Deccan Education Society's

Navinchandra Mehta Institute of Technology and Development

CERTIFICATE

This is to certify that Mr. **Nimesh Bharat Trivedi** of M.C.A. Semester III with Roll No. **C22127** has completed **All** practicals of MCAE332 **Deep Learning Lab** under my supervision in this college during the year 2022-2023.

CO	R1 (Attendance)	R2 (Performance during lab session)	R3 (Innovation in problem solving technique)	R4 (Mock Viva)	R5 (Variation in implementation of learnt topics on projects)
CO1					
CO2					
CO3					
CO4					

Practical-in-charge

Head of Department
MCA Department
(NMITD)

INDEX

Lab 1: Implement Feed Forward Neural Network for Handwritten Digit Recognition

Lab 1: Understanding Keras Deep Learning Library and Implementing a Simple NeuralNetwork for Handwritten Digit Recognition using Keras

Date: 27 September 2023

Problem Description: Using Keras build a Neural Network model and train it using MNIST dataset for doing Handwritten Digit Recognition.
Finally evaluate the performnance of the model on Test Dataset provided in MNIST

Step 1 - Setting up Keras and Numpy & Loading of MNIST dataset

```
[ ] import numpy as np
    import tensorflow as tf
    from tensorflow import keras
    from keras.datasets import mnist
    import matplotlib.pyplot as plt
```

Step 2 - Have a look at the data

```
[ ] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

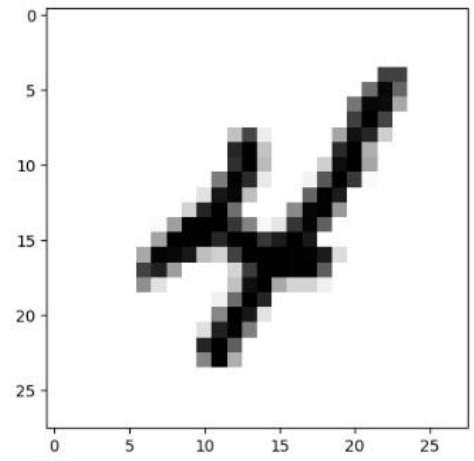
```
[ ] train_images.shape
```

```
(60000, 28, 28)
```

```
[ ] print(train_labels)
```

```
[5 0 4 ... 5 6 8]
```

```
[ ] #plot sample image from dataset
    digit = train_images[9]
    plt.imshow(digit, cmap=plt.cm.binary)
    plt.show()
    print("The image is of: ",train_labels[9])
```




```
The image is of:  4
```

Step 3 - Building the layered Neural Network Model for Handwritten Digit Recognition

```
[ ] from keras import models
    from keras import layers

    network = models.Sequential()
```

```
[ ]
    network = models.Sequential()
    network.add(layers.Dense(512, activation='relu', input_shape=(28*28,)))
    network.add(layers.Dense(10, activation='softmax'))
```

Step 4 -Compiling the Neural Network Model

```
[ ] network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

Step 5 - Input Data Preparation or Pre-Processing

```
[ ] train_images = train_images.reshape((60000,28*28))
    train_images = train_images.astype('float32')/255

    test_images = test_images.reshape((10000,28*28))
    test_images = test_images.astype('float32')/255

    print(train_labels[15])
    print(test_labels[15])
```

```
7
5
```

```
from keras.utils import to_categorical

train_labels=to_categorical(train_labels)
test_labels=to_categorical(test_labels)

print(train_labels[15])
print(test_labels[15])
```

```
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

Step 6 - Ready to train the network.

```
[ ] network.fit(train_images, train_labels, epochs = 5, batch_size=128)
```

```
Epoch 1/5
469/469 [==============================] - 4s 9ms/step - loss: 0.2657 - accuracy: 0.9230
Epoch 2/5
469/469 [==============================] - 5s 11ms/step - loss: 0.1064 - accuracy: 0.9688
Epoch 3/5
469/469 [==============================] - 4s 8ms/step - loss: 0.0708 - accuracy: 0.9788
Epoch 4/5
469/469 [==============================] - 4s 8ms/step - loss: 0.0518 - accuracy: 0.9844
Epoch 5/5
469/469 [==============================] - 6s 14ms/step - loss: 0.0385 - accuracy: 0.9889
<keras.src.callbacks.History at 0x7aa05b144970>
```

Step 7 - Evaluating the Performance of our Trained NN Model on Test Data

```
test_loss, test_acc = network.evaluate(test_images, test_labels)
print('Test Accuracy', test_acc)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0733 - accuracy: 0.9765
Test Accuracy 0.9764999747276306
```

Step 8 - The Final Step - Using the Trained and Tested Model to do actual work of Prediction

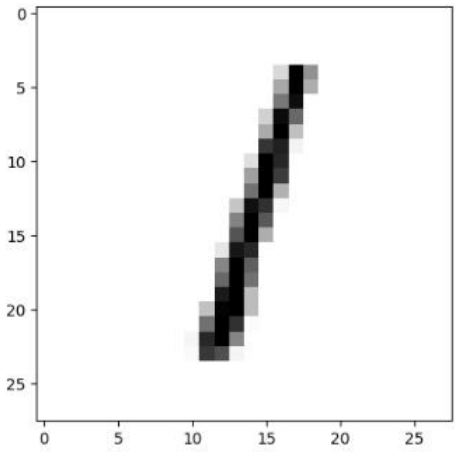
```
[ ] network.predict(test_images[2:3])
```

```
1/1 [==============================] - 0s 49ms/step
array([[1.7246172e-07, 9.9943465e-01, 6.7098219e-05, 2.7175661e-06,
        5.2490799e-05, 4.1399971e-06, 9.1766628e-07, 1.6501462e-04,
        2.7194276e-04, 8.2326329e-07]], dtype=float32)
```

```
[ ] network.predict(test_images[2:3])
```

```
1/1 [==============================] - 0s 49ms/step
array([[1.7246172e-07, 9.9943465e-01, 6.7098219e-05, 2.7175661e-06,
        5.2490799e-05, 4.1399971e-06, 9.1766628e-07, 1.6501462e-04,
        2.7194276e-04, 8.2326329e-07]], dtype=float32)
```

```
[ ] test_images_copy = test_images.reshape((10000,28,28))
    digit = test_images_copy[2]

    plt.imshow(digit, cmap=plt.cm.binary)
    plt.show()
```



Lab 2: Implement Tensors and Tensor Operations – Slicing, Broadcasting and Reshaping

Lab 2 - Implement Tensors and Tensor Operations – Slicing, Broadcasting and Reshaping

```
[1] from keras.datasets import mnist
    import numpy as np
    (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

Tensor Slicing Example 1:

```
[2] train_images.shape
    slice_images = train_images[15]
    # Select the 15th image from the dataset of 60,000 images
    slice_images.shape
```

```
(28, 28)
```

```
[3] # Selecting all images from 10th upto 100th (90 images)
    slice_images = train_images[10:100]
    slice_images.shape
```

```
(90, 28, 28)
```

```
[4] # this is equivalent to previous code
    slice_images = train_images[10:100,0:28, 0:28]
    slice_images.shape
```

```
(90, 28, 28)
```

Tensor Slicing - Exercise 1: Apply tensor slicing to get the 18 x 18 pixels from the bottom right corner of all images

```
[5] slice_images = train_images[:,10:28,10:28]
    slice_images.shape
```

```
(60000, 18, 18)
```

Tensor Slicing - Exercise 2: Apply tensor slicing to get the 18 x 18 pixels from the middle of all images

```
[6] slice_images = train_images[:,5:23,5:23]
    slice_images.shape
```

```
(60000, 18, 18)
```

Tensor Slicing - Exercise 3: Apply tensor slicing to select first three batches of images. Assume batch-size of 128

```
[7] slice_images = train_images[0:128,:,:]
    slice_images.shape
```

```
(128, 28, 28)
```

```
[8] slice_images = train_images[128:256,:,:]
    slice_images.shape
```

```
(128, 28, 28)
```

```
[9] slice_images = train_images[256:384,:,:]
    slice_images.shape
```

```
(128, 28, 28)
```

Exercise 4: Try to generalize the selection of batches such that if asked to get nth batch your generalized code can fetch the same. Assume

Exercise 4: Try to generalize the selection of batches such that if asked to get nth batch your generalized code can fetch the same. Assume that you get the batch# as a parameter

The smaller tensor is usually broadcasted to match the shape of the larger tensor. Broadcasting consists of two steps:

1. Axes are added to the smaller tensor to match the ndim of the larger tensor.
2. The smaller tensor is repeated along-side the new axes to match the full shape of the larger tensor

```
[10] n = 2
     slice_images = train_images[128*n:128*(n+1)]
     slice_images.shape
```

```
(128, 28, 28)
```

Topic 2 - Understanding Broadcasting

```
# Example 1 - Successful Broadcasting
# x is a 1 D tensor with shape (4,)
x = np.array([1,4,6,9])
x.ndim
```

```
1
```

```
[12] x.shape
```

```
(4,)
```

```
[13] # y is a matrix (i.e., a 2-D tensor) with shape (3,4)
     y = np.array([
         [24,57,96,99],
         [35,14,75,89],
```

0s completed at 11:11 PM

```
[13] # y is a matrix (i.e., a 2-D tensor) with shape (3,4)
     y = np.array([
         [24,57,96,99],
         [35,14,75,89],
         [15,26,49,72]
         ])
     y.ndim
```

```
2
```

```
[14] y.shape
```

```
(3, 4)
```

```
# They are actually not compatible for tensor operation of addition
# However because of implicit broadcasting the shape of the smaller tensor x will be matched with shape of y
x + y # successful because of broadcasting. Here due to broadcasting x is considered as:
# [[1,4,6,9],
# [1,4,6,9],
# [1,4,6,9]] for the tensor operation
np.array([
    [24,57,96,99],
    [35,14,75,89],
    [15,26,49,72]
    ])
```

```
array([[24, 57, 96, 99],
       [35, 14, 75, 89],
       [15, 26, 49, 72]])
```

```
[16] x.shape
```

```
(4,)
```

```
[17] # Example 2 - Illustrating how broadcasting might fail in certain cases
     # If the shape of tensors cannot be made compatible through broadcasting you will get an error
```

```
[18] # Here the smaller tensor p has shape (5,)
     # The larger tensor q has shape (3,4)
     # Thus they cannot be made compatible even through broadcasting.
     # Therefore we get error when we try to add them as seen below:
     p + q
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-18-b29edab970ef> in <cell line: 5>()
      3 # Thus they cannot be made compatible even through broadcasting.
      4 # Therefore we get error when we try to add them as seen below:
----> 5 p + q

ValueError: operands could not be broadcast together with shapes (5,) (3,4)
```

EXPLAIN ERROR

Reshaping a tensor means rearranging the elements in the tensor or rearranging its rows and columns to match a target shape. The reshaped tensor has the same no of elements as the original tensor. Below are some simple examples of tensor reshaping:

```
[19] x = np.array([[1,2,3],
                   [4,5,6]
                   ])
     x.shape
```

```
(2, 3)
```

```
[20] x = x.reshape((6,1))
     x
```

```
array([[1],
       [2],
       [3],
```

```
[20] x
```

```
array([[1],
       [2],
       [3],
       [4],
       [5],
       [6]])
```

```
[21] # Transpose operation is an example of reshaping
     x = np.array([[1,2,3],
                   [4,5,6]
                  ])
     print(x)
```

```
[[1 2 3]
 [4 5 6]]
```

```
[22] x = np.transpose(x)
     print(x)
```

```
[[1 4]
 [2 5]
 [3 6]]
```

```
x.shape
```

```
(3, 2)
```

Lab 3: Implement Multi-Layered Perceptron Network for Multi-Class Classification using Fashion MNIST Data

Lab 3 - Implement Multi-Layered Perceptron Network for Multi-Class Classification using Fashion MNIST Data

```
[1] import tensorflow as tf
    from tensorflow import keras
    import numpy as np
    import matplotlib.pyplot as plt
```

```
[2] fm_dataset = tf.keras.datasets.fashion_mnist
```

```
[3] (train_images, train_labels), (test_images, test_labels) = fm_dataset.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
```

