WEEK-1

Outcome: Students are able to implement the basic logic gates using McCullough Pitt's Model.

Pre Lab:

1) What is Deep Learning and how deep learning became one of the powerful branches of machine learning?

A) Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers. These neural networks attempt to simulate the behaviour of the human brain—albeit far from matching its ability—allowing it to "learn" from large amounts of data. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help to optimise and refine for accuracy. Deep learning eliminates some of data pre-processing that is typically involved with machine learning. These algorithms can ingest and process unstructured data, like text and images, and it automates feature extraction, removing some of the dependency on human experts.

2) What is the Difference between Machine Learning and Deep learning?

A) Deep learning distinguishes itself from classical machine learning by the type of data that it works with and the methods in which it learns. Machine learning algorithms leverage structured, labelled data to make predictions—meaning that specific features are defined from the input data for the model and organised into tables. This doesn't necessarily mean that it doesn't use unstructured data; it just means that if it does, it generally goes through some pre-processing to organise it into a structured format. Machine learning is about computers being able to think and act with less human intervention; deep learning is about computers learning to think using structures modelled on the human brain. Machine learning requires less computing power; deep learning typically needs less ongoing human intervention.

3) Explain mankind's first mathematical model of a biological neuron and linear separability.

A) The first computational model of a neuron was proposed by Warren McCulloch (neuroscientist) and Walter Pitts (logician) in 1943 i.e., McCulloch-Pitts Neuron. The first part, g takes an input (ahem dendrite ahem), performs an aggregation and based on the aggregated value the second part, f makes a decision.

In Lab:
EXP1:

a) Implement the basic logic gates AND & OR using McCullough Pit s model.
b) Draw the linear separability line for the above Boolean functions.

Program:

```
'''
a)  Implement the basic logic gates AND & OR using McCullough Pit s model.

b)  Draw the linear separability line for the above Boolean functions.
'''
```

```
import numpy as np
import matplotlib.pyplot as plt
from itertools import product
```

```
inp = np.random.choice([0, 1], 3)
bias = 2
print('inputs: ', inp, "\nbias: ", bias)
```

```
inputs:  [0 1 1]
bias:  2
```

```
dot_product = np.dot(inp, bias)
print(dot_product)
```

```
[0 2 2]
```

```
combinations = get_all_binary_combinations(2)
threshold = float('inf')

firing_cases = []
non_firing_cases = []

for point in combinations:
```

```
    and_value = logical_and(point)
    if(and_value == True and sum(point) < threshold):
        threshold = sum(point)
```

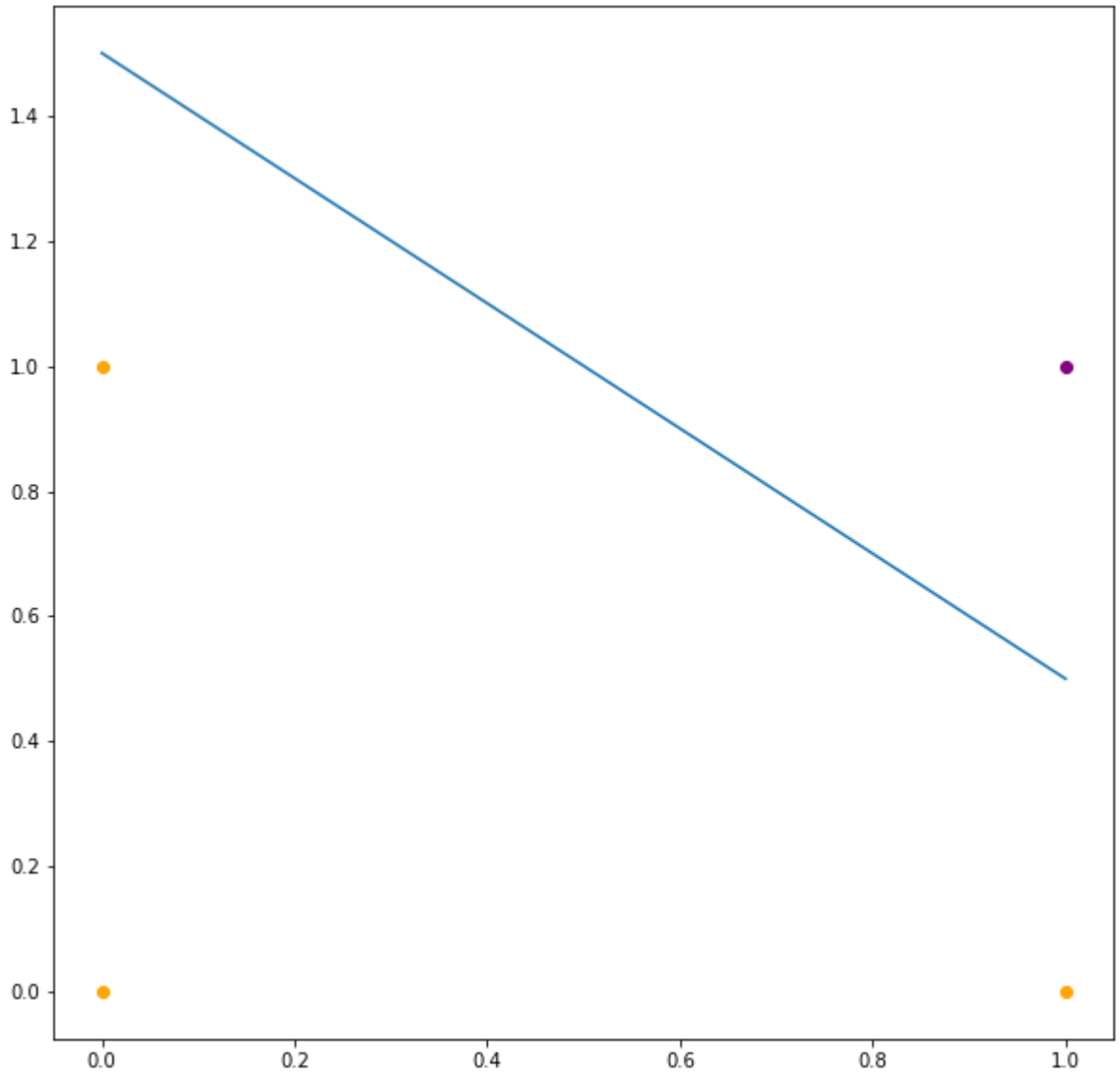
```
    print(point, and_value)

for point in combinations:

    if(sum(point) >= threshold):
        firing_cases.append(point)
    else:
        non_firing_cases.append(point)
```

```
(0, 0) False
(0, 1) False
(1, 0) False
(1, 1) True
```

```
plt.figure(figsize=(10, 10))

for point in firing_cases:

    plt.scatter(x=point[0], y = point[1], color='purple')

for point in non_firing_cases:

    plt.scatter(x=point[0], y = point[1], color='orange')


x = np.linspace(0, 1, 10)
print(x)
y = 1.5 - x
plt.plot(x, y)

plt.show()
```

Output:

```
[0.         0.11111111 0.22222222 0.33333333 0.44444444 0.55555556
 0.66666667 0.77777778 0.88888889 1.        ]
```


1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Post Lab:

Solve the given polynomial equation using TensorFlow X^2 -4X+4=0

Program:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import cmath
import time
#evaluated using Horner's method,

#p(x) = coeffs[n-1] + x * (coeffs[n-2] + ... + x * (coeffs[1] + x * coeffs[0]))
```

```
#Solve the given polynomial equation using Tensorflow X^2 -4X+4=0
import numpy as np
coeffs = [1, -4, 4]
print("coeffs: ", coeffs)

roots =np.roots(coeffs)
print(roots)

x = 5
res = tf.math.polyval(coeffs, x)

# Printing the result
print('Result: ', res)
```

Output:

coeffs: [1, -4, 4]

[2. 2.]

Result: tf.Tensor(9, shape=(), dtype=int32)

WEEK-2

Outcome: Students are able to implement a linear classifier using multilayer perceptron in TensorFlow.

Pre Lab:

1) **What are differences between Single layer perceptron and multilayer perceptron?**

A)

A single-layer neural network represents the most simple form of neural network, in which there is only one layer of input nodes that send weighted inputs to a subsequent layer of receiving nodes, or in some cases, one receiving node. This single-layer design was part of the foundation for systems which have now become much more complex. A multilayer perceptron (MLP) is a class of feedforward artificial neural network (ANN).

A Multi Layer Perceptron (MLP) contains one or more hidden layers (apart from one input and one output layer). While a single layer perceptron can only learn linear functions, a multi layer perceptron can also learn non – linear functions

2) **Define Delta Learning Rule?**

A) The learning rule is a technique or a mathematical logic which encourages a neural network to gain from the existing condition and uplift its performance. It is an iterative procedure. The delta rule in an artificial neural network is a specific kind of backpropagation that assists in refining the machine learning/artificial intelligence network, making associations among input and outputs with different layers of artificial neurons. Delta rule is introduced by Widrow and Hoff, which is the most significant learning rule that depends on supervised learning. This rule states that the change in the weight of a node is equivalent to the product of error and the input.

The given equation gives the mathematical equation for delta learning rule:

$\Delta w = \mu.x.z$

$\Delta w = \mu(t-y)x$

Δw = weight change; μ = the constant and positive learning rate;X = the input value from presynaptic neuron.z= (t-y) is the difference between the desired input t and the actual output y.

3) **Briefly explain about the impact of back propagation in multi layer perceptrons.**

A) Backpropagation is an algorithm used in artificial intelligence (AI) to fine-tune mathematical weight functions and improve the accuracy of an artificial neural network's outputs. Backpropagation, short for "backward propagation of errors," is an algorithm for supervised learning of artificial neural networks using gradient descent. Given an artificial neural network and an error function, the method calculates the gradient of the error function with respect to the neural network's weights. It is a

generalisation of the delta rule for perceptrons to multilayer feedforward neural networks

In Lab:

EXP2:

a) Implement a linear classifier (binary) for the given input data using multi multi-layer perceptron using TensorFlow.

b) John Successfully implemented AND & OR gates using single layer perceptron but was unable to implement XOR Gate. He got to know that it can be implemented by a multi-layer perceptron using TensorFlow.

Program:

A)

```
# %%
#a) Implement a linear classifier (binary) for the given input data using
# multi layer perceptron using TensorFlow.

# %%
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow import keras
```

```
# %%
df = pd.read_csv("diabetes.csv")
df
```

```
# %%
#1) First n samples of data
df.head(5)
```

```
# %%
#2) Last n samples of data
df.tail(5)
```

```
# %%
#3) Give the number of non-empty values in data
df.count()
```

```
Pregnancies                 2000
Glucose                     2000
BloodPressure               2000
SkinThickness               2000
Insulin                     2000
BMI                         2000
DiabetesPedigreeFunction    2000
Age                         2000
Outcome                     2000
dtype: int64
```

```
# %%
#4) information about the Data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               2000 non-null   int64
 1   Glucose                   2000 non-null   int64
 2   BloodPressure             2000 non-null   int64
 3   SkinThickness             2000 non-null   int64
 4   Insulin                   2000 non-null   int64
 5   BMI                       2000 non-null   float64
 6   DiabetesPedigreeFunction  2000 non-null   float64
 7   Age                       2000 non-null   int64
```

```
 8   Outcome                   2000 non-null   int64

dtypes: float64(2), int64(7)

memory usage: 140.8 KB
```

```
# %%
#5) Basic statistical details
df.describe()
```

```
Pregnancies                 0

Glucose                     0

BloodPressure               0

SkinThickness               0

Insulin                     0

BMI                         0

DiabetesPedigreeFunction    0

Age                         0

Outcome                     0

dtype: int64
```

```
# %%
#6) Check any samples to be null
df.isnull().sum()
```

```
# %%
#7) dimensions of the dataset
df.shape
```

```
# %%
#8a) Standardise the dataset
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df.drop('Outcome',axis=1))
scaled_features = scaler.transform(df.drop('Outcome',axis=1))
df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])
df_feat.head()
```

```
# %%
#8b) Standardise the dataset without using the sklearn library and
standardscaler
df2=(df-df.mean())/df.std()
df2
```

```
# %%
# 8c) Normalise the dataset
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(df.drop('Outcome',axis=1))
scaled_features = scaler.transform(df.drop('Outcome',axis=1))
df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])
df_feat.head()
```

```
'''
def normalise_dataset(data):
   return (data - data.min()) / (data.max() - data.min())


normalise_dataset(diabetes)
'''
```

```
# %%
'''
#9) Split train test into 80-20 ratio without train_test_split


def split_train_test_without_train_test_split(data, test_size):
   return data[:int(len(data) * (1 - test_size))], data[int(len(data) * (1 -
test_size)):] # train, test


split_train_test_without_train_test_split(diabetes, 0.2)

'''
```

```
# %%
#9) Split train test into 80-20 ratio
x = df.drop(df[["Outcome"]], axis = 1)
y = df["Outcome"]
```

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

```
# %%
class PrintDot(keras.callbacks.Callback):

   def on_epoch_end(self, epoch, logs):

       if epoch % 50 == 0:
           print(' ')
       print('.', end='')
```

```
#10)Using a single layer of 8 sigmoid neurons classify the dataset and show
the accuracy
# [use optimizer of your wish]

model = Sequential()
model.add(Dense(128, input_shape=(8,), activation=tf.nn.sigmoid))
model.add(Dense(64, activation=tf.nn.sigmoid))
model.add(Dense(1, activation=tf.nn.sigmoid))
```

```
# %%
model.summary()
```

```
# %%
#11) Also plot the loss and validation loss along with the estimates for the
test set and
# accuracy for the test set.

model.compile(loss='binary_crossentropy', optimizer='adam',
metrics='accuracy')
```

```
# %%
model.fit(x_train, y_train, epochs=150, batch_size=20,
callbacks=[PrintDot()])

# %%

accuracy = model.evaluate(x, y)
```

```
13/13 [==============================] - 0s 2ms/step - loss: 0.4494 -
accuracy: 0.7875
```

B) John Successfully implemented AND & OR gates using single layer perceptron but was unable to implement XOR Gate. He got to know that it can be implemented by multi layer perceptron using Tensor Flow.

```
#XOR Gate using Multi Layer Perceptron and Tensor Flow

import numpy as np
import tensorflow.compat.v1 as tf
import matplotlib.pyplot as plt

num_features = 2
num_iter = 10000
display_step = int(num_iter / 10)
learning_rate = 0.01

num_input = 2          # units in the input layer 28x28 images
num_hidden1 = 2        # units in the first hidden layer
num_output = 1         # units in the output, only one output 0 or 1
```

```
def multi_layer_perceptron_xor(x, weights, biases):

   hidden_layer1 = tf.add(tf.matmul(x, weights['w_h1']), biases['b_h1'])
   hidden_layer1 = tf.nn.sigmoid(hidden_layer1)

   out_layer = tf.add(tf.matmul(hidden_layer1, weights['w_out']),
biases['b_out'])

   return out_layer
```

```
x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], np.float32)  # 4x2, input
y = np.array([0, 1, 1, 0], np.float32)                      # 4, correct
output, AND operation
y = np.reshape(y, [4,1])                                    # convert to 4x1

# trainum_inputg data and labels
X = tf.placeholder('float', [None, num_input])     # training data
Y = tf.placeholder('float', [None, num_output])
```

```
# weights and biases
weights = {
   'w_h1' : tf.Variable(tf.random_normal([num_input, num_hidden1])), # w1,
from input layer to hidden layer 1
   'w_out': tf.Variable(tf.random_normal([num_hidden1, num_output])) # w2,
from hidden layer 1 to output layer
}
biases = {
```

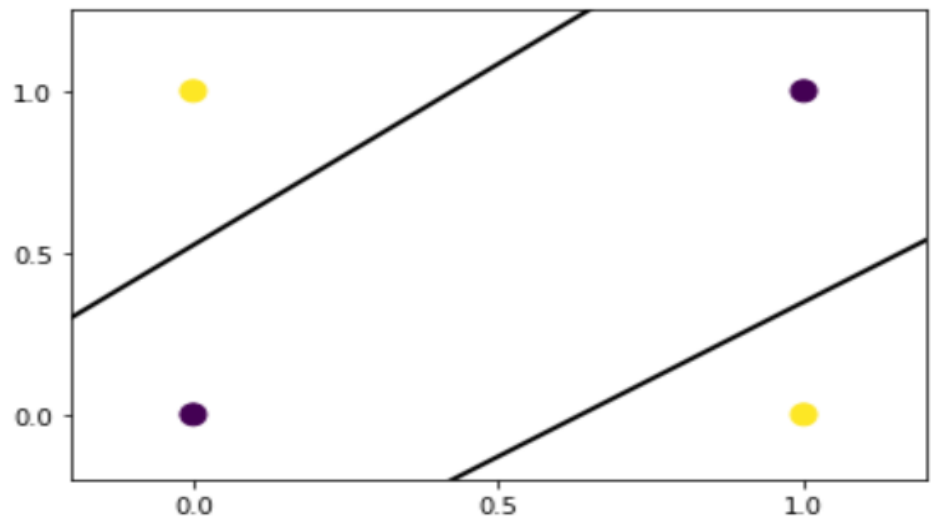
```
    'b_h1' : tf.Variable(tf.zeros([num_hidden1])),
    'b_out': tf.Variable(tf.zeros([num_output]))
}

model = multi_layer_perceptron_xor(X, weights, biases)
```

```
'''
- cost function and optimization
- sigmoid cross entropy -- single output
- softmax cross entropy -- multiple output, normalized
'''
loss_func =
tf.reduce_sum(tf.nn.sigmoid_cross_entropy_with_logits(logits=model,
labels=Y))
optimizer =
tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(loss
_func)
```

```
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

for k in range(num_iter):
    tmp_cost, _ = sess.run([loss_func, optimizer], feed_dict={X: x, Y: y})
    if k % display_step == 0:
        #print('output: ', sess.run(model, feed_dict={X:x}))
        print('loss= ' + "{:.5f}".format(tmp_cost))

# separates the input space
W = np.squeeze(sess.run(weights['w_h1']))   # 2x2
b = np.squeeze(sess.run(biases['b_h1']))    # 2,

sess.close()
```

```
# Now plot the fitted line. We need only two points to plot the line
plot_x = np.array([np.min(x[:, 0] - 0.2), np.max(x[:, 1]+0.2)])
plot_y =  -1 / W[1, 0] * (W[0, 0] * plot_x + b[0])
plot_y = np.reshape(plot_y, [2, -1])
plot_y = np.squeeze(plot_y)

plot_y2 = -1 / W[1, 1] * (W[0, 1] * plot_x + b[1])
plot_y2 = np.reshape(plot_y2, [2, -1])
plot_y2 = np.squeeze(plot_y2)

