DHAANISH AHMED COLLEGE OF ENGINEERING

Dhaanish Nagar, Padappai, Chennai – 601301

Approved By AICTE, New Delhi,

Affiliated to Anna University, Chennai.

www.dhaanish.in

Department of Artificial Intelligence & Data Science

Lab Manual

AD3511 – Deep Learning Laboratory

Year/Sem : III/V

DHAANISH AHMED COLLEGE OF ENGINEERING

Vision

To establish a world-class institution that is recognized as a “Centre of Excellence” offering education and research in engineering, technology and management with a blend of social and moral values to serve the community with a futuristic perspective.

Mission

To produce eminent engineers and managers with academic excellence in their chosen fields, which would be able to take up the challenges in the modern era and fulfill the expectations of the organization they join, with moral values and social ethics.

Department of Artificial Intelligence and Data Science

Vision

To impart quality Education, Industry Collaboration, promote Research and produce Graduate Industry-ready Engineers in the field of Artificial Intelligence and Data Science to serve the society.

Mission

- To provide a conducive learning environment for quality education in the field of Artificial Intelligence and Data Science.
- To promote industry-institute interaction and collaborative research activities.
- To empower the students with ethical values and social responsibilities in their profession.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

- Show proficiency in the knowledge of basic sciences, mathematics, Artificial Intelligence, data science and statistics to build systems that require management and analysis of large volume of data.
- Demonstrate technical skills to pursue pioneering research in the field of AI and Data Science and create disruptive and sustainable solutions for the welfare of ecosystems.
- Exhibit effective communication skills, team work and lead their profession with ethics.

Program Specific Outcome (PSO)

PSO1: Evolve AI based efficient domain specific processes for Effective decision making in several domains such as business and governance domains.

PSO2: Create, select and apply the theoretical knowledge of AI and Analytics along with practical industrial tools and techniques to manage and solve societal problems.

INDEX

S. No.	Date	Name of the Experiment	Page No.	Marks	Sign
1		XOR Problem Using DNN			
2		Character recognition using CNN			
3		Face Recognition Using CNN			
4		Language Modeling using RNN			
5		Sentimental Analysis using LSTM			
6		Part of Speech tagging			
7		Machine Translation			
8		Image Augmentation			
9		Traffic Prediction – Mini project			

XOR Problem Using DNN

Ex. No. : 1

Date:

Aim:

To write a python program for the implementation of XOR operation using the DNN algorithm

Algorithm:

Step1: Import the required Python libraries
Step2: Define Activation Function: Sigmoid Function
Step3: Initialize neural network parameters (weights, bias)
Step4: Define model hyperparameters (number of iterations, learning rate)
Step5: Forward Propagation
Step6: Backward Propagation
Step7: Update weight and bias parameters
Step8: Train the learning model
Step9: Plot Loss value vs Epoch
Step10: Test the model performance

Program:

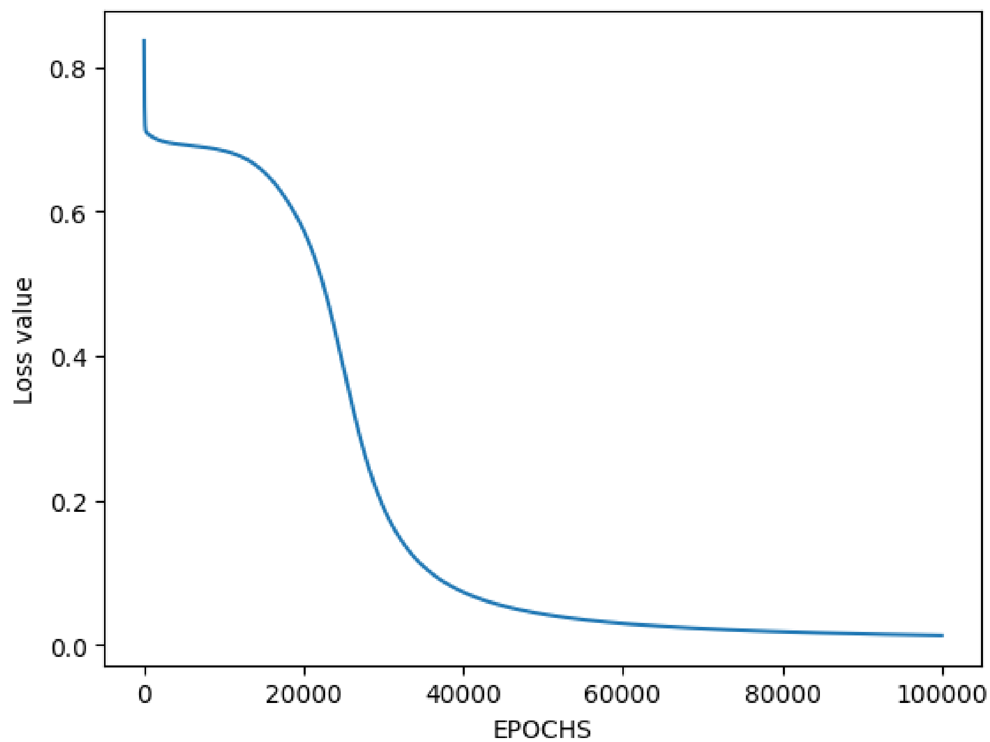
```
# import Python Libraries

import numpy as np
from matplotlib import pyplot as plt

# Sigmoid Function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Initialization of the neural network parameters
# Initialized all the weights in the range of between 0 and 1
# Bias values are initialized to 0

def initializeParameters(inputFeatures, neuronsInHiddenLayers, outputFeatures):
    W1 = np.random.randn(neuronsInHiddenLayers, inputFeatures)
    W2 = np.random.randn(outputFeatures, neuronsInHiddenLayers)
```

```
    b1 = np.zeros((neuronsInHiddenLayers, 1))
    b2 = np.zeros((outputFeatures, 1))

    parameters = {"W1" : W1, "b1": b1,
                  "W2" : W2, "b2": b2}
    return parameters

# Forward Propagation

def forwardPropagation(X, Y, parameters):
    m = X.shape[1]
    W1 = parameters["W1"]
    W2 = parameters["W2"]
    b1 = parameters["b1"]
    b2 = parameters["b2"]

    Z1 = np.dot(W1, X) + b1
    A1 = sigmoid(Z1)
    Z2 = np.dot(W2, A1) + b2
    A2 = sigmoid(Z2)

    cache = (Z1, A1, W1, b1, Z2, A2, W2, b2)
    logprobs = np.multiply(np.log(A2), Y) + np.multiply(np.log(1 - A2), (1 - Y))
    cost = -np.sum(logprobs) / m
    return cost, cache, A2

# Backward Propagation

def backwardPropagation(X, Y, cache):
    m = X.shape[1]
    (Z1, A1, W1, b1, Z2, A2, W2, b2) = cache

    dZ2 = A2 - Y
    dW2 = np.dot(dZ2, A1.T) / m
    db2 = np.sum(dZ2, axis = 1, keepdims = True)

    dA1 = np.dot(W2.T, dZ2)
    dZ1 = np.multiply(dA1, A1 * (1- A1))
    dW1 = np.dot(dZ1, X.T) / m
    db1 = np.sum(dZ1, axis = 1, keepdims = True) / m
```

```
    gradients = {"dZ2": dZ2, "dW2": dW2, "db2": db2,
                 "dZ1": dZ1, "dW1": dW1, "db1": db1}
    return gradients

# Updating the weights based on the negative gradients
def updateParameters(parameters, gradients, learningRate):
    parameters["W1"] = parameters["W1"] - learningRate * gradients["dW1"]
    parameters["W2"] = parameters["W2"] –
          learningRate * gradients["dW2"]
    parameters["b1"] = parameters["b1"] –
          learningRate * gradients["db1"]
    parameters["b2"] = parameters["b2"] - learningRate * gradients["db2"]
    return parameters

# Model to learn the XOR truth table

X = np.array([[0, 0, 1, 1], [0, 1, 0, 1]]) # XOR input
Y = np.array([[0, 1, 1, 0]]) # XOR output

# Define model parameters

neuronsInHiddenLayers = 2 # number of hidden layer neurons (2)
inputFeatures = X.shape[0] # number of input features (2)
outputFeatures = Y.shape[0] # number of output features (1)
parameters = initializeParameters(inputFeatures, neuronsInHiddenLayers, outputFeatures)
epoch = 100000
learningRate = 0.01
losses = np.zeros((epoch, 1))

for i in range(epoch):
    losses[i, 0], cache, A2 = forwardPropagation(X, Y, parameters)
    gradients = backwardPropagation(X, Y, cache)
    parameters = updateParameters(parameters, gradients, learningRate)

# Evaluating the performance

plt.figure()
plt.plot(losses)
plt.xlabel("EPOCHS")
```

```
plt.ylabel("Loss value")
plt.show()

# Testing

X = np.array([[1, 1, 0, 0], [0, 1, 0, 1]]) # XOR input
cost, _, A2 = forwardPropagation(X, Y, parameters)
prediction = (A2 > 0.5) * 1.0
# print(A2)
print(prediction)
```

Output:




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

Result:

Thus the XOR has been implemented using DNN algorithm successfully.

