NLP Assignment-4

Name:P.Rahul

Roll No:2203A51439

Batch:12

1. Load data from keras.datasets and perform following computational analysis:- [CO2]

(a) Preprocessing of the Data

(b) Divide data into training and testing data set

(c) Build the Recurrent Neural network (RNN) Model

(d) Training the RNN Model

(e) Evaluate the model on the test dataset to see how well it generalizes.

2. Develop LSTM (Long Short-Term Memory) by utilizing data set from https://www.kaggle.com/code/amirrezaeian/time-series-data-analysis-using-lstm-tutorial Links to an external site. or take any time series data. [CO2]
3. Demonstrate Vanishing and Exploding Gradients on deep neural network. [CO2]

Task-1:

(a) Preprocessing of the Data

Task 1: RNN with IMDB Dataset

```
from keras.datasets import imdb
from keras.preprocessing import sequence

# Load the dataset
max_words = 10000  # Top 10,000 most frequent words in the dataset
maxlen = 500  # Limit the review length to 500 words

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_words)

# Preprocessing: Pad sequences to ensure uniform length
x_train = sequence.pad_sequenc
x_test = sequence.pad_sequence

print(f'Training data shape: {x_train.shape}')
print(f'Testing data shape: {x_test.shape}')
```

ndarray: x_train

ndarray with shape (25000, 500)

```
Training data shape: (25000, 500)
Testing data shape: (25000, 500)
```

(b) Divide data into training and testing data set

The data is already divided into training and testing data set

(c) Build the Recurrent Neural network (RNN) Model

```
from keras.models import Sequential
from keras.layers import SimpleRNN, Dense, Embedding

# RNN Model
model = Sequential()
model.add(Embedding(max_words, 128))  # Embedding layer to convert words into vectors
model.add(SimpleRNN(64))  # Simple RNN with 64 units
model.add(Dense(1, activation='sigmoid'))  # Output layer with a single neuron for binary classification

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

Model: "sequential_1"

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	?	0 (unbuilt)
simple_rnn_1 (SimpleRNN)	?	0 (unbuilt)
dense_1 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

(d) Training the RNN Model

```
# Train the RNN
history = model.fit(x_train, y_train, epochs=5, batch_size=64, validation_split=0.2)
```

```
Epoch 1/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 78s 241ms/step - accuracy: 0.6057 - loss: 0.6482 - val_accuracy: 0.7004 - val_loss: 0.5710
Epoch 2/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 79s 252ms/step - accuracy: 0.7617 - loss: 0.5027 - val_accuracy: 0.7056 - val_loss: 0.5680
Epoch 3/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 76s 242ms/step - accuracy: 0.8142 - loss: 0.4144 - val_accuracy: 0.8098 - val_loss: 0.4323
Epoch 4/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 75s 241ms/step - accuracy: 0.8895 - loss: 0.2748 - val_accuracy: 0.5812 - val_loss: 0.6646
Epoch 5/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 73s 234ms/step - accuracy: 0.7709 - loss: 0.4844 - val_accuracy: 0.7476 - val_loss: 0.5522
```

(e) Evaluate the model on the test dataset to see how well it generalizes.

```
[8]  # Evaluate the model
     test_loss, test_acc = model.evaluate(x_test, y_test)
     print(f'Test Accuracy: {test_acc:.4f}')

782/782 ━━━━━━━━━━━━━━━━━━━━ 32s 40ms/step - accuracy: 0.7466 - loss: 0.5580
Test Accuracy: 0.7495
```

Task-2:

Task 2: Long Short-Term Memory (LSTM) for Time Series Data

```
from keras.models import Sequential
from keras.layers import LSTM, Dense
import numpy as np

# Simulating a time series dataset (random for demonstration)
n_timesteps = 100
n_features = 1
X = np.random.randn(1000, n_timesteps, n_features)
y = np.random.randn(1000, 1)

