Semester	B.E. Semester VII
Subject	Deep Learning
Subject Professor In-charge	Dr. Nayana Mahajan
Laboratory	M201B

Student Name	Harsh Jain	Division	B
Roll Number	22108B0054	Batch	4
Grade and Subject Teacher's Signature			

Experiment Number	8
Experiment Title	Create an RNN model that can classify the sentiment of tweets in real-time.
Resources / Apparatus Required	Software:

Algorithm	**Import libraries** • **Install and import necessary packages (pandas, numpy, tensorflow, sklearn).** • **These handle data loading, preprocessing, and model training.** **Load dataset** • **Read the CSV file into a DataFrame (df).** • **Inspect the columns → ['textID','text','selected_text','sentiment'].** **Select relevant columns** • **Use only text (input) and sentiment (label).** • **Drop unused columns like textID and selected_text.** **Label encoding** • **Convert sentiment classes (positive, negative, neutral) into numeric IDs using a dictionary.** • **Map them back for interpretation later.** **Text preprocessing** • **Convert all text to lowercase.** • **Tokenize the words using Tokenizer (Keras).** • **Convert sentences into integer sequences.** • **Pad sequences to ensure fixed length.** **Train/test split**

	• **Split dataset into training and testing sets (e.g., 80/20) using train_test_split.** **Define RNN model** • **Input: Embedding layer (turns word IDs into dense vectors).** • **Hidden: LSTM/GRU layer to capture sequence dependencies.** • **Dense layer with softmax activation for classification into 3 classes.** **Compile model** • **Loss: categorical crossentropy.** • **Optimizer: Adam.** • **Metric: accuracy.** **Train model** • **Fit the model on training data.** • **Validate with test data.** • **Track accuracy and loss.** **Evaluate model** • **Run evaluation on test set to get final accuracy.** • **Optionally, plot training curves.** **Real-time prediction** • **Take a new tweet as input.** • **Preprocess (tokenize + pad) it the same way as training data.**

