Natural Language Processing-Assignment-08

Name: Adithi Shinde

Enrollment No:2203A54032

Batch:40

1. Use a small dataset of English to French sentence pairs. You can replace this with any other language pair dataset.

data = [("hello", "bonjour"), ("how are you", "comment ça va"), ("I am fine", "je vais bien"), ("what is your name", "comment tu t'appelles"), ("my name is", "je m'appelle"), ("thank you", "merci"), ("goodbye", "au revoir")

(a) Take input (English) and target (French):

```python
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Dataset of English to French sentence pairs
data = [
    ("hello", "bonjour"),
    ("how are you", "comment ça va"),
    ("I am fine", "je vais bien"),
    ("what is your name", "comment tu t'appelles"),
    ("my name is", "je m'appelle"),
    ("thank you", "merci"),
    ("goodbye", "au revoir"),
]

# Separate the input and target sentences
input_sentences = [pair[0] for pair in data]
target_sentences = [pair[1] for pair in data]

# Print the separated sentences
print("Input Sentences (English):", input_sentences)
print("Target Sentences (French):", target_sentences)
```

```
Input Sentences (English): ['hello', 'how are you', 'I am fine', 'what is your name', 'my name is', 'thank you', 'goodbye']
Target Sentences (French): ['bonjour', 'comment ça va', 'je vais bien', "comment tu t'appelles", "je m'appelle", 'merci', 'au revoir']
```

(b) Building the Seq2Seq Model:

```python
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense

# Tokenize the input and target sentences
input_tokenizer = Tokenizer()
input_tokenizer.fit_on_texts(input_sentences)
input_vocab_size = len(input_tokenizer.word_index) + 1

target_tokenizer = Tokenizer()
target_tokenizer.fit_on_texts(target_sentences)
target_vocab_size = len(target_tokenizer.word_index) + 1

# Define parameters
embedding_dim = 256
latent_dim = 256

# Encoder
encoder_inputs = Input(shape=(None,))
encoder_embedding = Embedding(input_vocab_size, embedding_dim)(encoder_inputs)
encoder_outputs, state_h, state_c = LSTM(latent_dim, return_state=True)(encoder_embedding)
encoder_states = [state_h, state_c]

# Decoder
decoder_inputs = Input(shape=(None,))
decoder_embedding = Embedding(target_vocab_size, embedding_dim)(decoder_inputs)
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_embedding, initial_state=encoder_states)
decoder_dense = Dense(target_vocab_size, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)
```

```python
# Define the model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Print the model summary
model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, None)	0	-
input_layer_1 (InputLayer)	(None, None)	0	-
embedding (Embedding)	(None, None, 256)	3,840	input_layer[0][0]
embedding_1 (Embedding)	(None, None, 256)	3,584	input_layer_1[0][0]
lstm (LSTM)	[(None, 256), (None, 256), (None, 256)]	525,312	embedding[0][0]
lstm_1 (LSTM)	[(None, None, 256), (None, 256), (None, 256)]	525,312	embedding_1[0][0], lstm[0][1], lstm[0][2]
dense (Dense)	(None, None, 14)	3,598	lstm_1[0][0]

Total params: 1,061,646 (4.05 MB)
Trainable params: 1,061,646 (4.05 MB)
Non-trainable params: 0 (0.00 B)

(c) Prepare the target sequences for training by shifting them by one position, which the model will use to predict the next word in the sequence:

```python
# Define the target sentences in French
target_sentences = [
    "bonjour",
    "comment ça va",
    "je vais bien",
    "comment tu t'appelles",
    "je m'appelle",
    "merci",
    "au revoir"
]

# Step to add start and end tokens
target_sentences_with_tokens = ['<start> ' + sentence + ' <end>' for sentence in target_sentences]

# Print the modified target sentences for verification
print("Target Sentences with Tokens:", target_sentences_with_tokens)
```

```
Target Sentences with Tokens: ['<start> bonjour <end>', '<start> comment ça va <end>', '<start> je vais bien <end>', "<start> comment
```

```python
# Tokenize the input sentences
input_tokenizer = Tokenizer()
input_tokenizer.fit_on_texts(input_sentences)

# Convert input sentences to sequences
input_sequences = input_tokenizer.texts_to_sequences(input_sentences)

# Pad the input sequences
max_input_length = max(len(seq) for seq in input_sequences)
input_sequences = pad_sequences(input_sequences, maxlen=max_input_length, padding='post')

# Convert target sentences to sequences
target_sequences = target_tokenizer.texts_to_sequences(target_sentences_with_tokens)

# Determine maximum target length
max_target_length = max(len(seq) for seq in target_sequences)

# Pad the target sequences
target_sequences = pad_sequences(target_sequences, maxlen=max_target_length, padding='post')

# Create decoder input sequences by shifting the target sequences
decoder_input_sequences = np.zeros(target_sequences.shape)

# Shift the target sequences to create decoder input sequences
for i in range(len(target_sequences)):
    decoder_input_sequences[i, 1:] = target_sequences[i, :-1]  # Shift by 1
    decoder_input_sequences[i, 0] = target_tokenizer.word_index['<start>']  # Set the start token

# Reshape target sequences to have an extra dimension for training
target_sequences = np.expand_dims(target_sequences, -1)  # Add an extra dimension
```

(d) After training, set up separate models for the encoder and decoder to perform inference (translation) on new sentences.

