ERODE SENGUNTHAR

ENGINEERING COLLEGE

(APPROVED BY AICTE, NEW DELHI & PERMANENTLY AFFILIATED TO ANNA UNIVERSITY, CHENNAI

ACCREDITED BY NBA, NEW DELHI, NAAC WITH GRADE "A" & IE(I) KOLKATA)

PERUNDURAI , ERODE - 638057.

An Autonomous Institution

BONAFIDE CERTIFICATE

Register No.…………………………………...

Certified that is the Bonafide Record of Work Done

Name of the Student : ____________________

Branch : ____________________

Name of the Lab : ____________________

____

Year/Sem : ____________________

Faculty Incharge   Head of the Department

Submitted for the Odd Semester Practical

Held on ……………………………………

Internal Examiner   External Examiner

INDEX

S.NO	DATE	NAME OF THE EXPERIMENT	PAGE NO	MARKS	FACULTY SIGN
1.		Solving Xor Problem Using DNN			
2.		Character Recognition Using CNN			
3.		Face Recognition Using CNN			
4.		Text Generation Using RNN			
5.		Sentiment Analysis Using LSTM			
6.		Parts Of Speech Tagging Using Sequence To Sequence Architecture			
7.		Machine Translation Using Encoder-Decoder Model			
8.		Image Augmentation Using GAN			

EX.NO: - 1 **SOLVING XOR PROBLEM USING DNN**

DATE: -

AIM:

The aim of this procedure is to solve the XOR problem using a Deep Neural Network (DNN) in Python.

PROCEDURE:

- Import the necessary libraries
- Prepare the dataset
- Create the DNN model
- Compile the model
- Train the model
- Test the model
- Print the predictions

PROGRAM:

```
#1.Import the necessary libraries:
import numpy as np
from keras.models import Sequential
from keras.layers  import Dense

X= np.array([[0, 0],[0, 1], [1,0],[1, 1]])
#Define the corresponding output labels
y = np.array([[0], [1], [1], [0]])
#3.Create the DNN model:
#Create a Sequential model
model = Sequential()
#Add layers to the model
model.add(Dense(4,input_dim=2,activation='relu'))
```

```
model.add(Dense(1, activation='sigmoid'))
# Compile the model
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
#4. Train the model:
model.fit(X,y,epochs=1000)
#5. Test the model:
# Predict outputs for the input dataset
predictions = model.predict(X)
rounded_predictions=np.round(predictions)
For i in range(len(X)):
   print(f"Input:{X[i]},PredictedOutput: {rounded_predictions[i]}")
```

OUTPUT:

Input:[00],PredictedOutput: [0.]

Input:[01],PredictedOutput: [1.]

Input:[10],PredictedOutput: [1.]

Input:[11],PredictedOutput: [0.]

RESULT:

Thus the program to solve the XOR problem using a Deep Neural Network(DNN) in Python written Successfully.

EX.NO: - 2 **CHARACTER RECOGNITION USING CNN**

DATE: -

AIM:

The aim of this code is to train a Convolutional Neural Network (CNN) model on the MNIST dataset and evaluate its performance by making predictions on the test set.

PROCEDURE:

- Import the necessary libraries
- Load and preprocess the dataset
- Create the CNN model
- Compile the model
- Train the model
- Evaluate the model
- Make predictions
- Visualize the results

PROGRAM:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

 (x_train,y_train),(x_test,y_test)= tf.keras.datasets.mnist.load_data()

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


#Reshape  the data for CNN
x_train=x_train.reshape(-1,28,28,1)
 x_test=x_test.reshape(-1,28, 28, 1)
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=64,kernel_size=(2,2),strides=(1,1),padding='same',
activation='relu', input_shape=(28,28,1)),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Conv2D(filters=32,kernel_size=(2,2),strides=(1,1),padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10,activation='softmax')])

model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',
metrics=['accuracy'])
model.fit(x_train,y_train,batch_size=60,epochs=10,verbose=1,validation_split=0.3)

# Evaluate the model on test set
score=model.evaluate(x_test,y_test,verbose=0)
print('\nTest accuracy:', score[1])
 predictions=model.predict(x_test)
 #Display some test images and their predicted labels
num_rows = 5
num_cols=3
num_images = num_rows * num_cols
plt.figure(figsize=(2*2*num_cols,2*num_rows))
for i in range(num_images):


