Natural Language Processing-Assignment-08

Name: Adithi Shinde

Enrollment No:2203A54032

Batch:40

i)Build the Transformer Model on above dataset:

```python
import numpy as np
import tensorflow as tf
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, Dense, GlobalAveragePooling1D

# Sample text data
text = """Once upon a time, there was a little girl named Red Riding Hood. She loved to visit her grandmother, who lived in the woods. One day, her mother asked her to take a basket

# Tokenize the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])
total_words = len(tokenizer.word_index) + 1

# Create input sequences and labels
input_sequences = []
for line in text.split('.'):
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i + 1]
        input_sequences.append(n_gram_sequence)

# Pad sequences
max_sequence_length = max(len(x) for x in input_sequences)
input_sequences = pad_sequences(input_sequences, maxlen=max_sequence_length, padding='pre')
X, y = input_sequences[:, :-1], input_sequences[:, -1]

# Convert y to categorical
y = tf.keras.utils.to_categorical(y, num_classes=total_words)
```

```python
input_sequences = pad_sequences(input_sequences, maxlen=max_sequence_length, padding='pre')
X, y = input_sequences[:, :-1], input_sequences[:, -1]

# Convert y to categorical
y = tf.keras.utils.to_categorical(y, num_classes=total_words)

# Define the model
def build_transformer_model(vocab_size, max_length):
    model = Sequential()
    model.add(Embedding(vocab_size, 100, input_length=max_length))
    model.add(GlobalAveragePooling1D())
    model.add(Dense(vocab_size, activation='softmax'))
    return model

# Instantiate the model
model = build_transformer_model(total_words, max_sequence_length)
model.summary()
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	?	0 (unbuilt)
global_average_pooling1d_1 (GlobalAveragePooling1D)	?	0 (unbuilt)
dense_3 (Dense)	?	0 (unbuilt)

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

ii) Train the model using 20, 60, 70 epochs:

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

# Train the model for 20 epochs
history_20 = model.fit(X, y, epochs=20, batch_size=32, verbose=1)

# Train the model for 60 epochs
history_60 = model.fit(X, y, epochs=60, batch_size=32, verbose=1)

# Train the model for 70 epochs
history_70 = model.fit(X, y, epochs=70, batch_size=32, verbose=1)
```

```
Epoch 1/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.0232 - loss: 3.7215
Epoch 2/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.0361 - loss: 3.7099
Epoch 3/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.1058 - loss: 3.7009
Epoch 4/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1314 - loss: 3.6935
Epoch 5/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1442 - loss: 3.6855
Epoch 6/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1082 - loss: 3.6804
Epoch 7/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.0954 - loss: 3.6709
Epoch 8/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.0954 - loss: 3.6616
```

iii) After training, use the model to generate new text by feeding it an initial seed text:

```
# Function to generate new text
def generate_text(seed_text, next_words, model, max_sequence_length):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=max_sequence_length - 1, padding='pre')
        predicted = model.predict(token_list, verbose=0)
        predicted_word_index = np.argmax(predicted, axis=-1)[0]
        output_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predicted_word_index:
                output_word = word
                break
        seed_text += " " + output_word
    return seed_text

# Example of generating new text
seed_text = "Once upon a time"
generated_text = generate_text(seed_text, 50, model, max_sequence_length)
print("Generated Text:\n", generated_text)
```

```
Generated Text:
 Once upon a time a a a little named red riding hood hood hood hood hood hood hood hood hood hood hood hood hood hood hood hood a hood hood hood hood hood hood hood hood hood
```

iv) Experimenting and Improving the Model by large dataset and hyper tune parameter:

```
# Hyperparameter tuning examples
def build_optimized_model(vocab_size, max_length, embedding_dim=100, dropout_rate=0.2):
    model = Sequential()
    model.add(Embedding(vocab_size, embedding_dim, input_length=max_length))
    model.add(GlobalAveragePooling1D())
    model.add(Dense(vocab_size, activation='softmax'))
    return model

# Experiment with different embedding dimensions and dropout rates
model1 = build_optimized_model(total_words, max_sequence_length, embedding_dim=150)
model2 = build_optimized_model(total_words, max_sequence_length, embedding_dim=200)

# Compile and train with a larger dataset (if available) and different epochs
# For example, using `model1.fit()` or `model2.fit()`
```

```python
import matplotlib.pyplot as plt

def plot_history(histories):
    plt.figure(figsize=(14, 5))

    for key in histories.keys():
        plt.plot(histories[key].history['accuracy'], label=f'Accuracy {key}')
        plt.plot(histories[key].history['loss'], label=f'Loss {key}')

    plt.title('Model Accuracy and Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Metrics')
    plt.legend()
    plt.grid()
    plt.show()

# Plot the training history
plot_history({'20': history_20, '60': history_60, '70': history_70})
```

[15]