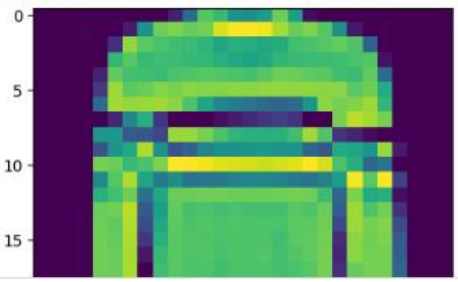
```
[4] print("Training data shape- ", train_images.shape)
    print("Training data lable- ", train_labels.shape)

    print("test data shape- ", test_images.shape)
    print("test data lable- ", test_labels.shape)

    print("Sample training image label " , train_labels[0])
```

```
Training data shape-  (60000, 28, 28)
Training data lable-  (60000,)
test data shape-  (10000, 28, 28)
test data lable-  (10000,)
Sample training image label  9
```

```
[5] # define text labels
    fashion_mnist_labels = ["T-shirt/top",
                            "Trouser",
                            "Pullover",
                            "Dress",
                            "Coat",
                            "Sandal",
                            "Shirt",
                            "Sneaker",
                            "Bag",
                            "Ankle boots",
                            ]
```

```
# Visualize some img by picking random img from 60,000 from training set
# any number between 0 to 59,999
i = 5

label = train_labels[i]
print("y = "+str(label)+" "+(fashion_mnist_labels[label]))

plt.imshow(train_images[i])
```

```
y = 2 Pullover
<matplotlib.image.AxesImage at 0x7d5524c70e50>
```



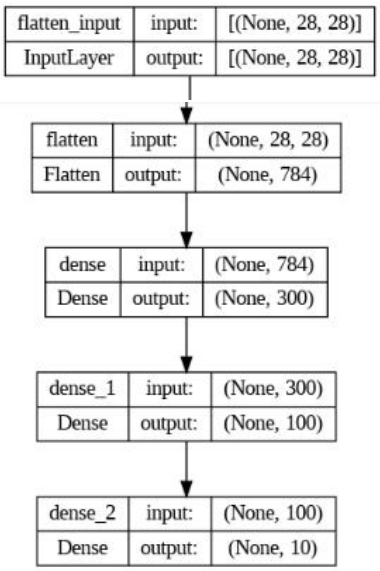
```
[7] # Step 1 : Defining our ANN
    # Total Params = 2,35,500 + 30,100 + 1010 = 2,66,610
    model = keras.models.Sequential()
    model.add(keras.layers.Flatten(input_shape=[28,28]))
    model.add(keras.layers.Dense(300, activation='relu')) #2,35,500 params
    model.add(keras.layers.Dense(100, activation='relu')) # 30,100 params
    model.add(keras.layers.Dense(10, activation='softmax')) # 1010 params
```

```
[8] model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 300)               235500

 dense_1 (Dense)             (None, 100)               30100

 dense_2 (Dense)             (None, 10)                1010

=================================================================
Total params: 266610 (1.02 MB)
Trainable params: 266610 (1.02 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
[9] from tensorflow.keras.utils import plot_model
```

```
plot_model(model, to_file="model_plot.png", show_shapes=True, show_layer_names=True)
```




```
[10]
```

```
[11] # Step 2: Compiling the defined Neural Network Model
     model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
```

```
[12] # Step 3: Training the Compiled NN Model Using Back-Propagation Method
     model.fit(train_images, train_labels, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 12s 6ms/step - loss: 1.7341 - accuracy: 0.7288
Epoch 2/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.5707 - accuracy: 0.7923
Epoch 3/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.4851 - accuracy: 0.8297
Epoch 4/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.4498 - accuracy: 0.8426
```

```
<keras.src.callbacks.History at 0x7d551c52c250>
```

```
[13] # Step 4 : Evaluating the trained model on Test Data
     test_loss, test_accuracy = model.evaluate(test_images, test_labels)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.4466 - accuracy: 0.8540
```

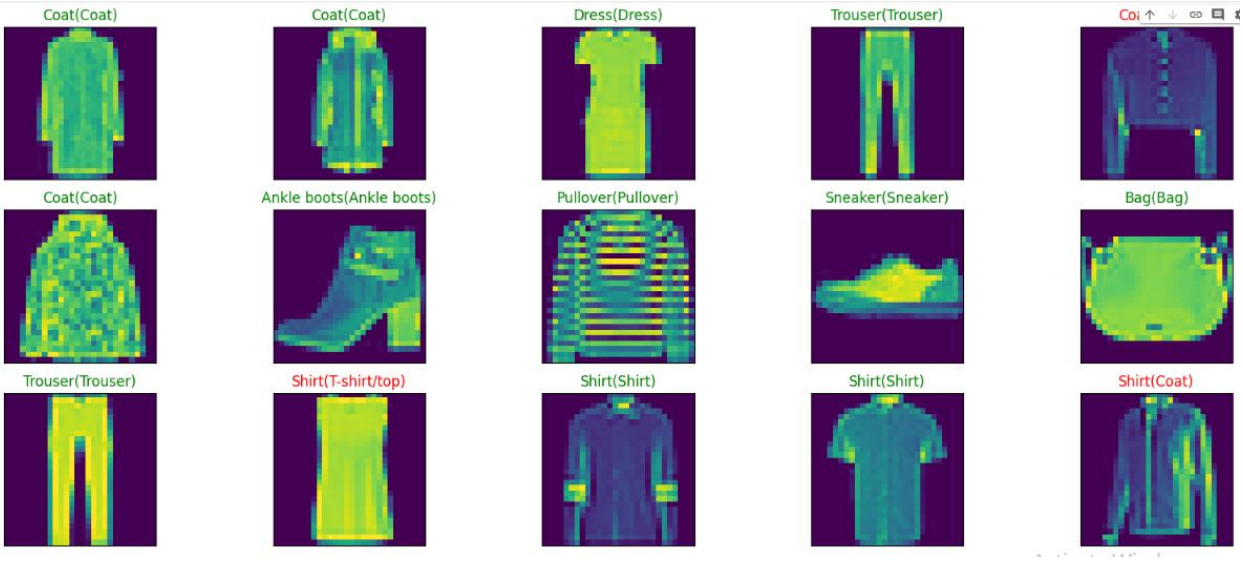
```
[14] # Print the test accuracy
     print('\n', 'Test Accuracy: ', test_accuracy)
```

```
 Test Accuracy:  0.8539999723434448
```

```
[15] result = np.random.choice(test_images.shape[0], size = 15, replace=False)
     y_predicted = model.predict(test_images)
     fashion_mnist_labels[np.argmax(y_predicted[2300])]
```

```
313/313 [==============================] - 1s 3ms/step
'Coat'
```

```
# Step 5 : Making Actual Predictions (Classification) on Real World Data
y_predicted = model.predict(test_images)
#Plot a random sample of 15 test images, their predicted labels and the ground truth (i.e. actual
figure = plt.figure(figsize=(20,8))
for i, index in enumerate(np.random.choice(test_images.shape[0], size = 15, replace=False)):
  ax = figure.add_subplot(3, 5, i + 1, xticks=[], yticks=[])
  # Display each image
  ax.imshow(np.squeeze(test_images[index]))
  predict_index = np.argmax(y_predicted[index])
  #true_index = np.argmax(test_labels[index])
  true_index = test_labels[index]
  # Set the title for each image
  ax.set_title("{}({})".format(fashion_mnist_labels[predict_index],
                               fashion_mnist_labels[true_index]),
                               color = "green" if predict_index == true_index else "red")
```



Lab 4: Demonstrate Data Exploratory & Preprocessing Techniques for Deep Learning

Lab 4: Implementing various Data Exploration and Preprocessing Techniques for Exploring and Preparing Data before using it to train Deep Learning Models

```
[1] import numpy as np # used for handling numbers
    import pandas as pd # to handle datasets
    from sklearn.impute import SimpleImputer # used for handling missing data
    from sklearn.preprocessing import LabelEncoder, OneHotEncoder # used for encoding categorical data
    from sklearn.model_selection import train_test_split # used for splitting training and testing data
    from sklearn.preprocessing import StandardScaler # used for feature scaling
```

```
[2] from google.colab import files
    uploaded = files.upload()
```

Choose Files DataPreprocessing.csv

- **DataPreprocessing.csv**(text/csv) - 236 bytes, last modified: 12/22/2023 - 100% done

Saving DataPreprocessing.csv to DataPreprocessing.csv

```
[3] import io
    df = pd.read_csv(io.BytesIO(uploaded['DataPreprocessing.csv']))
```

```
[4] df.shape
```

```
(11, 4)
```

```
[5] df
```

	Region	Age	Income	Online Shopper
0	India	49.0	86400.0	No
1	Brazil	32.0	57600.0	Yes

[5]

	Region	Age	Income	Online Shopper
0	India	49.0	86400.0	No
1	Brazil	32.0	57600.0	Yes
2	USA	35.0	64800.0	No
3	Brazil	43.0	73200.0	No
4	USA	45.0	NaN	Yes
5	India	40.0	69600.0	Yes
6	Brazil	NaN	62400.0	No
7	India	53.0	94800.0	Yes
8	USA	55.0	99600.0	No
9	India	42.0	80400.0	Yes
10	USA	55.0	99600.0	No

[6]
```
# Having a look at first 5 rows
df.head(5) # return only first 5 rows of the dataframe
```

	Region	Age	Income	Online Shopper
0	India	49.0	86400.0	No
1	Brazil	32.0	57600.0	Yes
2	USA	35.0	64800.0	No
3	Brazil	43.0	73200.0	No
4	USA	45.0	NaN	Yes

[7]
```
# Having a look at last 3 rows
df.tail(3) # return only last 3 rows of the dataframe
```

[7]

	Region	Age	Income	Online Shopper
8	USA	55.0	99600.0	No
9	India	42.0	80400.0	Yes
10	USA	55.0	99600.0	No

[8]
```
df.columns # returns the names of all columns
```

```
Index(['Region', 'Age', 'Income', 'Online Shopper'], dtype='object')
```

[9]
```
df.info() # to check the datatypes of columns and the no of (non-null) entries in each column
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11 entries, 0 to 10
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Region          11 non-null     object 
 1   Age             10 non-null     float64
 2   Income          10 non-null     float64
 3   Online Shopper  11 non-null     object 
dtypes: float64(2), object(2)
memory usage: 480.0+ bytes
```

[10]
```
# All non-numeric columns are by default omitted before summarising descriptive statistics for numerical columns
df.describe()
```

	Age	Income
count	10.00000	10.000000
mean	44.90000	78840.000000
std	8.07534	15677.244656
min	32.00000	57600.000000

[10]

25%	40.50000	66000.000000
50%	44.00000	76800.000000
75%	52.00000	92700.000000
max	55.00000	99600.000000

[11]
```
# Summarizing categorical variables
# Step 1 : to get only categorical columns extracted out
categorical = df.dtypes[df.dtypes == "object"].index
# Step 2: describe by indexing dataframe for only categorical values and applying describe() for those
df[categorical].describe()
```

	Region	Online Shopper
count	11	11
unique	3	2
top	India	No
freq	4	6

[12]
```
df["Region"].unique() #Display unique values of a particular column
```

```
array(['India', 'Brazil', 'USA'], dtype=object)
```

```
missing = np.where(df["Age"].isnull() == True)
missing
```

```
(array([6]),)
```

[14]
```
df.fillna(df.mean(), inplace=True)
df
```

[14]

	Region	Age	Income	Online Shopper
0	India	49.0	86400.0	No
1	Brazil	32.0	57600.0	Yes
2	USA	35.0	64800.0	No
3	Brazil	43.0	73200.0	No
4	USA	45.0	78840.0	Yes
5	India	40.0	69600.0	Yes
6	Brazil	44.9	62400.0	No
7	India	53.0	94800.0	Yes
8	USA	55.0	99600.0	No
9	India	42.0	80400.0	Yes
10	USA	55.0	99600.0	No