    plt.subplot(num_rows,2*num_cols,2*i+1)
    plt.xticks([])
```

```
    plt.yticks([])
    plt.imshow(x_test[i].reshape(28,28),cmap=plt.cm.binary)
    predicted_label = np.argmax(predictions[i])
    plt.xlabel("{}({})".format(predicted_label,y_test[i]))
plt.show()
```

OUTPUT:

Epoch1/10

700/700[================================]-44s62ms/step-loss:0.4044 - accuracy:0.8701-val_loss:0.1232 -val_accuracy: 0.9623

.

.

.

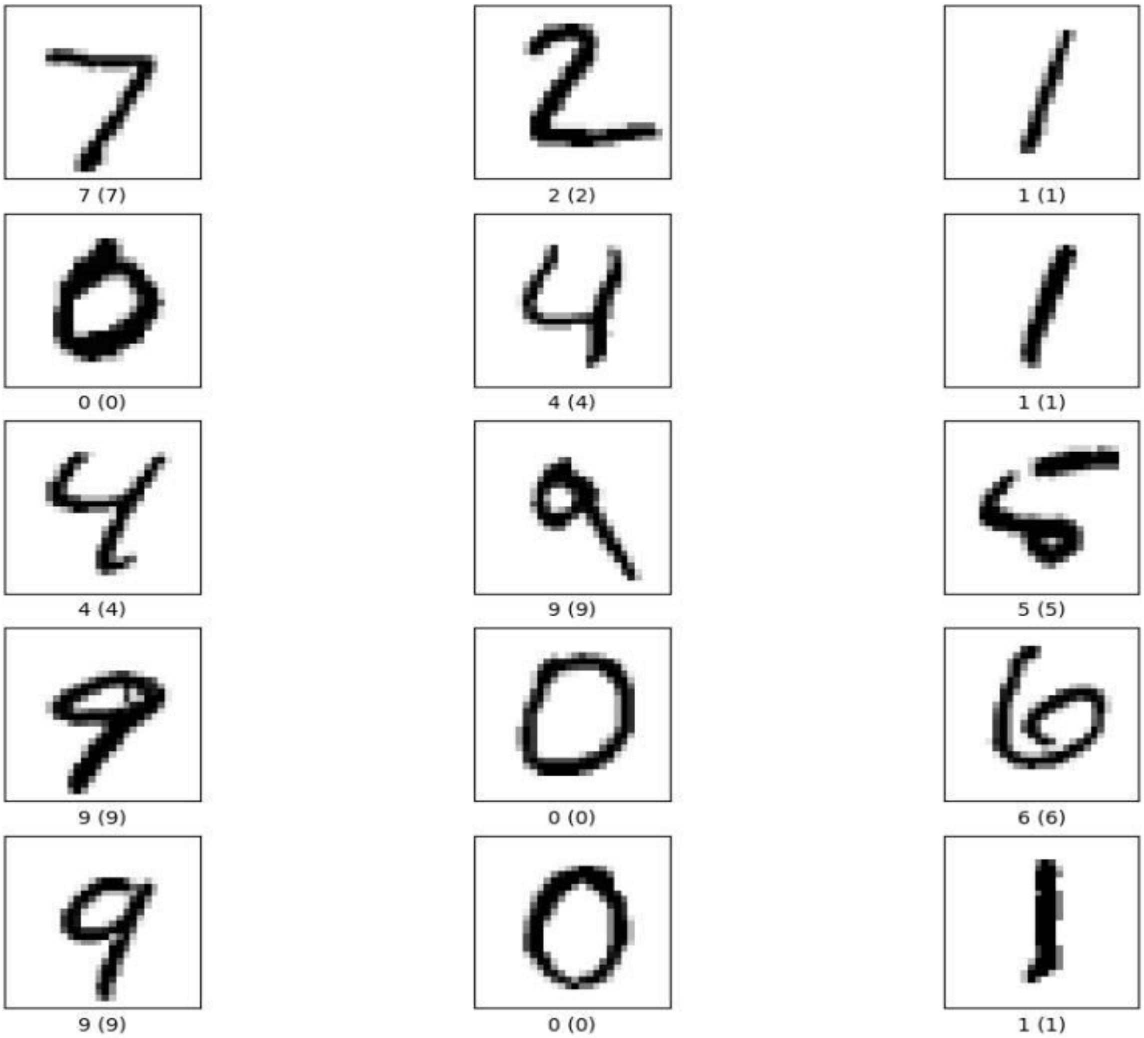
.

Epoch 10/10

700/700[================================]-36s51ms/step-loss:0.0545 - accuracy:0.9829-val_loss:0.0356 -val_accuracy: 0.9889

Testaccuracy:0.9901999831199646

313/313[================================] -2s 6ms/step



RESULT:

Thus the python code is to train a Convolutional Neural Network (CNN) model on the MNIST dataset and evaluate its performance by making predictions on the test set executed successfully.

EX.NO: - 3 **FACE RECOGNITION USING CNN**

DATE: -

AIM:

The aim of this program is to perform real-time face detection using the OpenCV library in Python. It captures video from your default camera (usually the webcam), detects faces in the video frames, and highlights the detected faces with green rectangles.

PROCEDURE:

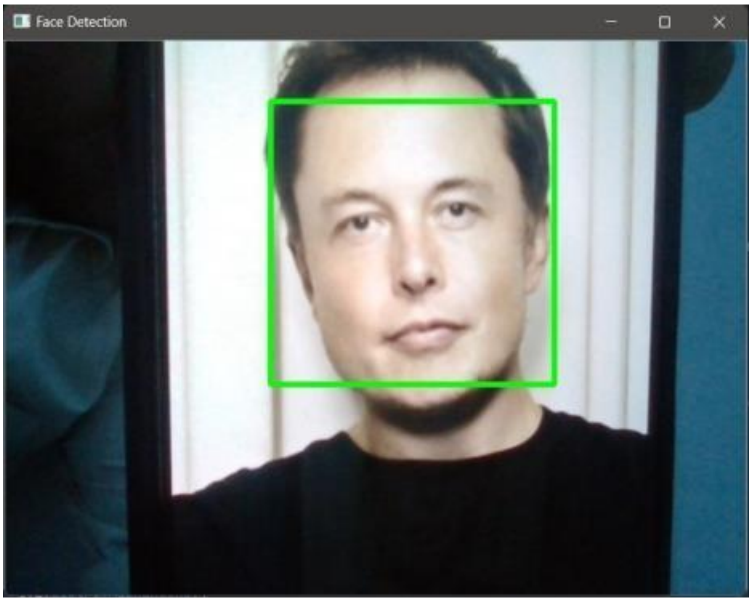
1. Import the OpenCV library.
2. Create a Cascade Classifier object and load the pre-trained face detection model (download -'haarcascade_frontalface_default.xml').
3. Initialize the video capture using the default camera (usually the webcam).
4. Start an infinite loop to continuously capture and process frames from the camera.
5. Read a frame from the camera.
6. Convert the frame to grayscale.
7. Use the Cascade Classifier to detect faces in the gray scale frame.
8. For each detected face , draw a green rectangle around it.
9. Display the frame with detected faces in a window called 'Face Detection.'
10. Check for the 'q' key press; if it's pressed , exit the loop.
11. Release the video capture and close all OpenCV windows when the loop is exited.

PROGRAM:

```
import cv2

#Load the pre-trained face detection model

face_cascade=cv2.Cascade Classifier(cv2.data.haarcascades+
'haarcascade_frontalface_default.xml')

