AD3511-Deep Learning Laboratory

1. XOR problem

AIM:

To write a python program to solve XOR problem using DNN.

ALGORITHM:

Step1:Import required modules for performing X-OR Algorithm.

Step2:Define Sigmoid Activation function and sigmoid derivative for backpropagation.

Step3:Define forward function with bias and backpropagation function.

Step4:Obtain values of variables X&Y using numpy module array function.

Step5:Initialize weights,learning rate,epochs.

Step6:Start training with forward and backpropagation.

Step7:Add cost to list for plotting.

Step8:Make predictions.

Step9:Plot cost.

PROGRAM:

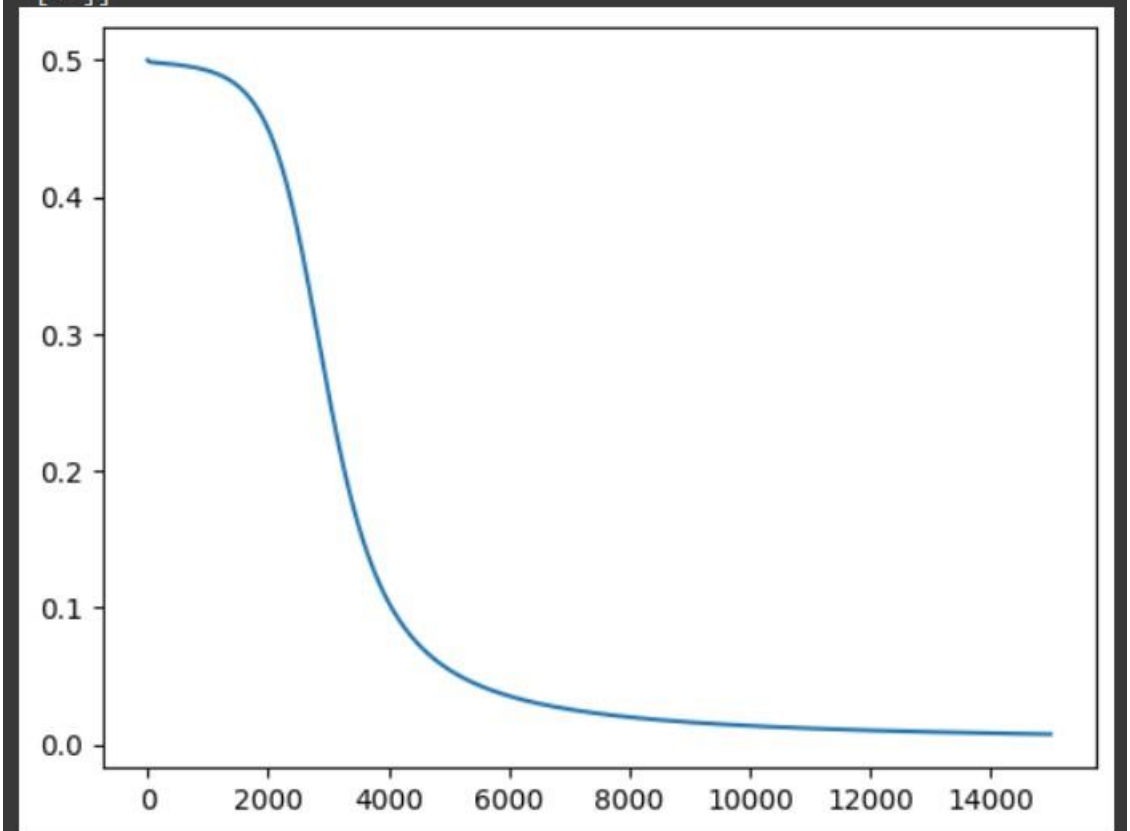
```
import numpy as np
import matplotlib.pyplot as plt
def sigmoid(X):
  return 1/(1+np.exp(-X))
```

```
def sigmoid_deriv(X):
  return sigmoid(X)*(1-sigmoid(X))
def forward(X,w1,w2,predict=False):
a1=np.matmul(X,w1)
  z1=sigmoid(a1)
  bias=np.ones((len(z1),1))
  z1=np.concatenate((bias,z1),axis=1)
  a2=np.matmul(z1,w2)
  z2=sigmoid(a2)
  if predict:
    return z2
  return a1,z1,a2,z2
def backprop(a1,z0,z1,z2,Y):
  delta2=z2-Y
  Delta2=np.matmul(z1.T,delta2)
  delta1=(delta2.dot(w2[1:,:].T))*sigmoid_deriv(a1)
  Delta1=np.matmul(z0.T,delta1)
  return delta2,Delta1,Delta2
X=np.array([[1,1,0],
        [1,0,1],
        [1,0,0],
        [1,1,1]])
Y=np.array([[1],[1],[0],[0]])
w1=np.random.randn(3,5)
```

```
w2=np.random.randn(6,1)
lr=0.09
costs=[]
epochs=15000
m=len(X)
for i in range(epochs):
  a1,z1,a2,z2=forward(X,w1,w2)
  delta2,Delta1,Delta2=backprop(a1,X,z1,z2,Y)
  w1-=lr*(1/m)*Delta1
  w2-=lr*(1/m)*Delta2
  c=np.mean(np.abs(delta2))
  costs.append(c)
  if i %100==0:
    print(f"Iteration: {i}. Error: {c}")
print("Training Complete")
z3=forward(X,w1,w2,True)
print("Percentages:")
print(z3)
print("Predictions")
print(np.round(z3))
plt.plot(costs)
plt.show()
```

