PART A:

Task 1:

Code: Encoder

```
# implementing embedding layer and encoder

# source embedding layer with input dim as source vocab size and output as em
bedding size
# these are trainable layer with mask_zero = true to get rid of the padding
embedding_source = Embedding(self.vocab_source_size, self.embedding_size, tra
inable=True, mask_zero = True)
# target embedding layer with input dim as target vocab size and output as em
bedding size
embedding_target = Embedding(self.vocab_target_size, self.embedding_size, tra
inable=True, mask_zero = True)
# passing source words through the source embedding layer and adding dropout
source_words_embeddings = embedding_source(source_words)
source_words_embeddings = Dropout(self.embedding_dropout_rate)(source_words_e
mbeddings)
# passing target words through the target embedding layer and adding dropout
target_words_embeddings = embedding_target(target_words)
target_words_embeddings = Dropout(self.embedding_dropout_rate)(target_words_e
mbeddings)
# LSTM layer with units = self.hidden_size
# return sequence is ture to get the output of all tokens
# return state is true to get the hidden state and cell state from the encode
r
encoder_lstm = LSTM(self.hidden_size, recurrent_dropout=self.hidden_dropout_r
ate, return_sequences=True, return_state=True)
# passing the source word embeddings to encoder_lstm
encoder_outputs, encoder_state_h, encoder_state_c = encoder_lstm(source_words
_embeddings)
```

Task 2:

Code: Decoder

```
# putting decoder state input h and state input c in a list
decoder_states = [decoder_state_input_h, decoder_state_input_c]
# passing targer_words_embeddings through the decoder lstm
# passing decoder states as the initial state for the decoder lstm
decoder_outputs_test, decoder_state_output_h, decoder_state_output_c = decode
r_lstm(target_words_embeddings,initial_state=decoder_states)
# check if attention layer should be used
# attention layer is used to provide the decoder with information from every
encoder hidden state
if self.use_attention:
   decoder_attention = AttentionLayer()
```

```
    decoder_outputs_test = decoder_attention([encoder_outputs_input,decoder_ou
tputs_test])
# passing output of the decoder lstm (attention = false) or decoder_attention
 (attention = true) to the decoder dense layer
decoder_outputs_test = decoder_dense(decoder_outputs_test)
```

Code: Print sample output at every epoch

```
# code to print the first 5 sample outputs: predicted outputs are candidates
and actual sentences are references
print("===== PRINTING OUTPUT====")
for i in range(5):
    print("{} Sample".format(i+1))
    print("Predicted Sentence: "+" ".join(candidates[i]))

    print("Actual Sentence: "+" ".join(references[i][0]))

print("========================")
```

Output

```
Time used for evaluate on dev set: 0 m 4 s
Training finished!
Time used for training: 6 m 35 s
Evaluating on test set:
Model BLEU score: 5.33
Time used for evaluate on test set: 0 m 4 s
```

Predicted (Attention = False)	Actual
the first <unk> of the <unk> <unk> was <unk> by <unk> <unk> .	the second quote is from the head of the u.k. financial services <unk> .
so the <unk> thing is .	it gets worse .
what happens when the <unk> is not <unk> ?	what 's happening here ? how can this be possible ?
it 's not just a <unk> .	unfortunately , the answer is yes .
but , there 's a <unk> <unk> , and i 'm not going to be a <unk> <unk> .	but there 's an <unk> solution which is coming from what is known as the science of <unk> .

Task 3:

Code: Attention Layer

```
# This attention mechanism computes the score between decoder_outpus and enco
der_outputs by matrix multiplication
```

```
# to multiply decoder_outputs and encoder_outputs, it is needed to transpose the last two dimensions of decoder_outputs
# the last two dimensions are transposed as given below
decoder_outputs_dim = K.permute_dimensions(decoder_outputs, (0,2,1))
# luong_score is calculated by using a dot product  of encoder_outputs and decoder_outputs
luong_score = K.batch_dot(encoder_outputs, decoder_outputs_dim, axes=None)
# softmax is applied to get the attention score
luong_score = K.softmax(luong_score, axis=1)
# expand the dimensions so that encoder_outputs and luong score(attention score) has the same shape
encoder_outputs = K.expand_dims(encoder_outputs, axis=2)
luong_score = K.expand_dims(luong_score, axis=3)
# encoded vector is created by doing element wise multiplication between encoder_outputs and luong_score
encoder_vector = encoder_outputs * luong_score
# sum the ecoder_vector along the axis = 1 (max_source_sent_len) to get the required encoder_vector
encoder_vector = K.sum(encoder_vector, axis=1)
```

Output:

```
Time used for evaluate on dev set: 0 m 4 s
Training finished!
Time used for training: 6 m 51 s
Evaluating on test set:
Model BLEU score: 15.40
Time used for evaluate on test set: 0 m 4 s
```

Predicted (Attention = True)	Actual
the second thing comes from people who lead the <unk> of the <unk> .	the second quote is from the head of the u.k. financial services <unk> .
it 's really much worse .	it gets worse .
what 's going on here ? what can you ?	what 's happening here ? how can this be possible ?
unfortunately , the answer is correct .	unfortunately , the answer is yes .
but , there 's a very funny solution to be a very interesting solution to be a collection of a complex scientific <unk> .	but there 's an <unk> solution which is coming from what is known as the science of <unk> .

PART B:

Task 1 and Task 2: Refer to the notebook “PART_B_Task _1_2.ipynb”

Task 3: Refer to the notebook “PART_B_Task_3.ipynb”