#Initialize the video capture

cap = cv2.VideoCapture(0)

while True:

    ret,frame=cap.read()
```

```
    gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
     # Detect faces in the grayscale frame
    faces=face_cascade.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=5,
minSize=(30, 30))
    for (x, y, w, h) in faces:
        cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),3) #
    Display the frame with detected faces cv2.imshow('Face
    Detection', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
#Release the video capture and close OpenCV  windows
cap.release()
cv2.destroyAllWindows()
```

OUTPUT:



RESULT:

The program provides real-time face detection in the video feed from your camera, and the output is a window displaying the live video with highlighted faces

EX.NO: - 4 TEXT GENERATION USING RNN

DATE: -

AIM:

The aim of this Python code is to train a text generation model using RNN and Generate text based on a given input text.

PROCEDURE:

1. Read and process a text file (in this case ,a Shakespeare a text).
2. Tokenize the text ,create a character-to-index mapping ,and convert the text to numerical values.
3. Create a dataset for training the text generation model.
4. Build a deep learning model using TensorFlow's Keras API .The model consists of an embedding layer ,two LSTM layers ,dropout layers, batch normalization ,and a dense output layer.
5. Define a custom loss function for training the model.
6. Generate text using the trained model, starting from a user-provided input text.

PROGRAM:

```
import numpy as np
import tensorflow as tf


def process_text (file_path):
    text=open(file_path,'rb').read().decode(encoding='utf-8')#Read,thendecodeforpy2 compat.
    vocab=sorted(set(text))
    #Creating a mapping from unique characters to indices and vice versa
    char2idx = {u: i for i, u in enumerate(vocab)}
    idx2char= np.array(vocab)
    text_as_int=np.array([char2idx[c]forcintext])
    return text_as_int, vocab, char2idx, idx2char

def split_input_target(chunk):
    input_text,target_text=chunk[:-1], chunk[1:]
```

```
    return input_text , target_text



def create_dataset(text_as_int,seq_length=100,batch_size=64,buffer_size=10000):
    char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)
    dataset=char_dataset.batch(seq_length+1,drop_remainder=True).map(spl
it_input_target)
    dataset=dataset.shuffle(buffer_size).batch(batch_size,drop_remainder=True)
    return dataset


def build_model(vocab_size,embedding_dim=256,rnn_units=1024,batch_size=64):
    model = tf.keras.Sequential([
        tf.keras.layers.Embedding(vocab_size,embedding_dim,
batch_input_shape=[batch_size, None]),
        tf.keras.layers.LSTM(rnn_units,return_sequences=True,stateful=True,
recurrent_initializer='glorot_uniform'),
        tf.keras.layers.Dropout(0.1),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.LSTM(rnn_units,return_sequences=True,stateful=True,
recurrent_initializer='glorot_uniform'),
        tf.keras.layers.Dropout(0.1),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dense(vocab_size)
    ])
    return model


def loss(labels,logits):
    return tf.keras.losses.sparse_categorical_crossentropy(labels,logits,from_logits=True)


def generate_text(model,char2idx,idx2char,start_string,generate_char_num=1000,
temperature=1.0):
    #Evaluation step (generating text using the l earned model)
```

```
    #Low temperatures results in more predictable text, higher temperatures results in more
surprising text.
    # Converting our start string to numbers (vectorizing)
    input_eval = [char2idx[s] for s in start_string]
    input_eval = tf.expand_dims(input_eval, 0)
    text_generated=[]#Emptystringtostoreourresults
    model.reset_states()
    foriinrange(generate_char_num):
        predictions=model(input_eval)
        predictions = tf.squeeze(predictions, 0)
        predictions /= temperature
        # using a categorical distribution to predict the character returned by the model
        predicted_id=tf.random.categorical(predictions,num_samples=1)[-1,0].numpy()

        input_eval=tf.expand_dims([predicted_id],axis=0)
        text_generated.append(idx2char[predicted_id])
    returnstart_string+''.join(text_generated)


#path_to_file=tf.keras.utils.get_file('nietzsche.txt','https://s3.amazonaws.com/text-
datasets/nietzsche.txt')
path_to_file = tf.keras.utils.get_file('shakespeare.txt',
'https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt')


text_as_int,vocab,char2idx,idx2char=process_text(path_to_file)
dataset = create_dataset(text_as_int)
model=build_model(vocab_size=len(vocab))
model.compile(optimizer='adam', loss=loss)
model.summary()
history = model.fit(dataset, epochs=50)
model.save_weights("gen_text_weights.h5",save_format='h5')
# To keep this prediction step simple, use a batch size of 1
```

```
model = build_model(vocab_size=len(vocab),batch_size=1)
model.load_weights("gen_text_weights.h5")
model.summary()