OUTPUT:

```
Training Complete
Percentages:
[[0.99743633]
 [0.98818835]
 [0.00507463]
 [0.01042947]]
Predictions
[[1.]
 [1.]
 [0.]
 [0.]]
```



RESULT:

Thus python program for XOR problem using DNN is created and executed successfully.

2. Character Recognition

AIM:

To write a python program for Character Recognition using CNN.

ALGORITHM:

Step1:Import required modules for performing character recognition.

Step2:load MNIST dataset.

Step3:Preprocess the data.

Step4:Build the CNN model.

Step5:Compile,train and evaluate the model.

Step6:Make the predictions.

Step7:Visualize the results.

Step8:Plot the results.

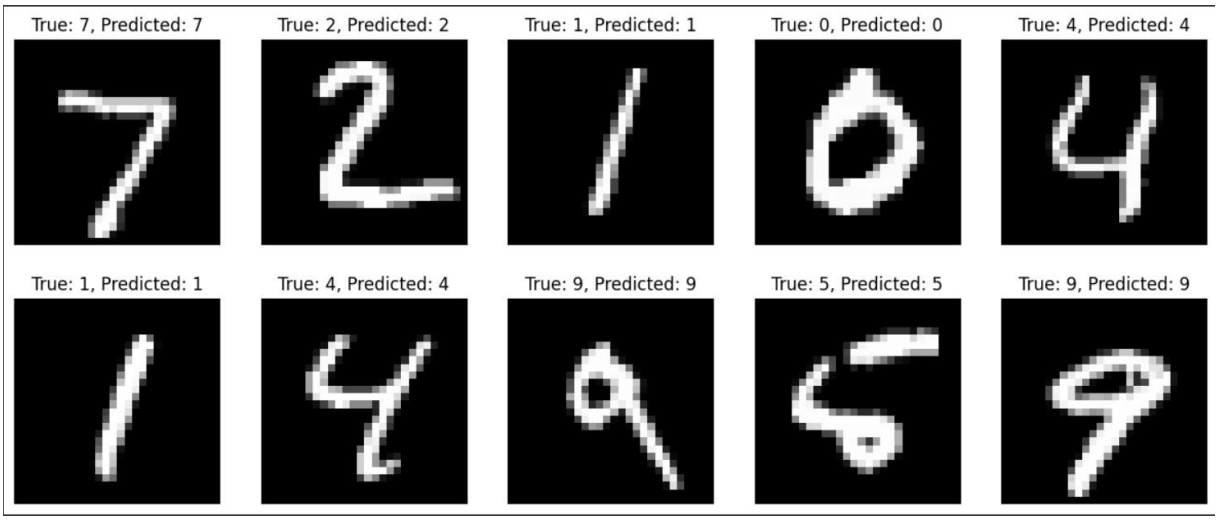
PROGRAM:

