Part A: Neural Machine Translation

Bikash Kumar Tiwary

Student Id: 190778031

Task 1: Implementing the encoder.

```
"""
Task 1 encoder

Start
"""

# Layer 1
embedding_source = Embedding(input_dim = self.vocab_source_size ,output_dim = self.embedding_size, mask_zero = True)(source_words)
# Layer 2
embedding_target = Embedding(input_dim = self.vocab_target_size ,output_dim = self.embedding_size ,mask_zero = True)(target_words)

# DropOut
source_words_embeddings = Dropout(self.embedding_dropout_rate)(embedding_source)
target_words_embeddings = Dropout(self.embedding_dropout_rate)(embedding_target)

#LSTM
encoder_outputs, encoder_state_h, encoder_state_c = LSTM(self.hidden_size, return_sequences = True, return_state = True)(source_words_embeddings)

"""
End Task 1
"""
encoder_states = [encoder_state_h,encoder_state_c]

decoder_lstm = LSTM(self.hidden_size,recurrent_dropout=self.hidden_dropout_rate,return_sequences=True,return_state=True)
decoder_outputs_train,_,_ = decoder_lstm(target_words_embeddings,initial_state=encoder_states)
```

The above code snippet shows the implementation of encoder. Where 2 embedding layers are created and dropout is applied followed with LSTM layer to process source_words_embeddings setting return_state and return_sequence to True.

Task 2: Implementing the decoder.

```
"""
Task 2 decoder for inference

Start
"""
decoder_states = [decoder_state_input_h,decoder_state_input_c]

decoder_outputs_test, decoder_state_output_h, decoder_state_output_c = decoder_lstm(target_words_embeddings,initial_state=decoder_states)

if self.use_attention:
    decoder_attention = AttentionLayer()
    decoder_outputs_test = decoder_attention([encoder_outputs,decoder_outputs_test])

# Pass LSTM or attention layer to decoder_dense
decoder_outputs_test = decoder_dense(decoder_outputs_test)

"""
End Task 2
"""
```

The above code snippet shows the implementation of decoder for inference. Where firstly the states are put together in a list, and passed target_word_embedding and decoder_states to LSTM. Finally, passing the LSTM into final layer of the decoder to assign probabilities of the next token.

BLEU score and a sample of output shown below (i.e. use_attention = False):

Starting training epoch 1/10

C:\Users\BIKASH\Anaconda3\envs\Tensorflow\lib\site-packages\tensorflow_core\python\framework\indexed_slices.py:433: UserWarning: Converting

sparse IndexedSlices to a dense Tensor of unknown shape. This may consume a large amount of memory.

"Converting sparse IndexedSlices to a dense Tensor of unknown shape. "

Epoch 1/1

24000/24000 [==============================] - 25s 1ms/step - loss: 2.1311 - accuracy: 0.2440

Time used for epoch 1: 0 m 27 s

Evaluating on dev set after epoch 1/10:

Model BLEU score: 1.57

Time used for evaluate on dev set: 0 m 5 s

Starting training epoch 2/10

Epoch 1/1

24000/24000 [==============================] - 24s 1ms/step - loss: 1.8316 - accuracy: 0.3026

Time used for epoch 2: 0 m 24 s

Evaluating on dev set after epoch 2/10:

Model BLEU score: 2.60

Time used for evaluate on dev set: 0 m 5 s

Starting training epoch 3/10

Epoch 1/1

24000/24000 [==============================] - 24s 1ms/step - loss: 1.7160 - accuracy: 0.3336

Time used for epoch 3: 0 m 24 s

Evaluating on dev set after epoch 3/10:

Model BLEU score: 3.04

Time used for evaluate on dev set: 0 m 5 s

Starting training epoch 4/10

Epoch 1/1

24000/24000 [==============================] - 25s 1ms/step - loss: 1.6378 - accuracy: 0.3517

Time used for epoch 4: 0 m 24 s

Evaluating on dev set after epoch 4/10:

Model BLEU score: 3.41

Time used for evaluate on dev set: 0 m 5 s

Starting training epoch 5/10

Epoch 1/1

24000/24000 [==============================] - 24s 992us/step - loss: 1.5823 - accuracy: 0.3633

Time used for epoch 5: 0 m 23 s

Evaluating on dev set after epoch 5/10:

Model BLEU score: 4.37

Time used for evaluate on dev set: 0 m 5 s

Starting training epoch 6/10

Epoch 1/1

24000/24000 [==============================] - 25s 1ms/step - loss: 1.5386 - accuracy: 0.3719

Time used for epoch 6: 0 m 25 s

Evaluating on dev set after epoch 6/10:

Model BLEU score: 4.42

Time used for evaluate on dev set: 0 m 6 s

Starting training epoch 7/10

Epoch 1/1

24000/24000 [==============================] - 24s 1ms/step - loss: 1.5042 - accuracy: 0.3785

Time used for epoch 7: 0 m 24 s

Evaluating on dev set after epoch 7/10:

