ASSIGNMENT 1E:

Neural Machine Translation.

Neural Networks and Natural Language Processing

Carolina Raymond Rodrigues.
Student ID: 210607000
ec21209@qmul.ac.uk

Contents

List of Figures

1 Introduction

For this part of the assignment, a Neural Machine Translation (NMT) model is build based on the sequence-to-sequence (seq2seq) model proposed in papers [1] and [2].

The presented model uses an Encoder-Decoder architecture. First, it will read the source sentence with help of an encoder. The encoder will be in charge of transforming the input into a context vector. This vector will represent the meaning of the input as numbers. Secondly, the decoder will process the sentence and emit a translation.

However, one problem may arise during the encoding process: the bottleneck problem (The encoder must capture all the information of a sentence, nevertheless, you can't use a vector of fixed dimensions to provide the meaning of a sentence). Therefore, the solution consists on focusing on a particular part of the source sequence in each step of the decoder.

In the following lines we will explain and provide the code used to implemented the NMT.

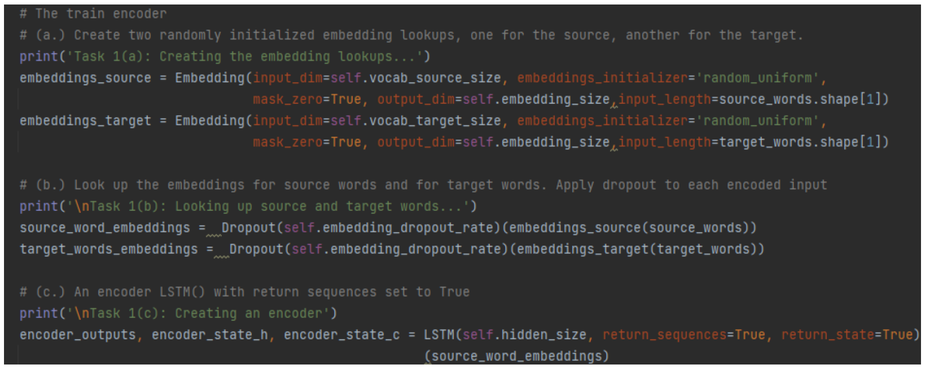
2 Implementing the encoder.

The encoder extracts the features from a source sentence. It consists on a stack of LSTM cells that will summarise the input in a context vector represented by the hidden state and cell state of the LSTM cell. These two vectors will be passed as inputs to the decoder, helping it to make more accurate predictions.

The model will be first composed by two embedding layers: one for the source and other for the target language. This layer will represent the input data in a way in which similar word meaning have a similar representations.

Then, the source embedding will be passed to the LSTM layer. That will be composed of 200 LSTM units. We will obtain three outcomes from the last LSTM cell: the output, the hidden state and the cell state. The last two will contain the meaning of the source and will be passed to the decoder as inputs.

In Figure 1 the implemented code built following the guideline has been provided.



```
# The train encoder
# (a.) Create two randomly initialized embedding lookups, one for the source, another for the target.
print('Task 1(a): Creating the embedding lookups...')
embeddings_source = Embedding(input_dim=self.vocab_source_size, embeddings_initializer='random_uniform',
                              mask_zero=True, output_dim=self.embedding_size,input_length=source_words.shape[1])
embeddings_target = Embedding(input_dim=self.vocab_target_size, embeddings_initializer='random_uniform',
                              mask_zero=True, output_dim=self.embedding_size,input_length=target_words.shape[1])

# (b.) Look up the embeddings for source words and for target words. Apply dropout to each encoded input
print('\nTask 1(b): Looking up source and target words...')
source_word_embeddings = Dropout(self.embedding_dropout_rate)(embeddings_source(source_words))
target_words_embeddings = Dropout(self.embedding_dropout_rate)(embeddings_target(target_words))

# (c.) An encoder LSTM() with return sequences set to True
print('\nTask 1(c): Creating an encoder')
encoder_outputs, encoder_state_h, encoder_state_c = LSTM(self.hidden_size, return_sequences=True, return_state=True)
                                                    (source_word_embeddings)
```