Character recognition using CNN

Ex. No. : 2

Date:

Aim:

To write a python program for the implementation of XOR operation using the DNN algorithm

Algorithm:

Step1: Import the required Python libraries
Step2: Import the dataset
Step3: Normalize the data
Step4: Split the training and testing data
Step5: Forward Propagation
Step6: Backward Propagation
Step7: Update weight and bias parameters
Step8: Train the learning model
Step9: Plot Loss value vs Epoch
Step10: Test the model performance

Program:

```
# importing packages

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K


#splitting training and testing data

(x_train, y_train), (x_test, y_test) = mnist.load_data()


# Train the model
```

```
num_of_trainImgs = x_train.shape[0] #60000 here
num_of_testImgs = x_test.shape[0] #10000 here
img_width = 28
img_height = 28

x_train = x_train.reshape(x_train.shape[0], img_height, img_width, 1)
x_test = x_test.reshape(x_test.shape[0], img_height, img_width, 1)
input_shape = (img_height, img_width, 1)

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


# categorizing the class

num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)


# Training the model

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
            activation='relu',
            input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
          optimizer=keras.optimizers.Adadelta(),
          metrics=['accuracy'])
```

```
# Testing the model

model.fit(x_train, y_train,
          batch_size=128,
          epochs=12,
          verbose=1,
          validation_data=(x_test, y_test))
```

Output:

```
Epoch 1/12
469/469 [==============================] - 184s 391ms/step - loss: 2.2798 - accuracy: 0.1673 - val_loss: 2.2508 - val_accuracy: 0.3252
Epoch 2/12
469/469 [==============================] - 186s 396ms/step - loss: 2.2330 - accuracy: 0.2693 - val_loss: 2.1952 - val_accuracy: 0.4243
Epoch 3/12
469/469 [==============================] - 186s 396ms/step - loss: 2.1753 - accuracy: 0.3505 - val_loss: 2.1203 - val_accuracy: 0.4765
Epoch 4/12
469/469 [==============================] - 186s 396ms/step - loss: 2.0949 - accuracy: 0.4137 - val_loss: 2.0156 - val_accuracy: 0.5637
Epoch 5/12
469/469 [==============================] - 188s 401ms/step - loss: 1.9832 - accuracy: 0.4745 - val_loss: 1.8736 - val_accuracy: 0.6202
Epoch 6/12
469/469 [==============================] - 181s 384ms/step - loss: 1.8422 - accuracy: 0.5197 - val_loss: 1.6953 - val_accuracy: 0.6680
Epoch 7/12
469/469 [==============================] - 190s 406ms/step - loss: 1.6728 - accuracy: 0.5675 - val_loss: 1.4938 - val_accuracy: 0.7133
Epoch 8/12
469/469 [==============================] - 188s 401ms/step - loss: 1.4991 - accuracy: 0.6019 - val_loss: 1.2938 - val_accuracy: 0.7504
Epoch 9/12
469/469 [==============================] - 182s 387ms/step - loss: 1.3442 - accuracy: 0.6309 - val_loss: 1.1194 - val_accuracy: 0.7797
Epoch 10/12
469/469 [==============================] - 186s 396ms/step - loss: 1.2124 - accuracy: 0.6572 - val_loss: 0.9786 - val_accuracy: 0.7965
Epoch 11/12
469/469 [==============================] - 185s 394ms/step - loss: 1.1081 - accuracy: 0.6791 - val_loss: 0.8696 - val_accuracy: 0.8111
Epoch 12/12
469/469 [==============================] - 179s 382ms/step - loss: 1.0297 - accuracy: 0.6955 - val_loss: 0.7853 - val_accuracy: 0.8240
<keras.callbacks.History at 0x7e0fa36188e0>
```

```
# Evaluation of the Model

score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Output:

```
Test loss: 0.7852669358253479
Test accuracy: 0.8240000009536743
```

Result:

Thus the program for character recognition using CNN has been successfully executed and the result has been verified.

Face Recognition Using CNN

Ex. No. : 3

Date:

Aim:

To write a python program for the implementation of face recognition using the CNN algorithm

Algorithm:

Step1: Import the required Python libraries
Step2: Define Activation Function: Sigmoid Function
Step3: Initialize neural network parameters (weights, bias)
Step4: Create the model
Step5: Train the learning model
Step6: Test the model
Step7: Evaluate the model

Program:

```
#load the LFW dataset

import numpy as np
import pandas as pd
from sklearn.datasets import fetch_lfw_people

faces = fetch_lfw_people(min_faces_per_person=100, resize=1.0, slice_=(slice(60, 188), slice(60, 188)), color=True)
class_count = len(faces.target_names)

print(faces.target_names)
print(faces.images.shape)
```

```
['Colin Powell' 'Donald Rumsfeld' 'George W Bush' 'Gerhard Schroeder'
 'Tony Blair']
(1140, 128, 128, 3)
```

```
#label the faces
%matplotlib inline
import matplotlib.pyplot as plt
```

```
import seaborn as sns
sns.set()

fig, ax = plt.subplots(3, 6, figsize=(18, 10))

for i, axi in enumerate(ax.flat):
   axi.imshow(faces.images[i] / 255) # Scale pixel values so Matplotlib doesn't clip everything
above 1.0
   axi.set(xticks=[], yticks=[], xlabel=faces.target_names[faces.target[i]])
```


```
#Normalizing the dataset
mask = np.zeros(faces.target.shape, dtype=np.bool)

for target in np.unique(faces.target):
   mask[np.where(faces.target == target)[0][:100]] = 1

x_faces = faces.data[mask]
y_faces = faces.target[mask]
x_faces = np.reshape(x_faces, (x_faces.shape[0], faces.images.shape[1], faces.images.shape[2],
faces.images.shape[3]))
x_faces.shape


#Splitting training and testing dataset
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications.resnet50 import preprocess_input
from sklearn.model_selection import train_test_split

face_images = preprocess_input(np.array(x_faces))
face_labels = to_categorical(y_faces)
```

x_train, x_test, y_train, y_test = train_test_split(face_images, face_labels, train_size=0.8, stratify=face_labels, random_state=0)

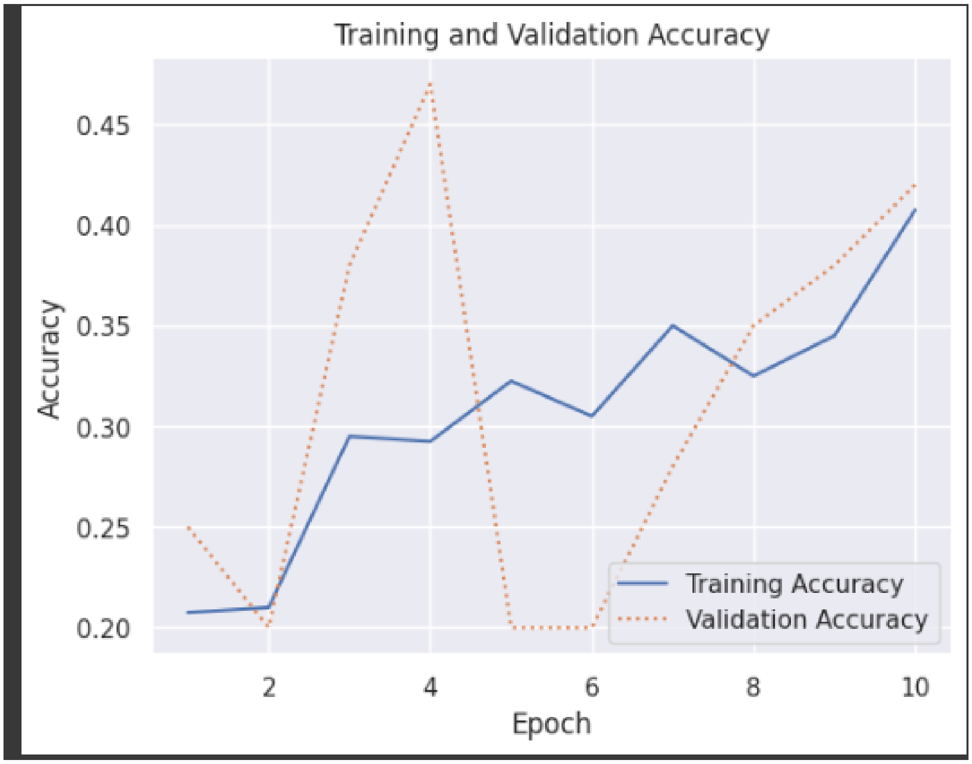
```
# Creating the model
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(face_images.shape[1:])))
model.add(MaxPooling2D(2, 2))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(class_count, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=20, batch_size=10)
```

```
Epoch 1/20
40/40 [==============================] - 26s 547ms/step - loss: 22.1749 - accuracy: 0.1975 - val_loss: 1.6095 - val_accuracy: 0.2000
Epoch 2/20
40/40 [==============================] - 24s 606ms/step - loss: 1.6103 - accuracy: 0.1575 - val_loss: 1.6095 - val_accuracy: 0.2000
Epoch 3/20
40/40 [==============================] - 29s 728ms/step - loss: 1.6100 - accuracy: 0.1600 - val_loss: 1.6095 - val_accuracy: 0.2000
Epoch 4/20
40/40 [==============================] - 22s 555ms/step - loss: 1.6096 - accuracy: 0.2000 - val_loss: 1.6094 - val_accuracy: 0.2000
Epoch 5/20
40/40 [==============================] - 25s 622ms/step - loss: 1.6098 - accuracy: 0.1750 - val_loss: 1.6094 - val_accuracy: 0.2000
Epoch 6/20
40/40 [==============================] - 23s 553ms/step - loss: 1.6097 - accuracy: 0.1850 - val_loss: 1.6094 - val_accuracy: 0.2000
Epoch 7/20
40/40 [==============================] - 24s 595ms/step - loss: 1.6097 - accuracy: 0.1725 - val_loss: 1.6094 - val_accuracy: 0.2000
Epoch 8/20
40/40 [==============================] - 23s 576ms/step - loss: 1.6097 - accuracy: 0.2000 - val_loss: 1.6094 - val_accuracy: 0.2000
Epoch 9/20
40/40 [==============================] - 24s 602ms/step - loss: 1.6096 - accuracy: 0.1500 - val_loss: 1.6094 - val_accuracy: 0.2000
Epoch 10/20
40/40 [==============================] - 24s 597ms/step - loss: 1.6097 - accuracy: 0.1725 - val_loss: 1.6094 - val_accuracy: 0.2000
Epoch 11/20
40/40 [==============================] - 27s 674ms/step - loss: 1.6096 - accuracy: 0.1550 - val_loss: 1.6094 - val_accuracy: 0.2000
Epoch 12/20
40/40 [==============================] - 22s 554ms/step - loss: 1.6098 - accuracy: 0.1750 - val_loss: 1.6094 - val_accuracy: 0.2000
Epoch 13/20
40/40 [==============================] - 23s 564ms/step - loss: 1.6098 - accuracy: 0.1800 - val_loss: 1.6094 - val_accuracy: 0.2000
Epoch 14/20
40/40 [==============================] - 23s 582ms/step - loss: 1.6096 - accuracy: 0.1950 - val_loss: 1.6094 - val_accuracy: 0.2000
Epoch 15/20
40/40 [==============================] - 30s 749ms/step - loss: 1.6096 - accuracy: 0.1650 - val_loss: 1.6094 - val_accuracy: 0.2000
```

