📅 Day 17 Report: Recurrent Neural Networks (RNN) & LSTM – Sentiment Analysis on IMDB Dataset

---

🔍 Introduction to Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are specialized neural architectures designed for sequential data such as text, speech, and time-series information. Unlike traditional feedforward networks, RNNs have loops that allow information to persist over time. This makes them powerful for tasks where context from earlier inputs is essential.

🔁 **Common Applications:**

- Text Classification
- Speech Recognition
- Language Translation
- Time-Series Forecasting
- Sentiment Analysis

🔗 **How RNN Works:**

At each timestep, the RNN receives an input and a hidden state (memory from the past). It then generates an output and updates the hidden state. However, standard RNNs face limitations like **vanishing gradients**, making it difficult to learn long-term dependencies.

LSTM – Long Short-Term Memory Networks

To overcome the shortcomings of basic RNNs, LSTM (Long Short-Term Memory) networks were introduced. LSTMs are designed to retain long-term dependencies in sequential data by using special structures called **gates**:

- 🔷 **Forget Gate** – Decides what to remove from the memory.
- 🔷 **Input Gate** – Decides what new data to store.
- 🔷 **Output Gate** – Determines what to output at each step.

These gates allow the network to remember relevant information for longer periods, making LSTMs ideal for complex NLP tasks like sentiment analysis.

💻 **Practical Task: Sentiment Analysis using LSTM on IMDB Movie Reviews**

🎯 **Objective:**

Build a neural network using LSTM that can classify IMDB movie reviews as **positive** or **negative**.

🛠 **Implementation Steps**

1️⃣ Load the IMDB Dataset

from tensorflow.keras.datasets import imdb

vocab_size = 10000

```
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=vocab_size)
```

- Contains 50,000 labeled movie reviews.
- We retain only the top 10,000 most frequent words to limit vocabulary size.

2️⃣ Preprocessing – Padding Sequences

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

max_length = 200

X_train = pad_sequences(X_train, maxlen=max_length)

X_test = pad_sequences(X_test, maxlen=max_length)
```

- All sequences are padded to the same length for uniformity.

3️⃣ Build the LSTM Model

```
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Embedding(input_dim=vocab_size, output_dim=64, input_length=max_length),
    tf.keras.layers.LSTM(64),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

- **Embedding Layer**: Converts word indices into dense vectors.
- **LSTM Layer**: Captures sequence dependencies.
- **Dense Layer**: Outputs the binary prediction.

4️⃣ Compile and Train the Model

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


history = model.fit(X_train, y_train, epochs=5, batch_size=64, validation_split=0.2)
```

- Trained using 80% data, validated on 20%.

5️⃣ Evaluate the Model

```
loss, acc = model.evaluate(X_test, y_test)

print(f"Test Accuracy: {acc:.4f}")
```

6️⃣ Plot Accuracy Trends

```
import matplotlib.pyplot as plt
```

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("LSTM Model – Accuracy over Epochs")
plt.legend()
plt.grid(True)
plt.show()
```

📌 Results and Observations

- The LSTM model achieved **strong accuracy** on test data.
- Training and validation accuracy curves show consistent performance.
- The model successfully distinguishes between positive and negative reviews.

📓 Key Takeaways

✅ Understood the architecture and working of both RNNs and LSTM.
✅ Explored how LSTM mitigates vanishing gradient issues.
✅ Built and trained a real-world **sentiment classification model** using TensorFlow.
✅ Visualized model performance using Matplotlib.