NITTE
EDUCATION TRUST

NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY
An Autonomous Institution Approved by UGC/AICTE/Govt. of Karnataka
Accredited by NBA (Tier-I) and NAAC 'A+' Grade
Affiliated to Visvesvaraya Technological University, Belagavi
Post Box No. 6429, Yelahanka, Bengaluru-560064, Karnataka, INDIA

KNOWLEDGE • CHARACTER • UNITY

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

21ADL74 NATURAL LANGUAGE PROCESSING LABORATORY

NAME :____________________________________

REGISTER NO. : ____________________________________

YEAR/SEM :____________________________________

SECTION :____________________________________

BRANCH : ____________________________________

RECORD NOTE BOOK

REGNO. . ____________________

Certified that this is a bonafide observation of Practical work done by Mr/Ms/Mrs…………………………………of the …………………………………

Semester………………………………………………. .... Branch during the Academic year……………in the ....laboratory.

Staff–in–Charge **Head of the Department**

Internal Examiner **External Examine**

Internal Evaluation

Date :

SL.NO	Particulars	Max Marks	Obtained Marks
1.	Programs execution during Regular Lab	10	
2.	Record Book	10	
3.	Execution of Program(part A and part B)	25	
4.	Viva	05	
	Total	50	

Total Marks secured : ____________

Signature of Faculty

LIST OF EXPERIMENTS

S.No	NAME OF THE PROGRAMS
1.	Write a Python program to perform Part of Speech Tagging with Stop words using NLTK.
2.	Write a Python program to find Term Frequency and Inverse Document Frequency (TF-IDF).
3.	Implement N-gram Language model using Python
4.	Implement word embedding using Word2Vec/Glove/fast using Python
5.	Implement text processing with LSTM. (Use the model to predict the next words in a sequence.)
6.	Build a small-scale GPT model for text generation by creating a transformer-based model using PyTorch or TensorFlow, training it on a small dataset, and evaluating the generated text for coherence.
7.	Implement self-attention from scratch by creating a small sequence dataset, computing attention weights and outputs manually, and visualizing the attention map.
8.	Build a Recurrent Neural Network (RNN) for binary sentiment classification by preprocessing a text dataset (e.g., IMDB reviews), training an RNN, and evaluating the model's accuracy to understand its limitations, such as vanishing gradients.

PROGRAM -01

Write a Python program to perform Part of Speech Tagging with Stop words using NLTK.

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

text="Natural Language Processing is a fascinating field of AI."
tokens= word_tokenize(text)
stop_words=set(stopwords.words('english'))
stop_words_tokens=[word for word in tokens if word.lower() in stop_words]
pos_tags =nltk.pos_tag(stop_words_tokens)
print("POS Tags of stop Words:", pos_tags)
```

OUTPUT:

```
POS Tags of stop Words: [('is', 'VBZ'), ('a', 'DT'), ('of', 'IN')]
```

PROGRAM -02

Write a Python program to find Term Frequency and Inverse Document Frequency (TF-IDF).

```
from sklearn.feature_extraction.text import TfidfVectorizer
docs= ["Natural Language processing is amazing.",
      "Machine learning and NLP go hand in hand.",
      "TF-IDF helps find important words in a document."]
vec= TfidfVectorizer()
tfidf_mat= vec.fit_transform(docs)
print("Feature Names:", vec.get_feature_names_out())
print("TF-IDF Matrix\n", tfidf_mat.toarray())
```

OUTPUT:

```
Feature Names: ['amazing' 'and' 'document' 'find' 'go' 'hand' 'helps' 'idf' 'important'
 'in' 'is' 'language' 'learning' 'machine' 'natural' 'nlp' 'processing'
 'tf' 'words']
TF-IDF Matrix
 [[0.4472136  0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.4472136  0.4472136
  0.         0.         0.4472136  0.         0.4472136  0.
  0.        ]
 [0.         0.32311233 0.         0.         0.32311233 0.64622465
  0.         0.         0.         0.24573525 0.         0.
  0.32311233 0.32311233 0.         0.32311233 0.         0.
  0.        ]
 [0.         0.         0.36325471 0.36325471 0.         0.
  0.36325471 0.36325471 0.36325471 0.27626457 0.         0.
  0.         0.         0.         0.         0.         0.36325471
  0.36325471]]
```

PROGRAM -03

Implement N-gram Language model using Python