plt.scatter(x[:, 0], x[:, 1], c=y, s=100, cmap='viridis')
```

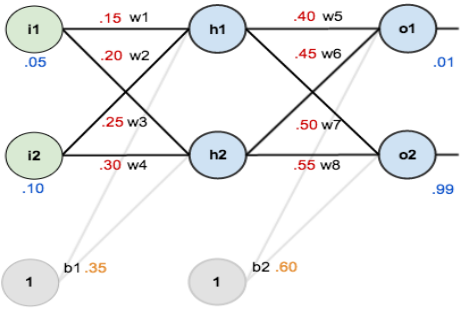
```
plt.plot(plot_x, plot_y, color='k', linewidth=2)    # line 1
plt.plot(plot_x, plot_y2, color='k', linewidth=2)   # line 2
plt.xlim([-0.2, 1.2]); plt.ylim([-0.2, 1.25]);
#plt.text(0.425, 1.05, 'XOR', fontsize=14)
plt.xticks([0.0, 0.5, 1.0]); plt.yticks([0.0, 0.5, 1.0])
plt.show()
```



Post Lab:

Analyze the forward pass and backward pass of back propagation algorithm for the network shown below. (*Note: Update the weights until the network gives the output that is exactly equals to target value.)*

Solution:



Input values

X1=0.05
X2=0.10

Initial weight

W1=0.15 w5=0.40
W2=0.20 w6=0.45
W3=0.25 w7=0.50
W4=0.30 w8=0.55

Bias Values

b1=0.35 b2=0.60

Target Values

T1=0.01
T2=0.99

Now, we first calculate the values of H1 and H2 by a forward pass.

Forward Pass

To find the value of H1 we first multiply the input value from the weights as

$$H1=x1\times w_1+x2\times w_2+b1$$
$$H1=0.05\times 0.15+0.10\times 0.20+0.35$$
$$\mathbf{H1=0.3775}$$

To calculate the final result of H1, we performed the sigmoid function as

$$H1_{final}=\frac{1}{1+\frac{1}{e^{H1}}}$$

$$H1_{final}=\frac{1}{1+\frac{1}{e^{0.3775}}}$$

$$\mathbf{H1_{final}=0.593269992}$$

We will calculate the value of H2 in the same way as H1

$$H2=x1\times w_3+x2\times w_4+b1$$
$$H2=0.05\times 0.25+0.10\times 0.30+0.35$$
$$\mathbf{H2=0.3925}$$

To calculate the final result of H1, we performed the sigmoid function as

$$H2_{final}=\frac{1}{1+\frac{1}{e^{H2}}}$$

$$H2_{final}=\frac{1}{1+\frac{1}{e^{0.3925}}}$$

$$\mathbf{H2_{final}=0.596884378}$$

Now, we calculate the values of y1 and y2 in the same way as we calculate the H1 and H2.

To find the value of y1, we first multiply the input value i.e., the outcome of H1 and H2 from the weights as

$$y1=H1\times w_5+H2\times w_6+b2$$
$$y1=0.593269992\times 0.40+0.596884378\times 0.45+0.60$$
$$\mathbf{y1=1.10590597}$$

To calculate the final result of y1 we performed the sigmoid function as

$$y1_{final}=\frac{1}{1+\frac{1}{e^{y1}}}$$

$$y1_{final}=\frac{1}{1+\frac{1}{e^{1.10590597}}}$$

$$\mathbf{y1_{final}=0.75136507}$$

We will calculate the value of y2 in the same way as y1

$$y2=H1\times w_7+H2\times w_8+b2$$
$$y2=0.593269992\times 0.50+0.596884378\times 0.55+0.60$$
$$\mathbf{y2=1.2249214}$$

To calculate the final result of H1, we performed the sigmoid function as

$$y2_{final} = \frac{1}{1 + \frac{1}{e^{y2}}}$$

$$y2_{final} = \frac{1}{1 + \frac{1}{e^{1.2249214}}}$$

$$\mathbf{y2_{final} = 0.772928465}$$

Our target values are 0.01 and 0.99. Our y1 and y2 value is not matched with our target values T1 and T2.

Now, we will find the **total error**, which is simply the difference between the outputs from the target outputs. The total error is calculated as

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

So, the total error is

$$= \frac{1}{2}(t1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2$$

$$= \frac{1}{2}(0.01 - 0.75136507)^2 + \frac{1}{2}(0.99 - 0.772928465)^2$$

$$= 0.274811084 + 0.0235600257$$

$$\mathbf{E_{total} = 0.29837111}$$

Now, we will backpropagate this error to update the weights using a backward pass.

Backward pass at the output layer

To update the weight, we calculate the error correspond to each weight with the help of a total error. The error on weight w is calculated by differentiating total error with respect to w.

$$Error_w = \frac{\partial E_{total}}{\partial w}$$

We perform backward process so first consider the last weight w5 as

$$\text{Error}_{w5} = \frac{\partial E_{total}}{\partial w5} \ldots\ldots\ldots (1)$$

$$E_{total} = \frac{1}{2}(T1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2 \ldots\ldots\ldots (2)$$

From equation two, it is clear that we cannot partially differentiate it with respect to w5 because there is no any w5. We split equation one into multiple terms so that we can easily differentiate it with respect to w5 as

$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial y1_{final}} \times \frac{\partial y1\_final}{\partial y1} \times \frac{\partial y1}{\partial w5} \ldots\ldots\ldots (3)$$

Now, we calculate each term one by one to differentiate E_{total} with respect to w5 as

$$\frac{\partial E_{total}}{\partial y1_{final}} = \frac{\partial(\frac{1}{2}(T1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2)}{\partial y1_{final}}$$

$$= 2 \times \frac{1}{2} \times (T1 - y1_{final})^{2-1} \times (-1) + 0$$

$$= -(T1 - y1_{final})$$

$$= -(0.01 - 0.75136507)$$

$$\mathbf{\frac{\partial E_{total}}{\partial y1_{final}} = 0.74136507} \ldots\ldots\ldots (4)$$

$$y1_{final} = \frac{1}{1 + e^{-y1}} \ldots\ldots\ldots\ldots (5)$$

$$\frac{\partial y1_{final}}{\partial y1} = \frac{\partial(\frac{1}{1 + e^{-y1}})}{\partial y1}$$

$$= \frac{e^{-y1}}{(1 + e^{-y1})^2}$$

$$= e^{-y1} \times (y1_{final})^2 \ldots\ldots\ldots\ldots (6)$$

$$y1_{final} = \frac{1}{1 + e^{-y1}}$$

$$e^{-y1} = \frac{1 - y1_{final}}{y1_{final}} \ldots\ldots\ldots\ldots (7)$$

Putting the value of e^{-y} in equation (5)

$$= \frac{1 - y1_{final}}{y1_{final}} \times (y1_{final})^2$$

$$= y1_{final} \times (1 - y1_{final})$$

$$= 0.75136507 \times (1 - 0.75136507)$$

$$\mathbf{\frac{\partial y1_{final}}{\partial y1} = 0.186815602 \ldots\ldots\ldots (8)}$$

$$y1 = H1_{final} \times w5 + H2_{final} \times w6 + b2 \ldots\ldots\ldots\ldots\ldots\ldots (9)$$

$$\frac{\partial y1}{\partial w5} = \frac{\partial (H1_{final} \times w5 + H2_{final} \times w6 + b2)}{\partial w5}$$

$$= H1_{final}$$

$$\mathbf{\frac{\partial y1}{\partial w5} = 0.596884378 \ldots\ldots\ldots (10)}$$

So, we put the values of $\frac{\partial E_{total}}{\partial y1_{final}}, \frac{\partial y1_{final}}{\partial y1}$, and $\frac{\partial y1}{\partial w5}$ in equation no (3) to find the final result.

$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial y1_{final}} \times \frac{\partial y1_{final}}{\partial y1} \times \frac{\partial y1}{\partial w5}$$

$$= 0.74136507 \times 0.186815602 \times 0.593269992$$

$$\mathbf{Error_{w5} = \frac{\partial E_{total}}{\partial w5} = 0.0821670407 \ldots\ldots\ldots (11)}$$

Now, we will calculate the updated weight $w5_{new}$ with the help of the following formula

$$w5_{new} = w5 - \eta \times \frac{\partial E_{total}}{\partial w5} \text{ Here, } \eta = \text{learning rate} = 0.5$$

$$= 0.4 - 0.5 \times 0.0821670407$$

$$\mathbf{w5_{new} = 0.35891648 \ldots\ldots\ldots (12)}$$

In the same way, we calculate $w6_{new}$,$w7_{new}$, and $w8_{new}$ and this will give us the following values

$w5_{new}$=0.35891648
$w6_{new}$=408666186
$w7_{new}$=0.511301270
$w8_{new}$=0.561370121

Backward pass at Hidden layer

Now, we will backpropagate to our hidden layer and update the weight w1, w2, w3, and w4 as we have done with w5, w6, w7, and w8 weights.

We will calculate the error at w1 as

$$\text{Error}_{w1} = \frac{\partial E_{total}}{\partial w1}$$

$$E_{total} = \frac{1}{2}(T1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2$$

From equation (2), it is clear that we cannot partially differentiate it with respect to w1 because there is no any w1. We split equation (1) into multiple terms so that we can easily differentiate it with respect to w1 as

$$\frac{\partial E_{total}}{\partial w1} = \frac{\partial E_{total}}{\partial H1_{final}} \times \frac{\partial H1\_final}{\partial H1} \times \frac{\partial H1}{\partial w1} \dots\dots\dots\dots (13)$$

Now, we calculate each term one by one to differentiate E_{total} with respect to w1 as

$$\frac{\partial E_{total}}{\partial H1_{final}} = \frac{\partial(\frac{1}{2}(T1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2)}{\partial H1} \dots\dots\dots\dots (14)$$

We again split this because there is no any $H1^{final}$ term in E^{toatal} as

$$\frac{\partial E_{total}}{\partial H1_{final}} = \frac{\partial E_1}{\partial H1_{final}} + \frac{\partial E_2}{\partial H1_{final}} \dots\dots\dots (15)$$

$\frac{\partial E_1}{\partial H1_{final}}$ and $\frac{\partial E_2}{\partial H1_{final}}$ will again split because in E1 and E2 there is no H1 term. Splitting is done as

$$\frac{\partial E_1}{\partial H1_{final}} = \frac{\partial E_1}{\partial y1} \times \frac{\partial y1}{\partial H1_{final}} \dots\dots\dots (16)$$

$$\frac{\partial E_2}{\partial H1_{final}} = \frac{\partial E_2}{\partial y2} \times \frac{\partial y2}{\partial H1\_final} \dots\dots\dots (17)$$

We again Split both $\frac{\partial E_1}{\partial y1}$ and $\frac{\partial E_2}{\partial y2}$ because there is no any y1 and y2 term in E1 and E2. We split it as

$$\frac{\partial E_1}{\partial y1} = \frac{\partial E_1}{\partial y1_{fianl}} \times \frac{\partial y1_{final}}{\partial y1} \dots\dots\dots (18)$$

$$\frac{\partial E_2}{\partial y2} = \frac{\partial E_2}{\partial y2_{fianl}} \times \frac{\partial y2_{final}}{\partial y2} \dots\dots\dots (19)$$

Now, we find the value of $\frac{\partial E_1}{\partial y1}$ and $\frac{\partial E_2}{\partial y2}$ by putting values in equation (18) and (19) as

From equation (18)

$$\frac{\partial E_1}{\partial y1} = \frac{\partial E_1}{\partial y1_{fianl}} \times \frac{\partial y1_{final}}{\partial y1}$$

$$= \frac{\partial(\frac{1}{2}(T1 - y1_{final})^2)}{\partial y1_{final}} \times \frac{\partial y1_{final}}{\partial y1}$$

$$= 2 \times \frac{1}{2}(T1 - y1_{final}) \times (-1) \times \frac{\partial y1_{final}}{\partial y1}$$

From equation (8)

$$= 2 \times \frac{1}{2}(0.01 - 0.75136507) \times (-1) \times 0.186815602$$

$$\mathbf{\frac{\partial E_1}{\partial y1} = 0.138498562} \dots\dots\dots (20)$$

From equation (19)

$$\frac{\partial E_2}{\partial y2} = \frac{\partial E_2}{\partial y2_{fianl}} \times \frac{\partial y2_{final}}{\partial y2}$$

$$= \frac{\partial(\frac{1}{2}(T2 - y2_{final})^2)}{\partial y2_{final}} \times \frac{\partial y2_{final}}{\partial y2}$$

$$= 2 \times \frac{1}{2}(T2 - y2_{final}) \times (-1) \times \frac{\partial y2_{final}}{\partial y2} \dots\dots\dots (21)$$

$$y2_{final} = \frac{1}{1 + e^{-y2}} \dots\dots\dots\dots\dots\dots (22)$$

$$\frac{\partial y2_{fianl}}{\partial y2} = \frac{\partial(\frac{1}{1 + e^{-y2}})}{\partial y2}$$

$$= \frac{e^{-y2}}{(1 + e^{-y2})^2}$$

$$= e^{-y2} \times (y2_{final})^2 \dots\dots\dots\dots\dots (23)$$

$$y2_{final} = \frac{1}{1 + e^{-y2}}$$

$$e^{-y2} = \frac{1 - y2_{final}}{y2_{final}} \dots\dots\dots\dots\dots (24)$$

Putting the value of e^{-y2} in equation (23)

$$= \frac{1 - y2_{final}}{y2_{final}} \times (y2_{final})^2$$

$$= y2_{final} \times (1 - y2_{final})$$

$$= 0.772928465 \times (1 - 0.772928465)$$

$$\mathbf{\frac{\partial y2_{fianl}}{\partial y2} = 0.175510053 \dots\dots\dots (25)}$$

From equation (21)

$$= 2 \times \frac{1}{2}(0.99 - 0.772928465) \times (-1) \times 0.175510053$$

$$\mathbf{\frac{\partial E_1}{\partial y1} = -0.0380982366126414 \dots\dots\dots (26)}$$

Now from equation (16) and (17)

$$\frac{\partial E_1}{\partial H1_{final}} = \frac{\partial E_1}{\partial y1} \times \frac{\partial y1}{\partial H1_{final}}$$

$$= 0.138498562 \times \frac{\partial(H1_{final} \times w_5 + H2_{final} \times w_6 + b2)}{\partial H1_{final}}$$

$$= 0.138498562 \times \frac{\partial(H1_{final} \times w_5 + H2_{final} \times w_6 + b2)}{\partial H1_{final}}$$

$$= 0.138498562 \times w5$$

$$= 0.138498562 \times 0.40$$

$$\mathbf{\frac{\partial E_1}{\partial H1_{final}} = 0.0553994248 \ldots\ldots\ldots (27)}$$

$$\frac{\partial E_2}{\partial H1_{final}} = \frac{\partial E_2}{\partial y2} \times \frac{\partial y2}{\partial H1\_final}$$

$$= -0.0380982366126414 \times \frac{\partial(H1_{final} \times w_7 + H2_{final} \times w_8 + b2)}{\partial H1_{final}}$$

$$= -0.0380982366126414 \times w7$$

$$= -0.0380982366126414 \times 0.50$$

$$\mathbf{\frac{\partial E_2}{\partial H1_{final}} = -0.0190491183063207 \ldots\ldots\ldots (28)}$$

Put the value of $\frac{\partial E_1}{\partial H1_{final}}$ and $\frac{\partial E_2}{\partial H1_{final}}$ in equation (15) as

$$\frac{\partial E_{total}}{\partial H1_{final}} = \frac{\partial E_1}{\partial H1_{final}} + \frac{\partial E_2}{\partial H1_{final}}$$

$$= 0.0553994248 + (-0.0190491183063207)$$

$$\mathbf{\frac{\partial E_{total}}{\partial H1_{final}} = 0.0364908241736793 \ldots\ldots\ldots (29)}$$

We have $\frac{\partial E_{total}}{\partial H1_{final}}$, we need to figure out $\frac{\partial H1\_final}{\partial H1}$, $\frac{\partial H1}{\partial w1}$ as

$$\frac{\partial H1_{final}}{\partial H1} = \frac{\partial(\frac{1}{1+e^{-H1}})}{\partial H1}$$

$$= \frac{e^{-H1}}{(1+e^{-H1})^2}$$

$$e^{-H1} \times (H1_{final})^2 \dots\dots\dots (30)$$

$$H1_{final} = \frac{1}{1+e^{-H1}}$$

$$e^{-H1} = \frac{1 - H1_{final}}{H1_{final}} \dots\dots\dots\dots (31)$$

Putting the value of e^{-H1} in equation (30)

$$= \frac{1 - H1_{final}}{H1_{final}} \times (H1_{final})^2$$

$$= H1_{final} \times (1 - H1_{final})$$

$$= 0.593269992 \times (1 - 0.593269992)$$

$$\mathbf{\frac{\partial H1_{final}}{\partial H1} = 0.2413007085923199}$$

We calculate the partial derivative of the total net input to H1 with respect to w1 the same as we did for the output neuron:

$$H1 = H1_{final} \times w5 + H2_{final} \times w6 + b2 \dots\dots\dots\dots\dots (32)$$

$$\frac{\partial y1}{\partial w1} = \frac{\partial(x1 \times w1 + x2 \times w3 + b1 \times 1)}{\partial w1}$$

$$= x1$$

$$\mathbf{\frac{\partial H1}{\partial w1} = 0.05 \dots\dots\dots (33)}$$

So, we put the values of $\frac{\partial E_{total}}{\partial H1_{final}}$, $\frac{\partial H1_{final}}{\partial H1}$, and $\frac{\partial H1}{\partial w1}$ in equation (13) to find the final result.

$$\frac{\partial E_{total}}{\partial w1} = \frac{\partial E_{total}}{\partial H1_{final}} \times \frac{\partial H1_{final}}{\partial H1} \times \frac{\partial H1}{\partial w1}$$

$$= 0.0364908241736793 \times 0.2413007085923199 \times 0.05$$

$$\mathbf{Error_{w1} = \frac{\partial E_{total}}{\partial w1} = 0.000438568 \ldots\ldots\ldots (34)}$$

Now, we will calculate the updated weight $w1_{new}$ with the help of the following formula

$$w1_{new} = w1 - \eta \times \frac{\partial E_{total}}{\partial w1} \text{ Here } \eta = \text{learning rate} = 0.5$$

$$= 0.15 - 0.5 \times 0.000438568$$

$$\mathbf{w1_{new} = 0.149780716 \ldots\ldots\ldots (35)}$$

In the same way, we calculate $w2_{new}$,$w3_{new}$, and w4 and this will give us the following values

$w1_{new}$=0.149780716
$w2_{new}$=0.19956143
$w3_{new}$=0.24975114
$w4_{new}$=0.29950229

We have updated all the weights. We found the error 0.298371109 on the network when we fed forward the 0.05 and 0.1 inputs. In the first round of Backpropagation, the total error is down to 0.291027924. After repeating this process 10,000, the total error is down to 0.0000351085. At this point, the outputs neurons generate 0.159121960 and 0.984065734 i.e., nearby our target value when we feed forward the 0.05 and 0.1.

WEEK-3

Outcome: Students are able to implement classification and regression using ANN.

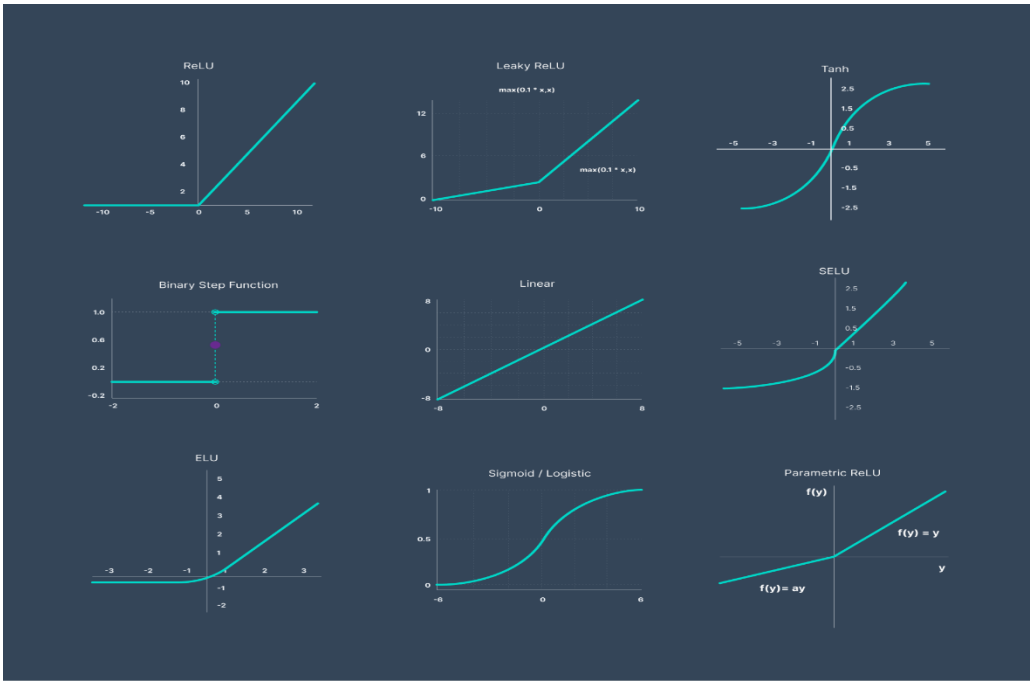
Pre Lab:

1) **What is an artificial neural network and how is it used in deep learning?**

A) Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another. Artificial neural networks (ANNs) are composed of a node layer, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.
it's worth noting that the "deep" in deep learning is just referring to the depth of layers in a neural network. A neural network that consists of more than three layers—which would be inclusive of the inputs and the output—can be considered a deep learning algorithm. A neural network that only has two or three layers is just a basic neural network.

2) **What is an activation function and different types of activation functions?**

A) An Activation Function decides whether a neuron should be activated or not. This means that it will decide whether the neuron's input to the network is important or not in the process of prediction using simpler mathematical operations. The role of the Activation Function is to derive output from a set of input values fed to a node (or a layer).



The linear activation function, also known as "no activation," or "identity function" (multiplied x1.0), is where the activation is proportional to the input.
F(x) = x

Sigmoid function takes any real value as input and outputs values in the range of 0 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0

Sigmoid / Logistic

$$f(x) = \frac{1}{1+e^{-x}}$$

Tanh Function (Hyperbolic Tangent)
Tanh function is very similar to the sigmoid/logistic activation function, and even has the same S-shape with the difference in output range of -1 to 1.

$$f(x) = \frac{\left(e^{x} - e^{-x}\right)}{\left(e^{x} + e^{-x}\right)}$$

3) **Why optimizers are very important in the field of deep learning and mention different types of optimizers used in deep learning?**

A) An optimizer is a function or an algorithm that modifies the attributes of the neural network, such as weights and learning rate. Thus, it helps in reducing the overall loss and improve the accuracy. The problem of choosing the right weights for the model is a daunting task, as a deep learning model generally consists of millions of parameters.

Types:

1. Gradient Descent
2. Stochastic Gradient Descent
3. Stochastic Gradient descent with momentum
4. Mini-Batch Gradient Descent
5. Adagrad
6. RMSProp
7. AdaDelta
8. Adam

In Lab:

EXP3:

a) This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. The "target" field refers to the presence of heart disease in the patient. It is integer valued 0 = no/less chance of heart attack and 1 = more chance of heart attack *Hint: Use ANN for performing classification(Optimizer: Adam)*

https://www.kaggle.com/nareshbhat/health-care-data-set-on-heart-attack-possibility

b) Perform ANN regression for the given data from 1c

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import tensorflow as tf
```