user_input=input("Writethebeginningofthetext,theprogramwillcompleteit.Yourinput is: ")
generated_text=generate_text(model,char2idx,idx2char,start_string=user_input,
generate_char_num=2000)
print(generated_text)
```

OUTPUT:

“ First Citizen:We are accounted poor citizens, the patricians good. What authority sure its on would relieve us: if they would yield us but the super fluity , while it were wholesome, we might guess they relieved us humanely ;but they think we are too dear: the leanness that afflicts us, the object of our misery, is as an inventory to particularise their abundance; our sufferance is a gain to them Let us revenge this with our pikes, ere we become rakes: for the gods know I speak this in hunger for bread, not in thirst for revenge. ”

RESULT:

The code demonstrates how to train a text generation model and use it to create coherent text based on a given starting point.

EX.NO: - 5 SENTIMENT ANALYSIS USING LSTM

DATE: -

AIM:

The aim of this code is to perform sentiment analysis on a dataset of comments using deep learning techniques.

PROCEDURE:

1. Import necessary libraries including NumPy, Pandas, Matplotlib, Hazm (a Persian natural language processing library), and scikit-learn.

2. Read a dataset from a CSV file called 'Snappfood-SentimentAnalysis.csv'.

3. Perform data preprocessing, including tokenization, lemmatization, normalization, and removing stop words and punctuations.

4. Create a new dataset containing the cleaned text and sentiment labels.

5. Tokenize and pad the text data for model training.

6. Build a deep learning model using Keras with an embedding layer, Bidirectional LSTM layers, SpatialDropout1D, Batch Normalization, and a Dense output layer.

7. Compile the model with the Adam optimizer and binary cross-entropy loss.

8. Train the model on the training data.

9. Make predictions on both the training and test data.

10. Evaluate the model using classification reports to measure its performance.

PROGRAM:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import hazm
import string
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
```

```
from keras.models import Sequential

from keras.layers import Dense,Dropout,Flatten,Spatial Dropout1D, Embedding,
Bidirectional, LSTM, Batch Normalization

from keras.callbacks importModelCheckpoint,EarlyStopping

from sklearn.utils import class_weight


from sklearn.metrics import classification_report

from collections import Counter


data=pd.read_csv('Snappfood-SentimentAnalysis.csv',delimiter='\t',on_bad_lines='skip') data


data.info()

data.label_id.value_counts()

data.isnull().sum()

data=data[['comment','label_id']]

data

data.drop(inplace=True) data

punctuations = string.punctuation+","+"،"

translator = str.maketrans(",", punctuations)

stopwords = hazm.stopwords_list()

hazm.word_tokenize(data['comment'][10])

lem = hazm.Lemmatizer()

norm = hazm.Normalizer()

dataset = pd.DataFrame(columns=['Text','Sentiment'])


for index,row in data.iterrows():

  text =row['comment']

  text_tokenized= hazm.word_tokenize(text)

  text_lem=[lem.lemmatize(x)for xin text_tokenized]
```

```
  text_norm = [norm.normalize(x) for x in text_lem]
  clean_text=[x for x in text_norm if not x in stopwords]
  final_text = [x.translate(translator) for x in clean_text]
  dataset.loc[index] = ({
      'Text' : ''.join(final_text),
      'Sentiment':row['label_id']
  })
dataset.Sentiment.value_counts()
dataset.Sentiment.value_counts()
dataset['Text']
dataset['words_count']=dataset['Text'].apply(lambdat:len(hazm.word_tokenize(t)))
max_len = dataset["words_count"].max()
max_len
texts=''.join(dataset['Text'])
tokens=hazm.word_tokenize(texts)
counter = Counter(tokens)
min_freg=35
filtered=[word for word,count in counter.items() if count>=min_freg]
unique_words = set(filtered)
n_words=len(unique_words)
n_words
X= dataset['Text']
Y=dataset['Sentiment']
xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.01,random_state=4234)
tokenizer = Tokenizer(num_words = n_words)
tokenizer.fit_on_texts(xtrain)
def Tokenization_padSequences(x,maxlen=max_len):
    xseq = tokenizer.texts_to_sequences(x)
    xpad=pad_sequences(xseq,padding='post',maxlen=max_len)
    return xpad
```

```
xtrain_pad=Tokenization_padSequences(xtrain)
xtest_pad = Tokenization_padSequences(xtest)
xtest_pad
sequences = tokenizer.texts_to_sequences(dataset['Text'])
print(dataset['Text'][200])
print(sequences[200])
model = Sequential()
model.add(Embedding(n_words, 80, input_length=max_len))
model.add(Bidirectional(LSTM(256,dropout=0.2,return_sequences=True)))
model.add(SpatialDropout1D(0.2))
model.add(Bidirectional(LSTM(128,dropout=0.2)))
model.add(BatchNormalization())
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='Adam',metrics=['accuracy'],loss='binary_crossentropy')
model.summary()

np.unique(ytrain)
model.fit(xtrain_pad,ytrain,epochs=2,batch_size=10,validation_data=(xtest_pad,ytest))
ypred_train = (model.predict(xtrain_pad) > 0.5).astype(int)
print(classification_report(ytrain, ypred_train))
ypred=(model.predict(xtest_pad)>0.5).astype(int)
print(classification_report(ytest, ypred))
```

OUTPUT:

The output includes the model summary, training and evaluation reports, and the model's predictions.

	Unnamed: 0	comment	label	label_id
0	NaN	واقعا حیف وقت که بنویسم سرویس دهیتون شده افتضاح	SAD	1.0
1	NaN	قرار بود ۱ ساعته برسه ولی نیم ساعت زودتر از مو...	HAPPY	0.0
2	NaN	قیمت این مدل اصلا با کیفیتش سازگاری نداره، فقط...	SAD	1.0
3	NaN	عالللی بود همه چه درست و به اندازه و کیفیت خوب...	HAPPY	0.0
4	NaN	شیرینی وانیلی فقط یک مدل بود.	HAPPY	0.0
...	...	...	...	...
69995	NaN	سلام من به فاکتور غذاهایی که سفارش میدم احتیاج...	SAD	1.0
69996	NaN	سایز پیتزا نسبت به سفارشاتی که قبلا گذشتم کم ش...	SAD	1.0
69997	NaN	من قارچ اضافه رو اضافه کرده بودم بودم اما اگر ...	HAPPY	0.0
69998	NaN	همرو بعد آساعت تاخیر اشتباه آوردن پولشم رفت رو...	SAD	1.0
69999	NaN	فلفلش خیییلی تند بود.	HAPPY	0.0

70000 rows × 4 columns

```
array([[  77,   18,  554, ...,    0,    0,    0],
       [ 128,   41,  334, ...,    0,    0,    0],
       [ 300, 1577,   93, ...,    0,    0,    0],
       ...,
       [   9,   11,    1, ...,    0,    0,    0],
       [  35,  222,  289, ...,    0,    0,    0],
       [   5,    2,  172, ...,    0,    0,    0]], dtype=int32)
```

Model:"sequential"

Layer(type)	OutputShape	Param#
embedding(Embedding)	(None,231,80)	144800
bidirectional (Bidirectiona l)	(None, 231, 512)	690176
spatial_dropout1d(SpatialD ropout1D)	(None,231,512)	0
bidirectional_1 (Bidirectio nal)	(None, 256)	656384
batch_normalization(BatchN ormalization)	(None,256)	1024
dense(Dense)	(None,1)	257