[15]
```
from google.colab import files
uploaded = files.upload()
```

```
Choose Files Salaries.csv
• Salaries.csv(text/csv) - 16257213 bytes, last modified: 12/22/2023 - 100% done
Saving Salaries.csv to Salaries.csv
```

[16]
```
# To store the data as a Pandas Dataframe
import io
df_sal = pd.read_csv(io.BytesIO(uploaded['Salaries.csv']))
df_sal.shape
```

```
<ipython-input-16-3afcd2d809c3>:3: DtypeWarning: Columns (3,4,5,6,12) have mixed types. Specify dtype option on import or set low_memory=False.
  df_sal = pd.read_csv(io.BytesIO(uploaded['Salaries.csv']))
(148654, 13)
```

```
[17] df_sal
```

	Id	EmployeeName	JobTitle	BasePay	OvertimePay	OtherPay	Benefits	TotalPay	TotalPayBenefits	Year	Notes	Agency	Status
0	1	NATHANIEL FORD	GENERAL MANAGER-METROPOLITAN TRANSIT AUTHORITY	167411.18	0.0	400184.25	NaN	567595.43	567595.43	2011	NaN	San Francisco	NaN
1	2	GARY JIMENEZ	CAPTAIN III (POLICE DEPARTMENT)	155966.02	245131.88	137811.38	NaN	538909.28	538909.28	2011	NaN	San Francisco	NaN
2	3	ALBERT PARDINI	CAPTAIN III (POLICE DEPARTMENT)	212739.13	106088.18	16452.6	NaN	335279.91	335279.91	2011	NaN	San Francisco	NaN
3	4	CHRISTOPHER CHONG	WIRE ROPE CABLE MAINTENANCE MECHANIC	77916.0	56120.71	198306.9	NaN	332343.61	332343.61	2011	NaN	San Francisco	NaN
4	5	PATRICK GARDNER	DEPUTY CHIEF OF DEPARTMENT,(FIRE DEPARTMENT)	134401.6	9737.0	182234.59	NaN	326373.19	326373.19	2011	NaN	San Francisco	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
148649	148650	Roy I Tillery	Custodian	0.00	0.00	0.00	0.00	0.00	0.00	2014	NaN	San Francisco	PT
148650	148651	Not provided	Not provided	Not Provided	Not Provided	Not Provided	Not Provided	0.00	0.00	2014	NaN	San Francisco	NaN
148651	148652	Not provided	Not provided	Not Provided	Not Provided	Not Provided	Not Provided	0.00	0.00	2014	NaN	San Francisco	NaN
148652	148653	Not provided	Not provided	Not Provided	Not Provided	Not Provided	Not Provided	0.00	0.00	2014	NaN	San Francisco	NaN
148653	148654	Joe Lopez	Counselor, Log Cabin Ranch	0.00	0.00	-618.13	0.00	-618.13	-618.13	2014	NaN	San Francisco	PT

148654 rows × 13 columns

```
[18] df_sal.describe() # statistical descriptive measures for numeric columns
```

	Id	TotalPay	TotalPayBenefits	Year	Notes
count	148654.000000	148654.000000	148654.000000	148654.000000	0.0
mean	74327.500000	74768.321972	93692.554811	2012.522643	NaN
std	42912.857795	50517.005274	62793.533483	1.117538	NaN
min	1.000000	-618.130000	-618.130000	2011.000000	NaN
25%	37164.250000	36168.995000	44065.650000	2012.000000	NaN
50%	74327.500000	71426.610000	92404.090000	2013.000000	NaN
75%	111490.750000	105839.135000	132876.450000	2014.000000	NaN
max	148654.000000	567595.430000	567595.430000	2014.000000	NaN

```
[19] # 1. Rescaling (Min-Max Normalization)
     ## -- simplest type of normalization where the features are rescaled to the range [0,1]
     # MinMax Normalization using MinMaxScaler in sklearn library
     from sklearn.preprocessing import MinMaxScaler
     scaler = MinMaxScaler()
     TotalPayReshaped = df_sal.TotalPay.values.reshape(-1,1)
     df_sal.TotalPay = scaler.fit_transform(TotalPayReshaped)
     # minimum is 0 and maximum is 1 for TotalPay column
     df_sal.describe()
```

	Id	TotalPay	TotalPayBenefits	Year	Notes
count	148654.000000	148654.000000	148654.000000	148654.000000	0.0
mean	74327.500000	0.132673	93692.554811	2012.522643	NaN
std	42912.857795	0.088905	62793.533483	1.117538	NaN
min	1.000000	0.000000	-618.130000	2011.000000	NaN
25%	37164.250000	0.064742	44065.650000	2012.000000	NaN
50%	74327.500000	0.126792	92404.090000	2013.000000	NaN

75%	111490.750000	0.187354	132876.450000	2014.000000	NaN
max	148654.000000	1.000000	567595.430000	2014.000000	NaN

```
[20] # 2.Standardization (Z-score normalization)
     ## -- Here the feature is scaled such that the resulting mean is zero and variance is one
     scaler = StandardScaler()
     TotalPayReshaped = df_sal.TotalPay.values.reshape(-1,1)
     df_sal.TotalPay = scaler.fit_transform(TotalPayReshaped)
     # Here the resulting mean is close to 0 and standard deviation is 1
     df_sal.describe()
```

	Id	TotalPay	TotalPayBenefits	Year	Notes
count	148654.000000	1.486540e+05	148654.000000	148654.000000	0.0
mean	74327.500000	-1.498959e-16	93692.554811	2012.522643	NaN
std	42912.857795	1.000003e+00	62793.533483	1.117538	NaN
min	1.000000	-1.492304e+00	-618.130000	2011.000000	NaN
25%	37164.250000	-7.640884e-01	44065.650000	2012.000000	NaN
50%	74327.500000	-6.615046e-02	92404.090000	2013.000000	NaN
75%	111490.750000	6.150586e-01	132876.450000	2014.000000	NaN
max	148654.000000	9.755700e+00	567595.430000	2014.000000	NaN

```
[21] df
```

	Region	Age	Income	Online Shopper
0	India	49.0	86400.0	No
1	Brazil	32.0	57600.0	Yes
2	USA	35.0	64800.0	No

```
[21]
```

4	USA	45.0	78840.0	Yes
5	India	40.0	69600.0	Yes
6	Brazil	44.9	62400.0	No
7	India	53.0	94800.0	Yes
8	USA	55.0	99600.0	No
9	India	42.0	80400.0	Yes
10	USA	55.0	99600.0	No

```
# encode categorical data
# particularly we apply label encoding to OnlineShopper column
le = LabelEncoder()
df['Online Shopper'] = le.fit_transform(df['Online Shopper'])
df
```

	Region	Age	Income	Online Shopper
0	India	49.0	86400.0	0
1	Brazil	32.0	57600.0	1
2	USA	35.0	64800.0	0
3	Brazil	43.0	73200.0	0
4	USA	45.0	78840.0	1
5	India	40.0	69600.0	1
6	Brazil	44.9	62400.0	0
7	India	53.0	94800.0	1
8	USA	55.0	99600.0	0
9	India	42.0	80400.0	1
10	USA	55.0	99600.0	0

```
[23] # encoding country (region) column using one hot encoding
     onehotencoder = OneHotEncoder()
     # reshape the 1-D country/region array to 2-D as fit_transform expects 2-D and finally fit the object
     X = onehotencoder.fit_transform(df.Region.values.reshape(-1,1)).toarray()
     X
```

```
array([[0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
[24] # counting unique places in Region
     n = len(pd.unique(df['Region']))
     n
```

```
3
```

```
# Adding X back into the dataframe with column names as Region1 Region2 and Region3
dfOneHot = pd.DataFrame(X, columns = ["Region_"+str(int(i)) for i in range(n)])
dfOneHot
```

	Region_0	Region_1	Region_2
0	0.0	1.0	0.0
1	1.0	0.0	0.0
2	0.0	0.0	1.0
3	1.0	0.0	0.0
4	0.0	0.0	1.0

```
[26] df = pd.concat([dfOneHot, df], axis = 1)
```

```
[27] # drop the region column
     df = df.drop(['Region'], axis=1)
     df
```

	Region_0	Region_1	Region_2	Age	Income	Online Shopper
0	0.0	1.0	0.0	49.0	86400.0	0
1	1.0	0.0	0.0	32.0	57600.0	1
2	0.0	0.0	1.0	35.0	64800.0	0
3	1.0	0.0	0.0	43.0	73200.0	0
4	0.0	0.0	1.0	45.0	78840.0	1
5	0.0	1.0	0.0	40.0	69600.0	1
6	1.0	0.0	0.0	44.9	62400.0	0
7	0.0	1.0	0.0	53.0	94800.0	1
8	0.0	0.0	1.0	55.0	99600.0	0
9	0.0	1.0	0.0	42.0	80400.0	1
10	0.0	0.0	1.0	55.0	99600.0	0

```
[28] # convert text to lower-case
     input_text = "The 5 Biggest countries population wise are China, India, Unites States, Indonesia, and Brazil as on 2017."
     input_text = input_text.lower()
     print(input_text)
```

```
the 5 biggest countries population wise are china, india, unites states, indonesia, and brazil as on 2017.
```

```
[29] # Remove numbers because they are not relevant to our analysis
     # We will use Regular Expressions for matching numbers within text and removing them
     import re
     input_text = 'Box A contains 12 balls whereas box B contains 6 balls. 7 out of 12 balls in Box A are red in colour'
[29] output_text = re.sub(r'\d+','',input_text)
     print(output_text)
```

```
Box A contains  balls whereas box B contains  balls.  out of  balls in Box A are red in colour
```

```
[30] # Remove Punctuation marks and special symbols
     # [!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~]:
     punctuation = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
     user_input = input("Enter a string: ")
     no_punc_input = ""
     for char in user_input:
       if char not in punctuation:
         no_punc_input = no_punc_input + char
     print("Punctuation free user input string : ", no_punc_input)
```

```
Enter a string: Hello!, What Text Do you want me to enter? I'm very excited.
Punctuation free user input string :  Hello What Text Do you want me to enter Im very excited
```

```
[31] # Remove whitespaces
     # Particularly stripping a text of its leading and trailing white spaces
     input_text = " \t a string example \t"
     input_text = input_text.strip()
     input_text
```

```
'a string example'
```

```
[32] # Remove stopwords
     !pip install -q wordcloud
```

```
[33] import wordcloud
     import nltk
     nltk.download("stopwords")
     nltk.download("wordnet")
     nltk.download("punkt")
     nltk.download("averaged_perceptron_tagger")
     nltk.download('omw-1.4')
     import matplotlib.pyplot as plt
```

```
[33] nltk.download('omw-1.4')
     import matplotlib.pyplot as plt
     import io
     import unicodedata
     import string
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
```

```
[34] from nltk.corpus import stopwords
     from nltk import word_tokenize
     example_sentence = "This is a sample sentence, showing off the stopwords filtration, You,re going to be amused. You all will be happy tha"
     stop_words = set(stopwords.words('english'))
     word_tokens = word_tokenize(example_sentence)
     filtered_sentence = []
     for w in word_tokens:
       if w not in stop_words:
         filtered_sentence.append(w)

     print(word_tokens)
     print(filtered_sentence)
```

```
['This', 'is', 'a', 'sample', 'sentence', ',', 'showing', 'off', 'the', 'stopwords', 'filtration', ',', 'You', ',', 're', 'going', 'to', 'be', 'amused', '.', 'You', 'all', 'will', 'be', 'happy', 'tha']
['This', 'sample', 'sentence', ',', 'showing', 'stopwords', 'filtration', ',', 'You', ',', 'going', 'amused', '.', 'You', 'happy', 'tha']
```

```
[35] # Stemming using NLTK
     # Stemming: process of reducing words to their stem or base form
     # Examples: books - book, looked - look, pens - pen
     stemmer = nltk.stem.PorterStemmer()
     input_text = 'There are several types of stemming algorithms'
     input_text = word_tokenize(input_text)
     for word in input_text:
       print(stemmer.stem(word))
```

```
there
are
sever
type
of
stem
algorithm
```

```
[36] # Lemmatization
     # to reduce inflectional forms to a common base form
     # As opposed to stemming, lemmatization does not simply chop off the inflections.
     # Instead it used lexical knowledge bases to get the correct base forms of words.
     lemmatizer = nltk.stem.WordNetLemmatizer()
     input_text = "been had languages cities mice"
     input_text = word_tokenize(input_text)
     for word in input_text:
       print(lemmatizer.lemmatize(word))
```

```
been
had
language
city
mouse
```