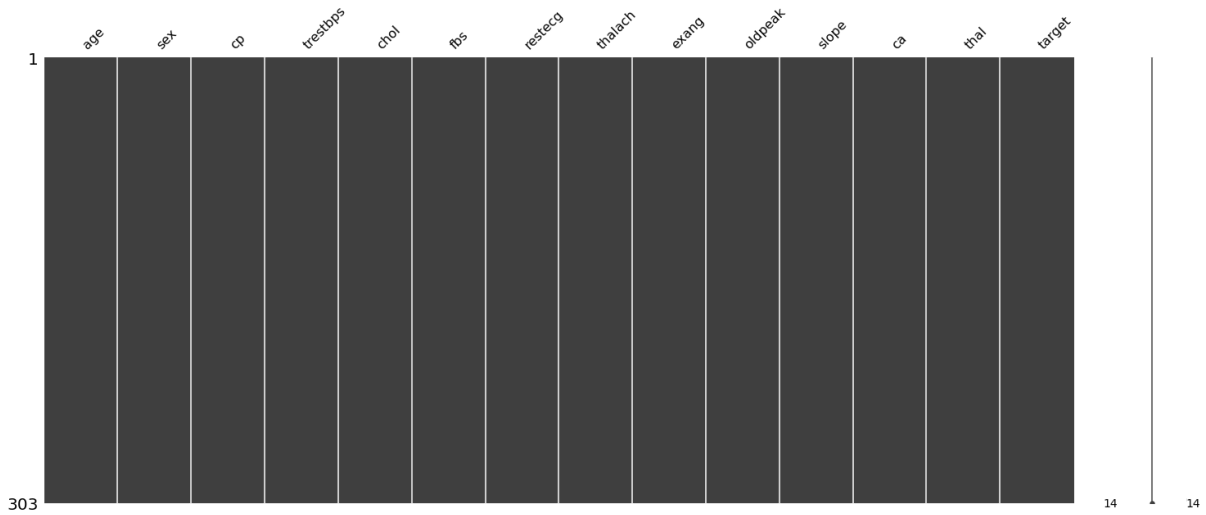
```
tf.test.is_gpu_available() # check gpu is available
```

```
df = pd.read_csv('heart.csv')
df.describe()
```

```
df.head()
df.isnull().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

```
msno.matrix(df)
```




```
scaler = StandardScaler()
X = df.drop('target', axis=1)

X = scaler.fit_transform(X)
Y = df['target']
```

```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2)
```

```
class PrintDot(keras.callbacks.Callback):

   def on_epoch_end(self, epoch, logs):

       if epoch % 100 == 0:
           print(' ')
       print('.', end='')
```

```
model =tf.keras.Sequential([

   tf.keras.layers.Dense(64, input_shape=[x_train.shape[1]]),
   tf.keras.layers.Dense(256, activation=tf.nn.tanh),
   tf.keras.layers.Dense(128, activation=tf.nn.tanh),
   tf.keras.layers.Dense(64, activation=tf.nn.softmax),
   tf.keras.layers.Dense(1)
])

optimizer = tf.keras.optimizers.Adam(0.01)
```

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['binary_accuracy'])
```

```
model.fit(x_train, y_train, epochs=1000, verbose=0, callbacks=[PrintDot()])
model.evaluate(x_test, y_test)
```

```
model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
```

```
8/8 [==============================] - 0s 2ms/step - loss: 0.1511 -
binary_accuracy: 0.9876
2/2 [==============================] - 0s 3ms/step - loss: 1.5335 -
binary_accuracy: 0.7541
```

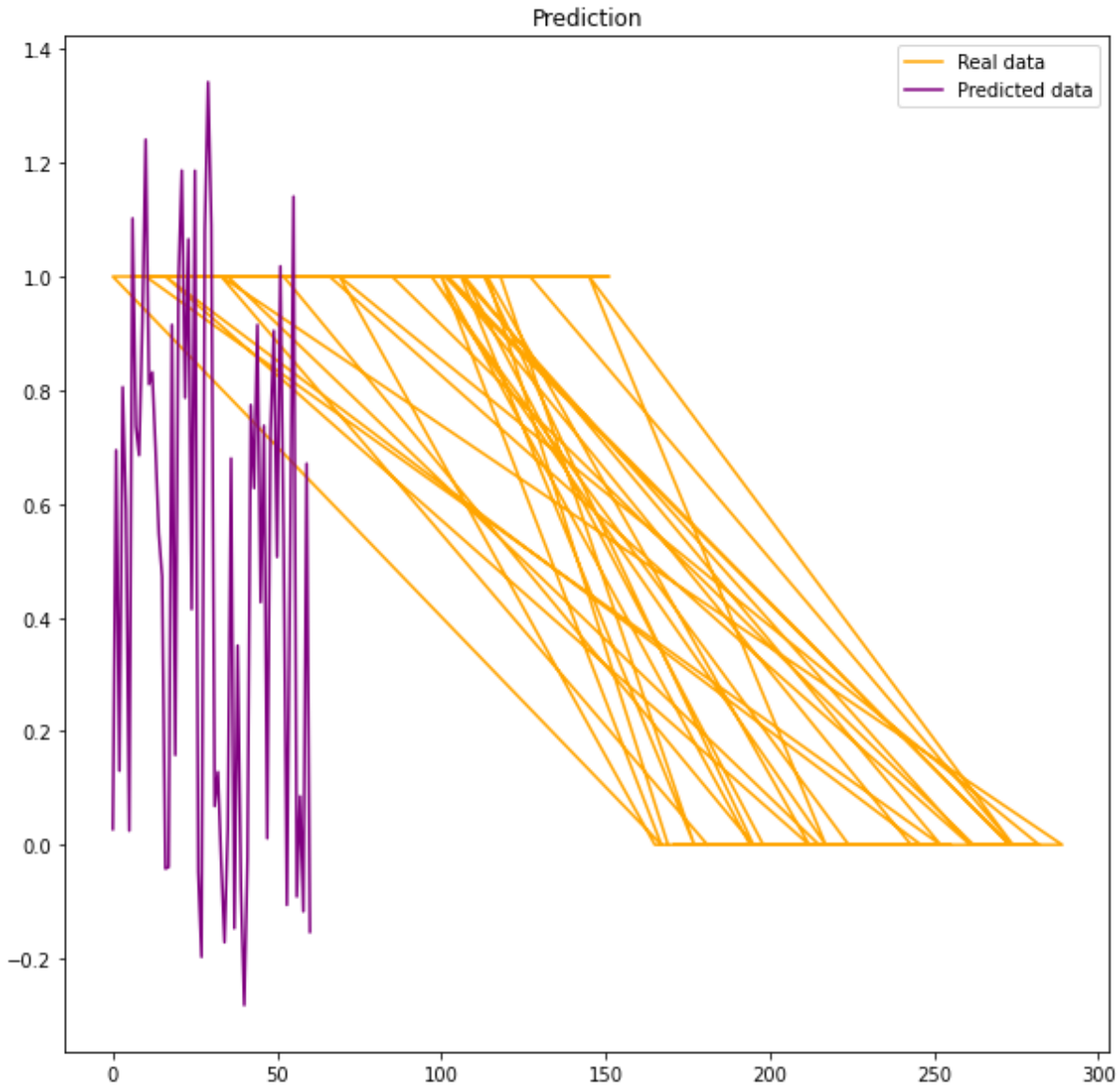
```
from sklearn.metrics import confusion_matrix, accuracy_score
conf = confusion_matrix(y_test, y_pred)
print(conf)
accuracy_score(y_test,y_pred)
```

```
import numpy as np
from keras.layers import Dense, Activation
from keras.models import Sequential
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

```
# Initialising the ANN
model = Sequential()

# Adding the input layer and the first hidden layer
model.add(Dense(32, activation = 'relu', input_dim = 13))

# Adding the second hidden layer
model.add(Dense(units = 32, activation = 'relu'))

# Adding the third hidden layer
model.add(Dense(units = 32, activation = 'relu'))

# Adding the output layer

model.add(Dense(units = 1))

#model.add(Dense(1))
# Compiling the ANN
```

```
model.compile(optimizer = 'adam', loss = 'mean_squared_error',
metrics=['accuracy'])

# Fitting the ANN to the Training set
model.fit(x_train, y_train, batch_size = 10, epochs = 100,
validation_data=(x_test, y_test),shuffle=True)

y_pred = model.predict(x_test)
```

```
plt.figure(figsize=(10, 10))
plt.plot(y_test, color = 'orange', label = 'Real data')
plt.plot(y_pred, color = 'purple', label = 'Predicted data')
plt.title('Prediction')
plt.legend()
plt.show()
```


Prediction
Real data
Predicted data
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
50
100
150
200
250
300

Post Lab:

Use a stochastic gradient descent optimizer for the similar classification problem above and compare the results of both the models.

Program:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import tensorflow as tf

df = pd.read_csv('heart.csv')
scaler = StandardScaler()
X = df.drop('target', axis=1)

X = scaler.fit_transform(X)
Y = df['target']
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2)
```

```
#Using SGD optimizer
model = tf.keras.models.Sequential([
   tf.keras.layers.Dense(10, activation='relu', input_shape=(13,)),
   tf.keras.layers.Dense(10, activation='relu'),
   tf.keras.layers.Dense(1, activation='sigmoid')

])

model.compile(optimizer='sgd', loss='binary_crossentropy',
metrics=['accuracy'])
model.fit(x_train, y_train, epochs=100)
```

```
print("For SGD optimizer, Training and Testing accuracy are: ")
```

```
model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)

print("For ADAM optimizer, Training and Testing accuracy are: ")
model.evaluate(X_train, Y_train)
model.evaluate(X_test, Y_test)
```

```
For SGD optimizer, Training and Testing accuracy are:
8/8 [==============================] - 0s 2ms/step - loss: 0.4029 - accuracy:
0.9669
```

```
2/2 [==============================] - 0s 4ms/step - loss: 0.5130 - accuracy:
0.9672
For ADAM optimizer, Training and Testing accuracy are:
8/8 [==============================] - 0s 1ms/step - loss: 0.1333 - accuracy:
0.9917
2/2 [==============================] - 0s 2ms/step - loss: 1.5825 - accuracy:
0.8689

[1.582500696182251, 0.868852436542511]
```

```
#By analysis, we find that compared with ADAM,
#SGD is more locally unstable and is more likely to converge to the
minima at the flat or
#asymmetric basins/valleys which often have better generalization
performance over other type minima.
#So our results can explain the better generalization performance of
SGD over ADAM.
```

WEEK-4

Outcome: Students are able to implement dimensionality reduction and classification using ANN.

Pre Lab:

1) **What is dimensionality reduction and what is the need for reducing the dimensions?**

A) The higher the number of features, the harder it gets to visualise the training set and then work on it. Sometimes, most of these features are correlated, and hence redundant. This is where dimensionality reduction algorithms come into play. Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction. When dealing with high dimensional data, it is often useful to reduce the dimensionality by projecting the data to a lower dimensional subspace which captures the "essence" of the data. This is called dimensionality reduction.

2) **What are different types of algorithms for performing dimensionality reduction?**

A)

1. Feature selection (Filter strategy, Wrapper strategy, Embedded strategy)
2. Feature extraction
3. Principal Component Analysis (PCA)
4. Non-negative matrix factorization (NMF)
5. Linear discriminant analysis (LDA)
6. Generalized discriminant analysis (GDA)
7. Missing Values Ratio
8. Backward Feature Elimination

3) **What is Batch Normalization and why is it used in deep neural networks?**

A) Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks. Batch normalization solves a major problem called internal covariate shift. It helps by making the data flowing between intermediate layers of the neural network look, this means you can use a higher learning rate. It has a regularizing effect which means you can often remove dropout.

In Lab:

EXP4:

a) In this dataset, there are various factors given, which are involved when a patient is hospitalized. On the basis of these factors, predict whether the *patient will survive or not*. But it has 85 columns so, perform Dimensionality reduction using PCA.

Data set: https://www.kaggle.com/mitishaagarwal/patient

b) Normalize the data in the given dataset and perform classification using ANN.

Program:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
data=pd.read_csv("dataset.csv")
```

```
data.shape
```

```
(91713, 85)
```

```
data.info()
```

```
RangeIndex: 91713 entries, 0 to 91712
Data columns (total 85 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   encounter_id                91713 non-null  int64
 1   patient_id                  91713 non-null  int64
 2   hospital_id                 91713 non-null  int64
 3   age                         87485 non-null  float64
 4   bmi                         88284 non-null  float64
 5   elective_surgery            91713 non-null  int64
 6   ethnicity                   90318 non-null  object
 7   gender                      91688 non-null  object
 8   height                      90379 non-null  float64
 9   icu_admit_source            91601 non-null  object
 10  icu_id                      91713 non-null  int64
 11  icu_stay_type               91713 non-null  object
 12  icu_type                    91713 non-null  object
```

```
 13  pre_icu_los_days                91713 non-null  float64
 14  weight                          88993 non-null  float64
 15  apache_2_diagnosis              90051 non-null  float64
 16  apache_3j_diagnosis             90612 non-null  float64
 17  apache_post_operative           91713 non-null  int64
 18  arf_apache                      90998 non-null  float64
 19  gcs_eyes_apache                 89812 non-null  float64
...
83  Unnamed: 83                     0 non-null      float64
 84  hospital_death                  91713 non-null  int64
dtypes: float64(71), int64(7), object(7)
```

```
data.isna().sum()
```

```
encounter_id                      0
patient_id                        0
hospital_id                       0
age                            4228
bmi                            3429
                               ...
solid_tumor_with_metastasis     715
apache_3j_bodysystem           1662
apache_2_bodysystem            1662
Unnamed: 83                   91713
hospital_death                    0
Length: 85, dtype: int64
```

```
data.drop(["encounter_id" ,"patient_id" ,"hospital_id","Unnamed: 83", ],axis
=1, inplace=True)
```

```
Num=[]
for col in data.columns:
   if (data[col].dtype==int)or (data[col].dtype==float):
        Num.append(col)
```

```
for col in data.columns:

   if col in Num :