```
#Testing
from tensorflow.keras.applications import ResNet50
```

```
base_model = ResNet50(weights='imagenet', include_top=False)
base_model.trainable = False
#Activting the hidden layers
from keras.models import Sequential
from keras.layers import Flatten, Dense, Resizing

model = Sequential()
model.add(Resizing(224, 224))
model.add(base_model)
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(class_count, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
Epoch 1/10
40/40 [==============================] - 180s 4s/step - loss: 13.7116 - accuracy: 0.2075 - val_loss: 2.6140 - val_accuracy: 0.2500
Epoch 2/10
40/40 [==============================] - 169s 4s/step - loss: 2.2675 - accuracy: 0.2100 - val_loss: 2.0198 - val_accuracy: 0.2000
Epoch 3/10
40/40 [==============================] - 170s 4s/step - loss: 1.7563 - accuracy: 0.2950 - val_loss: 2.0195 - val_accuracy: 0.3800
Epoch 4/10
40/40 [==============================] - 170s 4s/step - loss: 1.7580 - accuracy: 0.2925 - val_loss: 1.3700 - val_accuracy: 0.4700
Epoch 5/10
40/40 [==============================] - 153s 4s/step - loss: 1.8236 - accuracy: 0.3225 - val_loss: 2.0655 - val_accuracy: 0.2000
Epoch 6/10
40/40 [==============================] - 157s 4s/step - loss: 1.6553 - accuracy: 0.3050 - val_loss: 1.7280 - val_accuracy: 0.2000
Epoch 7/10
40/40 [==============================] - 155s 4s/step - loss: 1.4680 - accuracy: 0.3500 - val_loss: 1.5888 - val_accuracy: 0.2800
Epoch 8/10
40/40 [==============================] - 165s 4s/step - loss: 1.6130 - accuracy: 0.3250 - val_loss: 1.7193 - val_accuracy: 0.3500
Epoch 9/10
40/40 [==============================] - 156s 4s/step - loss: 1.5905 - accuracy: 0.3450 - val_loss: 1.4193 - val_accuracy: 0.3800
Epoch 10/10
40/40 [==============================] - 159s 4s/step - loss: 1.4059 - accuracy: 0.4075 - val_loss: 1.3703 - val_accuracy: 0.4200
[]
```



Result:

Thus the program for Facial recognition using CNN has been successfully executed and the result has been verified.

Language Modeling using RNN

Ex. No. : 4

Date:

Aim:

To write a python program for the implementation language modeling using the RNN algorithm.

Algorithm:

Step1: Import the required Python libraries
Step2: Import the dataset
Step3: Perform the data pre-processing
Step4: Remove the stopword from the corpus
Step5: Perform lemmatization
Step6: Build the model
Step7: Train the learning model
Step8: Test and evaluate the model

Program:

```
import numpy as np
class CharRNN(object):
    def __init__(self, corpus, hidden_size=128, seq_len=25, lr=1e-3, epochs=100):
        self.corpus = corpus
        self.hidden_size = hidden_size
        self.seq_len = seq_len
        self.lr = lr
        self.epochs = epochs
        chars = list(set(corpus))
        self.data_size, self.input_size, self.output_size = len(corpus), len(chars), len(chars)
        self.char_to_num = {c:i for i,c in enumerate(chars)}
        self.num_to_char = {i:c for i,c in enumerate(chars)}
        self.h = np.zeros((self.hidden_size , 1))
        self.W_xh = np.random.randn(self.hidden_size, self.input_size) * 0.01
        self.W_hh = np.random.randn(self.hidden_size, self.hidden_size) * 0.01
        self.W_hy = np.random.randn(self.output_size, self.hidden_size) * 0.01
```

```
        self.b_h = np.zeros((self.hidden_size, 1))
        self.b_y = np.zeros((self.output_size, 1))
def sample(self, seed, n):
    seq = []
    h = self.h
    x = np.zeros((self.input_size, 1))
    x[self.char_to_num[seed]] = 1
    for t in range(n):
        # forward pass
        h = np.tanh(np.dot(self.W_xh, x) + np.dot(self.W_hh, h) + self.b_h)
        y = np.dot(self.W_hy, h) + self.b_y
        p = np.exp(y) / np.sum(np.exp(y))
        # sample from the distribution
        seq_t = np.random.choice(range(self.input_size), p=p.ravel())
        x = np.zeros((self.input_size, 1))
        x[seq_t] = 1
        seq.append(seq_t)
    return ''.join(self.num_to_char[num] for num in seq)


def fit(self):
    smoothed_loss = -np.log(1. / self.input_size) * self.seq_len
    for e in range(self.epochs):
        for p in range(np.floor(self.data_size / self.seq_len).astype(np.int64)):
            # get a slice of data with length at most seq_len
            x = [self.char_to_num[c] for c in self.corpus[p * self.seq_len:(p + 1) * self.seq_len]]
            y = [self.char_to_num[c] for c in self.corpus[p * self.seq_len + 1:(p + 1) * self.seq_len +
1]]
            # compute loss and gradients
            loss, dW_xh, dW_hh, dW_hy, db_h, db_y = self.__loss(x, y)
            smoothed_loss = smoothed_loss * 0.99 + loss * 0.01
            if p % 1000 == 0: print('Epoch {0}, Iter {1}: Loss: {2:.4f}'.format(e+1, p,
smoothed_loss))
            # SGD update
            for param, dparam in zip([self.W_xh, self.W_hh, self.W_hy, self.b_h, self.b_y], [dW_xh,
dW_hh, dW_hy, db_h, db_y]):
                param += -self.lr * dparam


def __loss(self, X, Y):
    xs, hs, ys, ps = {}, {}, {}, {}
```

```
hs[-1] = np.copy(self.h)
# forward pass
loss = 0
for t in range(len(X)):
    xs[t] = np.zeros((self.input_size, 1))
    xs[t][X[t]] = 1
    hs[t] = np.tanh(np.dot(self.W_xh, xs[t]) + np.dot(self.W_hh, hs[t-1]) + self.b_h)
    ys[t] = np.dot(self.W_hy, hs[t]) + self.b_y
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t]))
    loss += -np.log(ps[t][Y[t], 0])
# backward pass
dW_xh = np.zeros_like(self.W_xh)
dW_hh = np.zeros_like(self.W_hh)
dW_hy = np.zeros_like(self.W_hy)
db_h = np.zeros_like(self.b_h)
db_y = np.zeros_like(self.b_y)
delta = np.zeros_like(hs[0])
for t in reversed(range(len(X))):
    dy = np.copy(ps[t])
    # backprop into y
    dy[Y[t]] -= 1
    dW_hy += np.dot(dy, hs[t].T)
    db_y += dy
    # backprop into h
    dh = np.dot(self.W_hy.T, dy) + delta
    dh_raw = (1 - hs[t] ** 2) * dh
    db_h += dh_raw
    dW_hh += np.dot(dh_raw, hs[t-1].T)
    dW_xh += np.dot(dh_raw, xs[t].T)
    # update delta
    delta = np.dot(self.W_hh.T, dh_raw)
for dparam in [dW_xh, dW_hh, dW_hy, db_h, db_y]:
    # gradient clipping to prevent exploding gradient
    np.clip(dparam, -5, 5, out=dparam)
# update last hidden state for sampling
self.h = hs[len(X) - 1]
return loss, dW_xh, dW_hh, dW_hy, db_h, db_y
```

Create a file input.txt in notepad

```
if __name__ == '__main__':
    with open('input.txt', 'r') as f:
        data = f.read()



char_rnn = CharRNN(data, epochs=10)
    char_rnn.fit()
    print(char_rnn.sample(data[0], 100))
```

Output:

```
Epoch 1/10
40/40 [==============================] - 180s 4s/step - loss: 13.7116 - accuracy: 0.2075 - val_loss: 2.6140 - val_accuracy: 0.2500
Epoch 2/10
40/40 [==============================] - 169s 4s/step - loss: 2.2675 - accuracy: 0.2100 - val_loss: 2.0198 - val_accuracy: 0.2000
Epoch 3/10
40/40 [==============================] - 170s 4s/step - loss: 1.7563 - accuracy: 0.2950 - val_loss: 2.0195 - val_accuracy: 0.3800
Epoch 4/10
40/40 [==============================] - 170s 4s/step - loss: 1.7580 - accuracy: 0.2925 - val_loss: 1.3700 - val_accuracy: 0.4700
Epoch 5/10
40/40 [==============================] - 153s 4s/step - loss: 1.8236 - accuracy: 0.3225 - val_loss: 2.0655 - val_accuracy: 0.2000
Epoch 6/10
40/40 [==============================] - 157s 4s/step - loss: 1.6553 - accuracy: 0.3050 - val_loss: 1.7280 - val_accuracy: 0.2000
Epoch 7/10
40/40 [==============================] - 155s 4s/step - loss: 1.4680 - accuracy: 0.3500 - val_loss: 1.5888 - val_accuracy: 0.2800
Epoch 8/10
40/40 [==============================] - 165s 4s/step - loss: 1.6130 - accuracy: 0.3250 - val_loss: 1.7193 - val_accuracy: 0.3500
Epoch 9/10
40/40 [==============================] - 156s 4s/step - loss: 1.5905 - accuracy: 0.3450 - val_loss: 1.4193 - val_accuracy: 0.3800
Epoch 10/10
40/40 [==============================] - 159s 4s/step - loss: 1.4059 - accuracy: 0.4075 - val_loss: 1.3703 - val_accuracy: 0.4200
```

Result:

Thus the language modeling has been done on successfully using the RNN deep learning model.

Sentimental Analysis using LSTM

Ex. No. : 5

Date:

Aim:

To write a python program for the implementation Sentimental analysis using the LSTM algorithm

Algorithm:

Step1: Import the required Python libraries
Step2: Import the dataset
Step3: Perform the data pre-processing
Step4: Remove the stopword from the corpus
Step5: Perform lemmatization
Step6: Build the model
Step7: Train the learning model
Step8: Test and evaluate the model

Importing libraries

