Assignment 5 - Perform Sentiment Analysis in the network graph using RNN

```
import networkx as nx
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
import pickle
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
import seaborn as sns
import matplotlib.pyplot as plt
```

Hyperparameters

```
vocab_size = 10000
max_length = 200
embedding_dim = 128
lstm_units = 128
epochs = 5
batch_size = 32
```

Step 1: Load and preprocess data

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

Step 2: Build the model

```
model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=max_length))
model.add(LSTM(units=lstm_units))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

Step 3: Train the model

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

Step 4: Evaluate the model

```
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_accuracy:.4f}, Test Loss: {test_loss:.4f}")

y_pred_probs = model.predict(X_test)
y_pred_classes = (y_pred_probs > 0.5).astype(int)  # Threshold at 0.5 for binary classification

conf_matrix = confusion_matrix(y_test, y_pred_classes)
```

Step 4.2: Visualize Confusion Matrix

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Negative', 'Positive'],
            yticklabels=['Negative', 'Positive'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for Sentiment Analysis')
plt.show()
```

Print Confusion Matrix metrics

```
TN, FP, FN, TP = conf_matrix.ravel()
accuracy_cm = (TP + TN) / (TP + TN + FP + FN)
precision_cm = TP / (TP + FP)
recall_cm = TP / (TP + FN)
f1_score_cm = 2 * (precision_cm * recall_cm) / (precision_cm + recall_cm)

print("\nConfusion Matrix Metrics:")
print(f"Accuracy : {accuracy_cm:.4f}")
print(f"Precision: {precision_cm:.4f}")
print(f"Recall   : {recall_cm:.4f}")
print(f"F1-Score : {f1_score_cm:.4f}")
```

Step 5: Save the model

```
model.save('sentiment_rnn_model.h5')
```

Step 6: Save the tokenizer

```
word_index = imdb.get_word_index()
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.word_index = word_index
with open('tokenizer.pickle', 'wb') as handle:
    pickle.dump(tokenizer, handle)

print("Model and tokenizer saved successfully!")
```

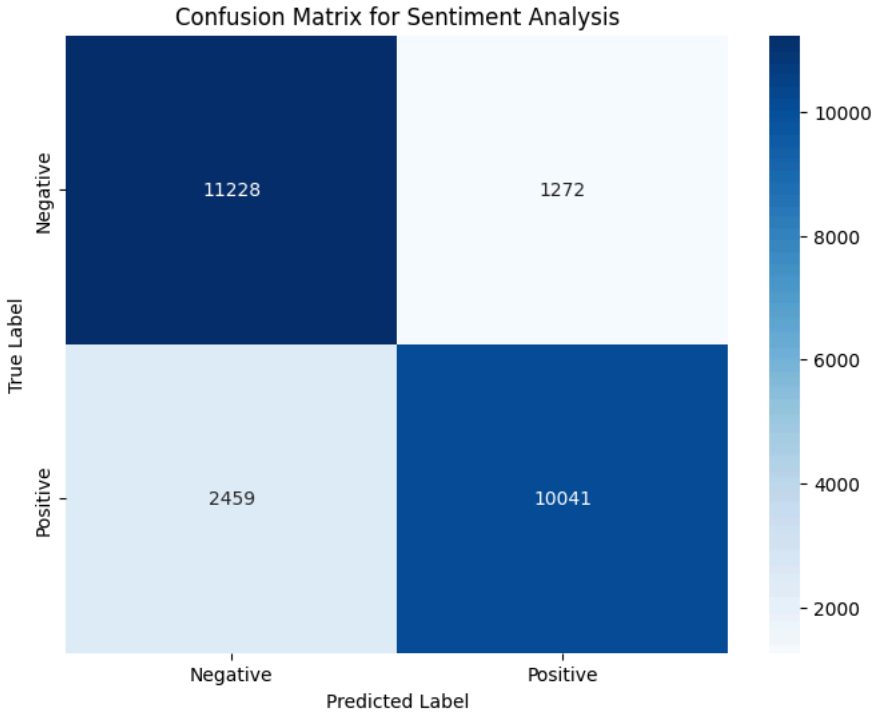
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is
  warnings.warn(
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	?	0 (unbuilt)
lstm_1 (LSTM)	?	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)

```
Epoch 1/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.6755 - loss: 0.5830 - val_accuracy: 0.8444 - val_loss: 0.3724
Epoch 2/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - accuracy: 0.8793 - loss: 0.2998 - val_accuracy: 0.8598 - val_loss: 0.3387
Epoch 3/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - accuracy: 0.9346 - loss: 0.1821 - val_accuracy: 0.8374 - val_loss: 0.3847
Epoch 4/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - accuracy: 0.9563 - loss: 0.1301 - val_accuracy: 0.8558 - val_loss: 0.4156
Epoch 5/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - accuracy: 0.9640 - loss: 0.1004 - val_accuracy: 0.8510 - val_loss: 0.4851
782/782 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.8516 - loss: 0.4875
Test Accuracy: 0.8508, Test Loss: 0.4864
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step
```




```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This fi

Confusion Matrix Metrics:
Accuracy : 0.8508
Precision: 0.8876
Recall   : 0.8033
F1-Score : 0.8433
Model and tokenizer saved successfully!
```

Perform Sentiment Analysis

```
1
2 def perform_sentiment_analysis(graph, model_path, tokenizer_path, max_length):
3     """
4     Perform sentiment analysis on a network graph using a pre-trained RNN model.
5
6     Parameters:
7     - graph (nx.Graph): NetworkX graph with 'text' attributes on nodes.
8     - model_path (str): Path to the pre-trained RNN model file (e.g., 'sentiment_rnn_model.h5')
9     - tokenizer_path (str): Path to the saved tokenizer file (e.g., 'tokenizer.pickle').
10    - max_length (int): Maximum sequence length for padding (must match the model's training).
11
12    Returns:
13    - nx.Graph: The input graph with 'sentiment' attributes added to nodes.
14    """
15
```

```
    # Step 1: Extract texts from nodes
    texts = [data['text'] for node, data in graph.nodes(data=True) if 'text' in data]
    if not texts:
        raise ValueError("No 'text' attributes found in the graph nodes.")

    # Step 2: Load the tokenizer and preprocess the texts
    with open(tokenizer_path, 'rb') as handle:
        tokenizer = pickle.load(handle)

    # Convert texts to sequences of integers
    sequences = tokenizer.texts_to_sequences(texts)
    # Pad sequences to ensure uniform length
    padded_sequences = pad_sequences(sequences, maxlen=max_length)

    # Step 3: Load the pre-trained RNN model
    model = load_model(model_path)

    # Step 4: Predict sentiments
    predictions = model.predict(padded_sequences)
    # Assuming binary classification: > 0.5 is positive, <= 0.5 is negative
    sentiments = ['positive' if p > 0.5 else 'negative' for p in predictions.flatten()]

    # Step 5: Attach sentiments back to nodes
    for node, sentiment in zip(graph.nodes(), sentiments):
        graph.nodes[node]['sentiment'] = sentiment

    return graph
```

Example usage

```

if __name__ == "__main__":
    # Create a sample graph
    G = nx.Graph()
    G.add_node(1, text="This is terrible.")
    G.add_node(2, text="It's absolutely amazing.")

    # Specify paths and parameters (replace with actual paths)
    model_path = 'sentiment_rnn_model.h5'
    tokenizer_path = 'tokenizer.pickle'
    max_length = 100  # Must match the model's training configuration

    try:
        # Perform sentiment analysis
```