Figure 1: Encoder code

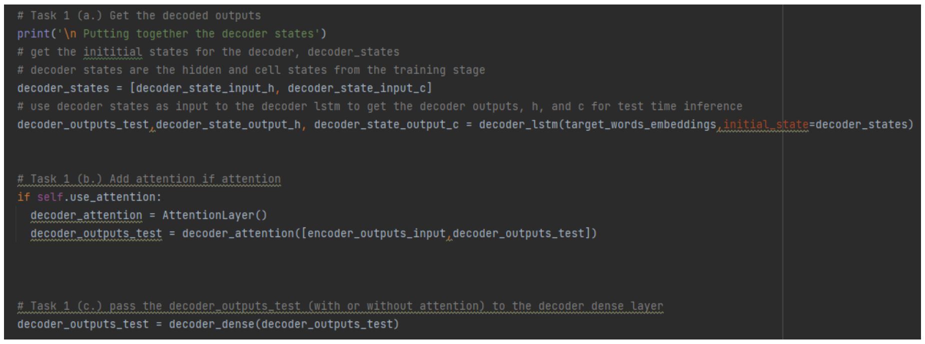
Note that the embedding layer has been randomly initialised using the "random uniform". This layer has also set the mask_zero to True in order to get rid of the paddings and a dropout of 20% has been applied.

3 Implementing the decoder.

The decoder layer will interpret the input obtained from the output of the encoder. In order to do that, it will generate a sequence of outputs that will be re-used for predicting subsequent sequences too. Just as the encoder, it will be composed of a stack of 200 LSTM cells that will predict one output at a time step.

When building the decoder, we must take into account the training and the inference process. In the case of the former one, we will process all the tokens that form the sentence in one single step. While in the case of the later one, we will process one token at a time.

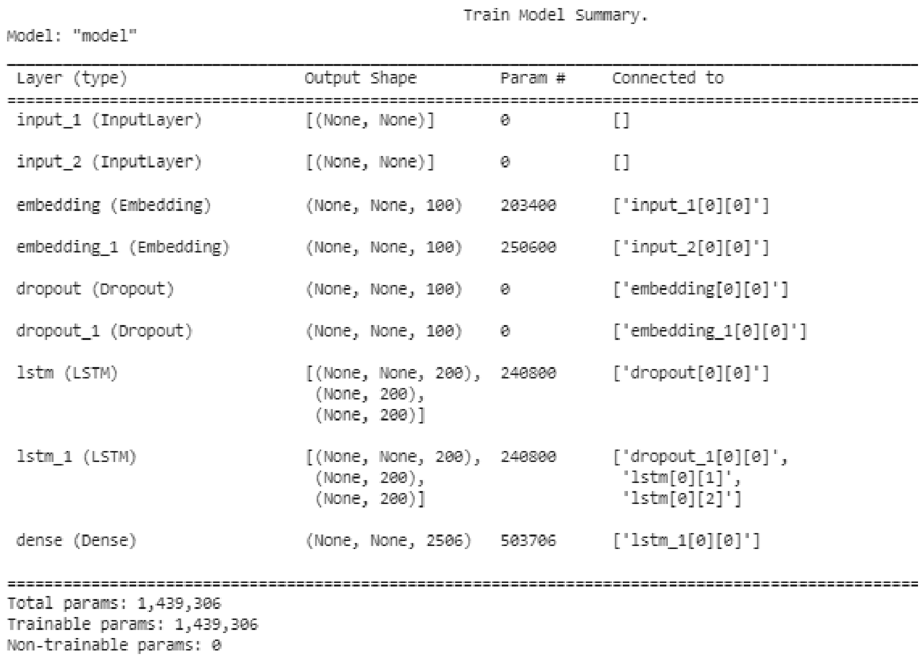
Taking all this into account, we can now implement the code for the decoder during inference. This code is similar to the one for the encoder as it can be seen in Figure 2