       data[col].fillna(data[col].median(), inplace=True)
```

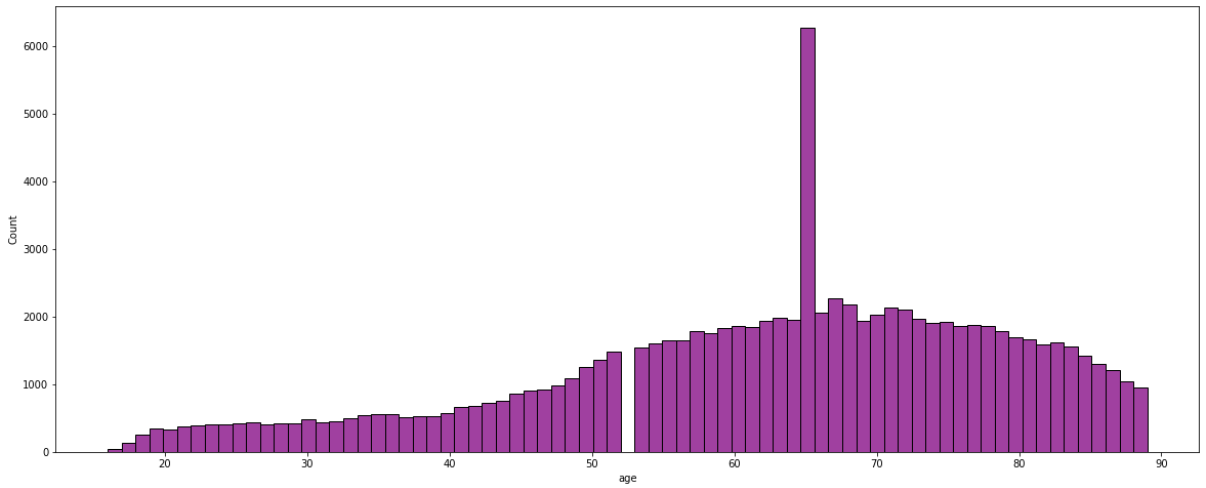
```
categorical=[]
for col in data.columns:
   if  (data[col].dtype==object):
       categorical.append(col)
```

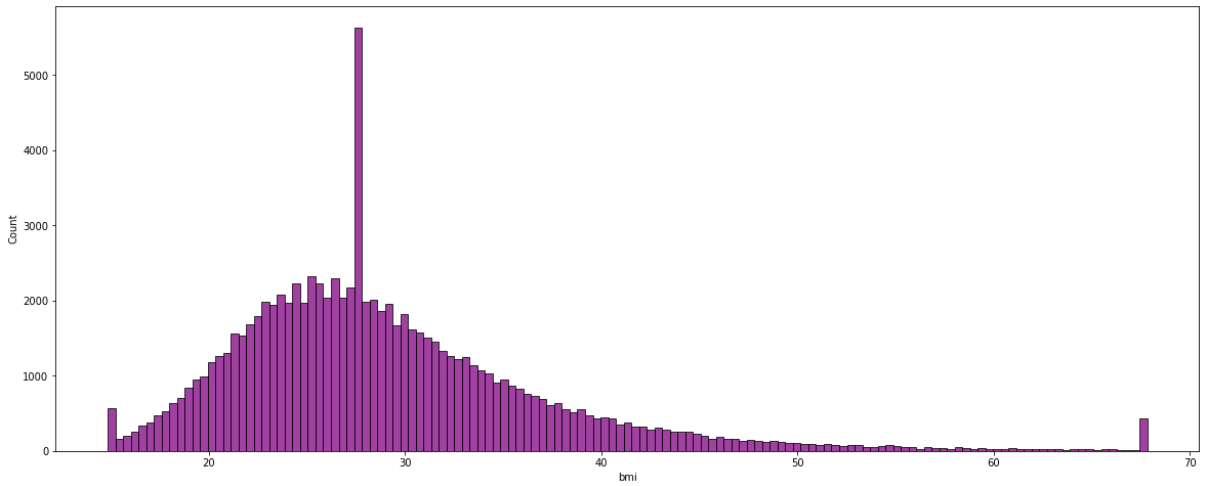
```
for col in data.columns:
   if col in categorical:
       data[col].fillna(data[col].mode(), inplace=True)
```

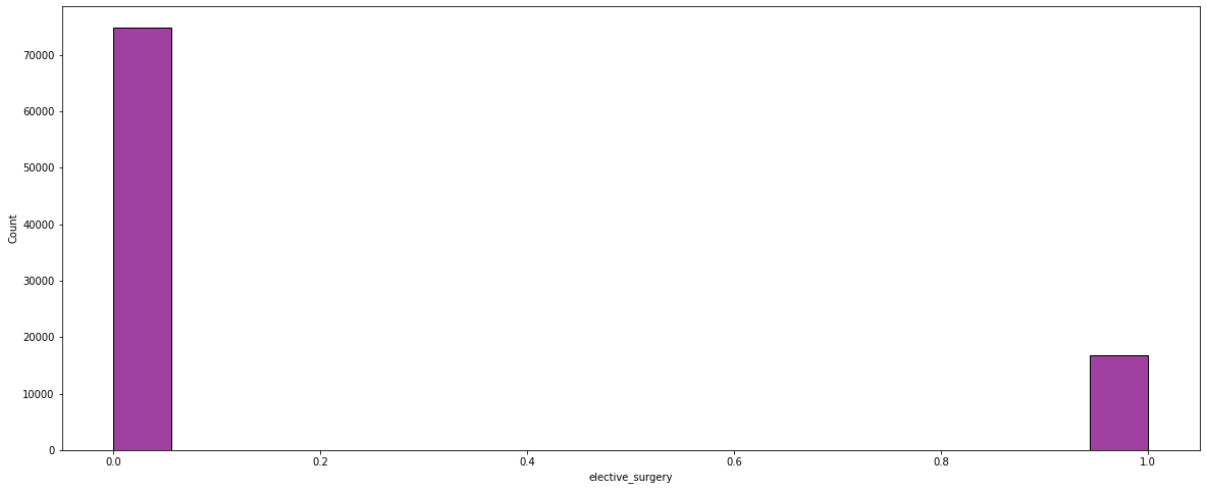
```
data["ethnicity"].mode()
```

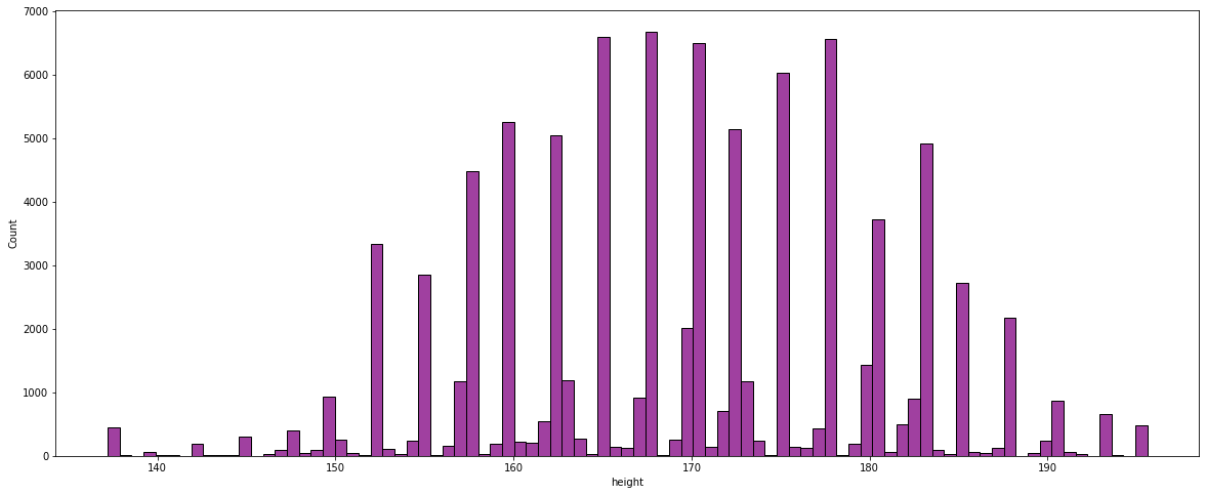
```
from tensorflow import keras
```

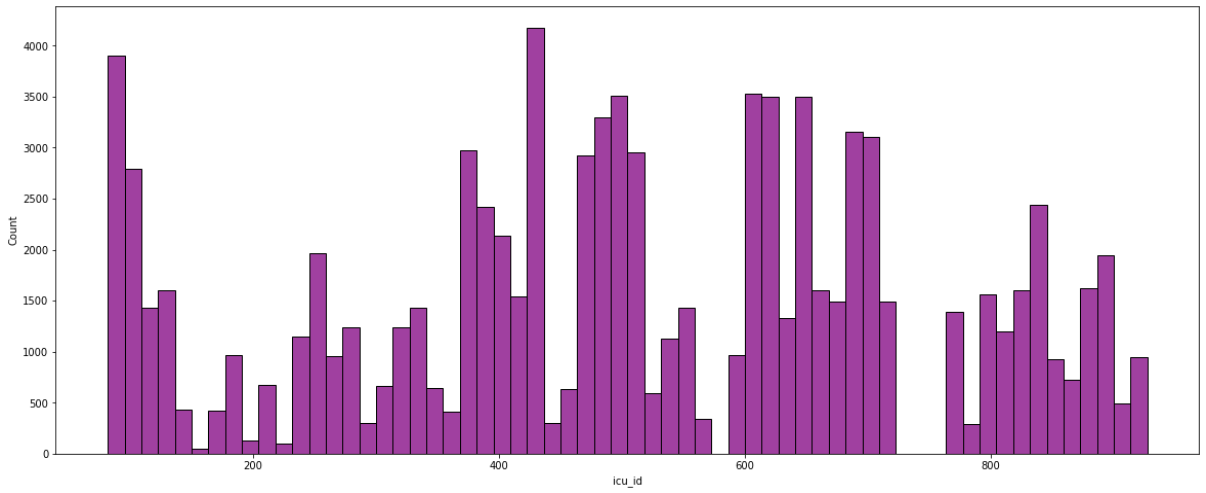
```
for col in Num:
   plt.figure(figsize=(20,8))
   sns.histplot(data[col], color="purple")
   plt.show()
```

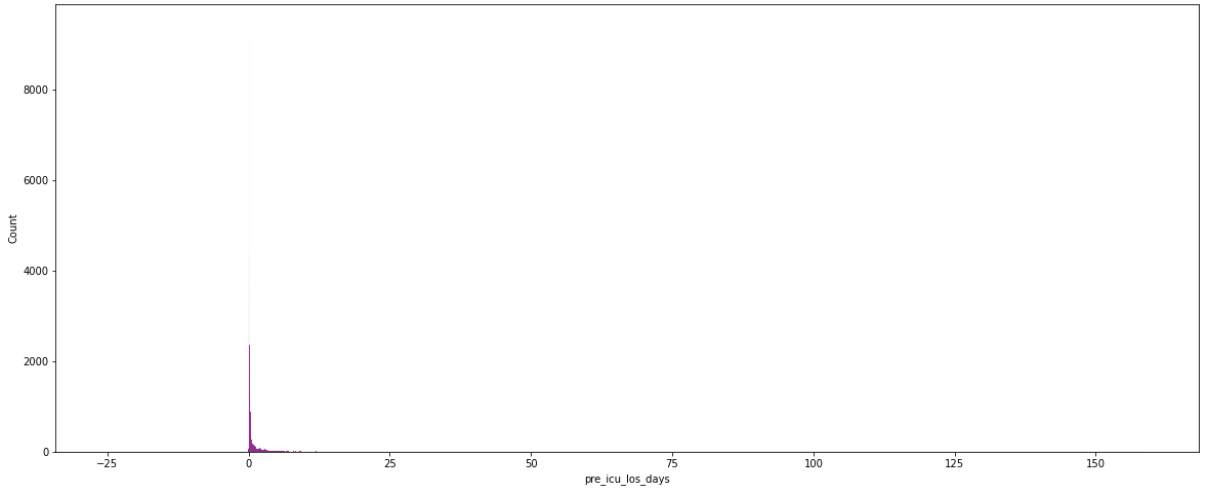


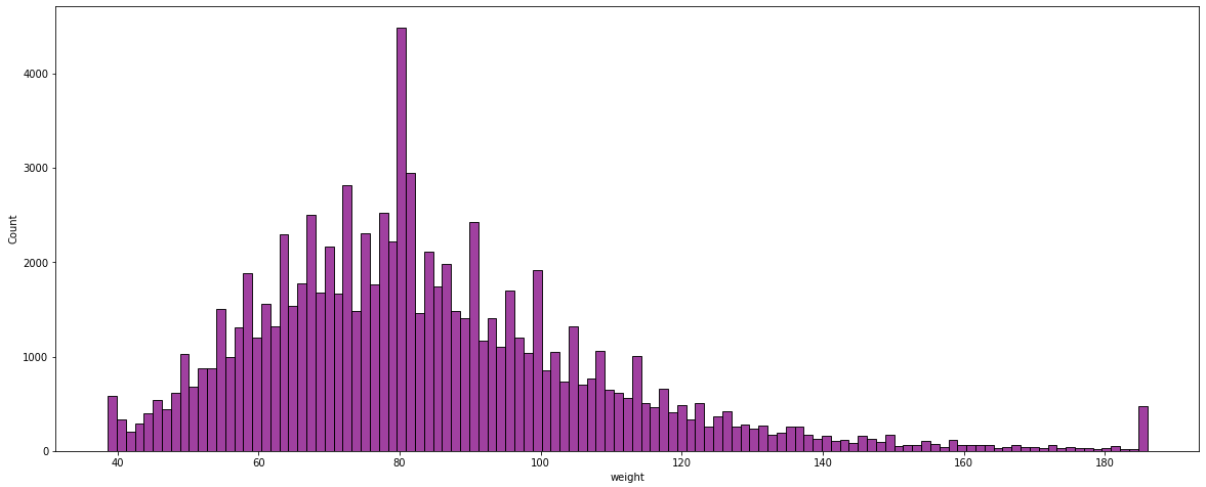

Count
bmi
0
1000
2000
3000
4000
5000
20
30
40
50
60
70

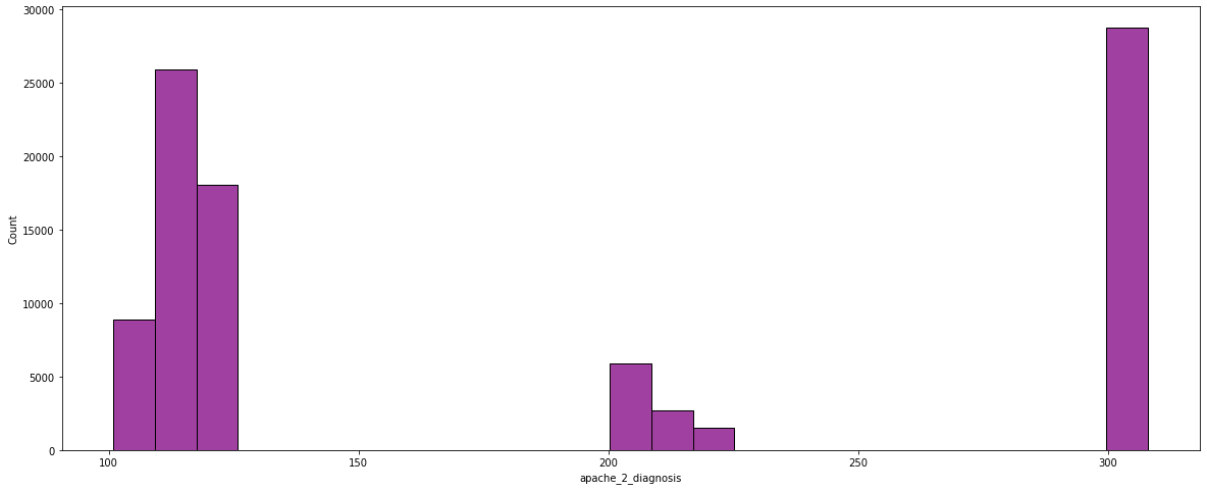

Count
elective_surgery
0
10000
20000
30000
40000
50000
60000
70000
0.0
0.2
0.4
0.6
0.8
1.0


Count
height
0
1000
2000
3000
4000
5000
6000
7000
140
150
160
170
180
190


Count
icu_id


Count
pre_icu_los_days


Count
weight




```
from category_encoders import CountEncoder
CE = CountEncoder(normalize=True, cols=categorical)
data = CE.fit_transform(data)
```

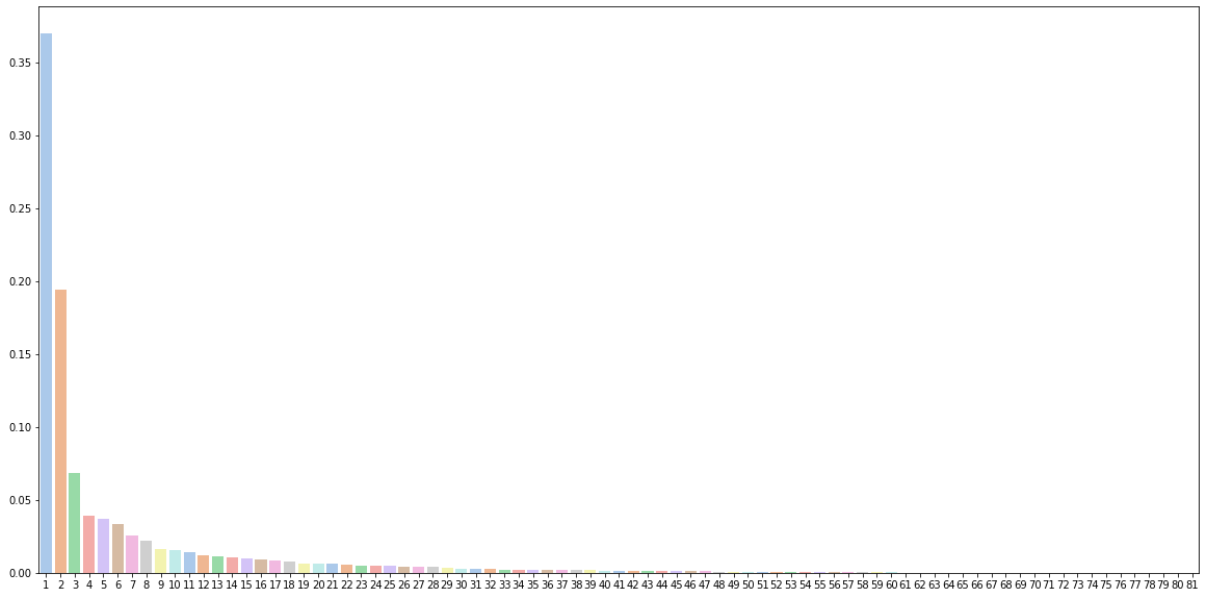
```
from sklearn.preprocessing import RobustScaler
RS = RobustScaler()
scale = RS.fit_transform(data)
data = pd.DataFrame(scale, columns=data.columns)
```

```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(data)
pca_data = pca.transform(data)
component_names = [f"Component {i}" for i in range(1,82 )]
component_names
```

```
pca_data = pd.DataFrame(pca_data, columns=component_names)
pca_data
```

```
pca.explained_variance_ratio_
```

```
plt.figure(figsize=(20,10))
sns.barplot(x=list(range(1, 82)), y=pca.explained_variance_ratio_,
palette="pastel")
```




```
import tensorflow as tf
from tensorflow import keras
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(data, y, test_size=0.2,
random_state=42)
```

```
class PrintDot(keras.callbacks.Callback):

   def on_epoch_end(self, epoch, logs):

       if epoch % 50 == 0:
           print(' ')
       print('.', end='')
```

```
model = keras.Sequential([
   keras.layers.Dense(input_shape=(81,), units=1, activation='sigmoid'),
   keras.layers.Dense(264, activation=tf.nn.tanh),
   keras.layers.Dropout(0.2),
   keras.layers.Dense(128, activation=tf.nn.tanh),
   keras.layers.Dropout(0.2),
   keras.layers.Dense(64, activation=tf.nn.sigmoid),
   keras.layers.Dropout(0.2),
   keras.layers.Dense(1)
   ])
```

```
earlyStopping = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=5)

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

model.fit(x_train, y_train, epochs=100, callbacks=[earlyStopping,
PrintDot()])
```

```
model.evaluate(x_test, y_test)
```

Post Lab:

1) Briefly describe the working of principle component analysis.

A) Principal Component Analysis (PCA) is a statistical procedure that uses an orthogonal transformation that converts a set of correlated variables to a set of uncorrelated variables. PCA is the most widely used tool in exploratory data analysis and in machine learning for predictive models. Moreover, PCA is an unsupervised statistical technique used to examine the interrelations among a set of variables. It is also known as a general factor analysis where regression determines a line of best fit.

The Principal Component Analysis is a popular unsupervised learning technique for reducing the dimensionality of data. It increases interpretability yet, at the same time, it minimizes information loss. It helps to find the most significant features in a dataset and makes the data easy for plotting in 2D and 3D. PCA helps in finding a sequence of linear combinations of variables. The Principal Components are a straight line that captures most of the variance of the data. They have a direction and magnitude. Principal components are orthogonal projections (perpendicular) of data onto lower-dimensional space.

WEEK-5

Outcome: Students are able to implement Multi class classification on MNIST dataset using ANN.

Pre Lab:

1) What is the difference between binary classification and multi class classification?
A)
Binary classification refers to those classification tasks that have two class labels.
Examples include:

- Email spam detection (spam or not).
- Churn prediction (churn or not).
- Conversion prediction (buy or not).

Typically, binary classification tasks involve one class that is the normal state and another class that is the abnormal state.
For example "*not spam*" is the normal state and "*spam*" is the abnormal state. Another example is "*cancer not detected*" is the normal state of a task that involves a medical test and "*cancer detected*" is the abnormal state.
The class for the normal state is assigned the class label 0 and the class with the abnormal state is assigned the class label 1.

Multi-Class Classification
Multi-Class Classification refers to those classification tasks that have more than two class labels.
Examples include:

- Face classification.
- Plant species classification.
- Optical character recognition.

Unlike binary classification, multi-class classification does not have the notion of normal and abnormal outcomes. Instead, examples are classified as belonging to one among a range of known classes.
The number of class labels may be very large on some problems. For example, a model may predict a photo as belonging to one among thousands or tens of thousands of faces in a face recognition system.
Problems that involve predicting a sequence of words, such as text translation models, may also be considered a special type of multi-class classification. Each word in the sequence of words to be predicted involves a multi-class classification where the size of the vocabulary defines the number of possible classes that may be predicted and could be tens or hundreds of thousands of words in size.

2) Briefly explain about SoftMax activation function?

A) Softmax is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of

each value in the vector.

The most common use of the softmax function in applied machine learning is in its use as an activation function in a neural network model. Specifically, the network is configured to output N values, one for each class in the classification task, and the softmax function is used to normalise the outputs, converting them from weighted sum values into probabilities that sum to one. Each value in the output of the softmax function is interpreted as the probability of membership for each class.

The softmax function is also a type of sigmoid function but is handy when we are trying to handle classification problems.

Nature :- non-linear

Uses :- Usually used when trying to handle multiple classes. The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.

Output:- The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.

3) Difference between softmax and sigmoid activation function

A) The sigmoid function is used for the two-class logistic regression, whereas the softmax function is used for the multiclass logistic regression (a.k.a. MaxEnt, multinomial logistic regression, softmax Regression, Maximum Entropy Classifier). In the two-class logistic regression, the predicted probabilities are as follows, using the sigmoid function:

$$\Pr(Y_i = 0) = \frac{e^{-\boldsymbol{\beta}\cdot\mathbf{X}_i}}{1 + e^{-\boldsymbol{\beta}\cdot\mathbf{X}_i}}$$
$$\Pr(Y_i = 1) = 1 - \Pr(Y_i = 0) = \frac{1}{1 + e^{-\boldsymbol{\beta}\cdot\mathbf{X}_i}}$$

In the multiclass logistic regression, with K classes, the predicted probabilities are as follows, using the softmax function:

$$\Pr(Y_i = k) = \frac{e^{\boldsymbol{\beta}_k\cdot\mathbf{X}_i}}{\sum_{0\le c\le K} e^{\boldsymbol{\beta}_c\cdot\mathbf{X}_i}}$$

In Lab:

EXP 5:

Perform Multi class classification on MNIST dataset using ANN.(Note: Use Softmax activation function for the output layer with 3-4 hidden layers consisting of ReLU activation function also evaluate the performance by using confusion matrix.)

Program:

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
```

```
data = keras.datasets.mnist
(x_train, y_train) , (x_test, y_test) = data.load_data()
plt.imshow(x_train[19])
plt.show()
```

```
x_train = x_train / 255
x_test = x_test / 255
x_valid, x_train = x_train[:2000], x_train[2000:4000]
y_valid, y_train = y_train[:2000], y_train[2000:4000]
```

```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28,28]))
model.add(keras.layers.Dense(264, activation="relu"))
model.add(keras.layers.Dense(128, activation="relu"))
model.add(keras.layers.Dense(64, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

```
weights, biases = model.layers[1].get_weights()
weights.shape, biases.shape
model.compile(loss = "sparse_categorical_crossentropy",
              optimizer = "adam",
              metrics = ["accuracy"])
```

```
model.summary()
```

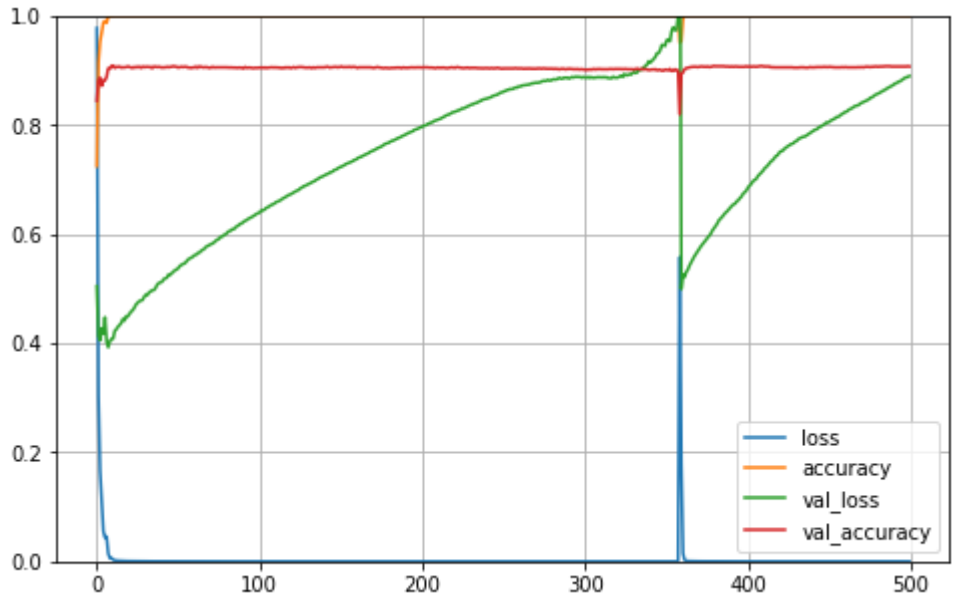
```
Model: "sequential_2"
```

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 flatten_2 (Flatten)          (None, 784)               0

 dense_6 (Dense)              (None, 264)               207240

 dense_7 (Dense)              (None, 128)               33920

 dense_8 (Dense)              (None, 64)                8256

 dense_9 (Dense)              (None, 10)                650

=================================================================
Total params: 250,066

Trainable params: 250,066

Non-trainable params: 0

_________________________________________________________________
```

```
class PrintDot(keras.callbacks.Callback):

   def on_epoch_end(self, epoch, logs):

       if epoch % 50 == 0:
           print(' ')
       print('.', end='')
```

```
model_history = model.fit(X_train, Y_train, epochs=500,
validation_data=(X_valid, Y_valid), callbacks=[PrintDot()], verbose=0)
```

```
pd.DataFrame(model_history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
```




```
model.evaluate(x_test, y_test)
```

313/313 [==============================] - 1s 2ms/step - loss: 0.6582 - accuracy: 0.9247

[0.6581530570983887, 0.9247000217437744]

Post Lab:

1) **List the applications of image classification in the real world.**

A) Image recognition technology has revolutionised online visualisation with its applications in

- In the areas of education
- Smart surveillance systems
- Facial recognition
- Image Correction, Sharpening, and Resolution Correction
- Filters on Editing Apps and Social Media
- Medical Technology
- Computer / Machine Vision

- Pattern recognition
- Video Processing

WEEK-6

Outcome: Students are able to build a prediction model from a given dataset.