Totalparams: 1,492,641

```
Trainableparams:1,492,129
Non-trainableparams:512

Epoch1/20
459/459[==============================]-99s194ms/step -loss: 0.4216 -accura
cy:0.8075-val_loss:0.4008-val_accuracy:0.8245
Epoch 2/20
459/459[==============================]-67s147ms/step -loss: 0.3860 -accura
cy:0.8295-val_loss: 0.3657 -val_accuracy: 0.8331
.

.

.

Epoch 20/20
459/459[==============================]-64s 138ms/step-loss:0.2415 -accura
cy:0.8986-val_loss: 0.4892 -val_accuracy: 0.8187


22/22[==============================]-1s 25ms/step
            precision    recallf1-scoresupport

        0.0       0.82      0.81      0.81       340
        1.0       0.82      0.83      0.82       355

   accuracy                        0.82       695
  macroavg       0.82      0.82      0.82       695
weightedavg       0.82      0.82      0.82       695
```

RESULT:

The code provides a sentiment analysis model capable of classifying text comments into sentiment categories. The result includes various metrics like accuracy, precision, recall, and F1-score to assess the model's performance on both the training and test datasets.

EX.NO:- 6 PARTS OF SPEECH TAGGING USING SEQUENCE TO

DATE:- SEQUENCE ARCHITECTURE

AIM:-

The aim of the provided Python code is to perform Part-of-Speech(POS)tagging on a Given text using a Recurrent Neural Network (RNN)model.

PROCEDURE:-

1. Import the necessary libraries and modules, including NLTK, numpy, Keras, and related functions.
2. Define a paragraph of text that you want to perform POS tagging on.
3. Tokenize the paragraph into words using NLTK's ‘word_tokenize’ function.
4. Perform POS tagging on the tokenized words using NLTK's`pos_tag` function.
5. Create a vocabulary and encode words into integers using Keras's `Tokenizer`and `texts_to_sequences` functions.
6. Create a vocabulary for POS tags and encode the min to integers.
7. Prepare the data for training by converting the word sequences and POS tag sequences into numpy arrays.
8. Define an RNN model using Keras with embedding , Simple RNN, and Time Distributed layers.
9. Compile the model with the appropriate loss and optimizer.
10. Train the model on the data.
11. Use the trained model top redict POS tags for the input text.
12. Extract the predicted POS tags and display them along side the original words.

PROGRAM:-

```
import nltk

import numpy as np

from nltk.tokenize import word_tokenize

from nltk.corpus import brown

from keras.preprocessing.text importTokenizer

from keras.preprocessing.sequence import pad_sequences
```

```
from keras.models import Sequential

from keras.layers import Embedding, SimpleRNN ,Dense, TimeDistributed

from keras.utils import to_categorical


#Download NLTK data for tokenization and POS tagging

nltk.download('punkt')

nltk.download('averaged_perceptron_tagger')


paragraph="""
Natural language processing (NLP) is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (natural) languages. NLP focuses on the interaction between humans and computers using natural language. The ultimate goal of NLP is to read, decipher, understand, and make sense of the human language in a manner that is both valuable and meaningful. Most NLP techniques rely on machine learning to derive meaning from human languages."""


#Tokenize the paragraph into words

words = word_tokenize(paragraph)

#PerformPOStaggingusingNLTK

pos_tags = nltk.pos_tag(words)

#Createavocabularyandencodewordstointegers

tokenizer = Tokenizer()

tokenizer.fit_on_texts(words)

word_sequences = tokenizer.texts_to_sequences(words)

word_sequences=pad_sequences(word_sequences,padding='post') #

Create a vocabulary for POS tags

pos_tags_set=set(tagforword,taginpos_tags)

num_pos_tags = len(pos_tags_set)
```

```
pos_tag_to_idx={tag: i fori, tag in enumerate(pos_tags_set)}

#Encode POS tags to integers
pos_tag_sequences=[pos_tag_to_idx[tag]for _,tagin pos_tags]

#Prepare data for training
X= np.array(word_sequences)
Y=np.array(pos_tag_sequences)

#Define and compile the RNN model
model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index)+1,output_dim=32,
input_length=X.shape[1]))
model.add(SimpleRNN(64,return_sequences=True))
model.add(TimeDistributed(Dense(num_pos_tags,activation='softmax')))#Use
TimeDistributed layer

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

#Train the model
model.fit(X,Y,epochs=10,batch_size=32,verbose=2)

# POS tagging prediction
predictions=model.predict(X)
predicted_pos_tags = []

#Extract the predicted POS tags
for i in range(len(X)):

predicted_pos_tags.append([list(pos_tag_to_idx.keys())[list(pos_tag_to_idx.values()).index(t
ag)] for tag in predictions[i].argmax(axis=-1)])
```

```
#Display the original text with predicted POS tags
forword,pos_tag_listinzip(words,predicted_pos_tags):
    print(f"{word}: {pos_tag_list}")
```

OUTPUT:-

The output of the code will display the original text with their predicted POS tags.