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt
mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images,
test_labels) = mnist.load_data()
model = models.Sequential([
layers.Conv2D(32, (3, 3), activation='relu',
```

```
input_shape=(28, 28, 1)),
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.Flatten(),
layers.Dense(64, activation='relu'),
layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])
epochs = 10
history = model.fit(train_images, train_labels,
epochs=epochs, validation_data=(test_images,
test_labels))
test_loss, test_acc = model.evaluate(test_images,
test_labels)
print("Test accuracy:"
, test_acc)
predictions = model.predict(test_images)
plt.figure(figsize=(15, 6))
for i in range(10):
  plt.subplot(2, 5, i+1)
```

```
  plt.imshow(test_images[i].reshape(28, 28),
cmap='gray')
  plt.title(f"True: {test_labels[i]}, Predicted:
{np.argmax(predictions[i])}")
  plt.axis('off')
plt.show()
```

OUTPUT:



RESULT:

Thus python program for character recognition using CNN is created and executed successfully.

3.Face Recognition

AIM:

To write a python program for face recognition using CNN.

ALGORITHM:

Step1:Import required modules for performing face recognition.

Step2:Define path of the face images.

Step3:Augument the data for training.

Step4:Build a CNN model.

Step5:Train and save the model.

Step6:Test the prediction of model.

PROGRAM:

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense
from tensorflow.keras.models import Sequential
train_data_dir = '/content/drive/MyDrive/Traning images'
test_data_dir = '/content/drive/MyDrive/Traning images'
image_size = (128, 128)
batch_size = 16
train_datagen = ImageDataGenerator(
   rescale=1.0/255,
   rotation_range=20,
   width_shift_range=0.2,
   height_shift_range=0.2,
   horizontal_flip=True
)
test_datagen = ImageDataGenerator(rescale=1.0/255)
train_generator = train_datagen.flow_from_directory(
   train_data_dir,
   target_size=image_size,
   batch_size=batch_size,
```

```
    class_mode='categorical'
)
test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical'
)
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(image_size[0],
image_size[1], 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(len(train_generator.class_indices), activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(train_generator, epochs=10,
validation_data=test_generator)
model.save('face_recognition_model.h5')
from tensorflow.keras.preprocessing import image
def predict_image(file_path):
    img = image.load_img(file_path, target_size=image_size)
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0
    prediction = model.predict(img_array)
    predicted_class = np.argmax(prediction)
    class_indices = train_generator.class_indices
    predicted_person = [person for person, index in
class_indices.items() if index == predicted_class][0]
```

```
    print("Predicted person:", predicted_person)
new_image_path = '/content/steve-jobs.jpg'
predict_image(new_image_path)
```

OUTPUT:

```
Predicted person: person1
```

RESULT:

Thus python program for face recognition using CNN is created and executed successfully.

4.Language Modeling

AIM:

To write a python program for language modeling using LSTM.

ALGORITHM:

Step1:Import required modules for performing language modeling.

Step2:Load the dataset.

Step3:Assign the hyperparameters..

Step4:Tokenize the text data.

Step5:Build RNN model.

Step6:Generate the text.

PROGRAM:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
poetry_data = [
   "Stars twinkling in the night sky",
   "Whispers of the wind, so gentle",
   "Moonlight casting shadows",
   "Dreams taking flight on wings of hope"
]
max_seq_length = 10
embedding_dim = 50
num_epochs = 100
tokenizer = Tokenizer()
tokenizer.fit_on_texts(poetry_data)
total_words = len(tokenizer.word_index) + 1
input_sequences = []
for line in poetry_data:
   token_list = tokenizer.texts_to_sequences([line])[0]
   for i in range(1, len(token_list)):
      n_gram_sequence = token_list[:i+1]
      input_sequences.append(n_gram_sequence)
input_sequences = pad_sequences(input_sequences,
maxlen=max_seq_length, padding='pre')
X, y = input_sequences[:, :-1], input_sequences[:, -1]
```