Lab 5: Demonstrate Regularization Technique of Early Stopping in Training Sentiment Classification Model for Movie Reviews

Lab 5: Classifying Movie Reviews Binary Classification and Demonstration of the Regularization Technique of Early Stopping

Date: 27 October 2023 Problem Description Use IMDB Movie Reviews Data for Sentiment Analysis (Binary) and Illustrate the use of Early Stopping to Avoid Overfitting In this lab you should train a model to predict the user sentiment (positive or negative) from his/her movie review. Its an instance of Binary Classification. Following are the main take-aways of this lab experiment It illustrates how to transform language text data to integers and integers to binary data. It illustrates how one can avoid overfitting to the training data by using Early Stopping while training the model For that purpose we divide the labelled moview review dataset into three parts: training set, validation set, and the test set We see for ourselves how Early Stopping helps us to have a model which performs well on test data. The resulting NN model generalizes better due to Early Stopping

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras.datasets import imdb
```

```
[24] (train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

```
[25] print(train_data[4]) # numbers 1, 2 and 3 are reserved for 'start', 'padding' and 'unknown'. They do not encode a english word
```

```
[1, 249, 1323, 7, 61, 113, 10, 10, 13, 1637, 14, 20, 56, 33, 2401, 18, 457, 88, 13, 2626, 1400, 45, 3171, 13, 70, 79, 49, 706, 919, 13, 16,
```

```
[26] print(train_labels[9]) # 1 denotes positive review and 0 denotes negative review
```

```
0
```

```
[27] max([max(sequence) for sequence in train_data]) # no word index will exceed 10,000
```

```
9999
```

```
[28] word_index = imdb.get_word_index()
     print(word_index)
```

```
{'fawn': 34701, 'tsukino': 52006, 'nunnery': 52007, 'sonja': 16816, 'vani': 63951, 'woods': 1408, 'spiders': 16115, 'hanging': 2345, 'woody':
```

```
[29] # Decoding reviews back to English words
     word_index = imdb.get_word_index()
     reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
     decoded_review = ' '.join([reverse_word_index.get(i-3,'?') for i in train_data[5]])
     print(decoded_review)
```

```
? begins better than it ends funny that the russian submarine crew ? all other actors it's like those scenes where documentary shots br br sp
```

```
print(word_index['this']+3,'',word_index['film']+3,'',word_index['was']+3,'',word_index['just']+3) # train_data has word-indices shfted 3
```

```
14  22  16  43
```

1 - Prepraring the data

You can't feed lists of integers into a neural network. You have to turn your lists into tensors. We will use One-Hot Encoding to do this. We will One-Hot Encode our lists into vectors of 0s and 1s. For instance the sequence [3,5] will turn into a 10,000 dimensional vector that would have all 0s except for indices 3 and 5, which would be 1s.

Doing this would allow us to use as the first layer in our network a Dense layer, capable of handling floating-point vector data.

[31]
```
# Encoding the integer sequences into a binary matrix
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
  results = np.zeros((len(sequences), dimension)) # Creates an all-zero matrix of shape (len(sequences), dimension)
  for i, sequence in enumerate(sequences):
    results[i, sequence] = 1. # Set specific indices of results[i] to 1s
  return results
```

[32]
```
x_train = vectorize_sequences(train_data) # vectorize training data
x_test = vectorize_sequences(test_data) # vectorize test data
x_train[0]
```

```
array([0., 1., 1., ..., 0., 0., 0.])
```

[33]
```
# Vectorizing our labels
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
y_test[0]
```

```
0.0
```

[34]
```
# The Three Layer Model Definition
from keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
```

[36]
```
x_val = x_train[:10000] # first 10000 reviews for validation
partial_x_train = x_train[10000:] # remaining 15000 reviews for actual training
y_val = y_train[:10000] # first 10000 labels for validation
partial_y_train = y_train[10000:] # remaining 15000 labels for actual training
```

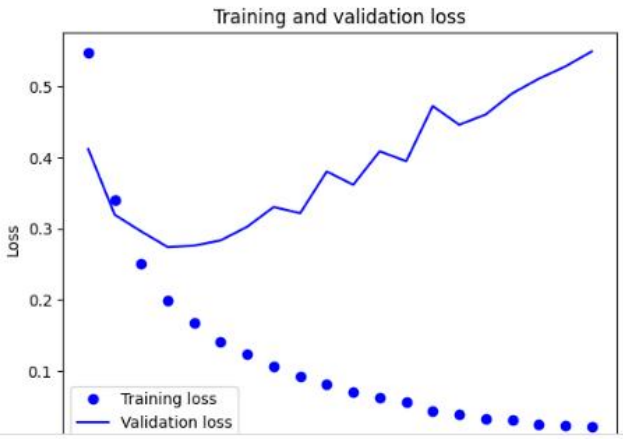
```
# Training our model
history = model.fit(partial_x_train, partial_y_train, epochs=20, batch_size=512, validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 [==============================] - 3s 66ms/step - loss: 0.5472 - accuracy: 0.7478 - val_loss: 0.4121 - val_accuracy: 0.8668
Epoch 2/20
30/30 [==============================] - 1s 33ms/step - loss: 0.3397 - accuracy: 0.8905 - val_loss: 0.3195 - val_accuracy: 0.8840
Epoch 3/20
30/30 [==============================] - 1s 49ms/step - loss: 0.2512 - accuracy: 0.9183 - val_loss: 0.2961 - val_accuracy: 0.8829
Epoch 4/20
30/30 [==============================] - 2s 57ms/step - loss: 0.1987 - accuracy: 0.9363 - val_loss: 0.2740 - val_accuracy: 0.8885
Epoch 5/20
30/30 [==============================] - 1s 43ms/step - loss: 0.1673 - accuracy: 0.9455 - val_loss: 0.2760 - val_accuracy: 0.8875
Epoch 6/20
30/30 [==============================] - 1s 33ms/step - loss: 0.1412 - accuracy: 0.9555 - val_loss: 0.2834 - val_accuracy: 0.8865
Epoch 7/20
30/30 [==============================] - 1s 33ms/step - loss: 0.1236 - accuracy: 0.9603 - val_loss: 0.3026 - val_accuracy: 0.8821
Epoch 8/20
30/30 [==============================] - 1s 33ms/step - loss: 0.1069 - accuracy: 0.9665 - val_loss: 0.3304 - val_accuracy: 0.8756
Epoch 9/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0924 - accuracy: 0.9731 - val_loss: 0.3216 - val_accuracy: 0.8810
Epoch 10/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0810 - accuracy: 0.9772 - val_loss: 0.3804 - val_accuracy: 0.8705
Epoch 11/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0698 - accuracy: 0.9814 - val_loss: 0.3616 - val_accuracy: 0.8790
Epoch 12/20
30/30 [==============================] - 1s 31ms/step - loss: 0.0621 - accuracy: 0.9831 - val_loss: 0.4087 - val_accuracy: 0.8691
Epoch 13/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0564 - accuracy: 0.9855 - val_loss: 0.3948 - val_accuracy: 0.8772
Epoch 14/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0432 - accuracy: 0.9904 - val_loss: 0.4724 - val_accuracy: 0.8626
Epoch 15/20
30/30 [==============================] - 1s 45ms/step - loss: 0.0389 - accuracy: 0.9911 - val_loss: 0.4461 - val_accuracy: 0.8727
```

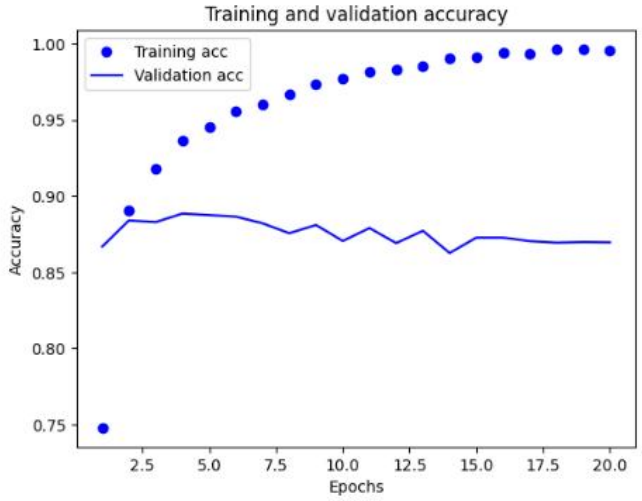
```
[38] history_dict.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
# Plotting the training and validation loss side by side using Matplotlib
import matplotlib.pyplot as plt
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, 'bo', label='Training loss') # "bo" stands for "blue dot"
plt.plot(epochs, val_loss_values, 'b', label='Validation loss') # "b" stands for "solod blue line"
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
[40] # Plotting the training and validation accuracy side by side using Matplotlib
plt.clf() # clears the figure
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Evaluating our model on Test Data We observed that during training the validation loss and accuracy started to increase after a point (they seemed to peak, i.e., be minimum, at the fourth epoch). In precise terms what we are seeing is overfitting: After the second epoch, we are overoptimizing on the training data, and end up learning representations that are specific to the training data and don't generalize to data outside the training set. Thus, to prevent overfitting, we could stop training after three epochs (one of the techniques to mitigate overfitting, known as early stopping) Below we retrain a new network from scratch for four epochs and then evaluate it on the test data

```
[41] model = models.Sequential()
     model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
     model.add(layers.Dense(16, activation='relu'))
     model.add(layers.Dense(1, activation='sigmoid'))
     model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
```

```
[42] # We are employing the strategy of Early Stopping
     model.fit(x_train, y_train, epochs=6, batch_size=512) # We are stopping early after only 4 epochs
```

```
Epoch 1/6
49/49 [==============================] - 2s 24ms/step - loss: 0.4586 - accuracy: 0.8199
Epoch 2/6
49/49 [==============================] - 1s 22ms/step - loss: 0.2724 - accuracy: 0.9012
Epoch 3/6
49/49 [==============================] - 1s 28ms/step - loss: 0.2145 - accuracy: 0.9212
Epoch 4/6
49/49 [==============================] - 2s 34ms/step - loss: 0.1835 - accuracy: 0.9334
Epoch 5/6
49/49 [==============================] - 2s 31ms/step - loss: 0.1612 - accuracy: 0.9424
Epoch 6/6
49/49 [==============================] - 1s 21ms/step - loss: 0.1433 - accuracy: 0.9502
<keras.src.callbacks.History at 0x7ec6253c5360>
```

```
[43] # Evaluating the new model on Test Data
     results = model.evaluate(x_test, y_test)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.3365 - accuracy: 0.8729
```

Using a trained network to generate predictions on new data After having trained a network, you'll want to use it in a practical setting. You can generate the likelihood of reviews being positive by using the predict method:

```
[44] model.predict(x_test)
```

```
782/782 [==============================] - 2s 2ms/step
array([[0.21605685],
       [0.99998784],
       [0.91671455],
       ...,
       [0.1553858 ],
       [0.07913048],
       [0.825039  ]], dtype=float32)
```

Lab 6: Implement Regularized Linear Regression Models

Lab 6 - Implementing Regularized Linear Regression Model

Problem Statement: Use Scikit Learn to implement Regularized Linear Regression Model that predicts average per capita life satisfaction index for a Country/Region given its GDP per capita value. Make use of OECD Better Life Index data along with IMF's GDP per capita data to train the Linear Regression Model.

First implement Ridge Regression Model and then implement Lasso Regression Model

```
## Code to understand how 'pivot' function of Pandas work
import pandas as pd
import numpy as np
df = pd.DataFrame({'foo':['one', 'one', 'one','two','two','two'],'bar': ['A', 'B', 'C', 'A','B','C'], 'baz':[1,2,3,4,5,6], 'zoo':['x','y','z','q','w','t']})
print('Original Dataframe')
print(df)
print('\n')
print('Reshaped Dataframe after pivot')
print(df.pivot(index='foo', columns='bar',values='baz'))
print('\n')
```

```
Original Dataframe
   foo bar  baz zoo
0  one   A    1   x
1  one   B    2   y
2  one   C    3   z
3  two   A    4   q
4  two   B    5   w
5  two   C    6   t


Reshaped Dataframe after pivot
bar  A  B  C
foo
one  1  2  3
two  4  5  6
```

The below function merges the OECD's life satisfaction data and the IMF's GDP per capita data

```
[2] def prepare_country_stats(oecd_bli, gdp_per_capita):
        oecd_bli = oecd_bli[oecd_bli["INEQUALITY"] == "TOT"]
        oecd_bli = oecd_bli.pivot(index="Country", columns="Indicator", values="Value")
        gdp_per_capita.rename(columns={"2015":"GDP per capita"},inplace=True)
        gdp_per_capita.set_index("Country", inplace=True)
        full_country_stats = pd.merge(left=oecd_bli, right=gdp_per_capita, left_index=True, right_index=True)
        full_country_stats.sort_values(by="GDP per capita", inplace=True)
        remove_indices = [0,1,6,8,33,34,35]
        keep_indices = list(set(range(36)) - set(remove_indices))
        return full_country_stats[["GDP per capita","Life satisfaction"]].iloc[keep_indices]
```

```
[3] import os
    datapath = os.path.join("datasets","lifesat","")
```

```
[4] # To plot pretty figures directly within Jupyter
    import matplotlib as mpl
    mpl.rc('axes',labelsize=14)
    mpl.rc('xtick', labelsize=12)
    mpl.rc('ytick', labelsize=12)
```

Step 1: Download the OECD life satisfaction csv and GDP per capita csv from the respective web sources.