```
from nltk import ngrams
from collections import Counter, defaultdict
txt= "Natural languga eprocessing s fun and challenging."
N=2
token= txt.lower().split()
n_grams=list(ngrams(token,N))
model=defaultdict(lambda:0)
counts=Counter(n_grams)
for ngram, count in counts.items():
   prefix=ngram[:-1]
   total_prefix_counts=sum(c for ng,c in counts.items() if ng[:-1] == prefix)
   model[ngram] =count/total_prefix_counts
print("N-gram Probabilities:", dict(model))
```

OUTPUT:

```
N-gram Probabilities: {('natural', 'languga'): 1.0,
('languga', 'eprocessing'): 1.0, ('eprocessing', 's'): 1.0,
('s', 'fun'): 1.0, ('fun', 'and'): 1.0, ('and',
'challenging.'): 1.0}
```

PROGRAM -04

Implement word embedding using Word2Vec/Glove/fast using Python

```
from gensim.models import Word2Vec
sent=[["natural","language","processing","is","fun"],
    ["machine","learning","and","nlp","go","together"]]
model= Word2Vec(sent, vector_size=50,window=3,min_count=1,sg=1)
vector=model.wv["language"]
print("word Vector for 'language':", vector)
```

OUTPUT:

```
word Vector for 'language': [ 0.00805391  0.00869487  0.01991474 -0.00894748 -0.00277853 -0.01463464
 -0.01939566 -0.01816051 -0.00204551 -0.01300658  0.00969946 -0.01232805
  0.00503837  0.00147888 -0.00678431 -0.00195845  0.01995825  0.01829177
 -0.00892366  0.01816605 -0.01128353  0.01186184 -0.00619444  0.0068635
  0.00603445  0.01380092 -0.00474777  0.01755007  0.01517886 -0.01909529
 -0.01601642 -0.01527579  0.00584651 -0.00558944 -0.01385904 -0.01625653
  0.01661836  0.00398098 -0.01865603 -0.00958543  0.00627348 -0.00942641
  0.01056169 -0.00846688  0.00528359 -0.01609137  0.01241977  0.00963778
  0.00157439  0.0060269 ]
```

PROGRAM -05

Implement text processing with LSTM. (Use the model to predict the next words in a sequence.)

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

# Sample text corpus
data = "The quick brown fox jumps over the lazy dog. The quick brown fox is fast and clever."

# Parameters
sequence_length = 3  # Number of words in the input sequence
num_words_to_predict = 3  # Number of words to predict

def preprocess_text(data):
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts([data])
    word_index = tokenizer.word_index
    sequences = []
    for line in data.split('. '):
        token_list = tokenizer.texts_to_sequences([line])[0]
        for i in range(1, len(token_list)):
            n_gram_sequence = token_list[:i + 1]
            sequences.append(n_gram_sequence)
    max_seq_length = max([len(seq) for seq in sequences])
    sequences = pad_sequences(sequences, maxlen=max_seq_length, padding='pre')
    return sequences, tokenizer, word_index, max_seq_length

# Preprocess the data
sequences, tokenizer, word_index, max_seq_length = preprocess_text(data)

# Prepare input and output
X, y = sequences[:, :-1], sequences[:, -1]
y = np.array(y)

# Vocabulary size
vocab_size = len(word_index) + 1

# Build the LSTM model
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=50, input_length=max_seq_length - 1),
    LSTM(100, return_sequences=False),
    Dense(vocab_size, activation='softmax')
])

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

```
# One-hot encode the output
y = np.eye(vocab_size)[y]

# Train the model
model.fit(X, y, epochs=50, verbose=1)

# Function to predict next words
def predict_next_words(seed_text, num_words, tokenizer, model, max_seq_length):
    for _ in range(num_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=max_seq_length - 1, padding='pre')
        predicted = np.argmax(model.predict(token_list), axis=-1)
        output_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                break
        seed_text += " " + output_word
    return seed_text

# Predict the next words
seed_text = "The quick brown"
print("Original text:", seed_text)
next_words = predict_next_words(seed_text, num_words_to_predict, tokenizer, model,
max_seq_length)
print("Predicted text:", next_words)
```

OUTPUT:

```
Original text: The quick brown
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 169ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
Predicted text: The quick brown fox jumps over
```

PROGRAM -06

Build a small-scale GPT model for text generation by creating a transformer-based model using PyTorch or TensorFlow, training it on a small dataset, and evaluating the generated text for coherence.