Model BLEU score: 4.79

Time used for evaluate on dev set: 0 m 5 s

Starting training epoch 8/10

Epoch 1/1

24000/24000 [==============================] - 25s 1ms/step - loss: 1.4735 - accuracy: 0.3841

Time used for epoch 8: 0 m 25 s

Evaluating on dev set after epoch 8/10:

Model BLEU score: 4.90

Time used for evaluate on dev set: 0 m 6 s

Starting training epoch 9/10

Epoch 1/1

24000/24000 [==============================] - 24s 1ms/step - loss: 1.4501 - accuracy: 0.3880

Time used for epoch 9: 0 m 24 s

Evaluating on dev set after epoch 9/10:

Model BLEU score: 4.82

Time used for evaluate on dev set: 0 m 5 s

Starting training epoch 10/10

Epoch 1/1

24000/24000 [==============================] - 25s 1ms/step - loss: 1.4289 - accuracy: 0.3915

Time used for epoch 10: 0 m 24 s

Evaluating on dev set after epoch 10/10:

Model BLEU score: 5.26

Time used for evaluate on dev set: 0 m 5 s

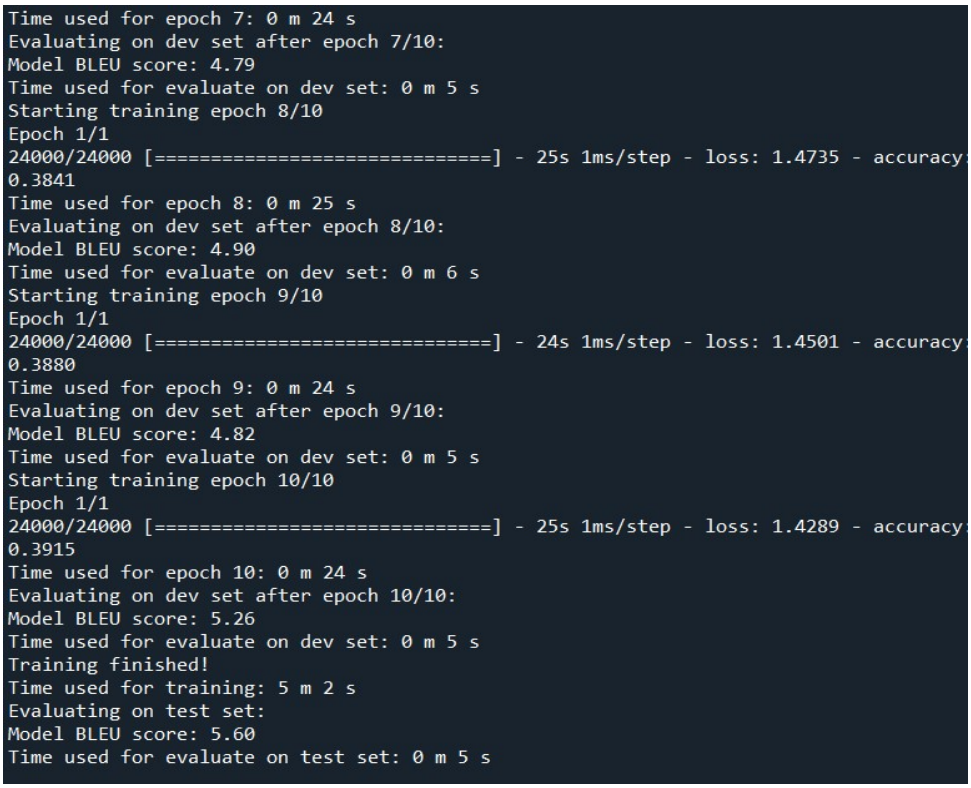
Training finished!

Time used for training: 5 m 2 s

Evaluating on test set:

Model BLEU score: 5.60

Time used for evaluate on test set: 0 m 5 s



(An output snippet)

3. Adding attention

```
"""
Task 3 attention

Start
"""
decoder_outputs = K.permute_dimensions(decoder_outputs, (0,2,1))

# multiply
luong_score = K.batch_dot(decoder_outputs, encoder_outputs)
# Apply Softmax, dim = max_sourse_sent_len
luong_score = softmax(luong_score, axis=1)

# Expand dimensions
luong_score = K.expand_dims(luong_score, axis = -1)
encoder_outputs = K.expand_dims(encoder_outputs, axis = 2)
#multiply 2 tensors with expanded dimension
mul = luong_score * encoder_outputs
encoder_vector = K.sum(mul, axis = 1)

"""
End Task 3
"""
# [batch,max_dec,2*emb]
new_decoder_outputs = K.concatenate([decoder_outputs, encoder_vector])

return new_decoder_outputs
```

The above code implements the attention layer, where first the last two dimensions are transposed to required shape. And batch_dot is used to multiple and get luong_score. Finally, encoder_vector is created by doing element-wise multiplication between encoder_outputs and their attention score.