```
[5] # Download the data
    import urllib.request
    DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
    os.makedirs(datapath, exist_ok=True)
    for filename in ("oecd_bli_2015.csv","gdp_per_capita.csv"):
        print("Downloading", filename)
        url = DOWNLOAD_ROOT + "datasets/lifesat/" + filename
        urllib.request.urlretrieve(url, datapath + filename)
```

Step 2: Load the csv files for OECD BLI Index data and IMF's GDP per capita data into respective pandas data frames.

Merge it into a single dataframe after pivoting the data having Country column as index.

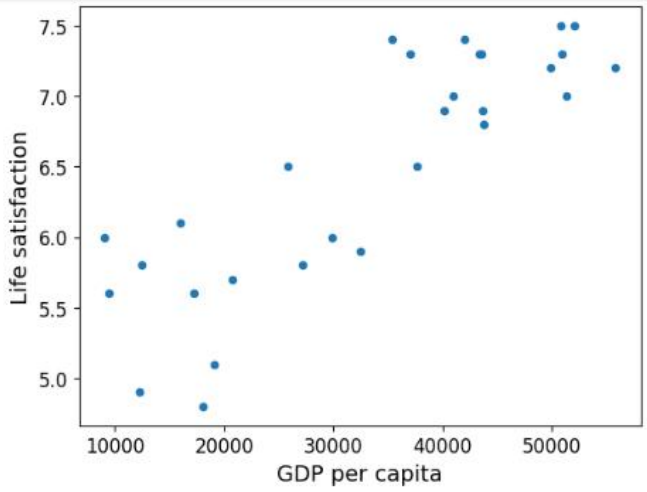
Visualize the correlation between GDP per capita with Life Satisfaction index using scatter plot.

Finally build, train and use the Scikit Learn's Linear Regression Model using the data.

```
# Code Example
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn.linear_model

# Load the data
oecd_bli = pd.read_csv(datapath + "oecd_bli_2015.csv", thousands=',')
gdp_per_capita = pd.read_csv(datapath + "gdp_per_capita.csv", thousands=',', delimiter='\t'
                             , encoding='latin1',na_values="n/a")

# Prepare the data
country_stats = prepare_country_stats(oecd_bli,gdp_per_capita)
X = np.c_[country_stats["GDP per capita"]]
y = np.c_[country_stats["Life satisfaction"]]

print(country_stats.head())

# Visualize the data
country_stats.plot(kind='scatter', x="GDP per capita", y="Life satisfaction")
```

```
                GDP per capita  Life satisfaction
Country
Russia                9054.914                6.0
Turkey                9437.372                5.6
Hungary              12239.894                4.9
Poland               12495.334                5.8
Slovak Republic      15991.736                6.1
<Axes: xlabel='GDP per capita', ylabel='Life satisfaction'>
```

[6]



Step 3

- Part 1: Implementing Ridge Regression

```
[7] # Select a ridge regression model
    from sklearn.linear_model import Ridge

    ridge_reg_model = Ridge(alpha=1, solver="cholesky")

    # Train the model
    ridge_reg_model.fit(X,y)
```

```
ridge_reg_model = Ridge(alpha=1, solver="cholesky")

# Train the model
ridge_reg_model.fit(X,y)

# Make a prediction for Cyprus
X_new = [[22587]] # Cyprus' GDP per capita
print(ridge_reg_model.predict(X_new)) # outputs [[5.96242338]]
```

```
[[5.96242338]]
```

Step 3

- Part 2: Implementing Lasso Regression

```
[8] # Select a ridge regression model
    from sklearn.linear_model import Lasso

    lasso_reg_model = Lasso(alpha=0.1)

    # Train the model
    lasso_reg_model.fit(X,y)

    # Make a prediction for Cyprus
    X_new = [[22587]] # Cyprus' GDP per capita
    print(lasso_reg_model.predict(X_new)) # outputs [[5.96242338]]
```

```
[5.96242859]
```

Lab 7: Implement Single-label Multi-class Topic Classification – Reuter's Newswire Dataset

Lab 7 - Implementing Single-Label, Multiclass Classification

Date: 01 Nov 2023

Problem Description: Build a network to classify Reuters newswires into 46 mutually exclusive topics.

The network should have an input layer, two hidden layer and one output layer. The hidden layers should use ReLU activation and have 64 units. The final layer should use Softmax as the activation function. Due to softmax layer the output layer will output 46 probability values.

You will use one-hot encoding (similar to what we used for IMDB Movie Reviews lab) to vectorize the training and test data into binary matrix of shape(dataset-size, 10,0000).

Even the training labels and test labels, which are originally integer labels should be converted into 46 dimensional binary vectors,using one-hot encoding (or categorical encoding). You may use, *to_categorical()* function from *keras.utils* module for this.

Use *categorical_crossentropy* as the loss function during training, and *rmsprop* as the optimizer, and *accuracy* as the monitoring metric.

During training phase, set apart initial 1000 training and test data as validation set for evaluation during the training. Train for 20 epochs with batch_size as 512.

Use the history datastructure to plot the *validation loss* and *training loss* in a single plot using matplotlib. Also do similar plotting for *validation accuracy* and *training accuracy*. Use these plots to identify the epoch for early stopping. Retrain a new neural network from scratch using all the training data for only this no of epochs to avoid overfitting.

Finally evaluate your regularized neural network model on the test data.

Note: Because there are many classes, this problem is an instance of *multi-class classification*. And because each data point should be classified into only one category, the problem is more specifically an instance of *single-label, multiclass classification*. If each data point could belong to multiple categories (in this case, topics), you'd be facing a *multilabel, multiclass classification* problem.

Step 1 - Load and Explore the Reuters Dataset

The Reuters dataset is a set of short newswires and their topics, published by Reuters in 1986.

It's a simple, widely used toy dataset for text classification.

There are 46 different topics; some topics are more represented than others, but each topic has at least 10 examples in the training set.

+ Code + Text

```
[1] from keras.datasets import reuters

    # the argument num_words=10000 restricts the data to the 10,000 most frequently occuring words found in the data
    (train_data, train_labels), (test_data, test_labels) = reuters.load_data(num_words=10000)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/reuters.npz
2110848/2110848 [==============================] - 0s 0us/step
```

```
[2] # The dataset has 8,982 training examples and 2,246 test examples
    print("training data size: ", len(train_data))
    print("test data size: ", len(test_data))
```

```
training data size:  8982
test data size:  2246
```

```
[3] # Each example is a list of integers (word indices)
    print(train_data[14])
```

```
[1, 4, 113, 23, 133, 6, 433, 226, 7, 1182, 407, 43, 2, 2, 69, 4, 7418, 5, 54, 2395, 521, 562, 6, 1681, 227, 108, 140, 645, 127, 108, 368, 8, 480
```

```
[4] # Decoding the examples back to words
    word_index = reuters.get_word_index()

    # print(word_index)

    reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])

    # print(reverse_word_index)
```

```
[4] reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])

    # print(reverse_word_index)

    # Note that the indices are offset by 3 because 0, 1, and 2 are reserved indices for "padding", "start of sequence", and "unknown".
    decoded_newswire = ' '.join([reverse_word_index.get(i - 3, '?') for i in train_data[10]])

    print(decoded_newswire)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/reuters_word_index.json
550378/550378 [==============================] - 0s 0us/step
? period ended december 31 shr profit 11 cts vs loss 24 cts net profit 224 271 vs loss 511 349 revs 7 258 688 vs 7 200 349 reuter 3
```

```
[5] # The labels associated with an example is an integer between 0 and 45 - topic index
    print(train_labels[14])
```

```
19
```

```
[5] Start coding or generate with AI.
```

Step 2 - Prepare the data

Here we will vectorize the data to transform each sample into a binary vector using one-hot encoding.

```
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
  results = np.zeros((len(sequences), dimension))
  for i, sequence in enumerate(sequences):
    results[i,sequence] = 1.
  return results
```

```
[7] x_train = vectorize_sequences(train_data) # vectorize training data
    x_test = vectorize_sequences(test_data) # vectorize test data
```

```
[8] # We will one hot encode the labels as well. One hot encoding is also known as categorical encoding
    # To one hot encode the labels, we will embed each integer label as an all-zero vector with a 1 in place of the label index

    # label is 3 --> 0001000000.....0
    # label is 46 --> 000000 .......1

    def to_one_hot(labels, dimension=46):
      results = np.zeros((len(labels),dimension))
      for i, label in enumerate(labels):
        results[i, label] = 1.
      return results
```

```
[9] one_hot_train_labels = to_one_hot(train_labels) # vectorize training labels
    one_hot_test_labels = to_one_hot(test_labels) # Vectorize test labels
```

```
[10] # Alternatively, we can use the built in function in Keras to do categorical (one hot) encoding
     from keras.utils import to_categorical

     one_hot_train_labels = to_categorical(train_labels)
     one_hot_test_labels = to_categorical(test_labels)
```

```
[11] print(one_hot_train_labels[10])
```

```
[0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
[12] # Vectorizing our labels
     y_train = np.asarray(one_hot_train_labels).astype('float32')
     y_test = np.asarray(one_hot_test_labels).astype('float32')
```

Step 3 - Building the Network

```
[13] # The Three Layer Model Definition
     from keras import models
     from keras import layers
     model = models.Sequential()
     model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
     model.add(layers.Dense(64, activation='relu'))
     model.add(layers.Dense(46, activation='softmax'))
```

```
[14] model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[15] x_val = x_train[:1000]
     partial_x_train = x_train[1000:]
     y_val = one_hot_train_labels[:1000]
     partial_y_train = one_hot_train_labels[1000:]

     print(partial_x_train)
```

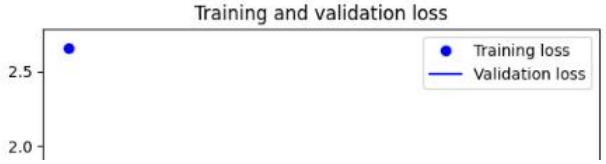
```
[[0. 1. 1. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]
 [0. 1. 1. ... 0. 0. 0.]
 ...
 [0. 1. 1. ... 0. 0. 0.]
 [0. 1. 1. ... 0. 0. 0.]
 [0. 1. 1. ... 0. 0. 0.]]
```

```
[16] history = model.fit(partial_x_train, partial_y_train, epochs=20, batch_size=512, validation_data=(x_val, y_val))