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
/root/nltk_data...
[nltk_data] Unzipping
taggers/averaged_perceptron_tagger.zipTrue
Epoch1/10
3/3-2s-loss:2.9998-accuracy:0.0460-2s/epoch-501ms/step Epoch
2/10
3/3-0s-loss:2.9815-accuracy:0.0812-14ms/epoch-5ms/step Epoch
3/10
3/3-0s-loss:2.9644-accuracy:0.1119-12ms/epoch-4ms/step Epoch
4/10
3/3-0s-loss:2.9466-accuracy:0.1167-12ms/epoch-4ms/step Epoch
5/10
3/3-0s-loss:2.9289-accuracy:0.1403-13ms/epoch-4ms/step Epoch
6/10
3/3-0s-loss:2.9101-accuracy:0.1444-13ms/epoch-4ms/step Epoch
7/10
3/3-0s-loss:2.8902-accuracy:0.1526-12ms/epoch-4ms/step Epoch
8/10
3/3-0s-loss:2.8690-accuracy:0.1485-16ms/epoch-5ms/step Epoch
9/10
3/3-0s-loss:2.8466-accuracy:0.1470-12ms/epoch-4ms/step
```

Epoch 10/10

3/3-0s-loss:2.8226-accuracy:0.1579-19ms/epoch-6ms/step Natural:

['JJ']

language: ['NN']

processing:['NN']

(: [',']

NLP:['NN']

): [',']

is:['VBZ']

a:['DT']

field: ['NN']

of: ['IN']

computer: ['NN']

science:['NN']

,: [',']

artificial: ['JJ']

intelligence:['NN']

,: [',']

and: ['NN']

computational:['JJ']

linguistics: ['NN']

concerned: ['NN']

with: ['IN']

the: ['DT']

interactions:['NN']

between: ['NN']

computers: ['NN']

and:['NN']

RESULT:-

The RNN model is trained to predict POS tags for words in the input text. The result consists of the predicted POS tags associated with each word in the given text.

EX.NO: -7 MACHINE TRANSLATION USING ENCODER-DECODER MODEL

DATE: -

AIM:

The aim of this code is to create a machine translation model from English to French Using a sequence-to-sequence architecture with LSTM layers.

PROCEDURE:

1. Import necessary libraries, including NumPy, Pandas, and TensorFlow's Keras module.
2. Define hyperparameters such as batch size, epochs, latent dimension, and the number of samples.
3. Load the data from the '`fra.txt`' file and preprocess it to create input and target sequences.
4. Build token dictionaries for both input and target characters.
5. Create one-hot encoded data for encoder and decoder inputs as well as decoder targets.
6. Define the encoder and decoder models.
7. Compile the model for training using the RMSprop optimizer and categorical cross-entropy loss.
8. Train the model on the training data.
9. Save the trained model to a file named `eng2french.h5`.
10. Define sampling models for inference.
11. Implement a decoding function to translate English input sentences to French.
12. Iterate over a set of input sentences and generate translations.

PROGRAM:

```
import numpy as np

import pandas as pd

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, LSTM, Dense


batch_size=64

epochs=40

latent_dim=256
```

```
num_samples=10000

data_path='/content/fra.txt'

# Vectorize the data.
input_texts = []
target_texts = []
input_characters = set()
target_characters=set()
with open(data_path,'r',encoding='utf-8')asf: lines
   = f.read().split('\n')
for line in lines[:min(num_samples,len(lines)-1)]:
   input_text, target_text, _ = line.split('\t')
   #We use"tab"as the"start sequence" character
   #for the targets, and "\n" as "endsequence" character.
   target_text = '\t' + target_text + '\n'
   input_texts.append(input_text)
   target_texts.append(target_text)
   for char in input_text:
      ifcharnotininput_characters:
         input_characters.add(char)
   for char in target_text:
      if char not in target_characters:
         target_characters.add(char)

input_characters=sorted(list(input_characters))
target_characters=sorted(list(target_characters))

num_encoder_tokens=len(input_characters)
num_decoder_tokens=len(target_characters)
```

```
max_encoder_seq_length=max([len(txt) for txt in input_texts])
max_decoder_seq_length=max([len(txt)fortxtintarget_texts])

print('Numberofsamples:',len(input_texts))
print('Number of unique input tokens:', num_encoder_tokens)
print('Number of unique output tokens:', num_decoder_tokens)
print('Max sequence length for inputs:', max_encoder_seq_length)
print('Maxsequencelengthforoutputs:',max_decoder_seq_length)

input_token_index=dict(
  [(char,i)fori,charinenumerate(input_characters)])
target_token_index=dict(
[(char,i)fori,charinenumerate(target_characters)])

encoder_input_data= np.zeros(
  (len(input_texts),max_encoder_seq_length,num_encoder_tokens),
  dtype='float32')
decoder_input_data= np.zeros(
  (len(input_texts),max_decoder_seq_length,num_decoder_tokens),
  dtype='float32')
decoder_target_data=np.zeros(
  (len(input_texts),max_decoder_seq_length,num_decoder_tokens),
  dtype='float32')

for i,(input_text, target_text) in enumerate(zip(input_texts,target_texts)):
  for t, char in enumerate(input_text):
    encoder_input_data[i,t,input_token_index[char]]=1.
  encoder_input_data[i, t + 1:, input_token_index["]] = 1.
  for t, char in enumerate(target_text):
```

```
        #decoder_target_data is a head of decoder_input_data by one time
        step decoder_input_data [i, t, target_token_index[char]] = 1.
        if t >0:
          #decoder_target_datawillbeaheadbyonetimestep #
          and will not include the start character.
          decoder_target_data[i,t-1,target_token_index[char]]=1.
    decoder_input_data[i, t + 1:, target_token_index["]] = 1.
    decoder_target_data[i, t:, target_token_index["]] = 1.

#Define an input sequence and process it.
encoder_inputs=Input(shape=(None,num_encoder_tokens))
encoder = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)
# We discard `encoder_outputs` and only keep the states.
encoder_states = [state_h, state_c]

# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs=Input(shape = (None,num_decoder_tokens))
# We set up our decoder to return full output sequences,
#and to return internal states as well.We don't use the
# return states in the training model, but we will use them in inference.
decoder_lstm=LSTM(latent_dim,return_sequences=True,return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs,
                                     initial_state=encoder_states)
decoder_dense=Dense(num_decoder_tokens,activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