```
import re
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
import keras
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
import math
import nltk
```

Load the data:

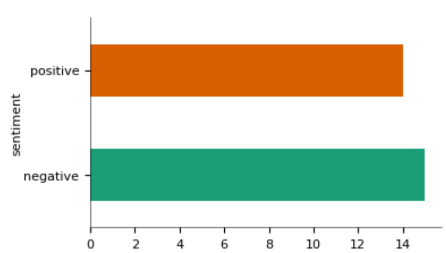
```
data = pd.read_csv('data.csv')
data
```

Output:

[ ]

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
5	Probably my all-time favorite movie, a story o...	positive
6	I sure would like to see a resurrection of a u...	positive
7	This show was an amazing, fresh & innovative i...	negative
8	Encouraged by the positive comments about this...	negative
9	If you like original gut wrenching laughter yo...	positive
10	Phil the Alien is one of those quirky films wh...	negative
11	I saw this movie when I was about 12 when it c...	negative
12	So im not a big fan of Boll's work but then ag...	negative
13	The cast played Shakespeare. Shakes...	negative
14	This a fantastic movie of three prisoners who ...	positive
15	Kind of drawn in by the erotic scenes, only to...	negative

Categorical distributions




```
def remove_tags(string):
    removelist = ""
    result = re.sub('','',string)          #remove HTML tags
    result = re.sub('https://.*','',result)   #remove URLs
    result = re.sub(r'[^w'+removelist+']', ' ',result)    #remove non-alphanumeric characters
```

```
    result = result.lower()
    return result
data['review']=data['review'].apply(lambda cw : remove_tags(cw))


nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
data['review'] = data['review'].apply(lambda x: ' '.join([word for word in x.split() if word not in
(stop_words)]))


import nltk
nltk.download('wordnet')


w_tokenizer = nltk.tokenize.WhitespaceTokenizer()
lemmatizer = nltk.stem.WordNetLemmatizer()
def lemmatize_text(text):
    st = ""
    for w in w_tokenizer.tokenize(text): st = st + lemmatizer.lemmatize(w) + " "
    return st
data['review'] = data.review.apply(lemmatize_text)
data
```

	review	sentiment
0	w w w w w w w w w w w w w w w w w w w w w w w ...	positive
1	w w w w w w w w w w w w w w w	positive
2	w w w w w w w w w w w w w w w w w w w	positive
3	w w w w w w w w w w w	negative
4	w w w w w w w w w w w w w w w w w w	positive
5	w w w w w w w	positive
6	w w w w w w w w w w w w w w w w w w w w w	positive
7	w w w w w w w w w w w w w w w w w w w	negative
8	w w w w w w w	negative
9	w w w	positive
10	w w w w w w	negative
11	w w w w w w w w w w w w w w w w	negative
12	w w w w w w w w w w w w w w w w w w w w w w w ...	negative
13	w w w w w w w w w w	negative
14	w w	positive
15	w w w w w w w w w w w w w w w w w	negative

```
s = 0.0
for i in data['review']:
    word_list = i.split()
    s = s + len(word_list)
print("Average length of each review : ",s/data.shape[0])
pos = 0
for i in range(data.shape[0]):
    if data.iloc[i]['sentiment'] == 'positive':
        pos = pos + 1
neg = data.shape[0]-pos
print("Percentage of reviews with positive sentiment is "+str(pos/data.shape[0]*100)+"%")
print("Percentage of reviews with negative sentiment is "+str(neg/data.shape[0]*100)+"%")
```

```
Average length of each review :  15.724137931034482
Percentage of reviews with positive sentiment is 48.275862068965516%
Percentage of reviews with negative sentiment is 51.724137931034484%
```

```
reviews = data['review'].values
labels = data['sentiment'].values
encoder = LabelEncoder()
encoded_labels = encoder.fit_transform(labels)

train_sentences, test_sentences, train_labels, test_labels = train_test_split(reviews,
encoded_labels, stratify = encoded_labels)

# Hyperparameters of the model
vocab_size = 3000 # choose based on statistics
oov_tok = ''
embedding_dim = 100
max_length = 200 # choose based on statistics, for example 150 to 200
padding_type='post'
trunc_type='post'
# tokenize sentences
tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(train_sentences)
word_index = tokenizer.word_index
# convert train dataset to sequence and pad sequences
train_sequences = tokenizer.texts_to_sequences(train_sentences)
# convert Test dataset to sequence and pad sequences
test_sequences = tokenizer.texts_to_sequences(test_sentences)

# model initialization
model = keras.Sequential([
   keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
   keras.layers.Bidirectional(keras.layers.LSTM(64)),
   keras.layers.Dense(24, activation='relu'),
   keras.layers.Dense(1, activation='sigmoid')
])
# compile model
model.compile(loss='binary_crossentropy',
```

```
        optimizer='adam',
        metrics=['accuracy'])
# model summary
model.summary()
```

Output:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 200, 100)          300000

 bidirectional_1 (Bidirectio  (None, 128)              84480
 nal)

 dense_2 (Dense)             (None, 24)                3096

 dense_3 (Dense)             (None, 1)                 25

=================================================================
Total params: 387,601
Trainable params: 387,601
Non-trainable params: 0
```

Result

Thus the sentimental analysis has been done using the LSTM model and the model has been evaluated.

Part of Speech tagging

Ex. No. : 6

Date:

Aim:

To write a python program for the implementation part of speech using the sequence to sequence architecture.

Algorithm:

Step1: Import the required Python libraries
Step2: Import the data
Step3: download nltk
Step4: tag the respective words from the corpus
Step5: compute probability
Step6: Build the model
Step7: Train the learning model
Step8: Test and evaluate the model

Program:

```
# Importing libraries
import nltk
import numpy as np
import pandas as pd
import random
from sklearn.model_selection import train_test_split
import pprint, time

#download the treebank corpus from nltk
nltk.download('treebank')

#download the universal tagset from nltk
nltk.download('universal_tagset')

# reading the Treebank tagged sentences
nltk_data = list(nltk.corpus.treebank.tagged_sents(tagset='universal'))

#print the first two sentences along with tags
```

```
print(nltk_data[:2])
```

```
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Unzipping taggers/universal_tagset.zip.
[[('Pierre', 'NOUN'), ('Vinken', 'NOUN'), (',', '.'), ('61', 'NUM'), ('years', 'NOUN'), ('old', 'ADJ'), (',', '.'), ('will', 'VERB'), ('join', 'VERB'), ('the',
```

```
#print each word with its respective tag for first two sentences
for sent in nltk_data[:2]:
  for tuple in sent:
    print(tuple)
```

```
('Pierre', 'NOUN')
('Vinken', 'NOUN')
(',', '.')
('61', 'NUM')
('years', 'NOUN')
('old', 'ADJ')
(',', '.')
('will', 'VERB')
('join', 'VERB')
('the', 'DET')
('board', 'NOUN')
('as', 'ADP')
('a', 'DET')
('nonexecutive', 'ADJ')
('director', 'NOUN')
('Nov.', 'NOUN')
('29', 'NUM')
('.', '.')
('Mr.', 'NOUN')
('Vinken', 'NOUN')
('is', 'VERB')
('chairman', 'NOUN')
('of', 'ADP')
('Elsevier', 'NOUN')
('N.V.', 'NOUN')
```

```
# split data into training and validation set in the ratio 80:20
train_set,test_set =train_test_split(nltk_data,train_size=0.80,test_size=0.20,random_state = 101)
```

```
# create list of train and test tagged words
train_tagged_words = [ tup for sent in train_set for tup in sent ]
test_tagged_words = [ tup for sent in test_set for tup in sent ]
```

```
print(len(train_tagged_words))
print(len(test_tagged_words))
```

```
80310
20366
```

```
# check some of the tagged words.
train_tagged_words[:5]
```

```
[('Drink', 'NOUN'),
 ('Carrier', 'NOUN'),
 ('Competes', 'VERB'),
 ('With', 'ADP'),
 ('Cartons', 'NOUN')]
```

```
#use set datatype to check how many unique tags are present in training data
tags = {tag for word,tag in train_tagged_words}
print(len(tags))
print(tags)

# check total words in vocabulary
vocab = {word for word,tag in train_tagged_words}
```

```
12
{'ADJ', 'X', '.', 'NOUN', 'PRON', 'DET', 'VERB', 'CONJ', 'ADV', 'NUM', 'ADP', 'PRT'}
```

```
# compute Emission Probability
def word_given_tag(word, tag, train_bag = train_tagged_words):
    tag_list = [pair for pair in train_bag if pair[1]==tag]
    count_tag = len(tag_list)#total number of times the passed tag occurred in train_bag
    w_given_tag_list = [pair[0] for pair in tag_list if pair[0]==word]
#now calculate the total number of times the passed word occurred as the passed tag.
    count_w_given_tag = len(w_given_tag_list)


    return (count_w_given_tag, count_tag)


# compute  Transition Probability
def t2_given_t1(t2, t1, train_bag = train_tagged_words):
```