Epoch 1/20
16/16 [==============================] - 3s 113ms/step - loss: 2.6568 - accuracy: 0.5242 - val_loss: 1.8147 - val_accuracy: 0.6170
Epoch 2/20
16/16 [==============================] - 1s 83ms/step - loss: 1.5258 - accuracy: 0.6776 - val_loss: 1.3854 - val_accuracy: 0.6960
Epoch 3/20
16/16 [==============================] - 1s 90ms/step - loss: 1.1792 - accuracy: 0.7399 - val_loss: 1.2182 - val_accuracy: 0.7300
Epoch 4/20
16/16 [==============================] - 2s 132ms/step - loss: 0.9620 - accuracy: 0.7938 - val_loss: 1.1091 - val_accuracy: 0.7700
Epoch 5/20
16/16 [==============================] - 2s 130ms/step - loss: 0.7971 - accuracy: 0.8317 - val_loss: 1.0384 - val_accuracy: 0.7850
Epoch 6/20
16/16 [==============================] - 1s 46ms/step - loss: 0.1845 - accuracy: 0.9518 - val_loss: 0.9338 - val_accuracy: 0.8020
```

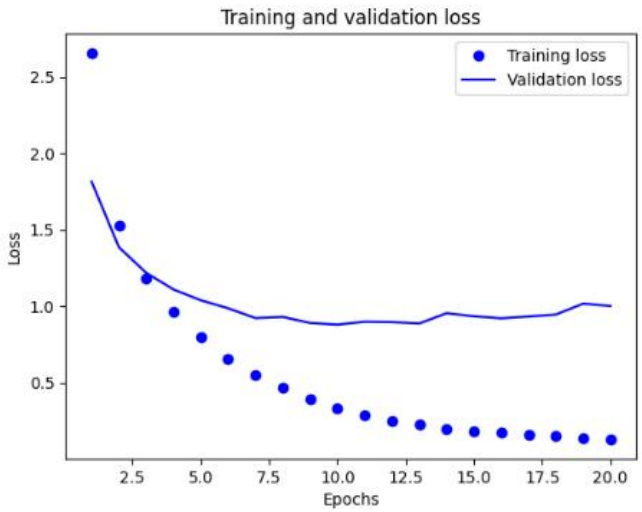
```
[16] Epoch 16/20
16/16 [==============================] - 1s 46ms/step - loss: 0.1700 - accuracy: 0.9525 - val_loss: 0.9204 - val_accuracy: 0.8230
Epoch 17/20
16/16 [==============================] - 1s 45ms/step - loss: 0.1550 - accuracy: 0.9531 - val_loss: 0.9325 - val_accuracy: 0.8180
Epoch 18/20
16/16 [==============================] - 1s 58ms/step - loss: 0.1475 - accuracy: 0.9548 - val_loss: 0.9451 - val_accuracy: 0.8150
Epoch 19/20
16/16 [==============================] - 1s 81ms/step - loss: 0.1391 - accuracy: 0.9557 - val_loss: 1.0162 - val_accuracy: 0.7970
Epoch 20/20
16/16 [==============================] - 1s 77ms/step - loss: 0.1291 - accuracy: 0.9575 - val_loss: 1.0013 - val_accuracy: 0.8060
```

```
[17] history_dict = history.history
     history_dict.keys()

dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
[18] # Plotting the training and validation loss side by side using Matplotlib
     import matplotlib.pyplot as plt
     loss_values = history_dict['loss']
     val_loss_values = history_dict['val_loss']
     epochs = range(1, len(loss_values) + 1)
     plt.plot(epochs, loss_values, 'bo', label='Training loss') # "bo" stands for "blue dot"
     plt.plot(epochs, val_loss_values, 'b', label='Validation loss') # "b" stands for "solod blue line"
     plt.title('Training and validation loss')
     plt.xlabel('Epochs')
     plt.ylabel('Loss')
     plt.legend()
     plt.show()
```



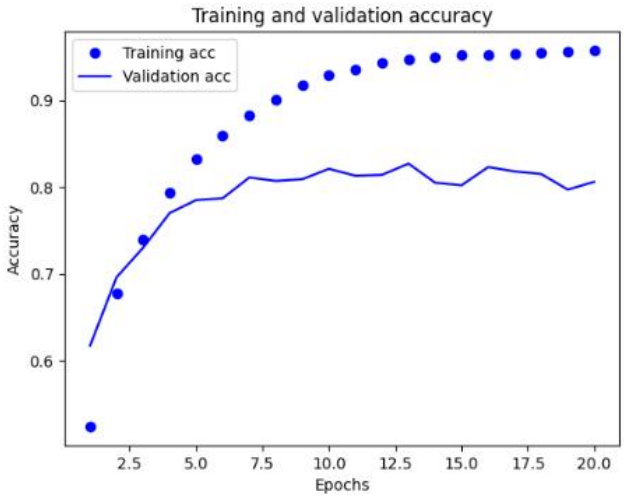
[18]



[19]

```
# Plotting the training and validation accuracy side by side using Matplotlib
plt.clf() # clears the figure
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

[19]




```
# Evaluating the performance on Test Dataset
results = model.evaluate(x_test, y_test)
```

```
71/71 [==============================] - 0s 3ms/step - loss: 1.0500 - accuracy: 0.7903
```

[21]

```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[21] model.add(layers.Dense(64, activation='relu'))
     model.add(layers.Dense(46, activation='softmax'))
     model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[22] model.fit(x_train, one_hot_train_labels, epochs=6, batch_size=512) # We are stopping early after only 4 epochs
```

```
Epoch 1/6
18/18 [==============================] - 1s 44ms/step - loss: 2.5631 - accuracy: 0.5327
Epoch 2/6
18/18 [==============================] - 1s 59ms/step - loss: 1.4496 - accuracy: 0.6880
Epoch 3/6
18/18 [==============================] - 1s 67ms/step - loss: 1.1299 - accuracy: 0.7506
Epoch 4/6
18/18 [==============================] - 1s 69ms/step - loss: 0.9205 - accuracy: 0.8008
Epoch 5/6
18/18 [==============================] - 1s 52ms/step - loss: 0.7730 - accuracy: 0.8304
Epoch 6/6
18/18 [==============================] - 1s 40ms/step - loss: 0.6385 - accuracy: 0.8607
<keras.src.callbacks.History at 0x7adfaddcd600>
```

```
# Evaluating the new model on Test Data
results = model.evaluate(x_test, y_test)
```

```
71/71 [==============================] - 0s 3ms/step - loss: 0.9875 - accuracy: 0.7716
```

```
[24] prediction = model.predict(x_test)
```

```
71/71 [==============================] - 0s 3ms/step
```

```
[25] np.argmax(prediction[0])
```

```
3
```

Lab 8: Implement Linear Regression Model to Predict House Prices – Boston House Price Dataset

Lab 8 - Training a Regression Model to Predict House Price using Boston Housing Dataset

Date: 22 November 2023

+ Code + Text

Problem Statement: Build and Train a Neural Network based Regression Model to predict the median price of houses in a given Boston suburb. Use The Boston Housing Price dataset for the purpose. The dataset contains house prices for houses from Boston suburb in the 1970s. Since the dataset size is relatively very less, use K-fold cross validation during the training phase. Apply appropriate feature scaling before inputting the data to the model. Finally evaluate the model on Test data and use the resulting trained and evaluated model to do the predictions on some data.

Step 1 - Data Exploration and Pre-processing

[1]
```
from keras.datasets import boston_housing

(train_data, train_targets), (test_data, test_targets) = boston_housing.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/boston_housing.npz
57026/57026 [==============================] - 0s 0us/step
```

[2]
```
train_data.shape
test_data.shape
```

```
(102, 13)
```

[3]
```
train_targets
```

```
array([15.2, 42.3, 50. , 21.1, 17.7, 18.5, 11.3, 15.6, 15.6, 14.4, 12.1,
       17.9, 23.1, 19.9, 15.7,  8.8, 50. , 22.5, 24.1, 27.5, 10.9, 30.8,
       32.9, 24. , 18.5, 13.3, 22.9, 34.7, 16.6, 17.5, 22.3, 16.1, 14.9,
       23.1, 34.9, 25. , 13.9, 13.1, 20.4, 20. , 15.2, 24.7, 22.2, 16.7,
       14.6, 13.4, 23.7, 50. , 21.7, 39.8, 38.7, 22.2, 34.9, 22.5, 31.1,
       28.7, 46. , 41.7, 21. , 26.6, 15. , 24.4, 13.3, 21.2, 11.7, 21.7,
       19.4, 50. , 22.8, 19.7, 24.7, 36.2, 14.2, 18.9, 18.3, 20.6, 24.6,
       18.2,  8.7, 44. , 10.4, 13.2, 21.2, 37. , 30.7, 22.9, 20. , 19.3,
       31.7, 32. , 23.1, 18.8, 10.9, 50. , 19.6,  5. , 14.4, 19.8, 13.8,
       19.6, 23.9, 24.5, 25. , 19.9, 17.2, 24.6, 13.5, 26.6, 21.4, 11.9,
       22.6, 19.6,  8.5, 23.7, 23.1, 22.4, 20.5, 23.6, 18.4, 35.2, 23.1,
       27.9, 20.6, 23.7, 28. , 13.6, 27.1, 23.6, 20.6, 18.2, 21.7, 17.1,
        8.4, 25.3, 13.8, 22.2, 18.4, 20.7, 31.6, 30.5, 20.3,  8.8, 19.2,
       19.4, 23.1, 23. , 14.8, 48.8, 22.6, 33.4, 21.1, 13.6, 32.2, 13.1,
       23.4, 18.9, 23.9, 11.8, 23.3, 22.8, 19.6, 16.7, 13.4, 22.2, 20.4,
       21.8, 26.4, 14.9, 24.1, 23.8, 12.3, 29.1, 21. , 19.5, 23.3, 23.8,
       17.8, 11.5, 21.7, 19.9, 25. , 33.4, 28.5, 21.4, 24.3, 27.5, 33.1,
       16.2, 23.3, 48.3, 22.9, 22.8, 13.1, 12.7, 22.6, 15. , 15.3, 10.5,
       24. , 18.5, 21.7, 19.5, 33.2, 23.2,  5. , 19.1, 12.7, 22.3, 10.2,
       13.9, 16.3, 17. , 20.1, 29.9, 17.2, 37.3, 45.4, 17.8, 23.2, 29. ,
       22. , 18. , 17.4, 34.6, 20.1, 25. , 15.6, 24.8, 28.2, 21.2, 21.4,
       23.8, 31. , 26.2, 17.4, 37.9, 17.5, 20. ,  8.3, 23.9,  8.4, 13.8,
        7.2, 11.7, 17.1, 21.6, 50. , 16.1, 20.4, 20.6, 21.4, 20.6, 36.5,
        8.5, 24.8, 10.8, 21.9, 17.3, 18.9, 36.2, 14.9, 18.2, 33.3, 21.8,
       19.7, 31.6, 24.8, 19.4, 22.8,  7.5, 44.8, 16.8, 18.7, 50. , 50. ,
       19.5, 20.1, 50. , 17.2, 20.8, 19.3, 41.3, 20.4, 20.5, 13.8, 16.5,
       23.9, 20.6, 31.5, 23.3, 16.8, 14. , 33.8, 36.1, 12.8, 18.3, 18.7,
       19.1, 29. , 30.1, 50. , 50. , 22. , 11.9, 37.6, 50. , 22.7, 20.8,
       23.5, 27.9, 50. , 19.3, 23.9, 22.6, 15.2, 21.7, 19.2, 43.8, 20.3,
       33.2, 19.9, 22.5, 32.7, 22. , 17.1, 19. , 15. , 16.1, 25.1, 23.7,
       28.7, 37.2, 22.6, 16.4, 25. , 29.8, 22.1, 17.4, 18.1, 30.3, 17.5,
       24.7, 12.6, 26.5, 28.7, 13.3, 10.4, 24.4, 23. , 20. , 17.8,  7. ,
       11.8, 24.4, 13.8, 19.4, 25.2, 19.4, 19.4, 29.1])
```