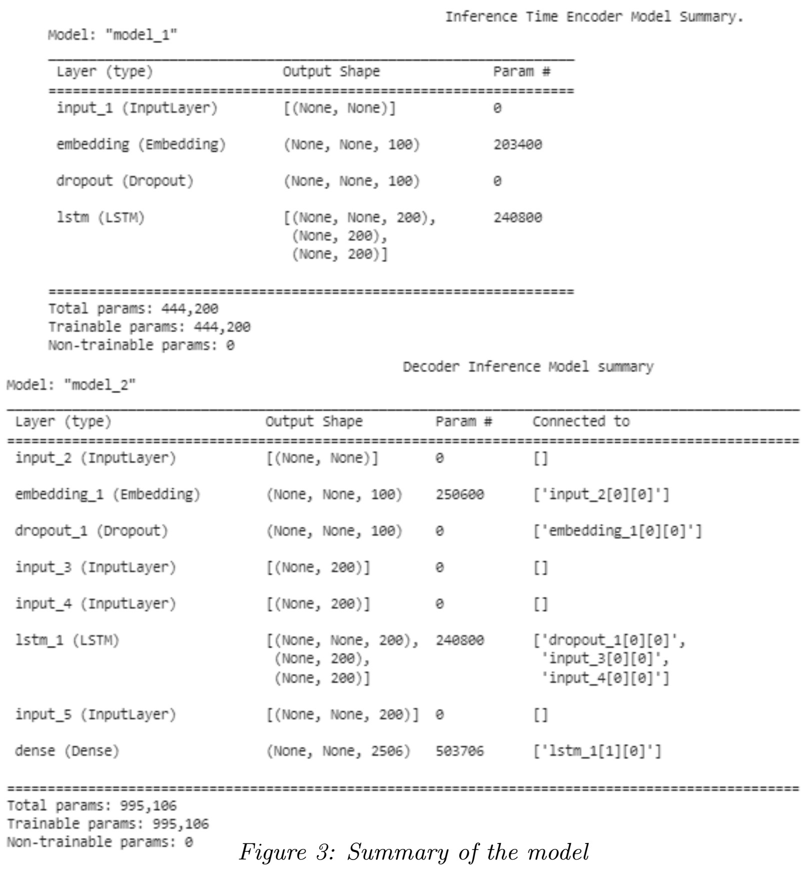
```python
# Task 1 (a.) Get the decoded outputs
print('\n Putting together the decoder states')
# get the inititial states for the decoder, decoder_states
# decoder states are the hidden and cell states from the training stage
decoder_states = [decoder_state_input_h, decoder_state_input_c]
# use decoder states as input to the decoder lstm to get the decoder outputs, h, and c for test time inference
decoder_outputs_test,decoder_state_output_h, decoder_state_output_c = decoder_lstm(target_words_embeddings,initial_state=decoder_states)


# Task 1 (b.) Add attention if attention
if self.use_attention:
  decoder_attention = AttentionLayer()
  decoder_outputs_test = decoder_attention([encoder_outputs_input,decoder_outputs_test])



# Task 1 (c.) pass the decoder_outputs_test (with or without attention) to the decoder dense layer
decoder_outputs_test = decoder_dense(decoder_outputs_test)
```

Figure 2: Decoder code

The summary of the built model can be seen in Figure 3



```
                                              Train Model Summary.
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, None)]       0           []

 input_2 (InputLayer)           [(None, None)]       0           []

 embedding (Embedding)          (None, None, 100)    203400      ['input_1[0][0]']

 embedding_1 (Embedding)        (None, None, 100)    250600      ['input_2[0][0]']

 dropout (Dropout)              (None, None, 100)    0           ['embedding[0][0]']

 dropout_1 (Dropout)            (None, None, 100)    0           ['embedding_1[0][0]']

 lstm (LSTM)                    [(None, None, 200),  240800      ['dropout[0][0]']
                                 (None, 200),
                                 (None, 200)]

 lstm_1 (LSTM)                  [(None, None, 200),  240800      ['dropout_1[0][0]',
                                 (None, 200),                     'lstm[0][1]',
                                 (None, 200)]                     'lstm[0][2]']

 dense (Dense)                  (None, None, 2506)   503706      ['lstm_1[0][0]']

==================================================================================================
Total params: 1,439,306
Trainable params: 1,439,306
Non-trainable params: 0
```



```
                                                  Inference Time Encoder Model Summary.
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None)]            0

 embedding (Embedding)       (None, None, 100)         203400

 dropout (Dropout)           (None, None, 100)         0

 lstm (LSTM)                 [(None, None, 200),       240800
                              (None, 200),
                              (None, 200)]

=================================================================
Total params: 444,200
Trainable params: 444,200
Non-trainable params: 0
```

```
                                              Decoder Inference Model summary
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, None)]       0           []

 embedding_1 (Embedding)        (None, None, 100)    250600      ['input_2[0][0]']

 dropout_1 (Dropout)            (None, None, 100)    0           ['embedding_1[0][0]']

 input_3 (InputLayer)           [(None, 200)]        0           []

 input_4 (InputLayer)           [(None, 200)]        0           []

 lstm_1 (LSTM)                  [(None, None, 200),  240800      ['dropout_1[0][0]',
                                 (None, 200),                     'input_3[0][0]',
                                 (None, 200)]                     'input_4[0][0]']

 input_5 (InputLayer)           [(None, None, 200)]  0           []

 dense (Dense)                  (None, None, 2506)   503706      ['lstm_1[1][0]']

==================================================================================================
Total params: 995,106
Trainable params: 995,106
Non-trainable params: 0
```