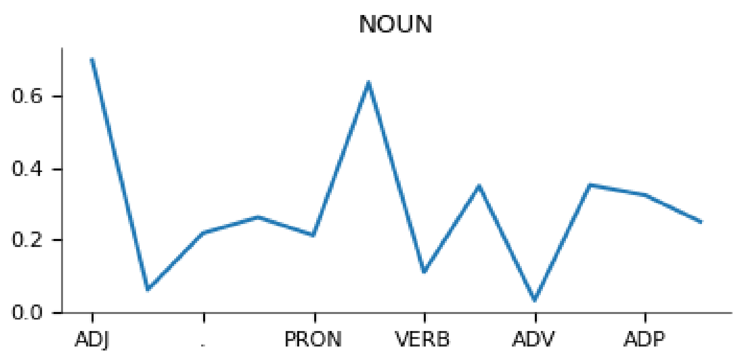
```
    tags = [pair[1] for pair in train_bag]
    count_t1 = len([t for t in tags if t==t1])
    count_t2_t1 = 0
    for index in range(len(tags)-1):
        if tags[index]==t1 and tags[index+1] == t2:
            count_t2_t1 += 1
    return (count_t2_t1, count_t1)
```

```
# creating t x t transition matrix of tags, t= no of tags
# Matrix(i, j) represents P(jth tag after the ith tag)

tags_matrix = np.zeros((len(tags), len(tags)), dtype='float32')
for i, t1 in enumerate(list(tags)):
    for j, t2 in enumerate(list(tags)):
        tags_matrix[i, j] = t2_given_t1(t2, t1)[0]/t2_given_t1(t2, t1)[1]
 print(tags_matrix)
```

```
[[6.33009672e-02 2.09708735e-02 6.60194159e-02 6.96893215e-01
  1.94174761e-04 5.24271838e-03 1.14563107e-02 1.68932043e-02
  5.24271838e-03 2.17475723e-02 8.05825219e-02 1.14563107e-02]
 [1.76821072e-02 7.57255405e-02 1.60868734e-01 6.16951771e-02
  5.41995019e-02 5.68902567e-02 2.06419379e-01 1.03786280e-02
  2.57543717e-02 3.07514891e-03 1.42225638e-01 1.85085520e-01]
 [4.61323895e-02 2.56410260e-02 9.23720598e-02 2.18538776e-01
  6.87694475e-02 1.72191828e-01 8.96899477e-02 6.00793920e-02
  5.25694676e-02 7.82104954e-02 9.29084867e-02 2.78940029e-03]
 [1.25838192e-02 2.88252197e-02 2.40094051e-01 2.62344331e-01
  4.65906132e-03 1.31063312e-02 1.49133503e-01 4.24540639e-02
  1.68945398e-02 9.14395228e-03 1.76826611e-01 4.39345129e-02]
 [7.06150308e-02 8.83826911e-02 4.19134386e-02 2.12756261e-01
  6.83371304e-03 9.56719834e-03 4.84738052e-01 5.01138950e-03
  3.69020514e-02 6.83371304e-03 2.23234631e-02 1.41230067e-02]
 [2.06410810e-01 4.51343954e-02 1.73925534e-02 6.35906279e-01
  3.30602261e-03 6.03708485e-03 4.02472317e-02 4.31220367e-04
  1.20741697e-02 2.28546783e-02 9.91806854e-03 2.87480245e-04]
 [6.63904250e-02 2.15930015e-01 3.48066315e-02 1.10589318e-01
  3.55432779e-02 1.33609578e-01 1.67955801e-01 5.43278083e-03
  8.38858187e-02 2.28360966e-02 9.23572779e-02 3.06629837e-02]
 [1.13611415e-01 9.33040585e-03 3.51262353e-02 3.49066973e-01
  6.03732169e-02 1.23490669e-01 1.50384188e-01 5.48847427e-04
  5.70801310e-02 4.06147093e-02 5.59824370e-02 4.39077942e-03]
 [1.30721495e-01 2.28859577e-02 1.39255241e-01 3.21955010e-02
  1.20248254e-02 7.13731572e-02 3.39022487e-01 6.98215654e-03
```

```
# convert the matrix to a df for better readability
#the table is same as the transition table shown in section 3 of article
tags_df = pd.DataFrame(tags_matrix, columns = list(tags), index=list(tags))
display(tags_df)
```

	ADJ	X	.	NOUN	PRON	DET	VERB	CONJ	ADV	NUM	ADP	PRT
ADJ	0.063301	0.020971	0.066019	0.696893	0.000194	0.005243	0.011456	0.016893	0.005243	0.021748	0.080583	0.011456
X	0.017682	0.075726	0.160869	0.061695	0.054200	0.056890	0.206419	0.010379	0.025754	0.003075	0.142226	0.185086
.	0.046132	0.025641	0.092372	0.218539	0.068769	0.172192	0.089690	0.060079	0.052569	0.078210	0.092908	0.002789
NOUN	0.012584	0.028825	0.240094	0.262344	0.004659	0.013106	0.149134	0.042454	0.016895	0.009144	0.176827	0.043935
PRON	0.070615	0.088383	0.041913	0.212756	0.006834	0.009567	0.484738	0.005011	0.036902	0.006834	0.022323	0.014123
DET	0.206411	0.045134	0.017393	0.635906	0.003306	0.006037	0.040247	0.000431	0.012074	0.022855	0.009918	0.000287
VERB	0.066390	0.215930	0.034807	0.110589	0.035543	0.133610	0.167956	0.005433	0.083886	0.022836	0.092357	0.030663
CONJ	0.113611	0.009330	0.035126	0.349067	0.060373	0.123491	0.150384	0.000549	0.057080	0.040615	0.055982	0.004391
ADV	0.130721	0.022886	0.139255	0.032196	0.012025	0.071373	0.339022	0.006982	0.081458	0.029868	0.119472	0.014740
NUM	0.035345	0.202428	0.119243	0.351660	0.001428	0.003570	0.020707	0.014281	0.003570	0.184220	0.037487	0.026062
ADP	0.107062	0.034548	0.038724	0.323589	0.069603	0.320931	0.008479	0.001012	0.014553	0.063275	0.016958	0.001266
PRT	0.082975	0.012133	0.045010	0.250489	0.017613	0.101370	0.401174	0.002348	0.009393	0.056751	0.019569	0.001174




```
def Viterbi(words, train_bag = train_tagged_words):

    state = []

    T = list(set([pair[1] for pair in train_bag]))



    for key, word in enumerate(words):

        #initialise list of probability column for a given observation

        p = []
```

```
        for tag in T:
            if key == 0:
                transition_p = tags_df.loc['.', tag]
            else:
                transition_p = tags_df.loc[state[-1], tag]

            # compute emission and state probabilities
            emission_p = word_given_tag(words[key], tag)[0]/word_given_tag(words[key], tag)[1]
            state_probability = emission_p * transition_p
            p.append(state_probability)

        pmax = max(p)
        # getting state for which probability is maximum
        state_max = T[p.index(pmax)]
        state.append(state_max)
    return list(zip(words, state))
# Let's test our Viterbi algorithm on a few sample sentences of test dataset
random.seed(1234)      #define a random seed to get same sentences when run multiple times

# choose random 10 numbers
rndom = [random.randint(1,len(test_set)) for x in range(10)]

# list of 10 sents on which we test the model
test_run = [test_set[i] for i in rndom]

# list of tagged words
test_run_base = [tup for sent in test_run for tup in sent]

# list of untagged words
test_tagged_words = [tup[0] for sent in test_run for tup in sent]
```

```
#Here We will only test 10 sentences to check the accuracy
#as testing the whole training set takes huge amount of time
start = time.time()
tagged_seq = Viterbi(test_tagged_words)
end = time.time()
difference = end-start
 print("Time taken in seconds: ", difference)
 # accuracy
check = [i for i, j in zip(tagged_seq, test_run_base) if i == j]
 accuracy = len(check)/len(tagged_seq)
print('Viterbi Algorithm Accuracy: ',accuracy*100)
```

Output:

```
Time taken in seconds:  62.58217978477478
Viterbi Algorithm Accuracy:  93.77990430622009
```

Result:

Thus the part of speech has been tagged using the sequence to sequence model and evaluated.

Machine Translation

Ex. No. : 7

Date:

Aim:

To write a python program to perform machine translation using encoder and decoder model.

Algorithm:

Step1: Import the required Python libraries
Step2: Import the data
Step3: download the file
Step4: import the file
Step5: create the buffer
Step6: Build the model
Step7: Train the learning model
Step8: Test and evaluate the model

Program:

```
!pip install "tensorflow-text>=2.11"
!pip install einops


import numpy as np

import typing
from typing import Any, Tuple

import einops
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

import tensorflow as tf
import tensorflow_text as tf_text


#@title
```

```
class ShapeChecker():
  def __init__(self):
    # Keep a cache of every axis-name seen
    self.shapes = {}

  def __call__(self, tensor, names, broadcast=False):
    if not tf.executing_eagerly():
      return

    parsed = einops.parse_shape(tensor, names)

    for name, new_dim in parsed.items():
      old_dim = self.shapes.get(name, None)

      if (broadcast and new_dim == 1):
        continue

      if old_dim is None:
        # If the axis name is new, add its length to the cache.
        self.shapes[name] = new_dim
        continue

      if new_dim != old_dim:
        raise ValueError(f"Shape mismatch for dimension: '{name}'\n"
                         f"    found: {new_dim}\n"
                         f"    expected: {old_dim}\n")


# Download the file
import pathlib

path_to_zip = tf.keras.utils.get_file(
    'spa-eng.zip', origin='http://storage.googleapis.com/download.tensorflow.org/data/spa-eng.zip',
    extract=True)

path_to_file = pathlib.Path(path_to_zip).parent/'spa-eng/spa.txt'


def load_data(path):
  text = path.read_text(encoding='utf-8')
```

```
  lines = text.splitlines()
  pairs = [line.split('\t') for line in lines]

  context = np.array([context for target, context in pairs])
  target = np.array([target for target, context in pairs])

  return target, context


target_raw, context_raw = load_data(path_to_file)
print(context_raw[-1])
```

```
Si quieres sonar como un hablante nativo, debes estar dispuesto a practicar diciendo la misma frase una y otra v
```

```
print(target_raw[-1])
```

```
If you want to sound like a native speaker, you must be willing to practice saying the same sentence over and over
```

```
BUFFER_SIZE = len(context_raw)
BATCH_SIZE = 64

is_train = np.random.uniform(size=(len(target_raw),)) < 0.8

train_raw = (
    tf.data.Dataset
    .from_tensor_slices((context_raw[is_train], target_raw[is_train]))
    .shuffle(BUFFER_SIZE)
    .batch(BATCH_SIZE))
val_raw = (
    tf.data.Dataset
    .from_tensor_slices((context_raw[~is_train], target_raw[~is_train]))
    .shuffle(BUFFER_SIZE)
    .batch(BATCH_SIZE))
for example_context_strings, example_target_strings in train_raw.take(1):
  print(example_context_strings[:5])
  print()
  print(example_target_strings[:5])
  break
```

```
tf.Tensor(
[b'Tom siempre grita cuando est\xc3\xa1 enfadado.'
 b'Los chicos escuchaban al maestro.'
 b'Ya me he hartado de tus respuestas sarc\xc3\xa1sticas.'
 b'Fui a su casa, pero no estaba.'
 b'Eres la persona m\xc3\xa1s importante en mi vida.'], shape=(5,), dtype=string)

tf.Tensor(
[b'Tom always shouts when he is angry.'
 b'The children were listening to the teacher.'
 b"I've had enough of your snide remarks."
 b'I went to her house, but she was not at home.'
 b"You're the most important person in my life."], shape=(5,), dtype=string)
```

```
example_text = tf.constant('¿Todavía está en casa?')
print(example_text.numpy())
print(tf_text.normalize_utf8(example_text, 'NFKD').numpy())
```

```
b'\xc2\xbfTodav\xc3\xada est\xc3\xa1 en casa?'
b'\xc2\xbfTodavi\xcc\x81a esta\xcc\x81 en casa?'
```

```
def tf_lower_and_split_punct(text):
  # Split accented characters.
  text = tf_text.normalize_utf8(text, 'NFKD')
  text = tf.strings.lower(text)
  # Keep space, a to z, and select punctuation.
  text = tf.strings.regex_replace(text, '[^ a-z.?!,¿]', '')
  # Add spaces around punctuation.
  text = tf.strings.regex_replace(text, '[.?!,¿]', r' \0 ')
  # Strip whitespace.
  text = tf.strings.strip(text)
  text = tf.strings.join(['[START]', text, '[END]'], separator=' ')
  return text
print(example_text.numpy().decode())
print(tf_lower_and_split_punct(example_text).numpy().decode())
```