[4]
```
#feature selection
mean=train_data.mean(axis=0)
train_data -= mean

std = train_data.std(axis=0)
train_data /= std

test_data -= mean
test_data /= std
```

Step 2 - Build your network

Since we have less data we should build a relatively small model, say only with two hidden layers with 64 units per layer.

```
[5] from keras import models
    from keras import layers

    def build_model():
      model = models.Sequential()
      model.add(layers.Dense(64, activation="relu", input_shape=(train_data.shape[1],)))
      model.add(layers.Dense(64, activation="relu"))
      model.add(layers.Dense(1))
      model.compile(optimizer='rmsprop', loss='mse', metrics=["mae"])
      return model
```

Step 3 - Validating the model using K-fold cross validation

If we use a single fold cross validation, we will have to split training data into a training set and a validation set. But because we have very few datapoints (only 404), the validation set will have too less data points (may be less than 100). Thus, validation scores can be misleading varying with each sample you chose for training and validation. The best practice in such case is to use K-fold cross validation and then use the avergage validation of the k iterations. This is more reliable

```
[6] import numpy as np

    k = 5
    num_val_samples = len(train_data) // k
    num_epochs = 100
    all_scores = []
    for i in range(k):
      print(f"Processing fold number: {i}")
      val_data = train_data[i*num_val_samples:(i+1)*num_val_samples]
      val_targets = train_targets[i*num_val_samples:(i+1)*num_val_samples]
      partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
        train_data[(i+1) * num_val_samples:]],
      partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
        train_targets[(i+1) * num_val_samples:]],
        axis=0)
      model = build_model()
      model.fit(partial_train_data, partial_train_targets, epochs=num_epochs, batch_size=16, verbose=0)
      val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)
      all_scores.append(val_mae)
```

```
Processing fold number: 0
Processing fold number: 1
Processing fold number: 2
Processing fold number: 3
Processing fold number: 4
WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_test_function.<locals>.test_function at 0x78ae9ceac430> triggered tf.functi
```

```
[7] all_scores
    np.mean(all_scores)
```

```
2.3773757219314575
```

Increasing the number of epochs and capturing and saving the per epoch validation scores for each fold We will try to train the network a bit longer: for 500 epochs. To keep a record of how well the model does at each epoch, we will modify the training loop to save the per epoch validation socre loss

```
[8] num_epochs = 500
    all_mae_history = []
    for i in range(k) :
      print(f"Processing fold #{i}")
      val_data = train_data[i * num_val_samples : (i + 1) * num_val_samples]
      val_targets = train_targets[i * num_val_samples : (i + 1) * num_val_samples]
      partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
        train_data[(i+1) * num_val_samples:]],
        axis=0)
```

```
[8]  partial_train_targets = np.concatenate(
       [train_targets[:i * num_val_samples],
       train_targets[(i+1) * num_val_samples:]],
       axis=0)
     model = build_model()
     history = model.fit(partial_train_data, partial_train_targets, validation_data=(val_data, val_targets), epochs=num_epochs, batch_size=16, verbose=0)
     mae_history = history.history["val_mae"]
     all_mae_history.append(mae_history)
```

```
Processing fold #0
WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_test_function.<locals>.test_function at 0x78ae9ccf05e0> triggered tf.function retrac
Processing fold #1
Processing fold #2
Processing fold #3
Processing fold #4
```

Calculating the average of per epoch MAE score We will build a history of successive mean K-fold validation scores per epoch.

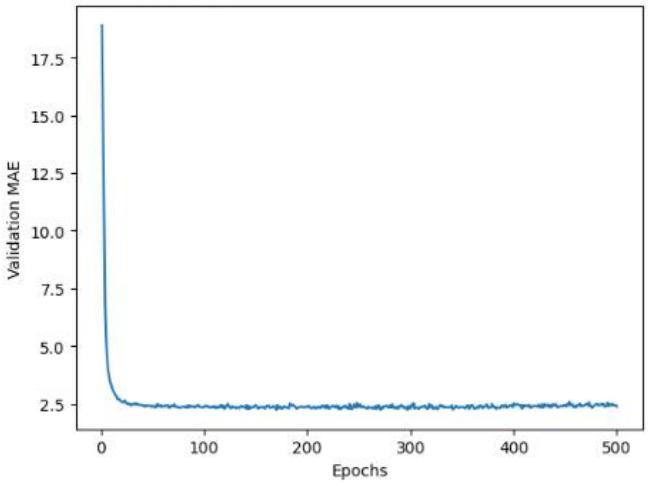
```
[9]  average_mae_history = [
     np.mean([x[i] for x in all_mae_history]) for i in range(num_epochs)]
```

```
[10] import matplotlib.pyplot as plt
     plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
     plt.xlabel("Epochs")
     plt.ylabel("Validation MAE")
```

```
Text(0, 0.5, 'Validation MAE')
```


```
[10] plt.xlabel("Epochs")
     plt.ylabel("Validation MAE")
```

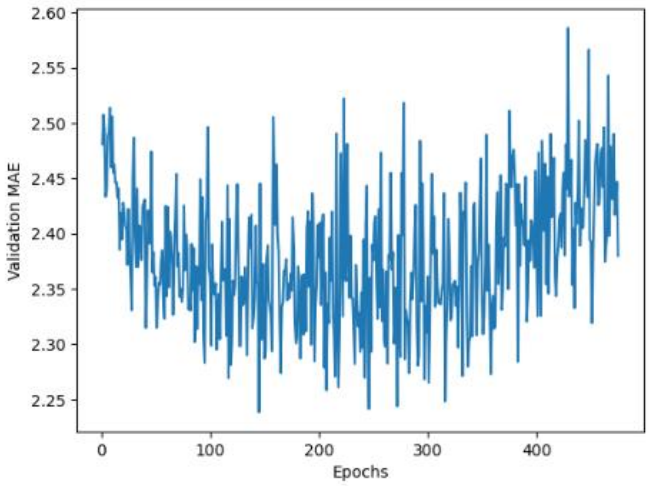
```
Text(0, 0.5, 'Validation MAE')
```




```
[11] truncated_mae_history = average_mae_history[25:]
     plt.plot(range(1, len(truncated_mae_history) + 1), truncated_mae_history)
     plt.xlabel("Epochs")
     plt.ylabel("Validation MAE")
```

```
Text(0, 0.5, 'Validation MAE')
```

[11]
```
Text(0, 0.5, 'Validation MAE')
```



Step 4: Training the final model from scratch with all the original training data

[12]
```
model = build_model()
model.fit(train_data, train_targets, epochs=150, batch_size=16, verbose=0)
test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)

test_mae_score
```

```
4/4 [==============================] - 0s 3ms/step - loss: 16.0834 - mae: 2.6397
2.639737367630005
```

Step 5: Generating predictions on new data

[13]
```
predictions = model.predict(test_data)
predictions[5]

test_targets[5]
```

```
4/4 [==============================] - 0s 3ms/step
24.5
```

Lab 9: Implement McCulloch-Pitts Neuron and Perceptron Models

Lab 9 - Artificial Neuron—McCulloch Pitts Neuron

[1]
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

import sklearn.datasets
```

[2]
```
cancer = sklearn.datasets.load_breast_cancer()

data = pd.DataFrame(cancer.data, columns=cancer.feature_names)

data["Class"] = cancer.target
data.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	0.6656
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238	0.1866
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444	0.4245
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098	0.8663
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374	0.2050

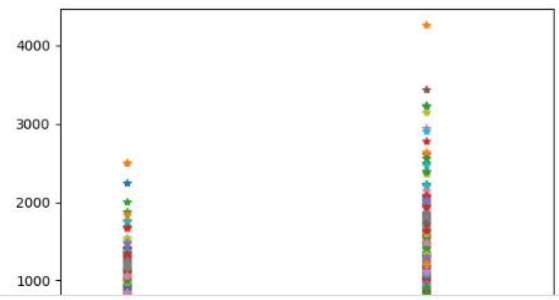
5 rows × 31 columns

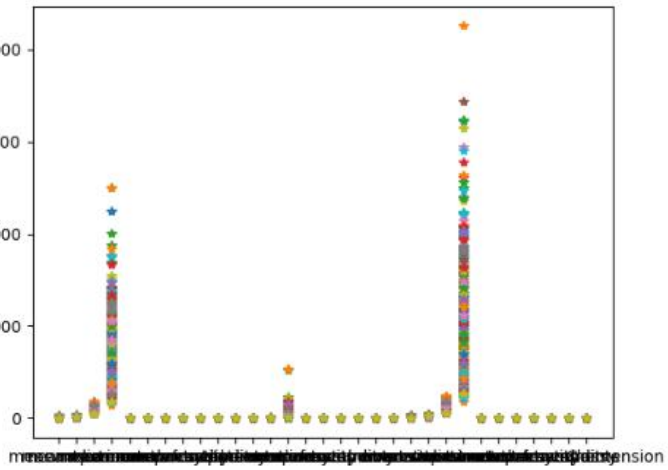
[3]
```
data.T.head()
```

	0	1	2	3	4	5	6	7	8	9	...	559	560	561	562
mean radius	17.9900	20.57000	19.6900	11.4200	20.2900	12.4500	18.25000	13.7100	13.0000	12.4600	...	11.51000	14.05000	11.20000	15.2200
mean texture	10.3800	17.77000	21.2500	20.3800	14.3400	15.7000	19.98000	20.8300	21.8200	24.0400	...	23.93000	27.15000	29.37000	30.6200
mean perimeter	122.8000	132.90000	130.0000	77.5800	135.1000	82.5700	119.60000	90.2000	87.5000	83.9700	...	74.52000	91.38000	70.67000	103.4000
mean area	1001.0000	1326.00000	1203.0000	386.1000	1297.0000	477.1000	1040.00000	577.9000	519.8000	475.9000	...	403.50000	600.40000	386.00000	716.9000
mean smoothness	0.1184	0.08474	0.1096	0.1425	0.1003	0.1278	0.09463	0.1189	0.1273	0.1186	...	0.09261	0.09929	0.07449	0.1048

5 rows × 569 columns

[4]
```
plt.plot(data.T,"*")
plt.show()
```






```
[5] x = data.drop('Class', axis=1)
    y = data['Class']

    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 3, stratify = y)
```

```
[6] print(type(x_train))
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
[7] x_train_binarized = x_train.apply(pd.cut, bins = 2, labels = [1,0]).values
    x_test_binarized = x_test.apply(pd.cut, bins = 2, labels = [1,0]).values
```

```
x_train_binarized = x_train.apply(pd.cut, bins = 2, labels = [1,0]).values
x_test_binarized = x_test.apply(pd.cut, bins = 2, labels = [1,0]).values
plt.figure(figsize=(15,5))
plt.plot(x_train_binarized.T,'*')
plt.xticks(rotation = 'vertical', c = 'white', size = 15)
plt.yticks(c = 'white', size = 15)
plt.show()
```

```
[8] x_train_binarized[:,0]
```

```
array([1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
[8] x_train_binarized[:,0]
```

```
array([1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0], dtype=object)
```

```
[9] class MP_Neuron:
      def __init__(self):
        self.b = 0

      def Model(self, x):
        return np.sum(x)>=self.b

      def fit(self, x, y):
        accuracy = {}

        for b in range(x.shape[1] + 1):
          self.b = b
          yhat = []
          for row in x:
            yhat.append(self.Model(row))
```