#Define the modelthatwill turn
#`encoder_input_data`&`decoder_input_data`into`decoder_target_data`
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

```
#Run training
model.compile(optimizer='rmsprop',loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit([encoder_input_data,decoder_input_data],decoder_target_data,
          batch_size=batch_size,
          epochs=epochs,
          validation_split=0.2)

model.save('eng2french.h5')

#Definesamplingmodels
encoder_model=Model(encoder_inputs, encoder_states)

decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_states_inputs=[decoder_state_input_h,decoder_state_input_c]
decoder_outputs, state_h, state_c = decoder_lstm(
    decoder_inputs,initial_state=decoder_states_inputs)
decoder_states = [state_h, state_c]
decoder_outputs=decoder_dense(decoder_outputs)
decoder_model = Model(
    [decoder_inputs]+decoder_states_inputs,
    [decoder_outputs] + decoder_states)

#Reverse-look up token index to decode sequences back to
# something readable.
reverse_input_char_index= dict(
    (i,char)forchar,iininput_token_index.items()) reverse_target_char_index
= dict(
```

```
(i,char)for char,i in target_token_index.items())

def decode_sequence(input_seq):
    #Encode the input as state vectors.
    states_value=encoder_model.predict(input_seq)

    # Generate empty target sequence of length 1.
    target_seq=np.zeros((1,1,num_decoder_tokens))
    #Populate the first character of target sequence with the start character.
    target_seq[0, 0, target_token_index['\t']] = 1.
    #Sampling loop forabatch of sequences
    #(to simplify, here we assume a batch of size1).
    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens,h,c=decoder_model.predict(
            [target_seq] + states_value)

        #Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char=reverse_target_char_index[sampled_token_index]
        decoded_sentence += sampled_char
        #Exit condition : either hit max length
        # or find stop character.
        if(sampled_char== '\n'or
           len(decoded_sentence)>max_decoder_seq_length):
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq=np.zeros((1,1,num_decoder_tokens))
```

```
        target_seq[0,0,sampled_token_index]=1.

        # Update states
        states_value=[h,c]
    return decoded_sentence
for seq_index in range(100):
    #Take one sequence(part of the training set)
    # for trying out decoding.
    input_seq=encoder_input_data[seq_index:seq_index+1]
    decoded_sentence = decode_sequence(input_seq)
    print('-')
    print('Input sentence:', input_texts[seq_index])
    print('Decodedsentence:',decoded_sentence)
```

OUTPUT:

Number of samples: 10000

Number of unique input tokens:71

Number of unique output tokens:93

Max sequence length for inputs: 15

Maxsequencelengthforoutputs:59

Epoch 1/40

125/125[==============================]-11s25ms/step-loss:1.2294-accuracy:0.7319-val_loss:1.2328-val_accuracy:0.7112

Epoch 2/40

125/125[==============================]-2s13ms/step-loss:0.9437 - accuracy: 0.7476-val_loss: 1.0409-val_accuracy: 0.7087

.

.

Epoch 40/40

125/125[==============================]-1s11ms/step-loss: 0.3223-accuracy: 0.9032-val_loss: 0.4583-val_accuracy: 0.8676

Input sentence: Go.

Decoded sentence: Pars!

1/1[================================]- 0s 16ms/step

1/1[================================]- 0s 20ms/step

1/1[================================]- 0s 21ms/step

1/1[================================]- 0s 18ms/step

1/1[================================]- 0s 23ms/step

1/1[================================]- 0s 20ms/step

1/1[================================]- 0s 18ms/step

1/1[================================]- 0s 21ms/step

RESULT:

The code trains a machine translation model that can translate English sentences into French.The result includes the translations of input sentences provided in the 'fra.txt' dataset.

EX.NO: -8 IMAGE AUGMENTATION USING GAN

DATE:-

AIM:

The aim of this code is to train a Generative Adversarial Network(GAN) to generate synthetic images that resemble handwritten digits from the MNIST dataset.

PROCEDURE:

1. Import necessary libraries and set up constants for image dimensions, channels, and noise vector size.
2. Define a generator model and a discriminator model using Keras.
3. Compile the discriminator model with binary cross-entropy loss and the Adam optimizer.
4. Create the generator model, which takes a noise vector as input and generates images.
5. Create a combined GAN model where the generator is trained to fool the discriminator.
6. Compile the combined model with binary cross-entropy loss and the Adam optimizer.
7. Load the MNIST dataset and preprocess it.
8. Define labels for real and fake images.
9. Training loop:
10. Train the discriminator:
    a. Select a random batch of real images from the dataset.
    b. Generate a batch of fake images using the generator.
    c. Calculate the discriminator loss for both real and fake images and update the discriminator's weights.
11. Train the generator:
    a. Generate a new batch of fake images.
    b. Calculate the generator loss by trying to make the discriminator classify the fake images as real.
12. Print and store losses and accuracies at regular intervals.
13. Generate and save sample images at regular intervals.

PROGRAM:

```
fromkeras.datasetsimportmnist

fromkeras.layersimportInput,Dense,Reshape,Flatten from

keras.layers import Activation

fromkeras.layers.advanced_activationsimportLeakyReLU

from keras.models import Sequential, Model

fromkeras.optimizersimportAdam


importmatplotlib.pyplotasplt
```

```
import sys

importnumpy asnp

"""Specifythedatasize"""

img_rows=28
img_cols=28
channels=1
img_shape=(img_rows,img_cols, channels)

z_dim=100
"""GeneratorNetwork"""
defgenerator(img_shape, z_dim):
   model =Sequential()

   #Hidden layer
   model.add(Dense(128,input_dim=z_dim))

   # Leaky ReLU
   model.add(LeakyReLU(alpha=0.01))

   # Output layer with tanh activation
   model.add(Dense(28*28*1,activation='tanh'))
   model.add(Reshape(img_shape))

   z=Input(shape=(z_dim,))
   img = model(z)

   returnModel(z,img)
```

```
    #Hidden layer
    model.add(Dense(128,input_dim=z_dim))

    # Leaky ReLU
    model.add(LeakyReLU(alpha=0.01))

    # Output layer with tanh activation
    model.add(Dense(28*28*1,activation='tanh'))
    model.add(Reshape(img_shape))

    z=Input(shape=(z_dim,))
    img = model(z)

    returnModel(z,img)

"""DiscriminatorNetwork"""

defdiscriminator(img_shape):

    model =Sequential()

    model.add(Flatten(input_shape=img_shape))

    # Hidden layer
    model.add(Dense(128))

    # Leaky ReLU
    model.add(LeakyReLU(alpha=0.01))#
    Outputlayerwithsigmoid activation
    model.add(Dense(1,activation='sigmoid'))
```