Pre Lab:

1) **What is the dropout rate?**

A) Dropout is a regularisation technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. The term "dropout" refers to dropping out units (both hidden and visible) in a neural network. A good value for dropout in a hidden layer is between 0.5 and 0.8. Input layers use a larger dropout rate, such as 0.8. The default interpretation of the dropout hyperparameter is the probability of training a given node in a layer, where 1.0 means no dropout, and 0.0 means no outputs from the layer.

2) **What is regularisation and types of regularisation?**

A) Regularisation is a technique which is used to solve the overfitting problem of the machine learning models.
There are two types of regularization as follows:
1. L1 Regularization or Lasso Regularization
2. L2 Regularization or Ridge Regularization

L1 Regularization or Lasso Regularization adds a penalty to the error function. The penalty is the sum of the absolute values of weights.

$$Min(\sum_{i=1}^{n}(y_i - w_i x_i)^2 + p\sum_{i=1}^{n}|w_i|)$$

p is the tuning parameter which decides how much we want to penalise the model.

L2 Regularization or Ridge Regularization also adds a penalty to the error function. But the penalty here is the sum of the squared values of weights.

$$Min(\sum_{i=1}^{n}(y_i - w_i x_i)^2 + p\sum_{i=1}^{n}(w_i)^2)$$

Similar to L1, in L2 also, p is the tuning parameter which decides how much we want to penalise the model.

3) **What is the need of using dropout and regularization?**

A) Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. The term "dropout" refers to dropping out units (both hidden and visible) in a neural network.

```
import numpy as np
import pandas as pd
import missingno as msno
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow import keras
```

```
import warnings
warnings.filterwarnings('ignore')
```

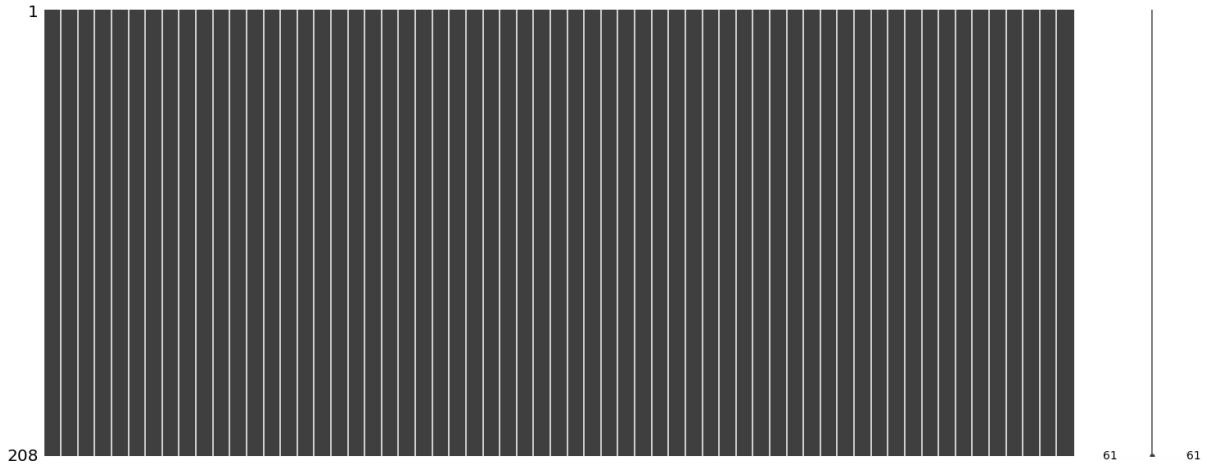
```
df = pd.read_csv('sonar_dataset.csv',header=None)
df.head()
```

```
df.shape
```

```
(208, 61)
```

```
df.isna().sum()
```

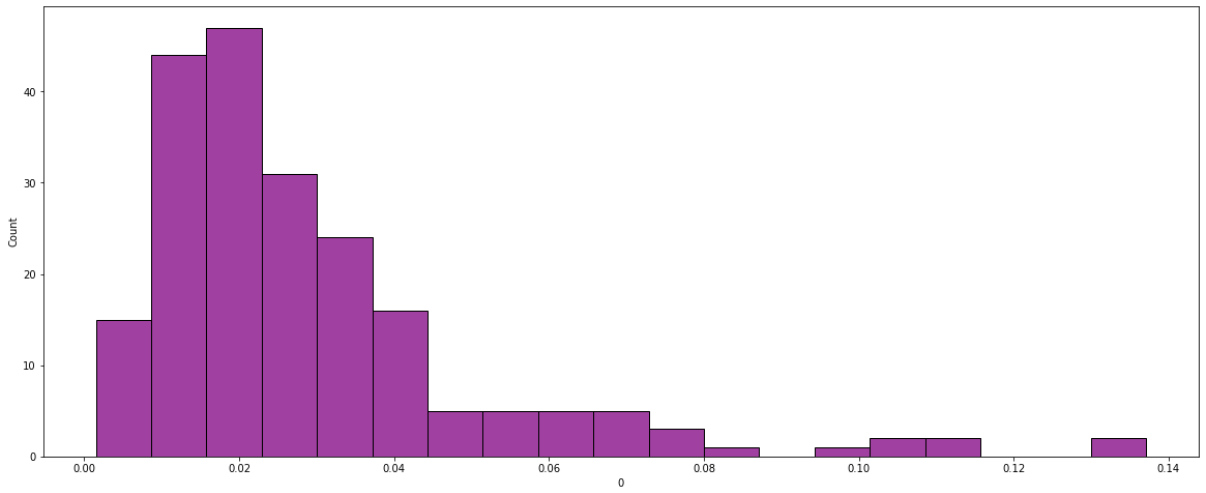
```
msno.matrix(df)
```

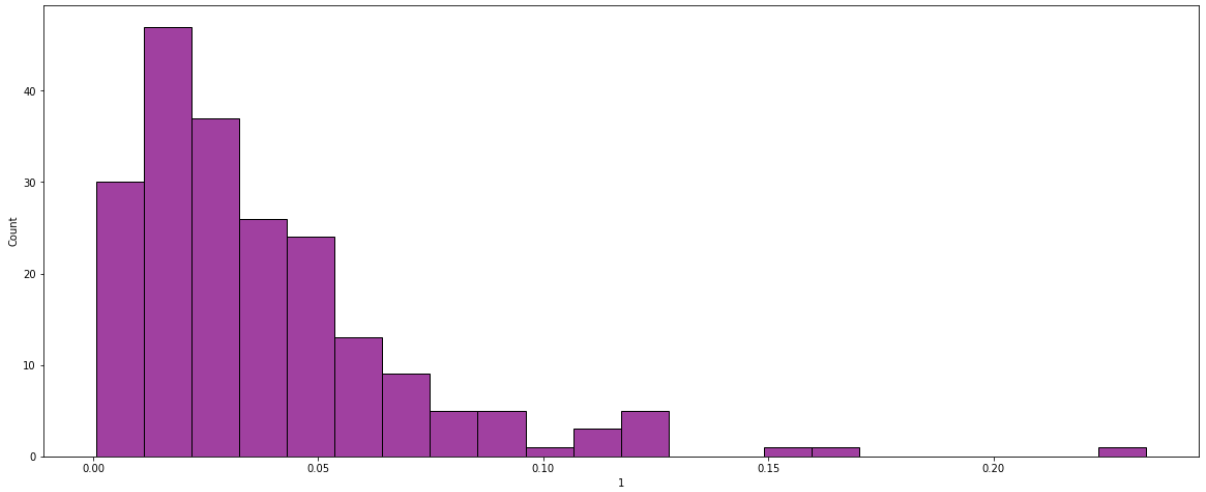


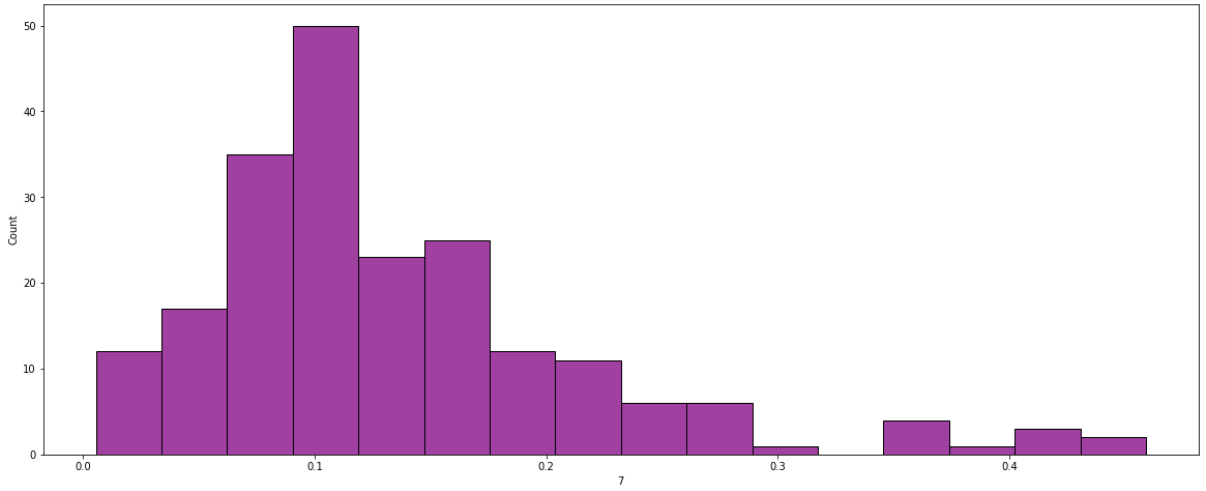

```
df[60].value_counts()
```

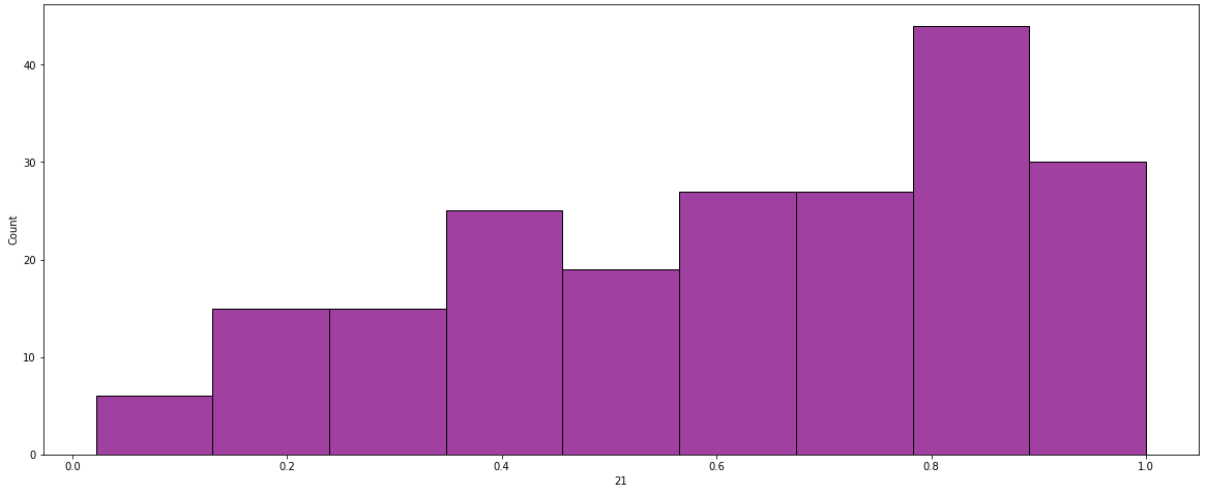
```
M    111
R     97
Name: 60, dtype: int64
```

```
for col in df.columns:
    plt.figure(figsize=(20,8))
    sns.histplot(df[col], color="purple")
    plt.show()
```


Count
0
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0
10
20
30
40


Count
1
0.00
0.05
0.10
0.15
0.20
0
10
20
30
40






```
x = df.drop(60,axis='columns')
y = df[60]
x.head()
```

```
y = pd.get_dummies(y,drop_first=True)
y.sample(5) # R = 1, M = 0
```

```
y.value_counts()
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.25,random_state=1)
```

```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((156, 60), (52, 60), (156, 1), (52, 1))
```

```
class PrintDot(keras.callbacks.Callback):

   def on_epoch_end(self, epoch, logs):

       if epoch % 10 == 0:
           print(' ')
       print('.', end='')
```

```
model = keras.Sequential([
   keras.layers.Dense(60,input_shape=(60,),activation='relu'),
   keras.layers.Dropout(0.5),
   keras.layers.Dense(32,activation='relu'),
   keras.layers.Dropout(0.5),
   keras.layers.Dense(16,activation='relu'),
   keras.layers.Dropout(0.5),
   keras.layers.Dense(1,activation='sigmoid')
   ])
```

```
#Compile model
model.compile(optimizer='adam',loss =
'binary_crossentropy',metrics=['binary_accuracy'])
model.fit(x_train,y_train,epochs=1000,batch_size=8, verbose=0,
callbacks=[PrintDot()])
```

```
model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
```

```
5/5 [==============================] - 0s 1ms/step - loss: 0.1870 - accuracy: 0.9487
2/2 [==============================] - 0s 2ms/step - loss: 0.4147 - accuracy: 0.7885

[0.4147123694419861, 0.7884615659713745]
```

```
y_pred = model.predict(x_test).reshape(-1)
y_pred[:10]
#rounding off the values to 0 or 1
y_pred = np.round(y_pred).astype(int)
```

```
from sklearn.metrics import confusion_matrix,classification_report
cm = confusion_matrix(y_test,y_pred)
cm
```

```
array([[24,  3],
       [ 8, 17]], dtype=int64)
```

```
print(classification_report(y_test,y_pred))
```

```
precision    recall  f1-score   support

           0       0.75      0.89      0.81        27
           1       0.85      0.68      0.76        25

    accuracy                           0.79        52
   macro avg       0.80      0.78      0.78        52
weighted avg       0.80      0.79      0.79        52
```

Post Lab:

1) **Write a detailed note on different types of regularization.**

A) Regularization refers to techniques that are used to calibrate machine learning models in order to minimise the adjusted loss function and prevent overfitting or underfitting. Using Regularization, we can fit our machine learning model appropriately on a given test set and hence reduce the errors in it.

There are two types of regularization as follows:
1. L1 Regularization or Lasso Regularization
2. L2 Regularization or Ridge Regularization

L1 Regularization or Lasso Regularization adds a penalty to the error function. The penalty is the sum of the absolute values of weights.

$$Min(\sum_{i=1}^{n}(y_i - w_i x_i)^2 + p\sum_{i=1}^{n}|w_i|)$$

p is the tuning parameter which decides how much we want to penalise the model.

WEEK-7

Outcome: Students are able to built CNN model and perform convolution operations.

Pre Lab:

1) **What is CNN?**

A) A convolutional neural network (CNN or convnet) is a subset of machine learning. It is one of the various types of artificial neural networks which are used for different applications and data types. A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data. There are other types of neural networks in deep learning, but for identifying and recognizing objects, CNNs are the network architecture of choice. This makes them highly suitable for computer vision (CV) tasks and for applications where object recognition is vital, such as self-driving cars and facial recognition.

2) **Discuss the different steps in CNN.**

A) Different steps in CNN are

1. Upload Dataset
2. The Input Layer
3. Padding
4. stride
5. Convolution Layer
6. Pooling Layer
7. Dense Layer
8. Output Layer

3) **Why is CNN used for image classification?**

A) CNN is made up of Neural Networks and Convolution Layers which extract the features from an image. Neural networks learn features directly from data with which they are trained, so specialists don't need to extract features manually. The power of neural networks comes from their ability to learn the representation in your training data and how to best relate it to the output variable that you want to predict.

In Lab:

Exp 7: Build CNN Model for COVID-19 Disease Detection

Program:

2000080132_LAB_7

October 11, 2022

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.utils import load_img, img_to_array
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
```

[ ]:
```
# Load the data

train_data_gen = ImageDataGenerator(rescale=1./255, shear_range=0.2,␣
 ↪zoom_range=0.2, horizontal_flip=True)
test_data_gen = ImageDataGenerator(rescale=1./255)

train = train_data_gen.flow_from_directory('train', target_size=(64, 64),␣
 ↪batch_size=32, class_mode='binary')
test = test_data_gen.flow_from_directory('test', target_size=(64, 64),␣
 ↪batch_size=32, class_mode='binary')
```

```
Found 181 images belonging to 2 classes.
Found 46 images belonging to 2 classes.
```

[ ]:
```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(64,␣
 ↪64, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(64, 64, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

[ ]:
```
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

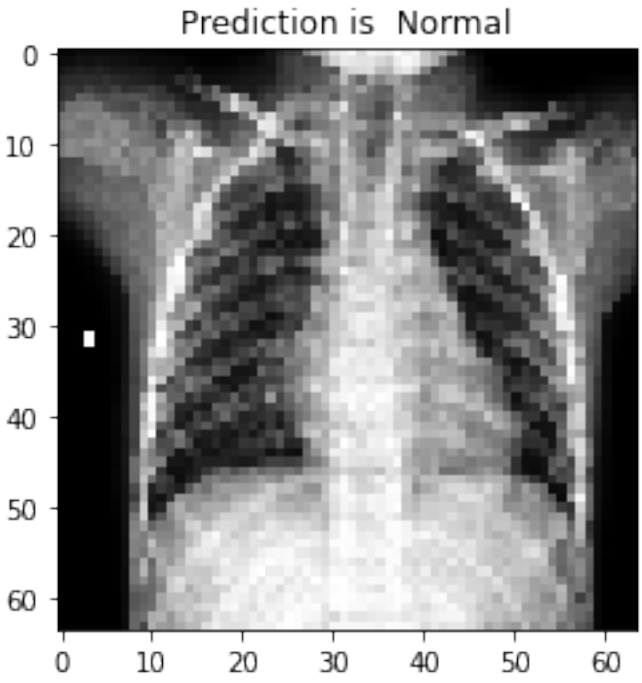
```
[ ]: model.compile(loss='binary_crossentropy', optimizer='adam',␣
      ↪metrics=['accuracy'])

     results = model.fit(train, epochs=20, validation_data=test, verbose=1)
```

```
Epoch 1/20
6/6 [==============================] - 6s 930ms/step - loss: 0.6709 - accuracy:
0.5801 - val_loss: 0.5371 - val_accuracy: 0.8913
Epoch 2/20
6/6 [==============================] - 5s 788ms/step - loss: 0.5606 - accuracy:
0.6851 - val_loss: 0.5075 - val_accuracy: 0.6304
Epoch 3/20
6/6 [==============================] - 5s 880ms/step - loss: 0.4522 - accuracy:
0.7901 - val_loss: 0.2759 - val_accuracy: 0.9130
Epoch 4/20
6/6 [==============================] - 5s 808ms/step - loss: 0.3191 - accuracy:
0.8619 - val_loss: 0.1390 - val_accuracy: 0.9348
Epoch 5/20
6/6 [==============================] - 5s 879ms/step - loss: 0.2765 - accuracy:
0.8674 - val_loss: 0.1792 - val_accuracy: 0.9130
Epoch 6/20
6/6 [==============================] - 5s 815ms/step - loss: 0.2152 - accuracy:
0.9116 - val_loss: 0.0899 - val_accuracy: 0.9565
Epoch 7/20
6/6 [==============================] - 5s 769ms/step - loss: 0.2157 - accuracy:
0.9227 - val_loss: 0.0415 - val_accuracy: 1.0000
Epoch 8/20
6/6 [==============================] - 5s 872ms/step - loss: 0.2949 - accuracy:
0.8619 - val_loss: 0.0370 - val_accuracy: 1.0000
Epoch 9/20
6/6 [==============================] - 5s 836ms/step - loss: 0.2829 - accuracy:
0.8895 - val_loss: 0.3306 - val_accuracy: 0.8478
Epoch 10/20
6/6 [==============================] - 5s 871ms/step - loss: 0.2693 - accuracy:
0.8895 - val_loss: 0.0708 - val_accuracy: 1.0000
Epoch 11/20
6/6 [==============================] - 5s 923ms/step - loss: 0.1842 - accuracy:
0.9337 - val_loss: 0.1801 - val_accuracy: 0.9130
Epoch 12/20
6/6 [==============================] - 5s 908ms/step - loss: 0.1784 - accuracy:
0.9392 - val_loss: 0.0627 - val_accuracy: 0.9783
Epoch 13/20
6/6 [==============================] - 5s 814ms/step - loss: 0.1354 - accuracy:
0.9503 - val_loss: 0.1128 - val_accuracy: 0.9348
Epoch 14/20
6/6 [==============================] - 5s 828ms/step - loss: 0.0949 - accuracy:
0.9669 - val_loss: 0.1720 - val_accuracy: 0.9130
Epoch 15/20
```