```
import torch
from torch.utils.data import Dataset, DataLoader
from transformers import GPT2LMHeadModel, GPT2Tokenizer

EPOCHS = 15
BATCH_SIZE = 8
LEARNING_RATE = 5e-6
MAX_LENGTH = 75

class TextDataset(Dataset):
    def __init__(self, text, tokenizer, max_length):
        self.input_ids = []
        self.attn_masks = []
        for line in text:
            encodings_dict = tokenizer(line, truncation=True, max_length=max_length,
padding="max_length")
            self.input_ids.append(torch.tensor(encodings_dict['input_ids']))
            self.attn_masks.append(torch.tensor(encodings_dict['attention_mask']))

    def __len__(self):
        return len(self.input_ids)

    def __getitem__(self, idx):
        return self.input_ids[idx], self.attn_masks[idx]

tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
if tokenizer.pad_token is None:
    tokenizer.add_special_tokens({'pad_token': '[PAD]'})

model = GPT2LMHeadModel.from_pretrained('gpt2')
model.resize_token_embeddings(len(tokenizer))

text_data = [
    "The quick brown fox jumps over the lazy dog.",
    "The sun sets in the west and rises in the east.",
    "Artificial Intelligence is transforming the world.",
    "Deep learning models are revolutionizing various industries.",
    "Natural Language Processing is a key area of artificial intelligence.",
    "Machine learning models are data-driven and improve over time.",
    "The future of technology lies in autonomous systems and robotics.",
    "Cloud computing has become the backbone of modern infrastructure."
]
dataset = TextDataset(text_data, tokenizer, max_length=MAX_LENGTH)
dataloader = DataLoader(dataset, batch_size=BATCH_SIZE, shuffle=True)
```

```
optimizer = torch.optim.AdamW(model.parameters(), lr=LEARNING_RATE)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)
model.train()

for epoch in range(EPOCHS):
    for batch in dataloader:
        input_ids, attn_masks = [x.to(device) for x in batch]
        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask=attn_masks, labels=input_ids)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
    print(f"Epoch {epoch + 1}/{EPOCHS}, Loss: {loss.item()}")

model.eval()
prompt = "Artificial Intelligence"
encoded_input = tokenizer(prompt, return_tensors='pt', padding=True).to(device)
generated_ids = model.generate(encoded_input['input_ids'], max_length=MAX_LENGTH,
num_return_sequences=1, pad_token_id=tokenizer.pad_token_id)
generated_text = tokenizer.decode(generated_ids[0], skip_special_tokens=True)

print("Generated Text:\n", generated_text)
```

OUTPUT:

```
Generated Text:
 Artificial Intelligence (AI)

The AI is a new type of artificial intelligence that is being developed
by the University of California, Berkeley. The AI is designed to be able
to perform tasks that are difficult to perform in real life.

The AI is designed to be able to perform tasks that are
difficult to perform in real life. The AI is designed to be
```

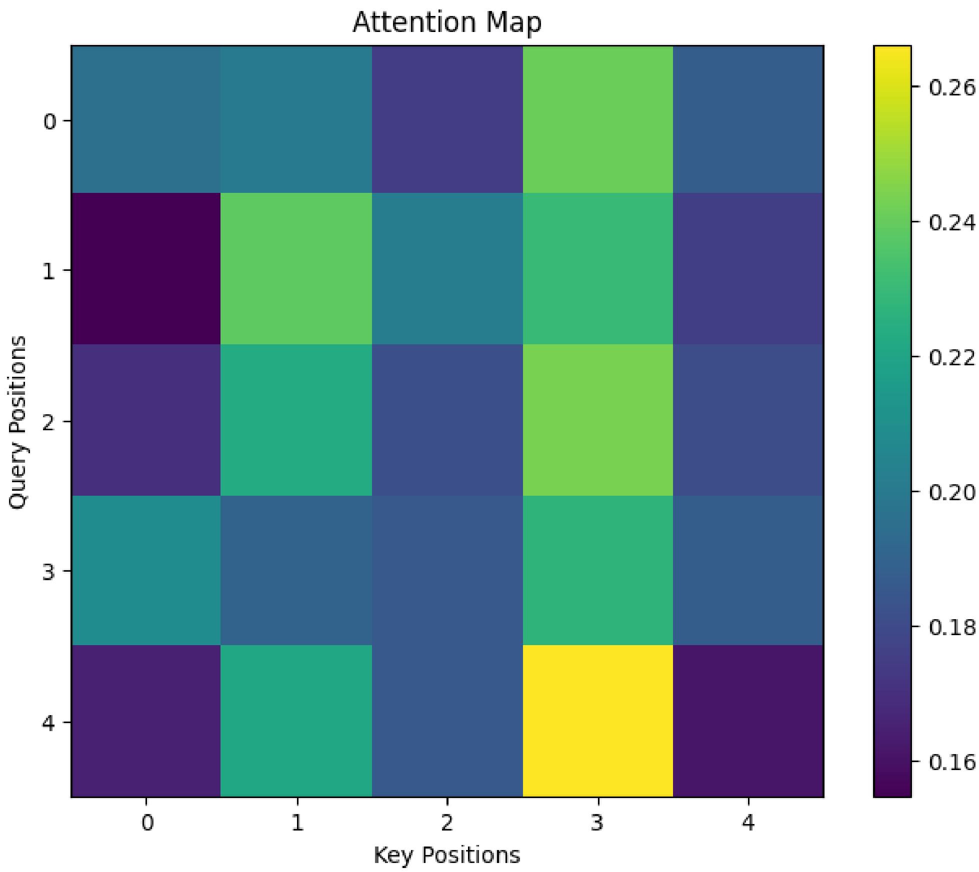
PROGRAM -07

Implement self-attention from scratch by creating a small sequence dataset, computing attention weights and outputs manually, and visualizing the attention map.