```
y = tf.keras.utils.to_categorical(y, num_classes=total_words)
model = Sequential([
    Embedding(total_words, embedding_dim,
input_length=max_seq_length - 1),
    LSTM(100),
    Dense(total_words, activation='softmax')
])
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.fit(X, y, epochs=num_epochs, verbose=1)
seed_text = "what is your "
num_words_to_generate = 10
generated_text = seed_text
for _ in range(num_words_to_generate):
    encoded_seed = tokenizer.texts_to_sequences([seed_text])[0]
    padded_seed = pad_sequences([encoded_seed],
maxlen=max_seq_length - 1, padding='pre')
    predicted_token = np.argmax(model.predict(padded_seed), axis=-1)
    predicted_word = tokenizer.index_word[predicted_token[0]]
    seed_text += " " + predicted_word
    generated_text += " " + predicted_word
print(generated_text)
```

OUTPUT:

```
what is your  casting casting shadows shadows the the night sky sky sky
```

RESULT:

Thus python program for Language modeling using RNN is created and executed successfully.

5.Sentiment Analysis

AIM:

To write a python program for sentiment analysis using LSTM.

ALGORITHM:

Step1:Import required modules for performing sentiment analysis.

Step2:Enter the sample data for positive and negative texts.

Step3:Combine positive and negative texts.

Step4:Create labels.

Step5:Tokenize and pad sequence.

Step6:Build LSTM models.

Step7:Train and test the models.

Step8:Display the result.

PROGRAM:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import
pad_sequences
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.models import Sequential

positive_texts = [
```

"I love this product!",

"This is amazing!",

"Great experience overall.",

"I'm really satisfied with this.",

"Absolutely fantastic!",

"Highly recommended.",

"Incredible quality.",

"A wonderful purchase.",

"So pleased with this.",

"Brilliant product!",

"Impressed with the performance.",

"Exactly what I needed.",

"Couldn't be happier with it.",

"Exceeded my expectations.",

"Top-notch product.",

"Thrilled with my purchase.",

"Well worth the price.",

"This is a game-changer!",

"Fantastic value.",

"I'm a fan!",

"Really impressed with the design.",

"This made my day!",

"Superb craftsmanship.",

"This deserves 5 stars.",

"Amazing customer service too.",

"I'm over the moon!",

"High-quality and reliable.",

"This improved my life.",

"I can't thank them enough.",

"Made a huge difference for me.",

"Impressive durability.",

"This is top-of-the-line.",

"So glad I found this.",

"This is worth every penny.",

"Changed the way I do things.",

"Excellent customer experience.",

"I'm recommending this to everyone!",

"This is a must-have.",

"Outstanding in every aspect.",

"This exceeded all my hopes.",

"I can't say enough good things.",

"This is pure gold.",

"This made my life easier.",

"I wish I had bought this sooner.",

"Unbelievably good!",

"This brings me so much joy.",

"Well done!",

"This is a winner.",

"I'm thoroughly impressed.",

"This is a life-saver.",

"This works like a charm.",

"I can't express how much I love this.",

"This deserves a standing ovation.",

"This is the real deal.",

"Can't imagine life without it.",

"This blew me away.",

"No complaints at all.",

"This is perfection.",

"This has a special place in my heart.",

"I'm head over heels for this!",

"I'm extremely happy with this.",

"This is the best investment.",

"This is beyond my dreams.",

"I'm in awe of this product.",

"This is a game-changing innovation.",

"This is worth its weight in gold.",

"This is an absolute gem.",

"This is the epitome of excellence.",

"This fills me with joy.",

"This is my new favorite thing.",

"This is an absolute delight.",

"This is genius.",

"This has made a huge impact on me.",

"This is a stroke of genius.",

"This is nothing short of outstanding.",

"This is pure genius.",

"This gets my highest recommendation.",

"This is a masterpiece.",

"This is the holy grail.",

"This is the answer to my prayers.",

"This is a true blessing.",

"This is a marvel.",

"This is pure magic.",

"This is revolutionary.",

"This is legendary.",

"This is iconic.",

"This is absolutely top-class.",

"This is like a dream come true.",

"This is the pinnacle of perfection.",

"This is pure elegance.",

"This is a source of inspiration.",

"This is worth every accolade.",

"This is poetry in motion.",

"This is a true work of art.",

"This is elegance personified.",

"This is a masterpiece of engineering.",

```
    "This is the ultimate solution.",
    "This is the epitome of sophistication.",
    "This is a treasure.",
    "This is the ultimate achievement.",
    "This is the epitome of luxury.",
    "This is a genuine miracle.",
    "This is the finest example of excellence.",
    "This is the key to happiness.",
    "This is pure satisfaction.",
    "This is true perfection.",
    "This is a triumph.",
    "This is a shining star.",
    "This is the embodiment of quality.",
    "This is the height of innovation.",
]

negative_texts = [
    "This is terrible.",
    "I regret buying this.",
    "Waste of time and money.",
    "Absolutely disappointing.",
    "I'm not impressed at all.",
    "Not worth the hype.",
    "I expected better.",
```