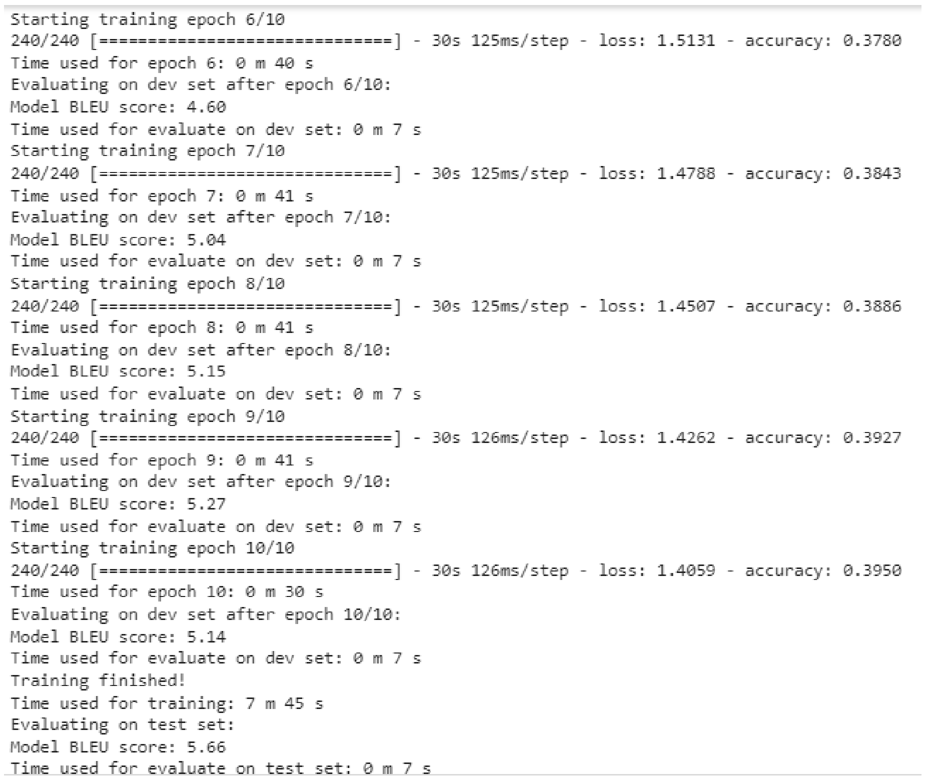
Figure 3: Summary of the model

Once we have implemented the code, we proceed to evaluate how the system works without using the Attention layer. To evaluate the performance of a NMT, adequacy and fluency is considered. The former one will check if the output conveys the same meaning as the input sentence. While the later one, involves grammatical correctness and idiomatic word choices.

All of this can be judged using the Bilingual Evaluation Understudy (BLEU). This metric will estimate a similarity score comparing the machine translation to one or several human translations. The score computation will be based on:

- n-gram precision
- Penalty for too-short system translations

As we can see in Figure 4, the last BLEU score we obtain from the training set is 5.14 and 5.66 for the test set. However, this value might improve if attention is added.