```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn

# Parameters
SEQ_LEN = 5
D_MODEL = 4

# Sample Sequence Dataset
sequence = torch.tensor([[1.0, 0.0, 1.0, 0.0],
                         [0.0, 2.0, 0.0, 1.0],
                         [1.0, 1.0, 1.0, 1.0],
                         [0.0, 0.0, 2.0, 1.0],
                         [1.0, 2.0, 0.0, 0.0]])

# Self-Attention Components
class SelfAttention(nn.Module):
    def __init__(self, d_model):
        super(SelfAttention, self).__init__()
        self.query = nn.Linear(d_model, d_model)
        self.key = nn.Linear(d_model, d_model)
        self.value = nn.Linear(d_model, d_model)

    def forward(self, x):
        Q = self.query(x)
        K = self.key(x)
        V = self.value(x)

        attention_scores = torch.matmul(Q, K.transpose(-2, -1)) / np.sqrt(K.size(-1))
        attention_weights = torch.softmax(attention_scores, dim=-1)
        attention_output = torch.matmul(attention_weights, V)
        return attention_output, attention_weights

# Initialize Self-Attention
self_attention = SelfAttention(D_MODEL)

# Compute Attention Outputs and Weights
attention_output, attention_weights = self_attention(sequence)

# Visualize Attention Map
plt.figure(figsize=(8, 6))
plt.imshow(attention_weights.detach().numpy(), cmap="viridis")
plt.colorbar()
plt.title("Attention Map")
```

```
plt.xlabel("Key Positions")
plt.ylabel("Query Positions")
plt.show()
```

OUTPUT:



PROGRAM -08

Build a Recurrent Neural Network (RNN) for binary sentiment classification by preprocessing a text dataset (e.g., IMDB reviews), training an RNN, and evaluating the model's accuracy to understand its limitations, such as vanishing gradients.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Embedding, Dense, SpatialDropout1D
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.text import Tokenizer

# Load the IMDB dataset (pre-split into training and testing sets)
max_features = 20000  # Vocabulary size
max_len = 100  # Max length of each input sequence

# Load the dataset
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# Pad the sequences to have the same length
x_train = pad_sequences(x_train, maxlen=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)

# Build the LSTM model
model = Sequential()
model.add(Embedding(input_dim=max_features, output_dim=128, input_length=max_len))
model.add(SpatialDropout1D(0.2))  # Dropout to prevent overfitting
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))  # LSTM layer
model.add(Dense(1, activation='sigmoid'))  # Sigmoid for binary classification (positive or negative
sentiment)

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

# Train the model
model.fit(x_train, y_train, epochs=3, batch_size=64, validation_data=(x_test, y_test))

# Evaluate the model
score, accuracy = model.evaluate(x_test, y_test, batch_size=64)
print(f"Test accuracy: {accuracy:.4f}")
```

OUTPUT:

```
Epoch 1/3
391/391 ━━━━━━━━━━━━━━━━━━━━ 132s 319ms/step - accuracy: 0.6960 - loss: 0.5540 - val_accuracy: 0.8447 - val_loss: 0.3575
Epoch 2/3
391/391 ━━━━━━━━━━━━━━━━━━━━ 128s 285ms/step - accuracy: 0.8704 - loss: 0.3143 - val_accuracy: 0.8394 - val_loss: 0.3770
Epoch 3/3
391/391 ━━━━━━━━━━━━━━━━━━━━ 142s 285ms/step - accuracy: 0.9141 - loss: 0.2265 - val_accuracy: 0.8389 - val_loss: 0.3895
391/391 ━━━━━━━━━━━━━━━━━━━━ 27s 69ms/step - accuracy: 0.8367 - loss: 0.3923
Test accuracy: 0.8389
```