```
6/6 [==============================] - 5s 852ms/step - loss: 0.2256 - accuracy:
0.9282 - val_loss: 0.1047 - val_accuracy: 0.9348
Epoch 16/20
6/6 [==============================] - 5s 846ms/step - loss: 0.0959 - accuracy:
0.9669 - val_loss: 0.0668 - val_accuracy: 0.9783
Epoch 17/20
6/6 [==============================] - 5s 988ms/step - loss: 0.1744 - accuracy:
0.9392 - val_loss: 0.1030 - val_accuracy: 0.9565
Epoch 18/20
6/6 [==============================] - 5s 848ms/step - loss: 0.1155 - accuracy:
0.9669 - val_loss: 0.0581 - val_accuracy: 0.9783
Epoch 19/20
6/6 [==============================] - 5s 884ms/step - loss: 0.0755 - accuracy:
0.9669 - val_loss: 0.0459 - val_accuracy: 0.9783
Epoch 20/20
6/6 [==============================] - 5s 816ms/step - loss: 0.0726 - accuracy:
0.9834 - val_loss: 0.0214 - val_accuracy: 1.0000
```

[ ]:
```python
# Testing
print(train.class_indices)
#Then load the image to be classified.
img = load_img('test/Normal/0101.jpeg', target_size=(64, 64))
imgplot = plt.imshow(img)
test_image = load_img('test/Normal/0101.jpeg', target_size=(64, 64))
test_image = img_to_array(test_image)
test_image = np.expand_dims(test_image, axis=0)
result = model.predict(test_image)

if result[0][0] == 1:
    prediction ='Normal'
else:
    prediction ='Covid-19'

plt.title('Prediction is  '+ prediction )
plt.show()
```

```
{'Covid': 0, 'Normal': 1}
1/1 [==============================] - 0s 26ms/step
```


Prediction is Normal
0
10
20
30
40
50
60
0
10
20
30
40
50
60

WEEK-8

Outcome: Students are implement classification using CNN.

Pre Lab:

1) What is convolution?

A) Convolution is the process of adding each element of the image to its local neighbors, weighted by the kernel. This is related to a form of mathematical convolution. The matrix operation being performed—convolution—is not traditional matrix multiplication, despite being similarly denoted by " * ".

2) What is done in max pooling layer?

A) Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

3) What is flattening and full connected layers in CNN?

A) Flattening is used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector. The flattened matrix is fed as input to the fully connected layer to classify the image.

Fully Connected Layer is simply, feed forward neural networks. Fully Connected Layers form the last few layers in the network.

In Lab:

Exp 8: Build a deep learning model which classifies cats and dogs using CNN.

Program:

LAB8

October 18, 2022

[ ]:
```
import os
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing import image
from keras import layers, models, optimizers
from keras.preprocessing.image import ImageDataGenerator
```

[ ]:
```
train_dir = r'cats_and_dogs_small\train'
test_dir = r'cats_and_dogs_small\test'
```

[ ]:
```
train_data_gen = ImageDataGenerator(rescale=1./255)
test_data_gen = ImageDataGenerator(rescale=1./255)

train_gen = train_data_gen.flow_from_directory(
    train_dir,
    target_size = (150, 150),
    batch_size = 20,
    class_mode = 'binary')

test_gen = test_data_gen.flow_from_directory(
    test_dir,
    target_size = (150, 150),
    batch_size = 20,
    class_mode = 'binary')
```

```
Found 1000 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
```

[ ]:
```
cnn_model = models.Sequential()

cnn_model.add(layers.Conv2D(32, (3, 3), activation = 'relu', input_shape =␣
 ↪(150, 150, 3)))
cnn_model.add(layers.MaxPooling2D(2, 2))
```

```
cnn_model.add(layers.Conv2D(64, (3, 3), activation = 'relu'))
cnn_model.add(layers.MaxPooling2D(2, 2))

cnn_model.add(layers.Conv2D(128, (3, 3), activation = 'relu'))
cnn_model.add(layers.MaxPooling2D(2, 2))

cnn_model.add(layers.Conv2D(128, (3, 3), activation = 'relu'))
cnn_model.add(layers.MaxPooling2D(2, 2))

cnn_model.add(layers.Flatten())

cnn_model.add(layers.Dense(512, activation = 'relu'))

cnn_model.add(layers.Dense(1, activation = 'sigmoid'))
```

[ ]:
```
cnn_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 max_pooling2d (MaxPooling2D  (None, 74, 74, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 64)       0
 2D)


_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 max_pooling2d (MaxPooling2D  (None, 74, 74, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 17, 17, 128)      0
```

```
 2D)

 conv2d_3 (Conv2D)           (None, 15, 15, 128)       147584

 max_pooling2d_3 (MaxPooling  (None, 7, 7, 128)        0
 2D)

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 512)               3211776

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

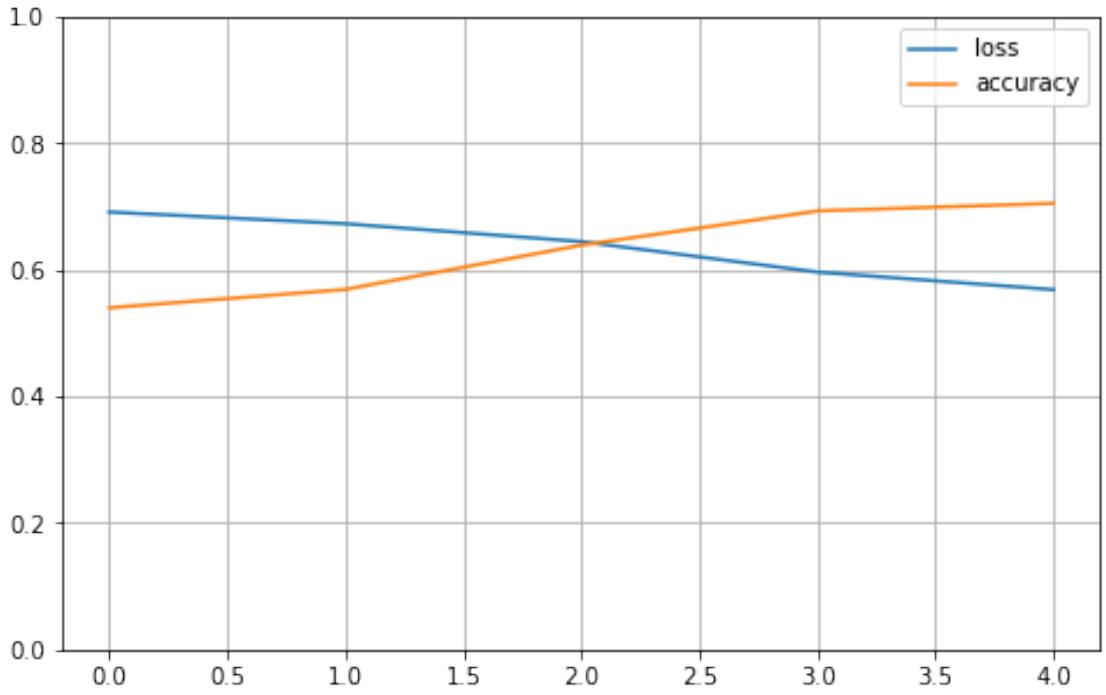
[ ]:
```
cnn_model.compile(loss = 'binary_crossentropy',
              optimizer = optimizers.RMSprop(lr = 1e-4),
              metrics = ['accuracy'])
```

```
c:\Users\Yash\AppData\Local\Programs\Python\Python39\lib\site-
packages\keras\optimizers\optimizer_v2\rmsprop.py:140: UserWarning: The `lr`
argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

[ ]:
```
model_history = cnn_model.fit(train_gen,
                                        steps_per_epoch = 50,
                                        epochs = 5)
```

```
Epoch 1/5
50/50 [==============================] - 51s 1s/step - loss: 0.6913 - accuracy:
0.5400
Epoch 2/5
50/50 [==============================] - 45s 912ms/step - loss: 0.6729 -
accuracy: 0.5690
Epoch 3/5
50/50 [==============================] - 48s 917ms/step - loss: 0.6446 -
accuracy: 0.6390
Epoch 4/5
50/50 [==============================] - 45s 904ms/step - loss: 0.5964 -
accuracy: 0.6930
Epoch 5/5
50/50 [==============================] - 48s 922ms/step - loss: 0.5687 -
accuracy: 0.7050
```

[ ]:
```
pd.DataFrame(model_history.history).plot(figsize = (8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```



[ ]:
```
cnn_model.evaluate(test_gen, steps = 50)
```

```
25/50 [==============>...] - ETA: 4s - loss: 0.6116 - accuracy:
0.6520WARNING:tensorflow:Your input ran out of data; interrupting training. Make
sure that your dataset or generator can generate at least `steps_per_epoch *
epochs` batches (in this case, 50 batches). You may need to use the repeat()
function when building your dataset.
50/50 [==============================] - 9s 172ms/step - loss: 0.6116 -
accuracy: 0.6520
```

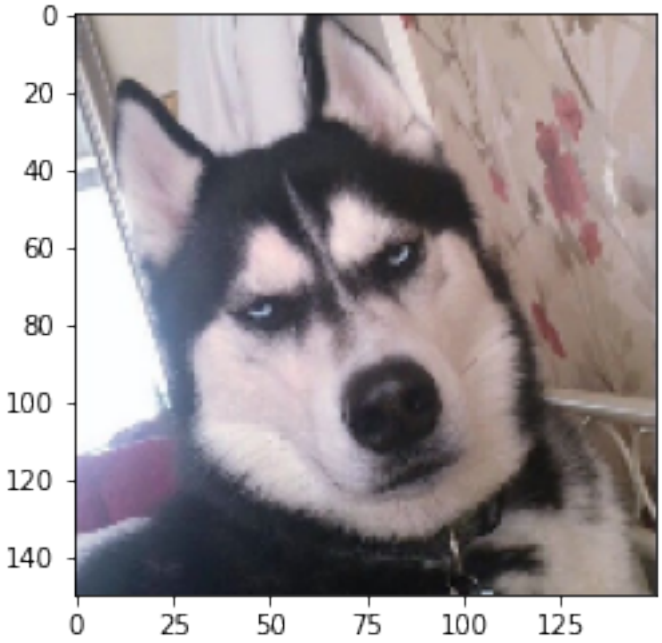
[ ]:
```
[0.6115962862968445, 0.6520000100135803]
```

[ ]:
```
img_path = "dog.jpg"
img_path1 = "cat.jpg"
```

[ ]:
```
img0 = image.load_img(img_path, target_size = (150, 150))
img1 = image.load_img(img_path1, target_size = (150, 150))
```

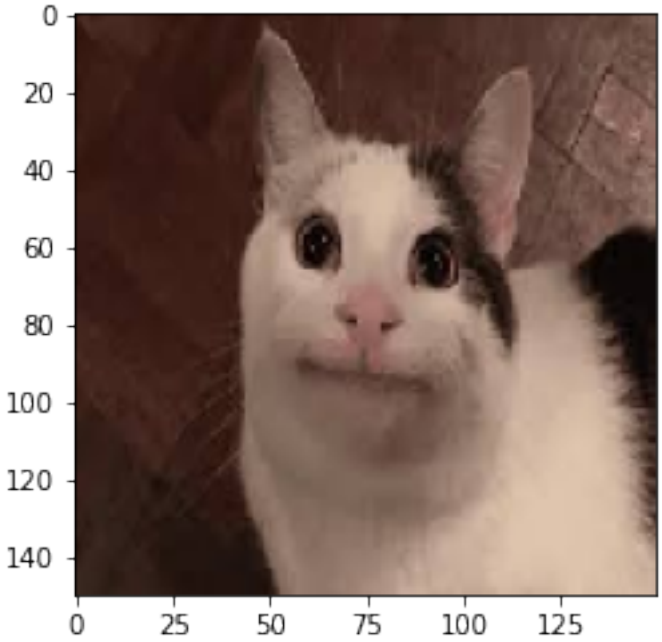
[ ]:
```
plt.imshow(img0)
```

[ ]: <matplotlib.image.AxesImage at 0x1e82fcfc340>



[ ]:
```
plt.imshow(img1)
```

[ ]: <matplotlib.image.AxesImage at 0x1e82fd5bd60>



[ ]:
```
img0 = image.img_to_array(img0)
img1 = image.img_to_array(img1)
```

[ ]:
```
x0 = np.expand_dims(img0, axis = 0)
x1 = np.expand_dims(img1, axis = 0)
```

[ ]:
```
img0 = np.vstack([x0])
img1 = np.vstack([x1])
```

[ ]:
```
result0 = cnn_model.predict(img0)
result1 = cnn_model.predict(img1)
```

```
1/1 [==============================] - 0s 214ms/step
1/1 [==============================] - 0s 214ms/step
1/1 [==============================] - 0s 36ms/step
```

[ ]:
```
# when the value is near to 1 it is classified as dog and
# when the result is near to 0 it is classified as a car
print(result0, result1)
```

```
[[1.7730022e-29]] [[1.]]
```

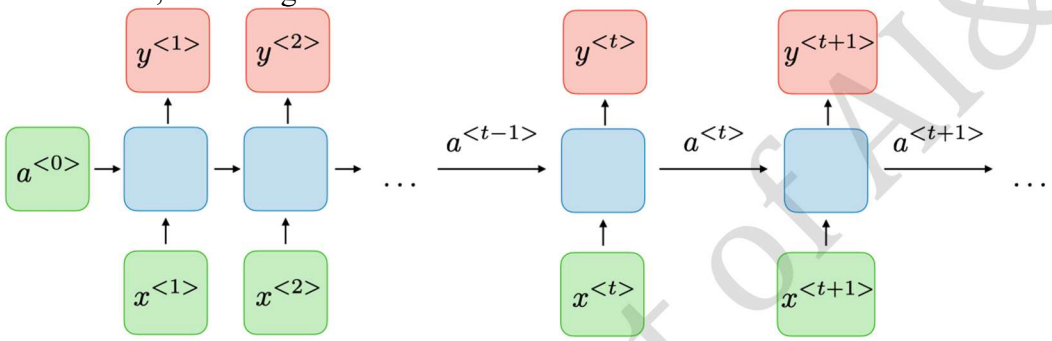
WEEK-9

Outcome: Students are able to implement simple RNN and visualize using matplotlib.

Pre Lab:

1) What is Recurrent Neural Network?

A) A recurrent neural network (RNN) is a type of artificial neural network which uses sequential data or time series data. These deep learning algorithms are commonly used for ordinal or temporal problems, such as language translation, natural language processing (nlp), speech recognition, and image captioning; they are incorporated into popular applications such as Siri, voice search, and Google Translate.



2) What is Time Stamp in RNN?

A) Time stamps have nothing to do with past, present, and future. Here time stamp represents a word or an item in a long sequence.
Example : Consider the sequence [“I”, “am”, “a”, “boy”]. Time stamp for “I” is x(0), “am” is x(1), “a” is x(2) and “boy” is x(3).
If t=1,
x(t) = “am” →“*Event at current time stamp”*
x(t-1) = “I” → “Event at previous time stamp”

3) How many types of RNN are there at present?

A) The different types of RNN are:

1. One to One RNN
2. One to Many RNN
3. Many to One RNN
4. Many to Many RNN

In Lab:

EXP. 9:

The stock price of the present day can be predicted by the stock prices obtained for past 50 days. Using Simple RNN, train the model with "Google_stock_price_train.csv" dataset, predict the stock prices for the dates given in "Google_stock_price_train.csv" dataset and visualize the actual, predicted prices using matplotlib.

Program:

LAB-9

October 18, 2022

[ ]:
```
"""
The stock price of the present day can be predicted by the stock prices␣
 ↪obtained for past 50 days.
Using Simple RNN, train the model with "Google_stock_price_train.csv" dataset,
predict the stock prices for the dates given in "Google_stock_price_train.csv"␣
 ↪dataset and
visualize the actual, predicted prices using matplotlib.
"""
```

[ ]:
```
'\nThe stock price of the present day can be predicted by the stock prices
obtained for past 50 days. \nUsing Simple RNN, train the model with
"Google_stock_price_train.csv" dataset, \npredict the stock prices for the dates
given in "Google_stock_price_train.csv" dataset and \nvisualize the actual,
predicted prices using matplotlib.\n'
```

[ ]:
```
# Importing the libraries

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
import keras
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN, Dropout
```

[ ]:
```
# Importing the training set

dataset_train = pd.read_csv('Google_Stock_Price_Train.csv')
dataset_train.head()
```

[ ]:

	Date	Open	High	Low	Close	Volume
0	1/3/2012	325.25	332.83	324.97	663.59	7,380,500
1	1/4/2012	331.27	333.87	329.08	666.45	5,749,400
2	1/5/2012	329.83	330.75	326.89	657.21	6,590,300
3	1/6/2012	328.34	328.77	323.68	648.24	5,405,900
4	1/9/2012	322.04	322.29	309.46	620.76	11,688,800

[ ]:
```
training_set = dataset_train.iloc[:, 1:2].values
training_set.shape
```

[ ]:
```
(1258, 1)
```

[ ]:
```
scaler = MinMaxScaler(feature_range = (0, 1))
scaled_training_set = scaler.fit_transform(training_set)
```

[ ]:
```
X_train = []
y_train = []

for i in range(50, 1258):
    X_train.append(scaled_training_set[i-50:i, 0])
    y_train.append(scaled_training_set[i, 0])

X_train, y_train = np.array(X_train), np.array(y_train)
```

[ ]:
```
#Print the shape of X_train and y_train

X_train.shape, y_train.shape
```

[ ]:
```
((1208, 50), (1208,))
```

[ ]:
```
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_train.shape
```

[ ]:
```
(1208, 50, 1)
```

[ ]:
```
regressor = Sequential()

regressor.add(SimpleRNN(units = 50,activation='relu', return_sequences = True,␣
 ↪input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(0.2))

regressor.add(SimpleRNN(units = 50, activation='relu', return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(SimpleRNN(units = 50, activation='tanh', return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(SimpleRNN(units = 50))
regressor.add(Dropout(0.2))

regressor.add(Dense(units = 1))
```