"Total letdown.",

"I'm very unhappy with this.",

"I wouldn't recommend it.",

"This is a huge disappointment.",

"I'm dissatisfied with this purchase.",

"This falls short of expectations.",

"Definitely not worth it.",

"I'm very displeased.",

"Not what I was hoping for.",

"This is a letdown.",

"I'm not satisfied with the quality.",

"This is a waste of money.",

"I'm unimpressed.",

"This is a complete failure.",

"I'm not happy with this.",

"This is a disaster.",

"I'm extremely dissatisfied.",

"This is a flop.",

"This is a disaster.",

"I'm very let down.",

"This is subpar.",

"This is a major disappointment.",

"I'm disappointed with this product.",

"This is a waste of resources.",

"I'm not pleased with this.",

"This is a flop.",

"I'm not content with this.",

"This is not up to par.",

"This is a total failure.",

"I'm not content with this.",

"This is a letdown.",

"I'm not satisfied with this.",

"This is a waste of time.",

"I'm very displeased with this.",

"This is a regrettable purchase.",

"I'm not happy with this at all.",

"This is a major letdown.",

"I'm not impressed by this.",

"This is not what I expected.",

"I'm not content with this.",

"This is not up to the mark.",

"I'm not satisfied at all.",

"This is not worth the price.",

"I'm very disappointed.",

"This is far from satisfactory.",

"I'm not pleased at all.",

"This is not worth it.",

"I'm not happy at all.",

```
    "This is not up to my standards.",
    "I'm not content with this.",
    "This is below my expectations.",
    "I'm not impressed with this.",
    "This is not worth the money.",
    "I'm not satisfied with this purchase.",
    "This is not what I was looking for.",
    "I'm very unsatisfied.",
    "This is not up to scratch.",
    "I'm not content with this.",
    "This is below par.",
    "I'm not happy with this at all.",
    "This is not worth the investment.",
    "I'm not impressed at all.",
    "This is not up to standard.",
    "I'm not satisfied with this.",
    "This is disappointing.",
    "I'm very displeased.",
    "This is not worth the cost."
]

texts = positive_texts + negative_texts
labels = [1] * len(positive_texts) + [0] * len(negative_texts)
max_words = 1000
```

```
max_sequence_length = 20

tokenizer = Tokenizer(num_words=max_words)

tokenizer.fit_on_texts(texts)

sequences = tokenizer.texts_to_sequences(texts)

padded_sequences = pad_sequences(sequences,
maxlen=max_sequence_length)
model = Sequential([

   Embedding(input_dim=max_words, output_dim=128,
input_length=max_sequence_length),

   LSTM(64),

   Dense(1, activation='sigmoid')

])

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

labels = np.array(labels)

model.fit(padded_sequences, labels, epochs=10, batch_size=2)

test_texts = [

   "Awesome!",

   "Not worth it.",

   "terrible"

]

test_sequences = tokenizer.texts_to_sequences(test_texts)
```

```
padded_test_sequences = pad_sequences(test_sequences, maxlen=max_sequence_length)

predictions = model.predict(padded_test_sequences)

for text, prediction in zip(test_texts, predictions):
    sentiment = "positive" if prediction > 0.5 else "negative"
    print(f"Text: {text}\nPredicted sentiment: {sentiment}\n")
```

OUTPUT:

```
Text: Awesome!
Predicted sentiment: positive

Text: Not worth it.
Predicted sentiment: negative

Text: terrible
Predicted sentiment: negative
```

RESULT:

Thus python program for sentiment analysis using LSTM is created and executed successfully.

6.Parts of speech tagging using Sequence to sequence architecture

AIM:

To write a python program for Parts of speech tagging using Sequence to sequence architecture.