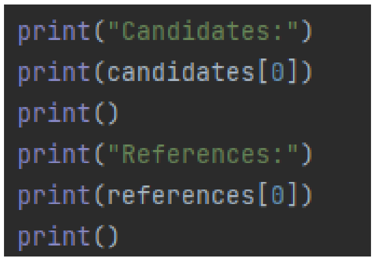


```
Starting training epoch 6/10
240/240 [==============================] - 30s 125ms/step - loss: 1.5131 - accuracy: 0.3780
Time used for epoch 6: 0 m 40 s
Evaluating on dev set after epoch 6/10:
Model BLEU score: 4.60
Time used for evaluate on dev set: 0 m 7 s
Starting training epoch 7/10
240/240 [==============================] - 30s 125ms/step - loss: 1.4788 - accuracy: 0.3843
Time used for epoch 7: 0 m 41 s
Evaluating on dev set after epoch 7/10:
Model BLEU score: 5.04
Time used for evaluate on dev set: 0 m 7 s
Starting training epoch 8/10
240/240 [==============================] - 30s 125ms/step - loss: 1.4507 - accuracy: 0.3886
Time used for epoch 8: 0 m 41 s
Evaluating on dev set after epoch 8/10:
Model BLEU score: 5.15
Time used for evaluate on dev set: 0 m 7 s
Starting training epoch 9/10
240/240 [==============================] - 30s 126ms/step - loss: 1.4262 - accuracy: 0.3927
Time used for epoch 9: 0 m 41 s
Evaluating on dev set after epoch 9/10:
Model BLEU score: 5.27
Time used for evaluate on dev set: 0 m 7 s
Starting training epoch 10/10
240/240 [==============================] - 30s 126ms/step - loss: 1.4059 - accuracy: 0.3950
Time used for epoch 10: 0 m 30 s
Evaluating on dev set after epoch 10/10:
Model BLEU score: 5.14
Time used for evaluate on dev set: 0 m 7 s
Training finished!
Time used for training: 7 m 45 s
Evaluating on test set:
Model BLEU score: 5.66
Time used for evaluate on test set: 0 m 7 s
```

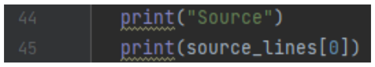
Figure 4: Results without attention

Additionally, it was asked to output one sample. Therefore, the code in Figure 5 was added. Figure 5a represents the code for printing the first source sentence in the dataset, and it will be included in the load_dataset() function. While Figure 5b is the code needed to print first candidates and the references (included in th eval() function). The former one represents machine-written translation from our NMT, and the later one, the human-written translation that will be compared to in order to compute the similarity score.

The printed results can be seen in Figure 6. The source word translation according to Google Translator should be: *"The science behind a climate headline"*. However, the translation is not accurate enough. The build NMT seems to struggle to make a good translation, most of the output tokens are unknown, what suggests that adding some contextual relations may improve the performance.



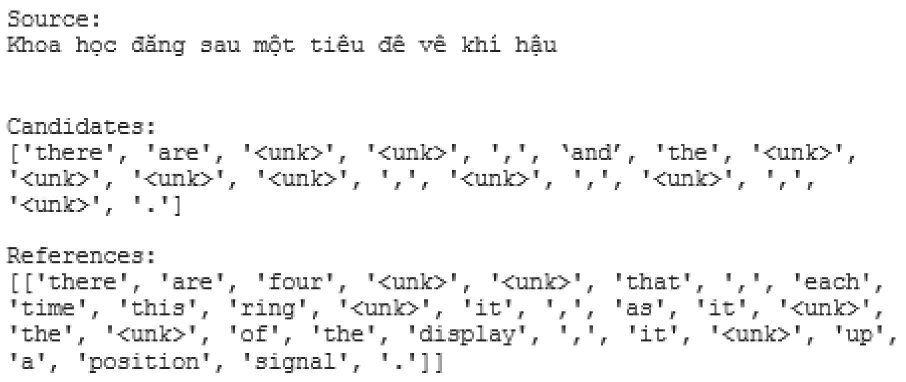
```
print("Candidates:")
print(candidates[0])
print()
print("References:")
print(references[0])
print()
```



```
print("Source")
print(source_lines[0])
```

(a) Print source. (b) Print candidate and reference.