Output:

```
¿Todavía está en casa?
[START] ¿ todavia esta en casa ? [END]
```

Result:

Thus the program for the machine translation has been done successfully and tested the model.

Image Augmentation

Ex. No. : 8

Date:

Aim:

To write a python program to perform image augmentation using GAN model.

Algorithm:

Step1: Import the required Python libraries
Step2: Import the data
Step3: classify the data
Step4: label the data
Step5: perform the scaling
Step6: Build the model
Step7: Train the learning model
Step8: Test and evaluate the model

Program:

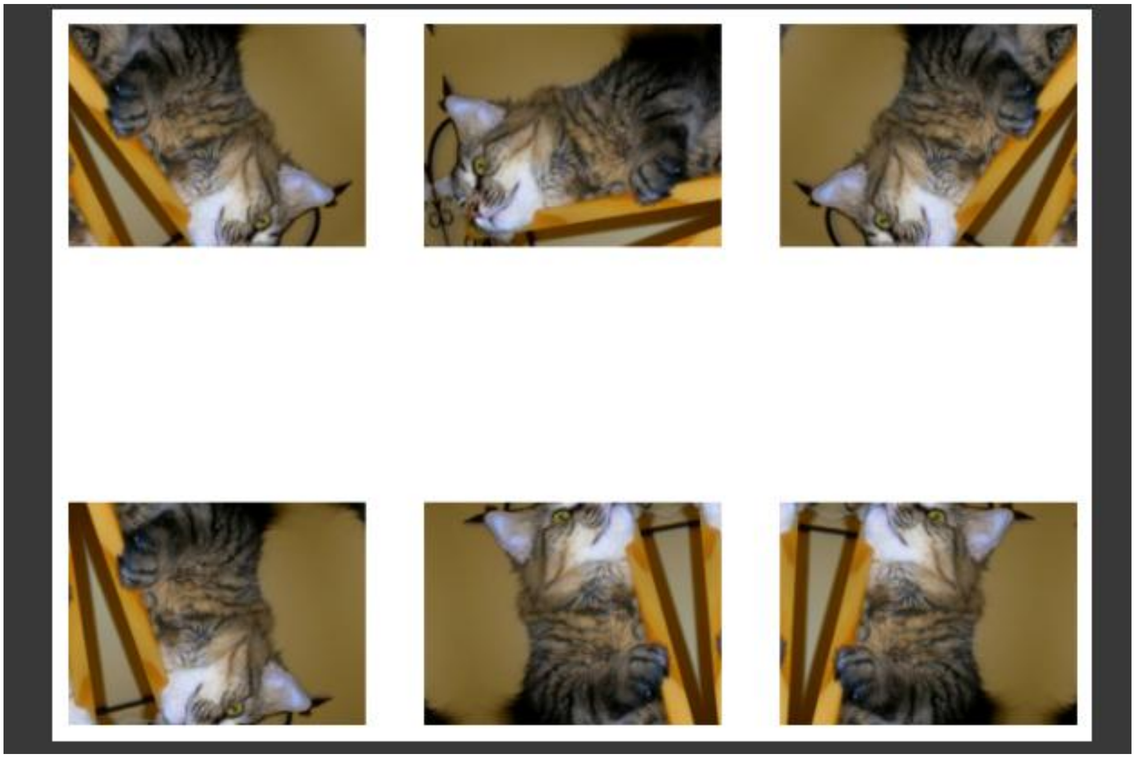
```
%%capture
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds

from keras import layers
import keras

%%capture
(train_ds, val_ds, test_ds), metadata = tfds.load(
    'cats_vs_dogs',
    split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
    with_info=True,
    as_supervised=True,


num_classes = metadata.features['label'].num_classes
print(num_classes)
```

```
2
```

```
get_label_name = metadata.features['label'].int2str
train_iter = iter(train_ds)
fig = plt.figure(figsize=(7, 8))
for x in range(4):
  image, label = next(train_iter)
  fig.add_subplot(1, 4, x+1)
  plt.imshow(image)
  plt.axis('off')
  plt.title(get_label_name(label));
```


```
IMG_SIZE = 180

resize_and_rescale = keras.Sequential([
  layers.Resizing(IMG_SIZE, IMG_SIZE),
  layers.Rescaling(1./255)
])

result = resize_and_rescale(image)
plt.axis('off')
plt.imshow(result);
```

```
data_augmentation = keras.Sequential([
  layers.RandomFlip("horizontal_and_vertical"),
  layers.RandomRotation(0.4),
])


plt.figure(figsize=(8, 7))
for i in range(6):
  augmented_image = data_augmentation(image)
  ax = plt.subplot(2, 3, i + 1)
  plt.imshow(augmented_image.numpy()/255)
  plt.axis("off")
```



```
model = keras.Sequential([
  # Add the preprocessing layers you created earlier.
  resize_and_rescale,
  data_augmentation,
  # Add the model layers
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(64, activation='relu'),
  layers.Dense(1,activation='sigmoid')
])


batch_size = 32
AUTOTUNE = tf.data.AUTOTUNE

def prepare(ds, shuffle=False, augment=False):
  # Resize and rescale all datasets.
  ds = ds.map(lambda x, y: (resize_and_rescale(x), y),
              num_parallel_calls=AUTOTUNE)
```

```
  if shuffle:
    ds = ds.shuffle(1000)

  # Batch all datasets.
  ds = ds.batch(batch_size)

  # Use data augmentation only on the training set.
  if augment:
    ds = ds.map(lambda x, y: (data_augmentation(x, training=True), y),
                num_parallel_calls=AUTOTUNE)

  # Use buffered prefetching on all datasets.
  return ds.prefetch(buffer_size=AUTOTUNE)


train_ds = prepare(train_ds, shuffle=True, augment=True)
val_ds = prepare(val_ds)
test_ds = prepare(test_ds)
model = keras.Sequential([
    layers.Conv2D(32, (3, 3), input_shape=(180,180,3), padding='same', activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(32, activation='relu'),
    layers.Dense(1,activation='softmax')
])
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
epochs=1
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
582/582 [==============================] - 357s 604ms/step - loss: 0.6953 - accuracy: 0.4961 - val_loss: 0.6935 - val_accuracy: 0.5185
```

Result

Thus the program has been done for the image augmentation and the model has been evaluated.

Traffic Prediction – Mini project

Ex. No. : 9

Date:

Aim:

To write a python program to perform traffic prediction using deep learning model.

Algorithm:

Step1: Import the required Python libraries
Step2: Import the data
Step3: classify the data
Step4: label the data
Step5: perform the scaling
Step6: Build the model
Step7: Train the learning model
Step8: Test and evaluate the model

Program:

Import libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import tensorflow
from statsmodels.tsa.stattools import adfuller
from sklearn.preprocessing import MinMaxScaler
from tensorflow import keras
from keras import callbacks
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, Dense, LSTM, Dropout, GRU,
Bidirectional
from tensorflow.keras.optimizers import SGD
import math
from sklearn.metrics import mean_squared_error

import warnings
```

```
warnings.filterwarnings("ignore")
```

Import Dataset

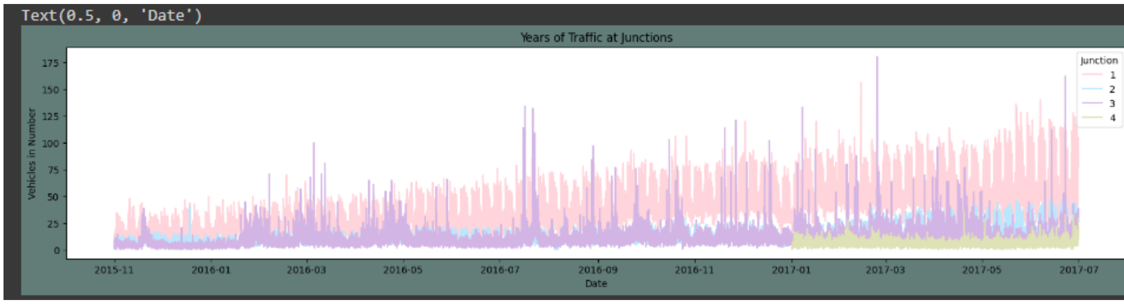
```
dataset = pd.read_csv("traffic.csv")
dataset.head()
```

	DateTime	Junction	Vehicles	ID
0	2015-11-01 00:00:00	1	15	20151101001
1	2015-11-01 01:00:00	1	13	20151101011
2	2015-11-01 02:00:00	1	10	20151101021
3	2015-11-01 03:00:00	1	7	20151101031
4	2015-11-01 04:00:00	1	9	20151101041