ALGORITHM:

Step1:Import required modules for performing parts of speech tagging.

Step2:Enter sample label data.

Step3:Tokenize the sentences.

Step4:Convert sentences into sentences of word indices.

Step5:Pad sequences for uniform input size.

Step6:Create label dictionary.

Step7:Convert label sentences to one-hot encoded vectors.

Step8:Build and train the LSTM model.

Step9:Convert predicted label probabilities to actual labels.

Step10:Display the result.

PROGRAM:

```
import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
sentences = [
    "I like to play soccer",
    "She sings beautifully",
    "He runs every morning",
    "They read books in the library",
    "The cat sleeps on the windowsill",
    "We eat dinner together",
    "The flowers bloom in spring",
    "She paints colorful pictures",
    "He writes poems",
    "The dog barks loudly",
    "Birds chirp in the trees",
    "The river flows peacefully",
    "The sun sets in the evening",
    "Stars twinkle in the night sky",
    "Children play in the park",
    "The wind rustles the leaves",
    "The rain falls gently",
    "The ocean waves crash",
    "People walk on the sidewalk",
```

"The fire crackles and burns",

"The clock ticks",

"The train rumbles by",

"The car honks its horn",

"The baby giggles",

"The teacher explains the lesson",

"The student asks a question",

"The doctor examines the patient",

"The chef prepares the meal",

"The gardener waters the plants",

"The mechanic fixes the car",

"The dancer performs on stage",

"The actor delivers lines",

"The musician plays the guitar",

"The artist sketches a portrait",

"The scientist conducts experiments",

"The programmer writes code",

"The writer creates stories",

"The photographer captures moments",

"The journalist reports the news",

"The detective solves the case",

"The firefighter extinguishes the flames",

"The police officer patrols the streets",

"The astronaut floats in space",

```
    "The farmer tends to the crops",
    "The baker kneads the dough",
    "The waiter serves the food",
    "The cashier handles transactions",
    "The lifeguard watches the pool",
    "The lifeguard watches the pool",
    "The lifeguard watches the pool",
]
labels = [
    "PRON VERB PART VERB NOUN",
    "PRON VERB ADV",
    "PRON VERB DET NOUN",
    "PRON VERB NOUN ADP DET NOUN",
    "DET NOUN VERB ADP DET NOUN",
    "PRON VERB NOUN ADV",
    "DET NOUN VERB ADP NOUN",
    "PRON VERB ADJ NOUN",
    "PRON VERB NOUN",
    "DET NOUN VERB ADV",
    "NOUN VERB ADP DET NOUN",
    "DET NOUN VERB ADV",
    "DET NOUN VERB ADP DET NOUN",
    "NOUN VERB ADP DET NOUN NOUN",
    "NOUN VERB ADP DET NOUN",
```