[ ]:
```
regressor.compile(optimizer='adam', loss='mse')
regressor.fit(X_train, y_train, epochs=100, batch_size=32)
```

```
Epoch 1/100
38/38 [==============================] - 16s 88ms/step - loss: 0.1810
Epoch 2/100
38/38 [==============================] - 3s 84ms/step - loss: 0.0604
Epoch 3/100
38/38 [==============================] - 4s 97ms/step - loss: 0.0323
Epoch 4/100
38/38 [==============================] - 3s 88ms/step - loss: 0.0210
Epoch 5/100
38/38 [==============================] - 3s 82ms/step - loss: 0.0145
Epoch 6/100
38/38 [==============================] - 3s 79ms/step - loss: 0.0103
Epoch 7/100
38/38 [==============================] - 3s 88ms/step - loss: 0.0103
Epoch 8/100
38/38 [==============================] - 3s 83ms/step - loss: 0.0084
Epoch 9/100
38/38 [==============================] - 3s 82ms/step - loss: 0.0083
Epoch 10/100
38/38 [==============================] - 3s 74ms/step - loss: 0.0075
Epoch 11/100
38/38 [==============================] - 3s 84ms/step - loss: 0.0081
Epoch 12/100
38/38 [==============================] - 4s 94ms/step - loss: 0.0065
Epoch 13/100
38/38 [==============================] - 3s 87ms/step - loss: 0.0061
Epoch 14/100
38/38 [==============================] - 3s 83ms/step - loss: 0.0061
Epoch 15/100
38/38 [==============================] - 3s 85ms/step - loss: 0.0057
Epoch 16/100
38/38 [==============================] - 3s 90ms/step - loss: 0.0065
Epoch 17/100
38/38 [==============================] - 4s 100ms/step - loss: 0.0053
Epoch 18/100
38/38 [==============================] - 3s 81ms/step - loss: 0.0049
Epoch 19/100
38/38 [==============================] - 4s 106ms/step - loss: 0.0049
Epoch 20/100
38/38 [==============================] - 3s 91ms/step - loss: 0.0050
Epoch 21/100
38/38 [==============================] - 4s 115ms/step - loss: 0.0048
Epoch 22/100
38/38 [==============================] - 4s 117ms/step - loss: 0.0041
Epoch 23/100
38/38 [==============================] - 4s 115ms/step - loss: 0.0046
Epoch 24/100
38/38 [==============================] - 4s 105ms/step - loss: 0.0039
```

```
Epoch 25/100
38/38 [==============================] - 4s 110ms/step - loss: 0.0042
Epoch 26/100
38/38 [==============================] - 4s 111ms/step - loss: 0.0040
Epoch 27/100
38/38 [==============================] - 4s 106ms/step - loss: 0.0039
Epoch 28/100
38/38 [==============================] - 4s 113ms/step - loss: 0.0042
Epoch 29/100
38/38 [==============================] - 4s 112ms/step - loss: 0.0042
Epoch 30/100
38/38 [==============================] - 4s 114ms/step - loss: 0.0035
Epoch 31/100
38/38 [==============================] - 4s 108ms/step - loss: 0.0036
Epoch 32/100
38/38 [==============================] - 4s 116ms/step - loss: 0.0038
Epoch 33/100
38/38 [==============================] - 4s 114ms/step - loss: 0.0033
Epoch 34/100
38/38 [==============================] - 5s 134ms/step - loss: 0.0038
Epoch 35/100
38/38 [==============================] - 4s 116ms/step - loss: 0.0040
Epoch 36/100
38/38 [==============================] - 5s 121ms/step - loss: 0.0033
Epoch 37/100
38/38 [==============================] - 4s 95ms/step - loss: 0.0037
Epoch 38/100
38/38 [==============================] - 4s 102ms/step - loss: 0.0032
Epoch 39/100
38/38 [==============================] - 4s 95ms/step - loss: 0.0030
Epoch 40/100
38/38 [==============================] - 4s 102ms/step - loss: 0.0032
Epoch 41/100
38/38 [==============================] - 4s 103ms/step - loss: 0.0031
Epoch 42/100
38/38 [==============================] - 4s 94ms/step - loss: 0.0034
Epoch 43/100
38/38 [==============================] - 4s 97ms/step - loss: 0.0029
Epoch 44/100
38/38 [==============================] - 4s 97ms/step - loss: 0.0029
Epoch 45/100
38/38 [==============================] - 4s 98ms/step - loss: 0.0031
Epoch 46/100
38/38 [==============================] - 4s 98ms/step - loss: 0.0031
Epoch 47/100
38/38 [==============================] - 4s 98ms/step - loss: 0.0031
Epoch 48/100
38/38 [==============================] - 4s 98ms/step - loss: 0.0027
```

```
Epoch 49/100
38/38 [==============================] - 4s 94ms/step - loss: 0.0028
Epoch 50/100
38/38 [==============================] - 4s 93ms/step - loss: 0.0034
Epoch 51/100
38/38 [==============================] - 4s 92ms/step - loss: 0.0029
Epoch 52/100
38/38 [==============================] - 4s 92ms/step - loss: 0.0032
Epoch 53/100
38/38 [==============================] - 4s 94ms/step - loss: 0.0026
Epoch 54/100
38/38 [==============================] - 4s 96ms/step - loss: 0.0030
Epoch 55/100
38/38 [==============================] - 4s 97ms/step - loss: 0.0029
Epoch 56/100
38/38 [==============================] - 4s 94ms/step - loss: 0.0029
Epoch 57/100
38/38 [==============================] - 4s 94ms/step - loss: 0.0028
Epoch 58/100
38/38 [==============================] - 4s 94ms/step - loss: 0.0025
Epoch 59/100
38/38 [==============================] - 4s 94ms/step - loss: 0.0025
Epoch 60/100
38/38 [==============================] - 4s 94ms/step - loss: 0.0027
Epoch 61/100
38/38 [==============================] - 4s 99ms/step - loss: 0.0026
Epoch 62/100
38/38 [==============================] - 4s 94ms/step - loss: 0.0026
Epoch 63/100
38/38 [==============================] - 4s 99ms/step - loss: 0.0027
Epoch 64/100
38/38 [==============================] - 4s 93ms/step - loss: 0.0025
Epoch 65/100
38/38 [==============================] - 4s 93ms/step - loss: 0.0024
Epoch 66/100
38/38 [==============================] - 4s 95ms/step - loss: 0.0026
Epoch 67/100
38/38 [==============================] - 4s 105ms/step - loss: 0.0025
Epoch 68/100
38/38 [==============================] - 4s 105ms/step - loss: 0.0022
Epoch 69/100
38/38 [==============================] - 4s 104ms/step - loss: 0.0023
Epoch 70/100
38/38 [==============================] - 4s 96ms/step - loss: 0.0025
Epoch 71/100
38/38 [==============================] - 4s 98ms/step - loss: 0.0020
Epoch 72/100
38/38 [==============================] - 4s 95ms/step - loss: 0.0025
```

```
Epoch 73/100
38/38 [==============================] - 3s 90ms/step - loss: 0.0021
Epoch 74/100
38/38 [==============================] - 3s 90ms/step - loss: 0.0024
Epoch 75/100
38/38 [==============================] - 4s 103ms/step - loss: 0.0024
Epoch 76/100
38/38 [==============================] - 4s 102ms/step - loss: 0.0023
Epoch 77/100
38/38 [==============================] - 4s 101ms/step - loss: 0.0021
Epoch 78/100
38/38 [==============================] - 4s 101ms/step - loss: 0.0022
Epoch 79/100
38/38 [==============================] - 3s 79ms/step - loss: 0.0022
Epoch 80/100
38/38 [==============================] - 4s 92ms/step - loss: 0.0020
Epoch 81/100
38/38 [==============================] - 5s 141ms/step - loss: 0.0022
Epoch 82/100
38/38 [==============================] - 5s 135ms/step - loss: 0.0022
Epoch 83/100
38/38 [==============================] - 4s 115ms/step - loss: 0.0023
Epoch 84/100
38/38 [==============================] - 4s 111ms/step - loss: 0.0021
Epoch 85/100
38/38 [==============================] - 4s 109ms/step - loss: 0.0021
Epoch 86/100
38/38 [==============================] - 4s 106ms/step - loss: 0.0019
Epoch 87/100
38/38 [==============================] - 4s 101ms/step - loss: 0.0019
Epoch 88/100
38/38 [==============================] - 4s 98ms/step - loss: 0.0022
Epoch 89/100
38/38 [==============================] - 3s 92ms/step - loss: 0.0020
Epoch 90/100
38/38 [==============================] - 3s 91ms/step - loss: 0.0019
Epoch 91/100
38/38 [==============================] - 3s 90ms/step - loss: 0.0022
Epoch 92/100
38/38 [==============================] - 3s 92ms/step - loss: 0.0020
Epoch 93/100
38/38 [==============================] - 3s 84ms/step - loss: 0.0020
Epoch 94/100
38/38 [==============================] - 3s 83ms/step - loss: 0.0021
Epoch 95/100
38/38 [==============================] - 3s 87ms/step - loss: 0.0021
Epoch 96/100
38/38 [==============================] - 3s 83ms/step - loss: 0.0021
```

```
Epoch 97/100
38/38 [==============================] - 3s 81ms/step - loss: 0.0019
Epoch 98/100
38/38 [==============================] - 3s 83ms/step - loss: 0.0019
Epoch 99/100
38/38 [==============================] - 3s 85ms/step - loss: 0.0018
Epoch 100/100
38/38 [==============================] - 3s 82ms/step - loss: 0.0018
```

[ ]:
```
<keras.callbacks.History at 0x2b42d3e93d0>
```

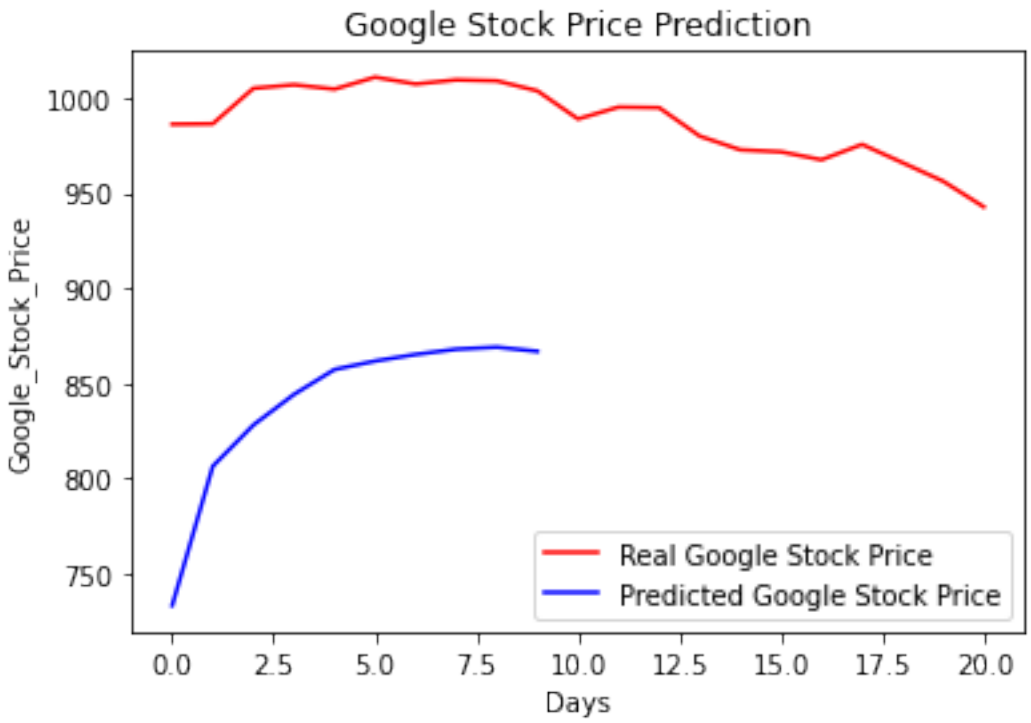
[ ]:
```
dataset_test = pd.read_csv('Google_Stock_Price_Test.csv')
actual_stock_price = dataset_test.iloc[:, 1:2].values
```

[ ]:
```
dataset_total = pd.concat((dataset_train['Open'], dataset_test['Open']), axis =␣
↪0)
inputs = dataset_total[len(dataset_total) - len(dataset_test) - 50:].values

inputs = inputs.reshape(-1,1)
inputs = scaler.transform(inputs)

X_test = []
for i in range(50, 60):
    X_test.append(inputs[i-50:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

[ ]:
```
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = scaler.inverse_transform(predicted_stock_price)
```

```
1/1 [==============================] - 1s 781ms/step
1/1 [==============================] - 1s 781ms/step
```

[ ]:
```
plt.plot(actual_stock_price, color='red', label = 'Real Google Stock Price')
plt.plot(predicted_stock_price, color='blue', label = 'Predicted Google Stock␣
↪Price')
plt.title('Google Stock Price Prediction')
plt.xlabel('Days')
plt.ylabel('Google_Stock_Price')
plt.legend()
plt.show()
```


Google Stock Price Prediction
Google_Stock_Price
1000
950
900
850
800
750
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Days
Real Google Stock Price
Predicted Google Stock Price

WEEK-10

Outcome: Students are able to use LSTM to model and visualize the result.

Pre Lab:

1) What is meant by vanishing gradient?

A) When the neural networks are trained with gradient-based learning methods and backpropagation, they encounter the vanishing gradient problem. In this problem, at the time of training, the gradient starts getting smaller in size which prevents the neural networks from getting trained by not letting the network weights be changed. The VGP occurs when the elements of the gradient (the partial derivatives with respect to the parameters of the NN) become exponentially small so that the update of the parameters with the gradient becomes almost insignificant.

2) Briefly Describe about LSTM?

A) Long Short Term Memory is a kind of recurrent neural network. In RNN output from the last step is fed as input in the current step. It tackled the problem of long-term dependencies of RNN in which the RNN cannot predict the word stored in the long-term memory but can give more accurate predictions from the recent information. As the gap length increases RNN does not give an efficient performance. LSTM can by default retain the information for a long period of time. It is used for processing, predicting, and classifying on the basis of time-series data.

3) Advantages of LSTM over simple RNN.

Ans)

- **A)** LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs.
- **B)** LSTM can handle the information in memory for the long period of time as compare to RNN.
- **C)** LSTMs deal with vanishing and exploding gradient problem by introducing new gates, such as input and forget gates, which allow for a better control over the gradient flow and enable better preservation of "long-range dependencies".
- **D)** The long range dependency in RNN is resolved by increasing the number of repeating layer in LSTM.

In Lab:

EXP 10:

Using LSTM train the model with train dataset, predict the covid death cases and confirmed cases. Visualize the actual, predict data using matplotlib. Use 20days time stamp.

Program

LAB-10(IN-LAB)

October 18, 2022

[ ]:
```
# Recurrent Neural Network

# Part 1 - Data Preprocessing

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

[ ]:
```
# Importing the training set
dataset_train = pd.read_csv('covid_india_train.csv')
training_set = dataset_train.iloc[:,1:2].values
```

[ ]:
```
# Feature Scaling
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_set)
```

[ ]:
```
# Creating a data structure with 20 timesteps and t+1 output
X_train = []
y_train = []
for i in range(20, 125):
    X_train.append(training_set_scaled[i-20:i, 0])
    y_train.append(training_set_scaled[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)
```

[ ]:
```
# Reshaping
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
```

1 Part 2 - Building the RNN

[ ]:
```
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
```

[ ]:
```
# Initialising the RNN
regressor = Sequential()

# Adding the input layer and the LSTM layer
regressor.add(LSTM(units = 3,activation='sigmoid',input_shape = (None, 1)))

# Adding the output layer
regressor.add(Dense(units = 1))

# Compiling the RNN
regressor.compile(optimizer = 'rmsprop', loss = 'mean_squared_error')
```

[ ]:
```
# Fitting the RNN to the Training set
regressor.fit(X_train, y_train, epochs = 1000, batch_size = 32)
```

```
Epoch 1/1000
4/4 [==============================] - 3s 7ms/step - loss: 0.0815
Epoch 2/1000
4/4 [==============================] - 0s 6ms/step - loss: 0.0786
Epoch 3/1000
4/4 [==============================] - 0s 6ms/step - loss: 0.0775
Epoch 4/1000
4/4 [==============================] - 0s 5ms/step - loss: 0.0766
Epoch 5/1000
4/4 [==============================] - 0s 6ms/step - loss: 0.0760
Epoch 6/1000
4/4 [==============================] - 0s 8ms/step - loss: 0.0754
Epoch 7/1000
4/4 [==============================] - 0s 7ms/step - loss: 0.0749
Epoch 8/1000
4/4 [==============================] - 0s 6ms/step - loss: 0.0746
Epoch 9/1000
4/4 [==============================] - 0s 5ms/step - loss: 0.0739
Epoch 10/1000
4/4 [==============================] - 0s 6ms/step - loss: 0.0735
Epoch 11/1000
4/4 [==============================] - 0s 7ms/step - loss: 0.0728
Epoch 12/1000
4/4 [==============================] - 0s 7ms/step - loss: 0.0721
Epoch 13/1000
4/4 [==============================] - 0s 6ms/step - loss: 0.0716
Epoch 14/1000
4/4 [==============================] - 0s 6ms/step - loss: 0.0713
Epoch 15/1000
4/4 [==============================] - 0s 6ms/step - loss: 0.0706
Epoch 16/1000
4/4 [==============================] - 0s 6ms/step - loss: 0.0701
Epoch 17/1000
```

```
4/4 [==============================] - 0s 72ms/step - loss: 0.0012
Epoch 978/1000
4/4 [==============================] - 0s 63ms/step - loss: 0.0013
Epoch 979/1000
4/4 [==============================] - 0s 61ms/step - loss: 0.0012
Epoch 980/1000
4/4 [==============================] - 0s 60ms/step - loss: 0.0012
Epoch 981/1000
4/4 [==============================] - 0s 58ms/step - loss: 0.0012
Epoch 982/1000
4/4 [==============================] - 0s 57ms/step - loss: 0.0012
Epoch 983/1000
4/4 [==============================] - 0s 64ms/step - loss: 0.0013
Epoch 984/1000
4/4 [==============================] - 0s 66ms/step - loss: 0.0013
Epoch 985/1000
4/4 [==============================] - 0s 62ms/step - loss: 0.0012
Epoch 986/1000
4/4 [==============================] - 0s 75ms/step - loss: 0.0012
Epoch 987/1000
4/4 [==============================] - 0s 80ms/step - loss: 0.0012
Epoch 988/1000
4/4 [==============================] - 0s 65ms/step - loss: 0.0012
Epoch 989/1000
4/4 [==============================] - 0s 66ms/step - loss: 0.0012
Epoch 990/1000
4/4 [==============================] - 0s 10ms/step - loss: 0.0013
Epoch 991/1000
4/4 [==============================] - 0s 8ms/step - loss: 0.0013
Epoch 992/1000
4/4 [==============================] - 0s 7ms/step - loss: 0.0013
Epoch 993/1000
4/4 [==============================] - 0s 6ms/step - loss: 0.0012
Epoch 994/1000
4/4 [==============================] - 0s 6ms/step - loss: 0.0013
Epoch 995/1000
4/4 [==============================] - 0s 6ms/step - loss: 0.0012
Epoch 996/1000
4/4 [==============================] - 0s 7ms/step - loss: 0.0012
Epoch 997/1000
4/4 [==============================] - 0s 10ms/step - loss: 0.0012
Epoch 998/1000
4/4 [==============================] - 0s 7ms/step - loss: 0.0012
Epoch 999/1000
4/4 [==============================] - 0s 6ms/step - loss: 0.0012
Epoch 1000/1000
4/4 [==============================] - 0s 6ms/step - loss: 0.0012
```

[ ]: <keras.callbacks.History at 0x20fbe927160>

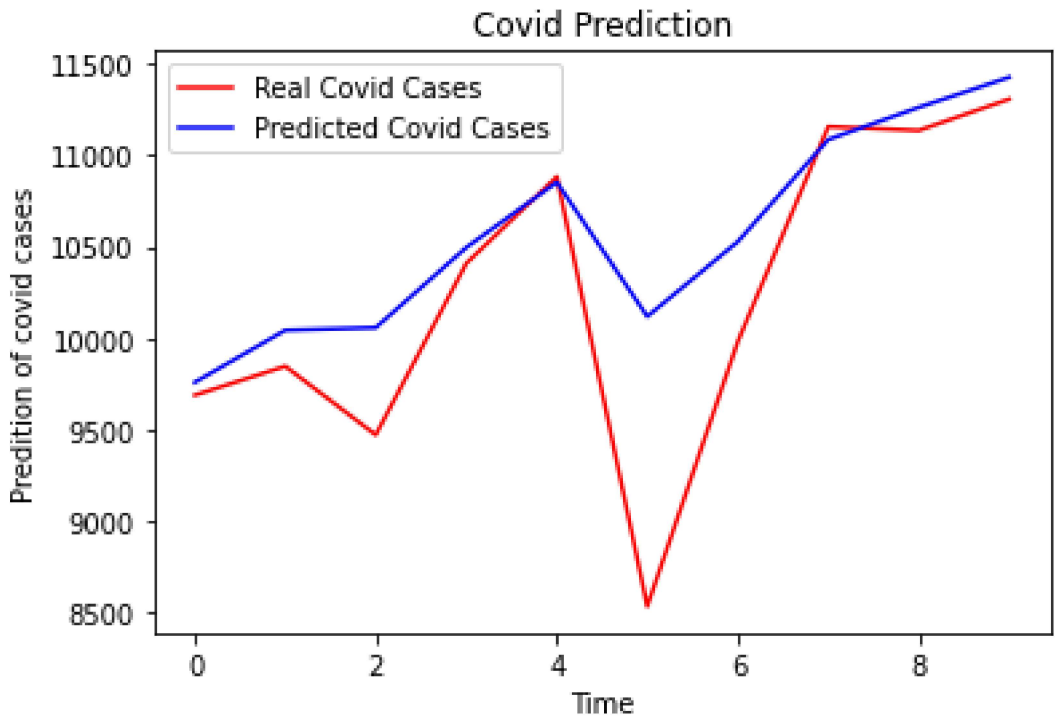
[ ]:
```
# Joining the train and test dataset under the real no. of covid cases in India
dataset_test = pd.read_csv('covid_india_test.csv')
test_set = dataset_test.iloc[:,1:2].values
real_covid_cases = np.concatenate((training_set[0:125], test_set), axis = 0)
```

[ ]:
```
# Getting the predicted number of cases
scaled_real_covid_cases= sc.fit_transform(real_covid_cases)
inputs = []
for i in range(126, 136):
    inputs.append(scaled_real_covid_cases[i-20:i, 0])
inputs = np.array(inputs)
inputs = np.reshape(inputs, (inputs.shape[0], inputs.shape[1], 1))
```

[ ]:
```
predicted_covid_cases = regressor.predict(inputs)

predicted_covid_cases = sc.inverse_transform(predicted_covid_cases)
predicted_covid_cases=predicted_covid_cases.astype('int64')
```

```
1/1 [==============================] - 0s 217ms/step
1/1 [==============================] - 0s 217ms/step
```

[ ]:
```
# Visualising the results
plt.plot(test_set, color = 'red', label = 'Real Covid Cases')
plt.plot(predicted_covid_cases, color = 'blue', label = 'Predicted Covid Cases')
plt.title('Covid Prediction')
plt.xlabel('Time')
plt.ylabel('Predition of covid cases')
plt.legend()
plt.show()
```



[ ]:
```
import math
from sklearn.metrics import mean_squared_error
rmse=math.sqrt(mean_squared_error(test_set,predicted_covid_cases))
```

[ ]:
```
#predict the next 2 days case
# Feature Scaling
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
real_covid_cases_scaled = sc.fit_transform(real_covid_cases)
input_next_pred=real_covid_cases_scaled.reshape(-1,1)
input_next_pred = np.asarray(input_next_pred).astype(np.float32)
```

[ ]:
```
#The problem's rooted in using lists as inputs, as opposed to Numpy arrays;␣
 ↪Keras/TF doesn't support former.
# A simple conversion is: x_array = np.asarray(x_list).

#Converting everything into np.float32 solved the issue!
#x_train = np.asarray(x_train).astype(np.float32)
#y_train = np.asarray(y_train).astype(np.float32)

#or it may be due to null values in the dataset

list_input=list(input_next_pred)
```

```
from numpy import array
lst_output=[]
n_steps=135
i=0
while(i<2):

    if(len(list_input)>135):
        #print(temp_input)
        input_next_pred=np.array(list_input[1:])
        #print("{} day input {}".format(i,x_input))
        input_next_pred=input_next_pred.reshape(1,-1)
        input_next_pred = input_next_pred.reshape((1, n_steps, 1))
        input_next_pred  = input_next_pred.astype(np.float32)
        #print(x_input)
        y_pred = regressor.predict(input_next_pred, verbose=0)
        y_pred = sc.inverse_transform(y_pred)
        y_pred=y_pred.astype('int64')

        #print("{} day output {}".format(i,yhat))
        list_input.extend(y_pred[0].tolist())
        list_input=list_input[1:]
        #print(temp_input)
        lst_output.extend(y_pred.tolist())
        i=i+1
    else:
        input_next_pred= input_next_pred.reshape((1, n_steps,1))
        y_pred = regressor.predict(input_next_pred, verbose=0)
        y_pred = sc.inverse_transform(y_pred)
        y_pred=y_pred.astype('int64')
        print(y_pred[0])
        list_input.extend(y_pred[0].tolist())
        print(len(list_input))
        lst_output.extend(y_pred.tolist())
        i=i+1
```

```
[11521]
136
```