```python
# Parameters
embedding_dim = 50
latent_dim = 256
num_encoder_tokens = len(input_tokenizer.word_index) + 1
num_decoder_tokens = len(target_tokenizer.word_index) + 1

# Build the model
encoder_inputs = Input(shape=(max_input_length,))
encoder_embedding = Embedding(num_encoder_tokens, embedding_dim)(encoder_inputs)
encoder_lstm = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm(encoder_embedding)
encoder_states = [state_h, state_c]

decoder_inputs = Input(shape=(max_target_length,))
decoder_embedding = Embedding(num_decoder_tokens, embedding_dim)(decoder_inputs)
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_embedding, initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

# Compile the model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```python
# Define the encoder model for inference
encoder_model = Model(encoder_inputs, encoder_states)

# Define the decoder model for inference
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_hidden_state_input = Input(shape=(None, latent_dim))

decoder_embedding_inference = Embedding(target_vocab_size, embedding_dim)(decoder_inputs)
decoder_outputs, state_h, state_c = decoder_lstm(decoder_embedding_inference, initial_state=[decoder_state_input_h, decoder_state_input_c])
decoder_outputs = decoder_dense(decoder_outputs)

decoder_model = Model([decoder_inputs] + [decoder_state_input_h, decoder_state_input_c], [decoder_outputs] + [state_h, state_c])
```

```python
# Train the model
model.fit([input_sequences, decoder_input_sequences], target_sequences, epochs=100, batch_size=2)
```

```
Epoch 1/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 47ms/step - accuracy: 0.1333 - loss: 2.7636
Epoch 2/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - accuracy: 0.3719 - loss: 2.7078
Epoch 3/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 0.3886 - loss: 2.6062
Epoch 4/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.3719 - loss: 2.3727
Epoch 5/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.3605 - loss: 2.1971
```


```
Epoch 1/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 47ms/step - accuracy: 0.1333 - loss: 2.7636
Epoch 2/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - accuracy: 0.3719 - loss: 2.7078
Epoch 3/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 0.3886 - loss: 2.6062
Epoch 4/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.3719 - loss: 2.3727
Epoch 5/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.3605 - loss: 2.1971
```


```
Epoch 6/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.3886 - loss: 2.0938
Epoch 7/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - accuracy: 0.4186 - loss: 1.9524
Epoch 8/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.4348 - loss: 2.0631
Epoch 9/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.4995 - loss: 1.8449
Epoch 10/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.5110 - loss: 1.8126
Epoch 11/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.4924 - loss: 1.7968
Epoch 12/100
```

e) Use the trained model to translate new English sentences into French:

```python
# Inference Models
encoder_model = Model(encoder_inputs, encoder_states)

# Create decoder model
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]
decoder_embedding_inf = Embedding(num_decoder_tokens, embedding_dim)
decoder_outputs_inf, state_h_inf, state_c_inf = decoder_lstm(decoder_embedding_inf(decoder_inputs), initial_state=decoder_states_inputs)
decoder_states_inf = [state_h_inf, state_c_inf]
decoder_outputs_inf = decoder_dense(decoder_outputs_inf)

# Define the inference model for the decoder
decoder_model = Model(
    [decoder_inputs] + decoder_states_inputs,
    [decoder_outputs_inf] + decoder_states_inf
)
```

```python
def translate_sentence(input_sentence):
    # Tokenize and pad the input sentence
    input_seq = input_tokenizer.texts_to_sequences([input_sentence])
    input_seq = pad_sequences(input_seq, maxlen=max_input_length, padding='post')

    # Encode the input sentence
    states_value = encoder_model.predict(input_seq)

    # Generate the target sequence (starting with the <start> token)
    target_seq = np.array([[target_tokenizer.word_index.get('<start>', 0)]])

    # Sampling loop for a batch of sequences
    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = target_tokenizer.index_word.get(sampled_token_index, '')

        if sampled_char:
            decoded_sentence += ' ' + sampled_char

        # Exit condition: either hit max length or find stop token
        if (sampled_char == '<end>' or len(decoded_sentence.split()) > max_target_length):
            stop_condition = True

        # Update the target sequence (for the next time step)
        target_seq = np.array([[sampled_token_index]])
        states_value = [h, c]
```

```python
    return decoded_sentence.strip()

# Example translation
input_sentence = "hello"
translated_sentence = translate_sentence(input_sentence)
print(f"Translated '{input_sentence}' to '{translated_sentence}'")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 180ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
```

```python
def translate_sentence(input_sentence):
    # Tokenize and pad the input sentence
    input_seq = input_tokenizer.texts_to_sequences([input_sentence])
    input_seq = pad_sequences(input_seq, maxlen=max_input_length, padding='post')

    # Encode the input sentence
    states_value = encoder_model.predict(input_seq)

    # Generate the target sequence (starting with the <start> token)
    target_seq = np.array([[target_tokenizer.word_index.get('<start>', 0)]])

    # Sampling loop for a batch of sequences
    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = target_tokenizer.index_word.get(sampled_token_index, '')

        if sampled_char:
            decoded_sentence += ' ' + sampled_char

        # Exit condition: either hit max length or find stop token
        if (sampled_char == '<end>' or len(decoded_sentence.split()) > max_target_length):
            stop_condition = True

        # Update the target sequence (for the next time step)
        target_seq = np.array([[sampled_token_index]])
        states_value = [h, c]
```