# Split into training and testing sets
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

# LSTM Model
model = Sequential()
model.add(LSTM(64, input_shape=(n_timesteps, n_features)))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')
model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an
  super().__init__(**kwargs)
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 64)	16,896
dense_2 (Dense)	(None, 1)	65

```
Total params: 16,961 (66.25 KB)
Trainable params: 16,961 (66.25 KB)
Non-trainable params: 0 (0.00 B)
```

```
[12] # Train the LSTM Model
     history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

Epoch 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - loss: 0.9358 - val_loss: 1.0835
Epoch 2/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - loss: 0.8912 - val_loss: 1.0816
Epoch 3/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - loss: 0.9875 - val_loss: 1.0838
Epoch 4/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - loss: 0.9475 - val_loss: 1.0744
Epoch 5/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - loss: 0.8904 - val_loss: 1.0842
Epoch 6/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - loss: 0.8957 - val_loss: 1.0789
Epoch 7/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 64ms/step - loss: 0.9805 - val_loss: 1.0757
Epoch 8/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 67ms/step - loss: 0.9886 - val_loss: 1.0868
Epoch 9/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 67ms/step - loss: 0.9439 - val_loss: 1.0811
Epoch 10/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - loss: 0.9527 - val_loss: 1.0817
```

```
[13] # Evaluate the model
     test_loss = model.evaluate(X_test, y_test)
     print(f'Test Loss: {test_loss:.4f}')

7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.9977
Test Loss: 0.9880
```

Task-3:

Task 3: Demonstrate Vanishing and Exploding Gradients in Deep Neural Networks

```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
import tensorflow as tf

# Build a deep neural network with many layers
model = Sequential()
model.add(Dense(64, activation='tanh', input_shape=(n_timesteps,)))

# Add many layers to show the vanishing/exploding gradient problem
for _ in range(50):  # 50 layers
    model.add(Dense(64, activation='tanh'))

model.add(Dense(1))

# Using Stochastic Gradient Descent (SGD) to highlight the gradient problem
optimizer = SGD(learning_rate=0.01)
model.compile(optimizer=optimizer, loss='mse')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a la
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
[17] import tensorflow as tf

    # Custom Callback to Monitor Gradients
    class GradientMonitor(tf.keras.callbacks.Callback):
        def on_epoch_end(self, epoch, logs=None):
            # Manually select a batch from the training data for gradient monitoring
            batch_data = X_train[:32]  # Take the first batch (you can randomize this)
            batch_labels = y_train[:32]

            with tf.GradientTape() as tape:
                # Forward pass
                predictions = self.model(batch_data, training=True)
                loss_value = self.model.compiled_loss(batch_labels, predictions)

            # Compute gradients
            gradients = tape.gradient(loss_value, self.model.trainable_weights)

            # Extract max and min gradients
            max_grad = max([tf.reduce_max(tf.abs(g)) for g in gradients if g is not None])
            min_grad = min([tf.reduce_min(tf.abs(g)) for g in gradients if g is not None])

            print(f'Epoch {epoch+1}: Max Gradient = {max_grad:.6f}, Min Gradient = {min_grad:.6f}')

    # Train the model and monitor gradients
    history = model.fit(X_train, y_train, epochs=5, batch_size=32, validation_split=0.2, callbacks=[GradientMonitor()])
```

```
Epoch 1/5
11/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.8358 Epoch 1: Max Gradient = 0.336394, Min Gradient = 0.000000
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.8820 - val_loss: 1.0989
Epoch 2/5
/usr/local/lib/python3.10/dist-packages/keras/src/backend/tensorflow/trainer.py:607: UserWarning: `model.compiled_loss()` is deprecated. Instead, use `model.compute_loss(x, y, y_pred, sample_weight, training)`.
  warnings.warn(
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.9715Epoch 2: Max Gradient = 0.436690, Min Gradient = 0.000000
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - loss: 0.9703 - val_loss: 1.1001
Epoch 3/5
11/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.8555 Epoch 3: Max Gradient = 0.447012, Min Gradient = 0.000000
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.8866 - val_loss: 1.1174
Epoch 4/5
19/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.9292Epoch 4: Max Gradient = 0.211720, Min Gradient = 0.000000
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 0.9288 - val_loss: 1.1172
Epoch 5/5
19/20 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.8833Epoch 5: Max Gradient = 0.390337, Min Gradient = 0.000000
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 0.8855 - val_loss: 1.1101
```