	• **Run through the trained RNN model.** • **Output predicted sentiment label**

Program code

```
# ==========================
# STEP 1: Setup
# ==========================
!pip install pandas scikit-learn

import pandas as pd
import re
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from collections import Counter

DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", DEVICE)

# ==========================
# STEP 2: Load dataset
# ==========================
# Make sure your Tweets.csv is uploaded in /content
df = pd.read_csv("/content/Tweets.csv")

print(df.head())
print(df.columns)

# Use "text" for input and "sentiment" for label
TEXT_COL = "text"
LABEL_COL = "sentiment"

# Convert string labels to integers
label2id = {l:i for i,l in enumerate(df[LABEL_COL].unique())}
id2label = {i:l for l,i in label2id.items()}
df["label"] = df[LABEL_COL].map(label2id)

print("Labels mapping:", label2id)

# ==========================
# STEP 3: Preprocessing
# ==========================
def clean_tweet(text):
    text = str(text).lower()
    text = re.sub(r"http\S+", " <URL> ", text)
```

```
    text = re.sub(r"@\w+", " <USER> ", text)
    text = re.sub(r"[^a-z0-9<> ]+", " ", text)
    text = re.sub(r"\s+", " ", text).strip()
    return text

def tokenize(text):
    return text.split()

texts = df[TEXT_COL].astype(str).tolist()
labels = df["label"].tolist()

# Build vocab
counter = Counter()
for t in texts:
    counter.update(tokenize(clean_tweet(t)))

PAD, UNK = "<PAD>", "<UNK>"
vocab = {PAD:0, UNK:1}
for word, freq in counter.most_common(20000):  # limit vocab size
    if word not in vocab:
        vocab[word] = len(vocab)

MAX_LEN = 50

def text_to_seq(text):
    tokens = tokenize(clean_tweet(text))
    seq = [vocab.get(tok, vocab[UNK]) for tok in tokens[:MAX_LEN]]
    if len(seq) < MAX_LEN:
        seq += [vocab[PAD]] * (MAX_LEN - len(seq))
    return seq

# ========================
# STEP 4: Dataset + Loader
# ========================
class TweetDataset(Dataset):
    def __init__(self, texts, labels):
        self.data = [text_to_seq(t) for t in texts]
        self.labels = labels
    def __len__(self): return len(self.data)
    def __getitem__(self, idx):
        return torch.tensor(self.data[idx]), torch.tensor(self.labels[idx])

X_train, X_val, y_train, y_val = train_test_split(texts, labels,
test_size=0.2, random_state=42)

train_ds = TweetDataset(X_train, y_train)
val_ds = TweetDataset(X_val, y_val)
```

```
train_loader = DataLoader(train_ds, batch_size=64, shuffle=True)
val_loader = DataLoader(val_ds, batch_size=64)

# =========================
# STEP 5: RNN Model
# =========================
class SentimentRNN(nn.Module):
    def __init__(self, vocab_size, embed_dim=100, hidden_dim=128,
num_classes=len(label2id)):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim,
padding_idx=0)
        self.lstm = nn.LSTM(embed_dim, hidden_dim, batch_first=True,
bidirectional=True)
        self.fc = nn.Linear(hidden_dim*2, num_classes)
        self.dropout = nn.Dropout(0.3)
    def forward(self, x):
        emb = self.embedding(x)
        out, _ = self.lstm(emb)
        pooled = out.mean(dim=1)
        return self.fc(self.dropout(pooled))

model = SentimentRNN(len(vocab)).to(DEVICE)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
criterion = nn.CrossEntropyLoss()

# =========================
# STEP 6: Training
# =========================
def train_one_epoch(model, loader):
    model.train()
    total, correct, loss_sum = 0, 0, 0
    for X, y in loader:
        X, y = X.to(DEVICE), y.to(DEVICE)
        optimizer.zero_grad()
        logits = model(X)
        loss = criterion(logits, y)
        loss.backward()
        optimizer.step()
        loss_sum += loss.item() * X.size(0)
        preds = logits.argmax(1)
        correct += (preds == y).sum().item()
        total += X.size(0)
    return loss_sum/total, correct/total

def evaluate(model, loader):
```

```
    model.eval()
    total, correct, loss_sum = 0, 0, 0
    with torch.no_grad():
        for X, y in loader:
            X, y = X.to(DEVICE), y.to(DEVICE)
            logits = model(X)
            loss = criterion(logits, y)
            loss_sum += loss.item() * X.size(0)
            preds = logits.argmax(1)
            correct += (preds == y).sum().item()
            total += X.size(0)
    return loss_sum/total, correct/total

EPOCHS = 5
for epoch in range(EPOCHS):
    tr_loss, tr_acc = train_one_epoch(model, train_loader)
    val_loss, val_acc = evaluate(model, val_loader)
    print(f"Epoch {epoch+1}: train_loss={tr_loss:.4f},
train_acc={tr_acc:.4f}, val_loss={val_loss:.4f}, val_acc={val_acc:.4f}")

# ========================
# STEP 7: Prediction function
# ========================
def predict_sentiment(text):
    seq = torch.tensor([text_to_seq(text)]).to(DEVICE)
    model.eval()
    with torch.no_grad():
        pred = model(seq).argmax(1).item()
    return id2label[pred]

print(predict_sentiment("I love this service!"))
print(predict_sentiment("This  is the worst thing ever"))
print(predict_sentiment("It was okay, nothing special"))
```

Output

```
Using device: cpu
       textID                                          text  \
0  cb774db0d1                I`d have responded, if I were going
1  549e992a42      Sooo SAD I will miss you here in San Diego!!!
2  088c60f138                          my boss is bullying me...
3  9642c003ef                     what interview! leave me alone
4  358bd9e861   Sons of ****, why couldn`t they put them on t...

                        selected_text sentiment
0  I`d have responded, if I were going   neutral
1                             Sooo SAD  negative
2                          bullying me  negative
3                       leave me alone  negative
4                        Sons of ****,  negative
Index(['textID', 'text', 'selected_text', 'sentiment'], dtype='object')
Labels mapping: {'neutral': 0, 'negative': 1, 'positive': 2}
Epoch 1: train_loss=0.9633, train_acc=0.5176, val_loss=0.8672, val_acc=0.6012
Epoch 2: train_loss=0.7963, train_acc=0.6462, val_loss=0.7807, val_acc=0.6584
Epoch 3: train_loss=0.6780, train_acc=0.7177, val_loss=0.7446, val_acc=0.6806
Epoch 4: train_loss=0.5916, train_acc=0.7646, val_loss=0.7746, val_acc=0.6776
Epoch 5: train_loss=0.5087, train_acc=0.8050, val_loss=0.7766, val_acc=0.6755
positive
negative
positive
```

Conclusion	In this project, we built and trained an RNN-based model to classify the sentiment of tweets in real time. By preprocessing raw tweets into tokenized and padded sequences, we enabled the network to capture sequential dependencies in text. The model learned to distinguish between positive, negative, and neutral sentiments with good accuracy. This demonstrates the effectiveness of recurrent neural networks for natural language tasks where word order and context matter. With further tuning, larger embeddings, or pre-trained word vectors, the performance can be improved and extended to more complex sentiment analysis tasks.