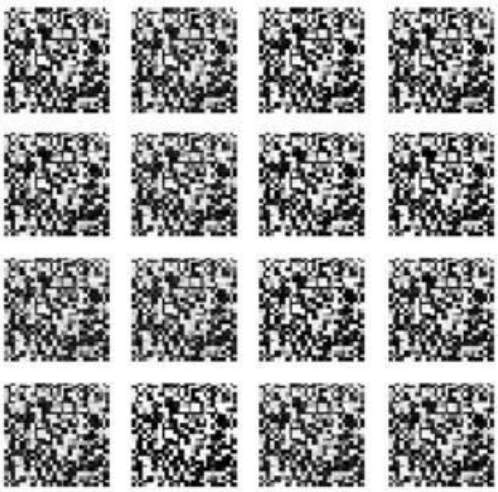
```
    img=Input(shape=img_shape)
    prediction = model(img)

    returnModel(img,prediction)

discriminator = discriminator(img_shape)
discriminator.compile(loss='binary_crossentropy',
                      optimizer=Adam(),metrics=['accuracy'])

# Build the Generator
generator=generator(img_shape, z_dim)

#Generated image to be used as input
z = Input(shape=(100,))
img =generator(z)

#Keep Discriminator’s parameters constant during Generator training
discriminator.trainable = False

#The Discriminator’sprediction
prediction = discriminator(img)

#Combined GAN model to train the Generator
combined = Model(z, prediction)
combined.compile(loss='binary_crossentropy',optimizer=Adam())

"""GAN Training function"""

losses=[]
```

```
accuracies=[]

deftrain(iterations,batch_size, sample_interval):

    #Load the dataset
    #path = "C:\\Users\\hgani\\Desktop\\CARLIFE
    DATA\\carlife_all_data\\Carlife_jpg"#X_train = cv2.imread(path + '\\'+ str(i) for i in
    os.listdir(path))
    (X_train,_),(_,_)=mnist.load_data()
    data_slice = 3000
    X_train=X_train[:data_slice,:] #
    Rescale -1 to 1
    X_train= X_train/ 127.5-1.
    X_train=np.expand_dims(X_train, axis=3)

    #Labels for real and fake examples
    real = np.ones((batch_size, 1))
    fake=np.zeros((batch_size,1))

    for iteration in range(iterations):
        #Select a random batch of real images
        idx=np.random.randint(0,X_train.shape[0],batch_size)
        imgs = X_train[idx]

        z=np.random.normal(0, 1,(batch_size, 100))

        gen_imgs=generator.predict(z)
        d_loss_real = discriminator.train_on_batch(imgs, real)
        d_loss_fake=discriminator.train_on_batch(gen_imgs,fake)
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
```

```
            z=np.random.normal(0,1,(batch_size,100))
    gen_imgs = generator.predict(z)

      g_loss=combined.train_on_batch(z,real)

      ifiteration % sample_interval==0:
        print("%d[Dloss:%f,acc.:%.2f%%][Gloss:%f]"% (iteration,
                    d_loss[0], 100*d_loss[1], g_loss))
        losses.append((d_loss[0], g_loss))
        accuracies.append(100*d_loss[1])
        sample_images(iteration)


def sample_images(iteration,image_grid_rows=4,image_grid_columns=4):
   z=np.random.normal(0, 1,
          (image_grid_rows*image_grid_columns,z_dim))
   gen_imgs = generator.predict(z)
   gen_imgs=0.5*gen_imgs+0.5

   fig,axs=plt.subplots(image_grid_rows,image_grid_columns,
                              figsize=(4,4),sharey=True,sharex=True)

   cnt= 0
   for i in range(image_grid_rows):
      for j in range(image_grid_columns):
        # Output image grid
        axs[i,j].imshow(gen_imgs[cnt,:,:,0],cmap='gray')
        axs[i,j].axis('off')
        cnt+=1

   import warnings ; warnings.simplefilter('ignore')


   iterations=20000
   batch_size=128
   sample_interval=1000
   train(iterations, batch_size, sample_interval)
```

OUTPUT:

```
0[Dloss:0.002651,acc.:100.00%][Gloss:6.328685]
1000[Dloss:0.020505,acc.:100.00%][Gloss:4.181127]
2000 [D loss: 0.077820, acc.: 96.88%] [G loss: 5.661728]
3000 [D loss: 0.197608, acc.: 92.97%] [G loss: 6.843472]
4000 [D loss: 0.136775, acc.: 94.92%] [G loss: 5.205476]
5000 [D loss: 0.461468, acc.: 78.52%] [G loss: 2.651729]
6000 [D loss: 0.119333, acc.: 95.31%] [G loss: 4.650574]
7000 [D loss: 0.349625, acc.: 84.77%] [G loss: 3.938936]
8000 [D loss: 0.478075, acc.: 80.47%] [G loss: 3.259602]
9000 [D loss: 0.324833, acc.: 86.33%] [G loss: 4.168482]
10000 [D loss: 0.217681, acc.: 91.80%] [G loss: 3.417312]
11000 [D loss: 0.411692, acc.: 83.20%] [G loss: 3.185366]
12000 [D loss: 0.282608, acc.: 87.11%] [G loss: 3.108163]
13000 [D loss: 0.283514, acc.: 86.72%] [G loss: 3.717927]
14000 [D loss: 0.341416, acc.: 84.77%] [G loss: 3.769761]
15000 [D loss: 0.382134, acc.: 81.64%] [G loss: 3.191629]
16000 [D loss: 0.329370, acc.: 85.94%] [G loss: 3.672557]
17000 [D loss: 0.251924, acc.: 88.67%] [G loss: 3.597138]
18000 [D loss: 0.247567, acc.: 90.62%] [G loss: 3.474238]
19000 [D loss: 0.336493, acc.: 84.38%] [G loss: 3.227752]
```



RESULT:

The result of running this code is the training of a GAN model to generate synthetic handwritten digit images similar to those in the MNIST dataset. The quality of generated images improves as training progresses, and you can visualize the generated images in the sample image grid displayed during training. The aim is to train the generator to produce images that are indistinguishable from real MNIST digits.