```
dataset["DateTime"]= pd.to_datetime(dataset["DateTime"])
dataset = dataset.drop(["ID"], axis=1) #dropping IDs column
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48120 entries, 0 to 48119
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   DateTime  48120 non-null  datetime64[ns]
 1   Junction  48120 non-null  int64
 2   Vehicles  48120 non-null  int64
dtypes: datetime64[ns](1), int64(2)
memory usage: 1.1 MB
```

```
# dataframe to be used for EDA
dataframe=dataset.copy()

# Let's plot the Timeseries
colors = [ "#FFD4DB","#BBE7FE","#D3B5E5","#dfe2b6"]
plt.figure(figsize=(20,4),facecolor="#627D78")
Time_series=sns.lineplot(x=dataframe['DateTime'],y="Vehicles",data=dataframe,
hue="Junction", palette=colors)
Time_series.set_title("Years of Traffic at Junctions")
Time_series.set_ylabel("Vehicles in Number")
Time_series.set_xlabel("Date")
```

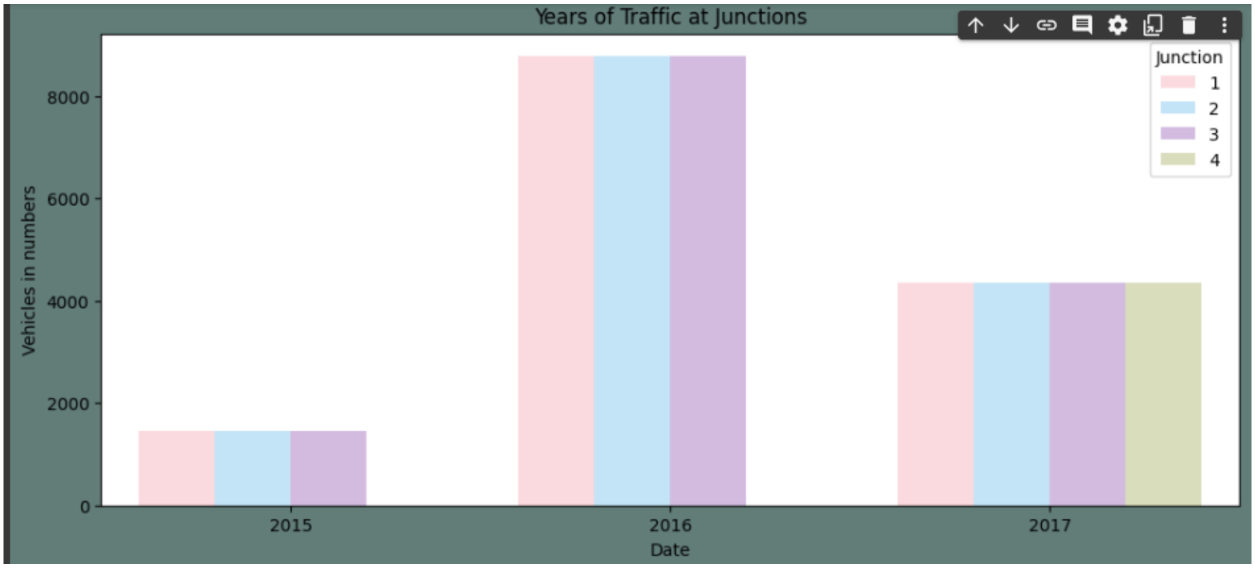



```
# Exploring more features
dataframe["Year"]= dataframe['DateTime'].dt.year
dataframe["Month"]= dataframe['DateTime'].dt.month
dataframe["Date_no"]= dataframe['DateTime'].dt.day
dataframe["Hour"]= dataframe['DateTime'].dt.hour
dataframe["Day"]= dataframe.DateTime.dt.strftime("%A")
dataframe.head()
```

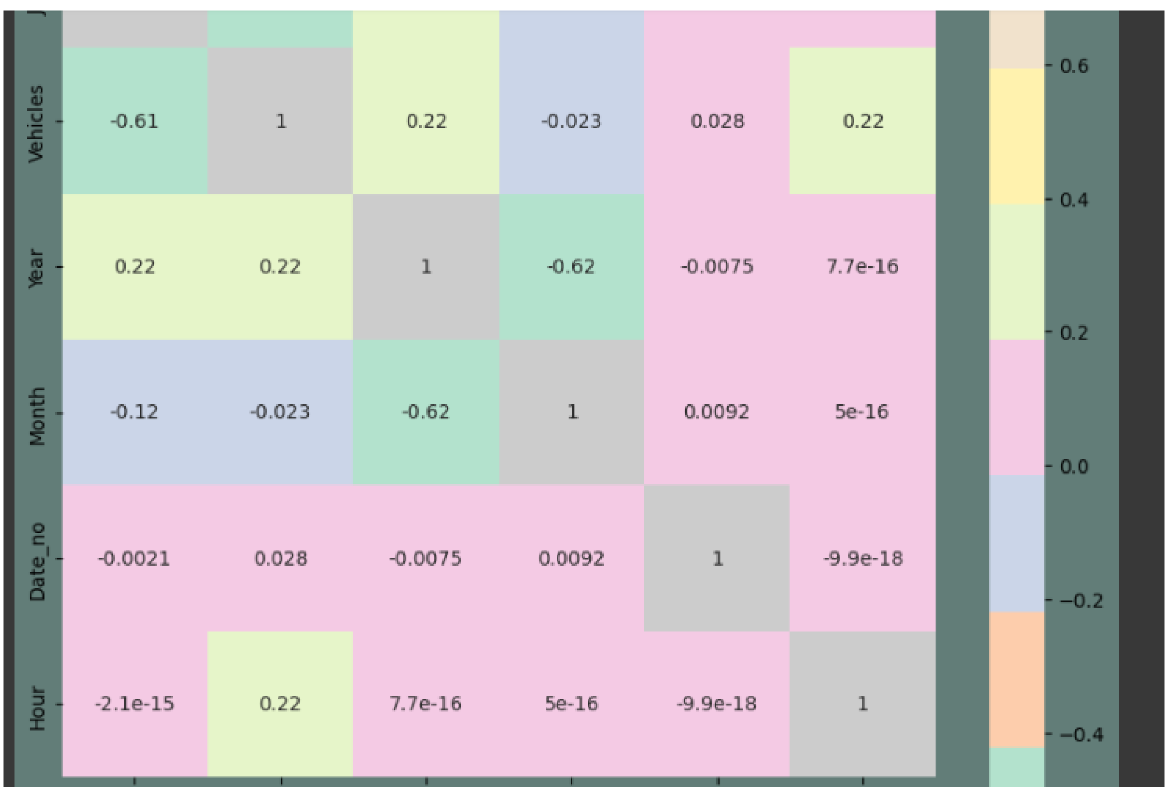
	DateTime	Junction	Vehicles	Year	Month	Date_no	Hour	Day
0	2015-11-01 00:00:00	1	15	2015	11	1	0	Sunday
1	2015-11-01 01:00:00	1	13	2015	11	1	1	Sunday
2	2015-11-01 02:00:00	1	10	2015	11	1	2	Sunday
3	2015-11-01 03:00:00	1	7	2015	11	1	3	Sunday
4	2015-11-01 04:00:00	1	9	2015	11	1	4	Sunday

```
# Let's plot the Timeseries
new_features = [ "Year","Month", "Date_no", "Hour", "Day"]

for i in new_features:
    plt.figure(figsize=(10,2),facecolor="#627D78")
    ax=sns.lineplot(x=dataframe[i],y="Vehicles",data=dataframe, hue="Junction", palette=colors
)
    plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)


plt.figure(figsize=(12,5),facecolor="#627D78")
count = sns.countplot(data=dataframe, x =dataframe["Year"], hue="Junction", palette=colors)
count.set_title("Years of Traffic at Junctions")
count.set_ylabel("Vehicles in numbers")
```

```
count.set_xlabel("Date")
```




```
corrmat = dataframe.corr()
plt.subplots(figsize=(10,10),facecolor="#627D78")
sns.heatmap(corrmat,cmap= "Pastel2",annot=True,square=True, )
```



```
sns.pairplot(data=dataframe, hue= "Junction",palette=colors)

# Pivoting dataset from junction
dataframe_junction = dataset.pivot(columns="Junction", index="DateTime")
dataframe_junction.describe()
```

Junction	1	2	3	4
count	14592.000000	14592.000000	14592.000000	4344.000000
mean	45.052906	14.253221	13.694010	7.251611
std	23.008345	7.401307	10.436005	3.521455
min	5.000000	1.000000	1.000000	1.000000
25%	27.000000	9.000000	7.000000	5.000000
50%	40.000000	13.000000	11.000000	7.000000
75%	59.000000	17.000000	18.000000	9.000000
max	156.000000	48.000000	180.000000	36.000000

```
# Creating new dataframes
dataframe_1 = dataframe_junction[[('Vehicles', 1)]]
dataframe_2 = dataframe_junction[[('Vehicles', 2)]]
dataframe_3 = dataframe_junction[[('Vehicles', 3)]]
dataframe_4 = dataframe_junction[[('Vehicles', 4)]]
dataframe_4 = dataframe_4.dropna() #For only a few months, Junction 4 has only had minimal
data.