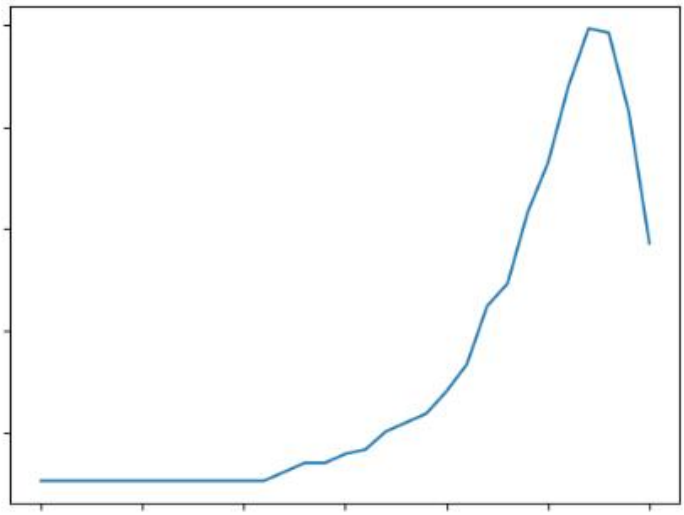
```
[9]       accuracy[b] = accuracy_score(yhat, y)
        best_b = max(accuracy, key = accuracy.get)
        self.b = best_b
        return [accuracy, best_b, accuracy[best_b]]

      def predict(self, x, y):
        yhat = []
        for row in x:
          yhat.append(self.Model(row))
        accuracy = accuracy_score(y, yhat)
        return accuracy
```

```
[10] neuron = MP_Neuron()
     accuracy, best_b, accuracy_model = neuron.fit(x_train_binarized,y_train)

     print("The optimal value of b is: ", best_b)
     print("accuracy_model on training data: ", accuracy_model*100)
```

```
The optimal value of b is:  27
accuracy_model on training data:  84.83516483516483
```

```
[11] accuracies = list(accuracy.values())
     plt.plot(accuracies)
     plt.xticks(c='white')
     plt.yticks(c='white')
     plt.show()
```

```
[11] plt.show()
```



```
accuracy = neuron.predict(x_test_binarized,y_test)
print("The accuracy of model on test data is :",accuracy*100)
```

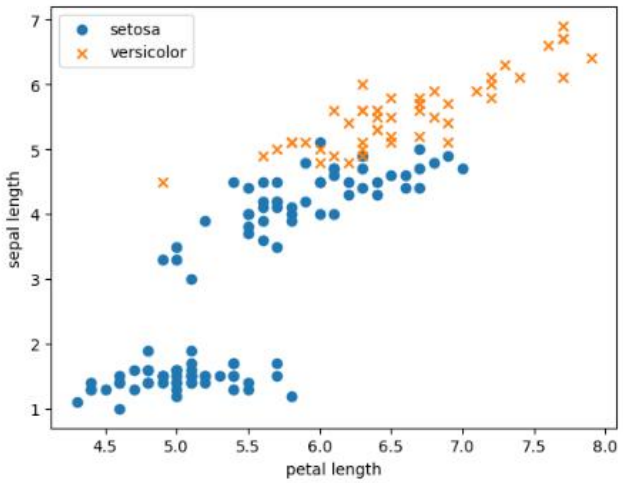
```
The accuracy of model on test data is : 87.71929824561403
```

Lab - 9 - Perceptron with Learning Algo

```
[ ] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    def load_data():
        URL_='https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
        data = pd.read_csv(URL_, header = None)
        print(data)

        # make the dataset linearly separable
        data = data[:200]
        data[4] = np.where(data.iloc[:, -1]=='Iris-setosa', 0, 1)
        data = np.asmatrix(data, dtype = 'float64')
        return data
    data = load_data()
```

```
       0    1    2    3              4
0    5.1  3.5  1.4  0.2    Iris-setosa
1    4.9  3.0  1.4  0.2    Iris-setosa
2    4.7  3.2  1.3  0.2    Iris-setosa
3    4.6  3.1  1.5  0.2    Iris-setosa
4    5.0  3.6  1.4  0.2    Iris-setosa
..   ...  ...  ...  ...            ...
145  6.7  3.0  5.2  2.3  Iris-virginica
146  6.3  2.5  5.0  1.9  Iris-virginica
147  6.5  3.0  5.2  2.0  Iris-virginica
148  6.2  3.4  5.4  2.3  Iris-virginica
149  5.9  3.0  5.1  1.8  Iris-virginica

[150 rows x 5 columns]
```

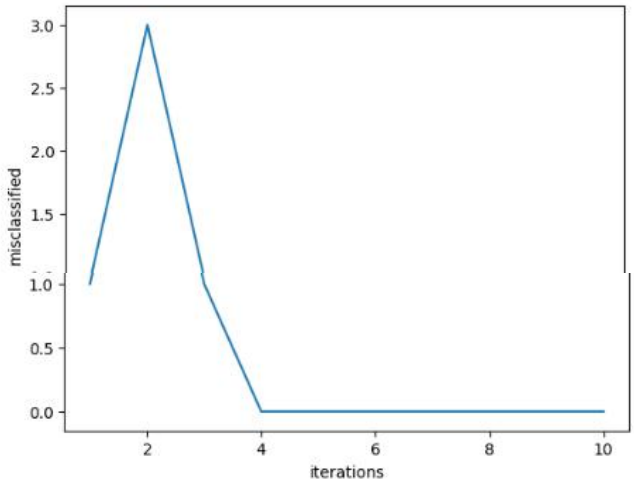
```
[ ] plt.scatter(np.array(data[:100,0]), np.array(data[:100,2]), marker='o', label='setosa')
    plt.scatter(np.array(data[100:,0]), np.array(data[100:,2]), marker='x', label='versicolor')
    plt.xlabel('petal length')
[ ] plt.ylabel('sepal length')
    plt.legend()
    plt.show()
```




```
[ ] def perceptron(data, num_iter):
        features = data[:, :-1]
        labels = data[:, -1]

        # set weights to zero
        w = np.zeros(shape=(1, features.shape[1]+1))

        misclassified_ = []

        for epoch in range(num_iter):
```

```
[ ]         misclassified = 0
            for x, label in zip(features, labels):
                x = np.insert(x,0,1)
                y = np.dot(w, x.transpose())
                target = 1.0 if (y > 0) else 0.0

                delta = (label.item(0,0) - target)

                if(delta): # misclassified
                    misclassified += 1
                    w += (delta * x)

            misclassified_.append(misclassified)
        return (w, misclassified_)

    num_iter = 10
    w, misclassified_ = perceptron(data, num_iter)
```

```
[ ] epochs = np.arange(1, num_iter+1)
    plt.plot(epochs, misclassified_)
    plt.xlabel('iterations')
    plt.ylabel('misclassified')
    plt.show()
```



Lab 10: Implement CNN Model to Classify Handwritten Digits in MNIST Dataset

Lab 10 - Implement CNN Model to Classify Handwritten Digits in MNIST Dataset

[1]
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras import layers
from keras import models
from tensorflow.keras.utils import to_categorical

from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

[2]
```
train_images.shape
```

```
(60000, 28, 28)
```

[3]
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64, (3,3), activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64, (3,3), activation='relu'))
```

[4]
```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

[5]
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2  (None, 13, 13, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 64)          36928

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 64)                36928

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 93322 (364.54 KB)
Trainable params: 93322 (364.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
[6] train_images = train_images.reshape((60000, 28, 28, 1))
    train_images = train_images.astype('float32') / 255

    test_images = test_images.reshape((10000, 28, 28, 1))
    test_images = test_images.astype('float32') / 255

    train_labels = to_categorical(train_labels)
    test_labels = to_categorical(test_labels)
```

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=6, batch_size=64)
```

```
Epoch 1/6
938/938 [==============================] - 53s 53ms/step - loss: 0.1748 - accuracy: 0.9456
Epoch 2/6
938/938 [==============================] - 57s 61ms/step - loss: 0.0475 - accuracy: 0.9854
Epoch 3/6
938/938 [==============================] - 48s 52ms/step - loss: 0.0316 - accuracy: 0.9905
Epoch 4/6
938/938 [==============================] - 49s 53ms/step - loss: 0.0236 - accuracy: 0.9926
Epoch 5/6
938/938 [==============================] - 51s 54ms/step - loss: 0.0183 - accuracy: 0.9943
Epoch 6/6
938/938 [==============================] - 48s 51ms/step - loss: 0.0148 - accuracy: 0.9953
<keras.src.callbacks.History at 0x7802b7916c50>
```

```
[8] test_loss, test_acc = model.evaluate(test_images, test_labels)
    test_acc
```

```
313/313 [==============================] - 3s 9ms/step - loss: 0.0278 - accuracy: 0.9932
0.9932000041007996
```

Lab 11: Implement RNN Model for Sentiment Analysis on IMDB Movie Reviews Dataset

Lab 11 - Implement RNN Model for Sentiment Analysis on IMDB Movie Reviews Dataset

```
[1] from keras.datasets import imdb
    from keras.preprocessing import sequence
    from keras.layers import Dense, Embedding, SimpleRNN
    from keras.models import Sequential
```

```
[2] max_features = 1200
    maxlen = 600
    batch_size = 32

    print("loading data....")

    (input_train, y_train), (input_test, y_test) = imdb.load_data(num_words = max_features)
    print(len(input_train), 'train sequence')
    print(len(input_test), 'tesst sequence')

    print('Pad sequence (simple x times)')

    input_train = sequence.pad_sequences(input_train, maxlen = maxlen)
    input_test = sequence.pad_sequences(input_test, maxlen = maxlen)

    print('input_train Shape:', input_train.shape)
    print('input_test Shape:', input_test.shape)
```

```
loading data....
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 1s 0us/step
25000 train sequence
25000 tesst sequence
Pad sequence (simple x times)
input_train Shape: (25000, 600)
input_test Shape: (25000, 600)
```

```
model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))

model.summary()

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(input_train, y_train, epochs = 10, batch_size = 128, validation_split = 0.2)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 32)          38400

 simple_rnn (SimpleRNN)      (None, 32)                2080

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 40513 (158.25 KB)
Trainable params: 40513 (158.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
157/157 [==============================] - 33s 199ms/step - loss: 0.6043 - acc: 0.6659 - val_loss: 0.4582 - val_acc: 0.8004
Epoch 2/10
157/157 [==============================] - 31s 197ms/step - loss: 0.4618 - acc: 0.7940 - val_loss: 0.4990 - val_acc: 0.7858
Epoch 3/10
157/157 [==============================] - 32s 207ms/step - loss: 0.4149 - acc: 0.8221 - val_loss: 0.4077 - val_acc: 0.8226
Epoch 4/10
157/157 [==============================] - 32s 202ms/step - loss: 0.3895 - acc: 0.8304 - val_loss: 0.4500 - val_acc: 0.7902
Epoch 5/10
```

```
=================================================================
Total params: 40513 (158.25 KB)
Trainable params: 40513 (158.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
157/157 [==============================] - 33s 199ms/step - loss: 0.6043 - acc: 0.6659 - val_loss: 0.4582 - val_acc: 0.8004
Epoch 2/10
157/157 [==============================] - 31s 197ms/step - loss: 0.4618 - acc: 0.7940 - val_loss: 0.4990 - val_acc: 0.7858
Epoch 3/10
157/157 [==============================] - 32s 207ms/step - loss: 0.4149 - acc: 0.8221 - val_loss: 0.4077 - val_acc: 0.8226
Epoch 4/10
157/157 [==============================] - 32s 202ms/step - loss: 0.3895 - acc: 0.8304 - val_loss: 0.4500 - val_acc: 0.7902
Epoch 5/10
157/157 [==============================] - 32s 202ms/step - loss: 0.3621 - acc: 0.8457 - val_loss: 0.4466 - val_acc: 0.7878
Epoch 6/10
157/157 [==============================] - 32s 204ms/step - loss: 0.3456 - acc: 0.8559 - val_loss: 0.4045 - val_acc: 0.8302
Epoch 7/10
157/157 [==============================] - 32s 202ms/step - loss: 0.3292 - acc: 0.8648 - val_loss: 0.4175 - val_acc: 0.8284
Epoch 8/10
157/157 [==============================] - 32s 206ms/step - loss: 0.3114 - acc: 0.8727 - val_loss: 0.4059 - val_acc: 0.8284
Epoch 9/10
157/157 [==============================] - 32s 207ms/step - loss: 0.3026 - acc: 0.8760 - val_loss: 0.4828 - val_acc: 0.7584
Epoch 10/10
157/157 [==============================] - 30s 193ms/step - loss: 0.2797 - acc: 0.8841 - val_loss: 0.4484 - val_acc: 0.8302
```

```
[4] test_loss, test_acc = model.evaluate(input_test, y_test)
    test_acc
```

```
782/782 [==============================] - 25s 32ms/step - loss: 0.4261 - acc: 0.8371
0.8370800018310547
```