"DET NOUN VERB ADP DET NOUN",

"DET NOUN VERB ADV",

"DET NOUN VERB",

"NOUN VERB ADP DET NOUN",

"DET NOUN VERB NOUN CONJ VERB",

"DET NOUN VERB",

"DET NOUN VERB ADV",

"DET NOUN VERB ADP DET NOUN",

"DET NOUN VERB ADP DET NOUN",

"DET NOUN VERB",

"DET NOUN VERB NOUN",

"DET NOUN VERB ADP DET NOUN",

"DET NOUN VERB",

"DET NOUN VERB ADP NOUN",

"DET NOUN VERB ADP NOUN",

"DET NOUN VERB ADP DET NOUN",

"DET NOUN VERB NOUN",

"DET NOUN VERB NOUN",

"DET NOUN VERB ADP DET NOUN",

"DET NOUN VERB",

"DET NOUN VERB NOUN",

"DET NOUN VERB",

"DET NOUN VERB ADP NOUN",

"DET NOUN VERB NOUN",

```
    "DET NOUN VERB NOUN",
    "DET NOUN VERB",
    "DET NOUN VERB",
    "DET NOUN VERB",
    "DET NOUN VERB",
    "DET NOUN VERB",
    "DET NOUN VERB",
    "DET NOUN VERB",
    "DET NOUN VERB",
    "DET NOUN VERB",
    "DET NOUN VERB"
]
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences)
vocab_size = len(tokenizer.word_index) + 1
sequences = tokenizer.texts_to_sequences(sentences)
max_sequence_length = max(len(seq) for seq in sequences)
padded_sequences = pad_sequences(sequences,
maxlen=max_sequence_length, padding='post')
label_dict = {}
for idx, label_seq in enumerate(labels):
    label_dict[idx] = label_seq.split()
num_labels = max(len(label_dict[idx]) for idx in label_dict)
```

```
label_vectors = np.zeros((len(label_dict), max_sequence_length, num_labels), dtype='float32')
for idx, label_seq in label_dict.items():
    for word_idx, label in enumerate(label_seq):
        label_vectors[idx, word_idx, label_dict[idx].index(label)] = 1
model = Sequential()
model.add(Embedding(vocab_size, 50, input_length=max_sequence_length))
model.add(LSTM(100, return_sequences=True))
model.add(Dense(num_labels, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(padded_sequences, label_vectors, epochs=100, verbose=1)
test_sentence = "The clock ticks"
test_sequence = tokenizer.texts_to_sequences([test_sentence])
padded_test_sequence = pad_sequences(test_sequence, maxlen=max_sequence_length, padding='post')
predicted_labels = model.predict(padded_test_sequence)[0]
predicted_label_seq = [label_dict[0][np.argmax(label_probs)] for label_probs in predicted_labels]
print(predicted_label_seq)
```

OUTPUT:

```
['PRON', 'VERB', 'PART', 'VERB', 'VERB', 'VERB']
```

RESULT:

Thus python program for Parts of speech tagging using sequence to sequence architecture is created and executed successfully.

7.Machine translation

AIM:

To write a python program for machine translation using encoder-decoder model.

ALGORITHM:

Step1:Import required modules for machine translation.

Step2:Build a model.

Step3:Tokenize the model.

Step4: Enter the code for translation.

Step5: Use encoder -decoder model.

Step6:Translate using generate function.

PROGRAM:

```
pip install transformers -U -q

pip install sentencepiece

pip freeze | grep transformers

from transformers import MBartForConditionalGeneration, MBart50TokenizerFast

model = MBartForConditionalGeneration.from_pretrained("facebook/mbart-large-50-one-to-many-mmt")

tokenizer = MBart50TokenizerFast.from_pretrained("facebook/mbart-large-50-one-to-many-mmt", src_lang="en_XX")

article_en = "what is your name"

model_inputs = tokenizer(article_en, return_tensors="pt")
```

```
generated_tokens = model.generate(
    **model_inputs,
    forced_bos_token_id=tokenizer.lang_code_to_id["ta_IN"]
)
translation = tokenizer.batch_decode(generated_tokens, skip_special_tokens=True)
translation
```

OUTPUT:

```
['உங்கள் பெயர் என்ன?']
```

RESULT:

Thus python program for machine translation suing encoder and decoder model is created and executed successfully.

8.Image Augmentation

AIM:

To write a python program for Image augmentation using GANs model.

ALGORITHM:

Step1:Import required modules for machine translation.

Step2:Load and display the image.

Step3:Convert the image to NumPy array.

Step4: Create an ImageDataGenerator instance with augmentation options.

Step5: Reshape the image array to match the input shape expected by the generator.

Step6: Generate augmented images and display a few of them.

Step7: Save augmented images.

PROGRAM:

Save augmented imagesimport numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array

img = load_img("/content/Screenshot_20230205_102452.png")

plt.imshow(img)

plt.axis('off')

plt.show()

img_array = img_to_array(img)

```
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
img_array = img_array.reshape((1,) + img_array.shape)
augmented_images = datagen.flow(img_array, batch_size=1)
plt.figure(figsize=(10, 10))
for i in range(9):
    augmented_image =
augmented_images.next()[0].astype('uint8')
    plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_image)
    plt.axis('off')
plt.show()
for i in range(9):
    augmented_image =
augmented_images.next()[0].astype('uint8')
    augmented_img_path = f'augmented_image_{i}.jpg'
    plt.imsave(augmented_img_path, augmented_image)
```

OUTPUT:

RESULT:

Thus python program for Image augmentation using GANs model is created and executed successfully.