# As DFS's data frame contains many indices, its index is lowering level one.
list_dfs = [dataframe_1, dataframe_2, dataframe_3, dataframe_4]
for i in list_dfs:
    i.columns= i.columns.droplevel(level=1)

# Creates comparison dataframe charts using this function
def Sub_Plots4(dataframe_1, dataframe_2,dataframe_3,dataframe_4,title):
    fig, axes = plt.subplots(4, 1, figsize=(15, 8),facecolor="#627D78", sharey=True)
    fig.suptitle(title)
    #J1
    pl_1=sns.lineplot(ax=axes[0],data=dataframe_1,color=colors[0])
    #pl_1=plt.ylabel()
    axes[0].set(ylabel ="Junction 1")
```

```
    #J2
    pl_2=sns.lineplot(ax=axes[1],data=dataframe_2,color=colors[1])
    axes[1].set(ylabel ="Junction 2")
    #J3
    pl_3=sns.lineplot(ax=axes[2],data=dataframe_3,color=colors[2])
    axes[2].set(ylabel ="Junction 3")
    #J4
    pl_4=sns.lineplot(ax=axes[3],data=dataframe_4,color=colors[3])
    axes[3].set(ylabel ="Junction 4")


# It is displayed to test for stationarity.
Sub_Plots4(dataframe_1.Vehicles,
dataframe_2.Vehicles,dataframe_3.Vehicles,dataframe_4.Vehicles,"Transformation of the
Dataframe Before")


# Normalize Function
def Normalize(dataframe,column):
    average = dataframe[column].mean()
    stdev = dataframe[column].std()
    df_normalized = (dataframe[column] - average) / stdev
    df_normalized = df_normalized.to_frame()
    return df_normalized, average, stdev

# Differencing Function
def Difference(dataframe,column, interval):
    diff = []
    for i in range(interval, len(dataframe)):
        value = dataframe[column][i] - dataframe[column][i - interval]
        diff.append(value)
    return diff
# Normalize Function
def Normalize(dataframe,column):
    average = dataframe[column].mean()
    stdev = dataframe[column].std()
    df_normalized = (dataframe[column] - average) / stdev
    df_normalized = df_normalized.to_frame()
    return df_normalized, average, stdev

# Differencing Function
```

```
def Difference(dataframe,column, interval):
    diff = []
    for i in range(interval, len(dataframe)):
        value = dataframe[column][i] - dataframe[column][i - interval]
        diff.append(value)
    return diff


# Stationary time series check Improved Dickey-Fuller test
def Stationary_check(dataframe):
    check = adfuller(dataframe.dropna())
    print(f"ADF Statistic: {check[0]}")
    print(f"p-value: {check[1]}")
    print("Critical Values:")
    for key, value in check[4].items():
        print('\t%s: %.3f' % (key, value))
    if check[0] > check[4]["1%"]:
        print("Time Series is Non-Stationary")
    else:
        print("Time Series is Stationary")


# examining the series' stationary state

List_df_ND = [ dataframe_N1["Diff"], dataframe_N2["Diff"], dataframe_N3["Diff"],
dataframe_N4["Diff"]]
print("Checking the transformed series for stationarity:")
for i in List_df_ND:
    print("\n")
    Stationary_check(i)
```

```
Checking the transformed series for stationarity:


ADF Statistic: -15.265303390415337
p-value: 4.798539876398l6e-28
Critical Values:
        1%: -3.431
        5%: -2.862
        10%: -2.567
Time Series is Stationary


ADF Statistic: -21.795891026940065
p-value: 0.0
Critical Values:
        1%: -3.431
        5%: -2.862
        10%: -2.567
Time Series is Stationary
```

```
# Several NA values were produced as a result of differencing using a week's worth of data.
dataframe_J1 = dataframe_N1["Diff"].dropna()
dataframe_J1 = dataframe_J1.to_frame()

dataframe_J2 = dataframe_N2["Diff"].dropna()
dataframe_J2 = dataframe_J2.to_frame()

dataframe_J3 = dataframe_N3["Diff"].dropna()
dataframe_J3 = dataframe_J3.to_frame()

dataframe_J4 = dataframe_N4["Diff"].dropna()
dataframe_J4 = dataframe_J4.to_frame()

# Splitting the dataset
def Split_data(dataframe):
    training_size = int(len(dataframe)*0.90)
    data_len = len(dataframe)
    train, test = dataframe[0:training_size],dataframe[training_size:data_len]
    train, test = train.values.reshape(-1, 1), test.values.reshape(-1, 1)
    return train, test
# Splitting the training and test datasets
Junction1_train, Junction1_test = Split_data(dataframe_J1)
Junction2_train, Junction2_test = Split_data(dataframe_J2)
Junction3_train, Junction3_test = Split_data(dataframe_J3)
```

```
Junction4_train, Junction4_test = Split_data(dataframe_J4)

# Target and Feature
def target_and_feature(dataframe):
    end_len = len(dataframe)
    X = []
    y = []
    steps = 32
    for i in range(steps, end_len):
        X.append(dataframe[i - steps:i, 0])
        y.append(dataframe[i, 0])
    X, y = np.array(X), np.array(y)
    return X ,y

# fixing the shape of X_test and X_train
def FeatureFixShape(train, test):
    train = np.reshape(train, (train.shape[0], train.shape[1], 1))
    test = np.reshape(test, (test.shape[0],test.shape[1],1))
    return train, test

# Assigning features and target
X_train_Junction1, y_train_Junction1 = target_and_feature(Junction1_train)
X_test_Junction1, y_test_Junction1 = target_and_feature(Junction1_test)
X_train_Junction1, X_test_Junction1 = FeatureFixShape(X_train_Junction1, X_test_Junction1)

X_train_Junction2, y_train_Junction2 = target_and_feature(Junction2_train)
X_test_Junction2, y_test_Junction2 = target_and_feature(Junction2_test)
X_train_Junction2, X_test_Junction2 = FeatureFixShape(X_train_Junction2, X_test_Junction2)

X_train_Junction3, y_train_Junction3 = target_and_feature(Junction3_train)
X_test_Junction3, y_test_Junction3 = target_and_feature(Junction3_test)
X_train_Junction3, X_test_Junction3 = FeatureFixShape(X_train_Junction3, X_test_Junction3)

X_train_Junction4, y_train_Junction4 = target_and_feature(Junction4_train)
x_test_Junction4, y_test_Junction4 = target_and_feature(Junction4_test)
X_train_Junction4, x_test_Junction4 = FeatureFixShape(X_train_Junction4, x_test_Junction4)


#Model for the prediction
def GRU_model(X_Train, y_Train, X_Test):
```

```
    early_stopping = callbacks.EarlyStopping(min_delta=0.001,patience=10,
restore_best_weights=True)

    #The GRU model
    model = Sequential()
    model.add(GRU(units=150, return_sequences=True, input_shape=(X_Train.shape[1],1),
activation='tanh'))
    model.add(Dropout(0.2))
    model.add(GRU(units=150, return_sequences=True, input_shape=(X_Train.shape[1],1),
activation='tanh'))
    model.add(Dropout(0.2))
    model.add(GRU(units=50, return_sequences=True, input_shape=(X_Train.shape[1],1),
activation='tanh'))
    model.add(Dropout(0.2))
    model.add(GRU(units=50, return_sequences=True, input_shape=(X_Train.shape[1],1),
activation='tanh'))
    model.add(Dropout(0.2))

    model.add(GRU(units=50, input_shape=(X_Train.shape[1],1), activation='tanh'))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))

    # Compiling the model
    model.compile(optimizer=SGD(decay=1e-7, momentum=0.9),loss='mean_squared_error')
    model.fit(X_Train,y_Train, epochs=50, batch_size=150,callbacks=[early_stopping])
    pred_GRU= model.predict(X_Test)
    return pred_GRU

# To determine the root mean squared prediction error
def RMSE_Value(test,predicted):
    rmse = math.sqrt(mean_squared_error(test, predicted))
    print("The root mean squared error is {}.".format(rmse))
    return rmse

# Plotting the goal and forecast comparison plot
def PredictionsPlot(test,predicted,m):
    plt.figure(figsize=(12,5),facecolor="#627D78")
    plt.plot(test, color=colors[m],label="True Value",alpha=0.5 )
    plt.plot(predicted, color="#627D78",label="Predicted Values")
    plt.title("GRU Traffic Prediction Vs True values")
```

```
plt.xlabel("DateTime")
plt.ylabel("Number of Vehicles")
plt.legend()
plt.show()
```

Output:

```
Epoch 1/10
40/40 [==============================] - 180s 4s/step - loss: 13.7116 - accuracy: 0.2075 - val_loss: 2.6140 - val_accuracy: 0.2500
Epoch 2/10
40/40 [==============================] - 169s 4s/step - loss: 2.2675 - accuracy: 0.2100 - val_loss: 2.0198 - val_accuracy: 0.2000
Epoch 3/10
40/40 [==============================] - 170s 4s/step - loss: 1.7563 - accuracy: 0.2950 - val_loss: 2.0195 - val_accuracy: 0.3800
Epoch 4/10
40/40 [==============================] - 170s 4s/step - loss: 1.7580 - accuracy: 0.2925 - val_loss: 1.3700 - val_accuracy: 0.4700
Epoch 5/10
40/40 [==============================] - 153s 4s/step - loss: 1.8236 - accuracy: 0.3225 - val_loss: 2.0655 - val_accuracy: 0.2000
Epoch 6/10
40/40 [==============================] - 157s 4s/step - loss: 1.6553 - accuracy: 0.3050 - val_loss: 1.7280 - val_accuracy: 0.2000
Epoch 7/10
40/40 [==============================] - 155s 4s/step - loss: 1.4680 - accuracy: 0.3500 - val_loss: 1.5888 - val_accuracy: 0.2800
Epoch 8/10
40/40 [==============================] - 165s 4s/step - loss: 1.6130 - accuracy: 0.3250 - val_loss: 1.7193 - val_accuracy: 0.3500
Epoch 9/10
40/40 [==============================] - 156s 4s/step - loss: 1.5905 - accuracy: 0.3450 - val_loss: 1.4193 - val_accuracy: 0.3800
Epoch 10/10
40/40 [==============================] - 159s 4s/step - loss: 1.4059 - accuracy: 0.4075 - val_loss: 1.3703 - val_accuracy: 0.4200
[]
```

Result:

Thus the mini project has been done on the traffic prediction using the deep learning model.