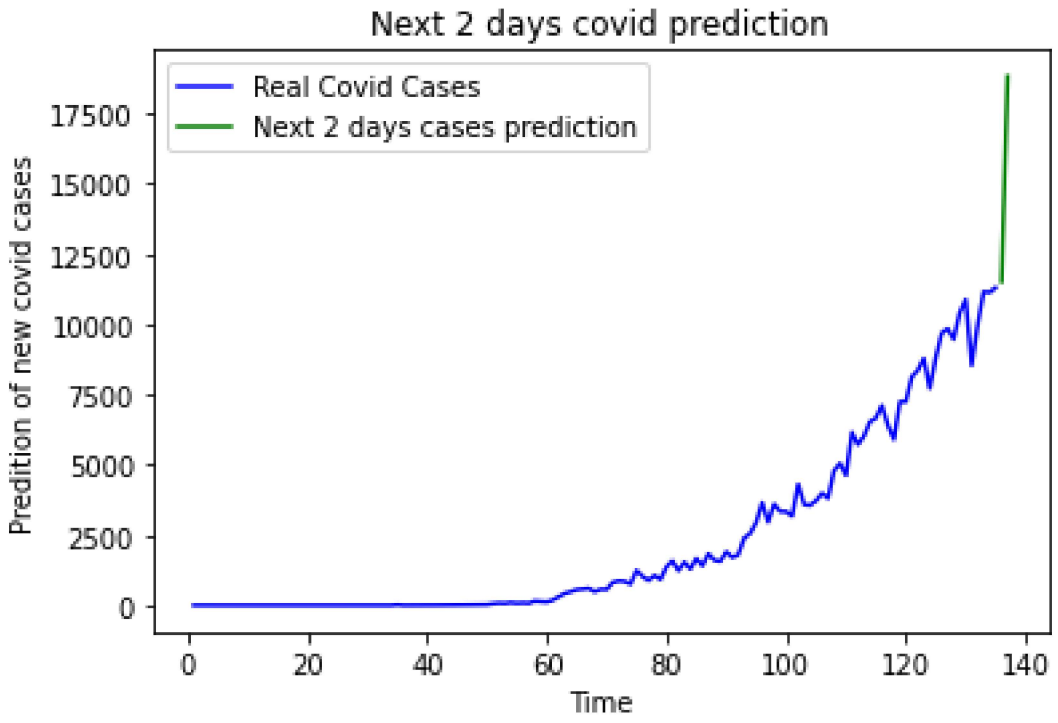
```
C:\Users\Yash\AppData\Local\Temp/ipykernel_15724/3301925112.py:19:
VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  input_next_pred=np.array(list_input[1:])
```

[ ]:
```
print("Total covid cases in India on 13 June will be apporx: ",lst_output[0])
print("Total covid cases in India on 14 June will be apporx: ",lst_output[1])
```

```
day_new=np.arange(1,136)
day_pred=np.arange(136,138)
day_new=day_new.reshape(-1,1)
```

```
Total covid cases in India on 13 June will be apporx:  [11521]
Total covid cases in India on 14 June will be apporx:  [18830]
```

[ ]:
```
plt.plot(day_new,real_covid_cases,color = 'blue', label = 'Real Covid Cases')
plt.plot(day_pred,lst_output,color = 'green', label = 'Next 2 days cases␣
 ↪prediction')
plt.title('Next 2 days covid prediction')
plt.xlabel('Time')
plt.ylabel('Predition of new covid cases')
plt.legend()
plt.show()
```



WEEK-12

Outcome: Students are able to implement denoisy auto encoder.

Pre Lab:

1) What are denoisy auto encoders?

A) A Denoising Autoencoder is a modification on the autoencoder to prevent the network learning the identity function. Specifically, if the autoencoder is too big, then it can just learn the data, so the output equals the input, and does not perform any useful representation learning or dimensionality reduction. Denoising autoencoders solve this problem by corrupting the input data on purpose, adding noise or masking some of the input values.

2) What is meant by Convolutional Auto Encoder?

A) A convolutional autoencoder is a neural network (a special case of an unsupervised learning model) that is trained to reproduce its input image in the output layer. An image is passed through an encoder, which is a ConvNet that produces a low-dimensional representation of the image. The decoder, which is another sample ConvNet, takes this compressed image and reconstructs the original image. The encoder is used to compress the data and the decoder is used to reproduce the original image. Therefore, autoencoders may be used for data, compression. Compression logic is data-specific, meaning it is learned from data rather than predefined compression algorithms such as JPEG, MP3, and so on.

3) What is the difference between self organizing maps and auto encoders?

A) The Autoencoder (AE) is a kind of artificial neural network, which is widely used for dimensionality reduction and feature extraction in unsupervised learning tasks. Analogously, the Self-Organizing Map (SOM) is an unsupervised learning algorithm to represent the high-dimensional data by a 2D grid map, thus achieving dimensionality reduction. Some recent work has shown improvement in performance by combining the AEs with the SOMs.

2000080132_WEEK-12

October 30, 2022

[ ]:
```
from keras.datasets import mnist
import numpy as np
import keras
from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D
from keras.callbacks import TensorBoard
import matplotlib.pyplot as plt
```

[ ]:
```
#making the noisy images using the mnist dataset

(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.


#normalizing the images
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
```

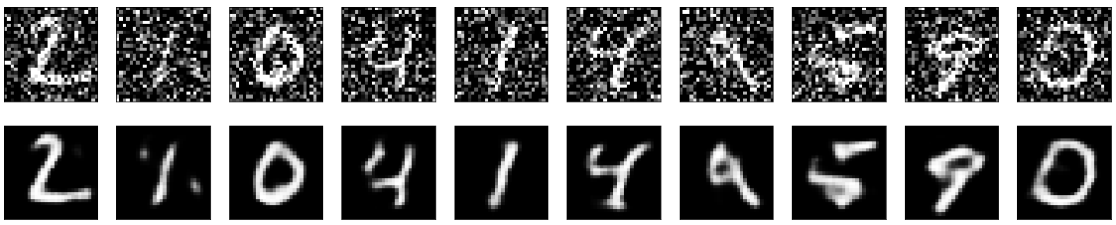
[ ]:
```
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0,␣
 ↪size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0,␣
 ↪size=x_test.shape)

x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

[ ]:
```
#The convolution autoencoder model
input_img = Input(shape=(28, 28, 1))

x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)

encoded = MaxPooling2D((2, 2), padding='same')(x)
```

[ ]:
```
#at this point representation is (7, 7, 32)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)

decoded =Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)
```

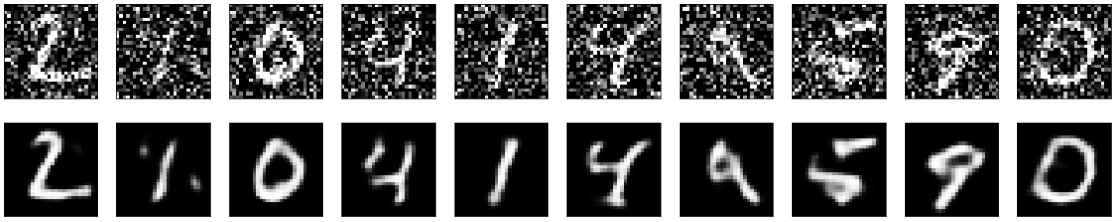
[ ]:
```
autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')␣
 ↪#optimizer='adadelta'

autoencoder.fit(x_train_noisy, x_train,
                epochs=2,
                batch_size=128,
                shuffle=True,
                validation_data=(x_test_noisy, x_test),
                callbacks=[TensorBoard(log_dir='/tmp/autoencoder',␣
 ↪histogram_freq=0, write_graph=False)])

decoded_imgs = autoencoder.predict(x_test_noisy)
```

```
Epoch 1/2
469/469 [==============================] - 137s 290ms/step - loss: 0.1677 -
val_loss: 0.1163
Epoch 2/2
469/469 [==============================] - 109s 232ms/step - loss: 0.1131 -
val_loss: 0.1085
```



[ ]:
```
n = 10
plt.figure(figsize=(20, 4))

for i in range(1,n+1):
    ax = plt.subplot(2, n, i)
    plt.imshow(x_test_noisy[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
```

```
    ax.get_yaxis().set_visible(False)

    ax =plt.subplot(2,n,i+n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()
```

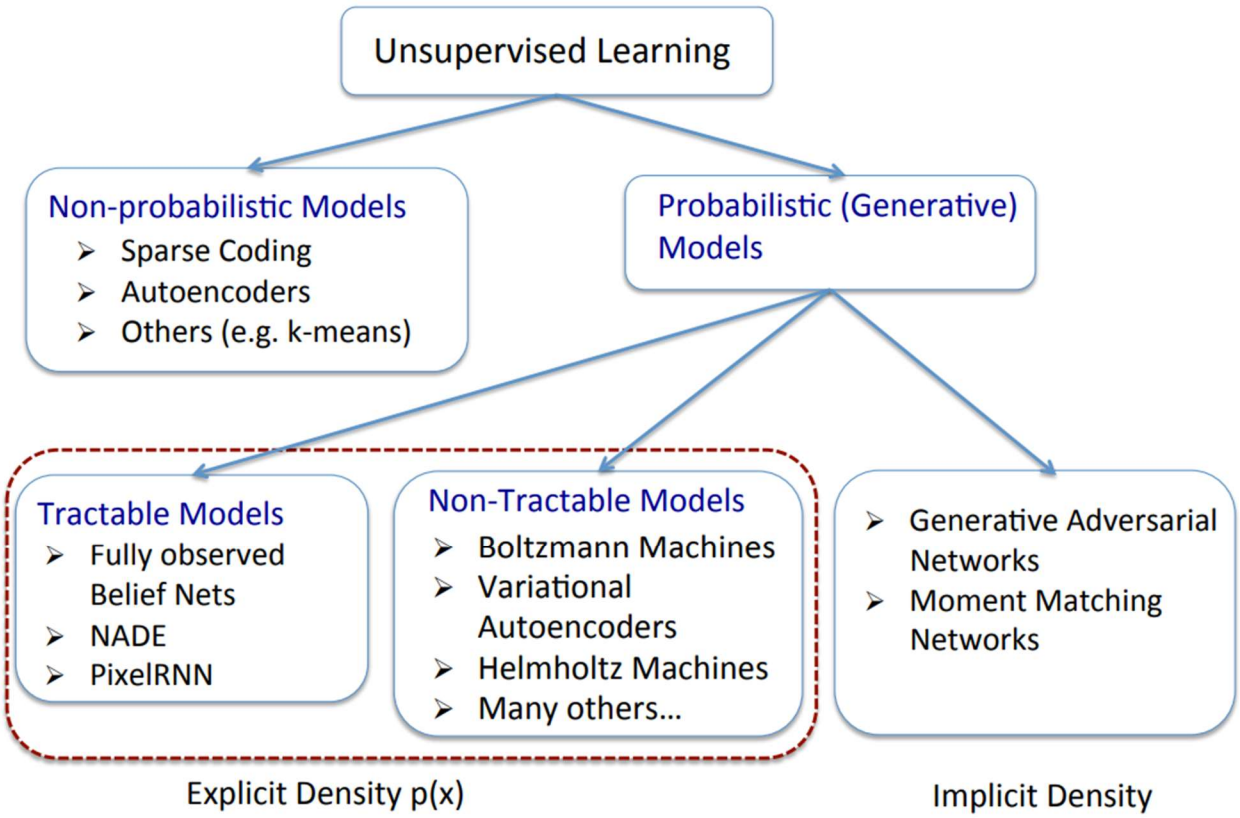


WEEK-11

Outcome: Students are able to build a simple auto encoded using keras library.

Pre Lab:

1) List out some unsupervised deep learning models.



Other: LSTM, RBF, SOM, DBF(both).

2) What are auto encoders and list out various types of auto encoders?

A) Autoencoder is an unsupervised artificial neural network that learns how to efficiently compress and encode data then learns how to reconstruct the data back from the reduced encoded representation to a representation that is as close to the original input as possible. It contains Encoder, Decoder.

Types of Auto Encoders:

- Vanilla autoencoder.
- Multilayer autoencoder.
- Convolutional autoencoder.
- Regularized autoencoder.
  Other are: Denoising Autoencoder, Sparse Autoencoder, Deep Autoencoder, Contractive Autoencoder, Variational Autoencoder

3) List out the applications of auto encoders.

A) Applications of Auto Encoders:

1. Dimensionality Reduction
2. Image Compression
3. Image Denoising
4. Feature Extraction
5. Image generation
6. Sequence to sequence prediction
7. Recommendation system

In Lab:

EXP 11:

Build a simple auto encoder on MNIST data using keras library and help them in building a simple Auto Encoder. (Note: Encode the image into from 784 to 32 pixels).

Program

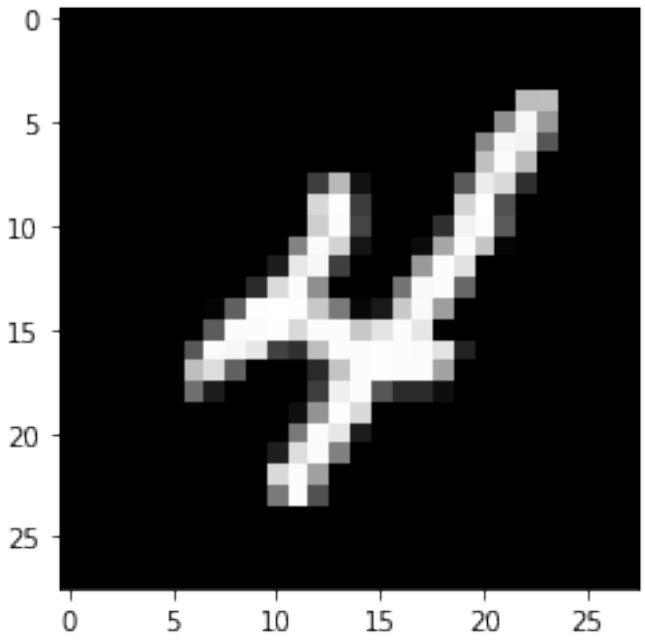
Autoencoders

October 18, 2022

[ ]:
```
#Auto Encoders

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import cv2
import tensorflow as tf
```

[ ]:
```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
plt.imshow(x_train[9], cmap='gray')
```

[ ]:
```
<matplotlib.image.AxesImage at 0x21114be15e0>
```



[ ]:
```
x_train[9].shape
```

[ ]:
```
(28, 28)
```

[ ]:

```
x_train[9]
```

[ ]:

```
array([[  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,   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,   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,   0,   0, 189, 190,   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, 143, 247, 153,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0, 136, 247, 242,  86,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0, 192, 252, 187,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  62,
        185,  18,   0,   0,   0,   0,  89, 236, 217,  47,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 216,
        253,  60,   0,   0,   0,   0, 212, 255,  81,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 206,
        252,  68,   0,   0,   0,  48, 242, 253,  89,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 131, 251,
        212,  21,   0,   0,  11, 167, 252, 197,   5,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  29, 232, 247,
         63,   0,   0,   0, 153, 252, 226,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  45, 219, 252, 143,
          0,   0,   0, 116, 249, 252, 103,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   4,  96, 253, 255, 253, 200,
        122,   7,  25, 201, 250, 158,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
```

```
[  0,   0,   0,   0,   0,   0,   0,  92, 252, 252, 253, 217, 252,
 252, 200, 227, 252, 231,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  87, 251, 247, 231,  65,  48, 189,
 252, 252, 253, 252, 251, 227,  35,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0, 190, 221,  98,   0,   0,   0,  42,
 196, 252, 253, 252, 252, 162,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0, 111,  29,   0,   0,   0,   0,  62,
 239, 252,  86,  42,  42,  14,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  15, 148,
 253, 218,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 121, 252,
 231,  28,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  31, 221, 251,
 129,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 218, 252, 160,
   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, 122, 252,  82,
   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,   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,   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]], dtype=uint8)
```

[ ]:
```
x_train = x_train/255
x_test = x_test/255
```

[ ]:
```
encoder_input = tf.keras.layers.Input(shape=(28,28,1),name='img')
x = tf.keras.layers.Flatten()(encoder_input)
encoder_output = tf.keras.layers.Dense(32, activation='relu')(x)
```

```
encoder = tf.keras.Model(encoder_input, encoder_output, name='encoder')

decoder_input = tf.keras.layers.Dense(784, activation='relu')(encoder_output)
decoder_output = tf.keras.layers.Reshape((28,28,1))(decoder_input)

opt = tf.keras.optimizers.Adam(lr=0.001)

autoencoder = tf.keras.Model(encoder_input, decoder_output, name='autoencoder')
autoencoder.summary()
```

```
Model: "autoencoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 32)                25120

 dense_1 (Dense)             (None, 784)               25872

 reshape (Reshape)           (None, 28, 28, 1)         0

=================================================================
Total params: 50,992
Trainable params: 50,992
Non-trainable params: 0
_________________________________________________________________
c:\Users\Yash\AppData\Local\Programs\Python\Python39\lib\site-
packages\keras\optimizers\optimizer_v2\adam.py:114: UserWarning: The `lr`
argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

[ ]:
```
autoencoder.compile(optimizer=opt, loss='mse')
autoencoder.fit(x_train, x_train, epochs=3, batch_size=32, validation_split=0.1)
```

```
Epoch 1/3
1688/1688 [==============================] - 7s 3ms/step - loss: 0.0226 -
val_loss: 0.0160
Epoch 2/3
1688/1688 [==============================] - 8s 4ms/step - loss: 0.0159 -
val_loss: 0.0156
Epoch 3/3
1688/1688 [==============================] - 6s 4ms/step - loss: 0.0157 -
val_loss: 0.0155
```

[ ]:
```
<keras.callbacks.History at 0x21114c8f790>
```

[ ]:
```
example = encoder.predict([x_test[9].reshape(1,28,28,1)])[0]
print(example)
```

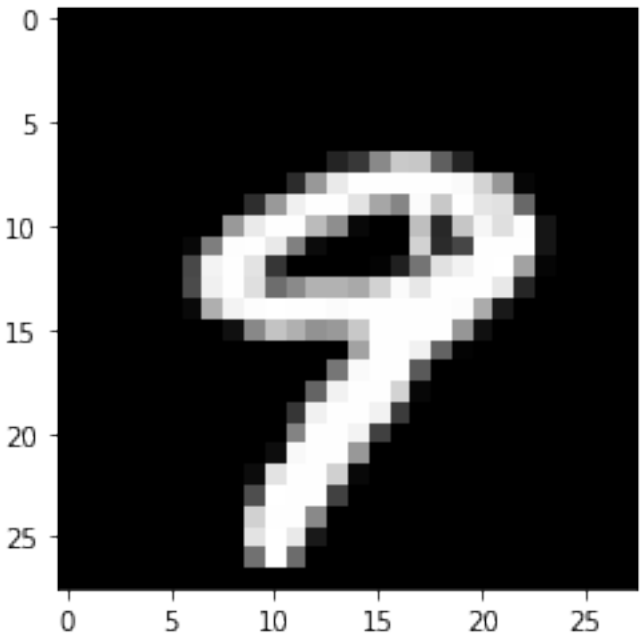
```
1/1 [==============================] - 0s 154ms/step
[2.5445952  4.459136   2.7209787  1.5872056  0.76784146 2.3575752
 1.3548634  2.8487937  0.72864115 3.2948034  1.6244698  2.0749702
 1.508508   1.9331096  1.8187125  1.7990305  3.2019749  3.2552347
 3.45871    4.0377545  3.590996   1.3729562  2.0083427  2.7692058
 5.4671154  1.6948321  1.9330242  2.6233985  5.5681143  2.3114822
 3.5919688  3.5784943 ]
1/1 [==============================] - 0s 154ms/step
[2.5445952  4.459136   2.7209787  1.5872056  0.76784146 2.3575752
 1.3548634  2.8487937  0.72864115 3.2948034  1.6244698  2.0749702
 1.508508   1.9331096  1.8187125  1.7990305  3.2019749  3.2552347
 3.45871    4.0377545  3.590996   1.3729562  2.0083427  2.7692058
 5.4671154  1.6948321  1.9330242  2.6233985  5.5681143  2.3114822
 3.5919688  3.5784943 ]
```

[ ]:
```
example.shape
```

[ ]:
```
(32,)
```

[ ]:
```
plt.imshow(example.reshape(8,4), cmap='gray')
plt.imshow(x_test[9], cmap='gray')
```

[ ]:
```
<matplotlib.image.AxesImage at 0x211127e9eb0>
```



[ ]:
```
ae_out = autoencoder.predict([x_test[9].reshape(1,28,28,1)])[0]
plt.imshow(ae_out.reshape(28,28), cmap='gray')
```

```
1/1 [==============================] - 0s 71ms/step
```

[ ]:
```
<matplotlib.image.AxesImage at 0x21112b476d0>
```