Figure 5: Code for printing output sample.



```
Source:
Khoa học đăng sau một tiêu đê vê khí hậu


Candidates:
['there', 'are', '<unk>', '<unk>', ',', 'and', 'the', '<unk>',
'<unk>', '<unk>', '<unk>', ',', '<unk>', ',', '<unk>', ',',
'<unk>', '.']

References:
[['there', 'are', 'four', '<unk>', '<unk>', 'that', ',', 'each',
'time', 'this', 'ring', '<unk>', 'it', ',', 'as', 'it', '<unk>',
'the', '<unk>', 'of', 'the', 'display', ',', 'it', '<unk>', 'up',
'a', 'position', 'signal', '.']]
```

Figure 6: Output sample without attention

4 Adding attention.

When it comes to long length sentences, the previous NMT will not be able to extract strong contextual relations from long semantic sentences, what affects to the performance of the model.

Therefore, to upgrade the accuracy of the model, Attention is added to the network. For that, the decoder will focus on a particular part of the input sentence and then relate them to elements in the output sequence when predicting the output at each time step in the output sequence.

The steps that the Attention operation perform are the following:

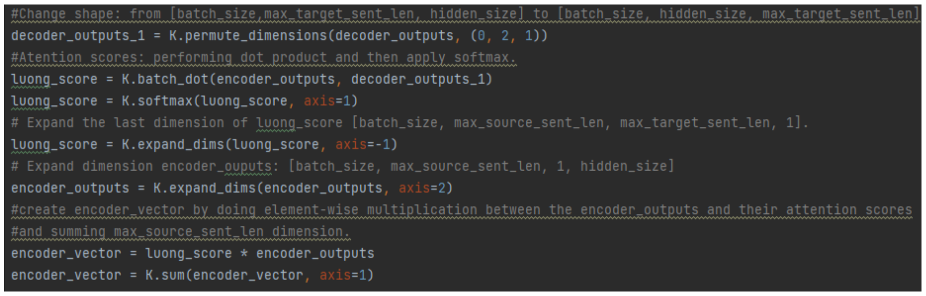
- The context vector (output of the encoder) and the decoder output value of the previous time step are used to compute the attention scores. The function used to calculate the score is the dot product.

- Once we have computed the dot product, the scores are normalised using a softmax function. This will hep to turn the scores into a probability distribution, indicating the likelihood of each encoded input time step being relevant to the current output time step or not.

- Use the attention distribution to take a weighted sum of the encoder hidden states. The attention output mostly contains information the hidden states that received high attention.

- Concatenate attention output with decoder hidden state, then use to compute the output as in the previous section

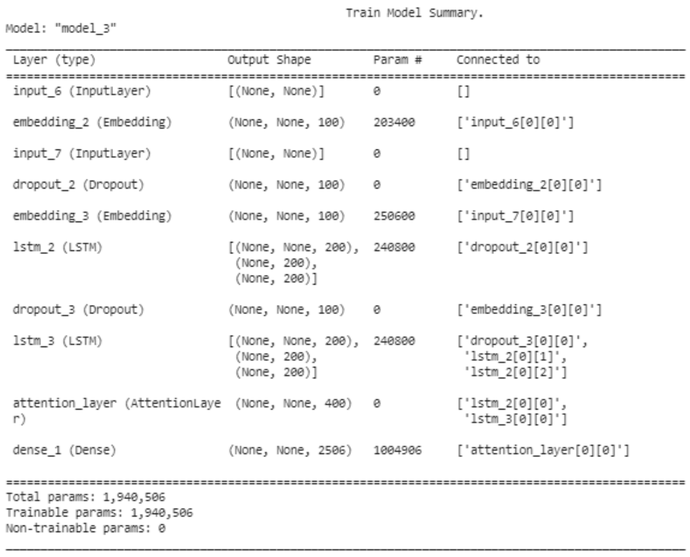
Taking all this into account, we can now implement the code (Figure 7 for this part of the assignment.



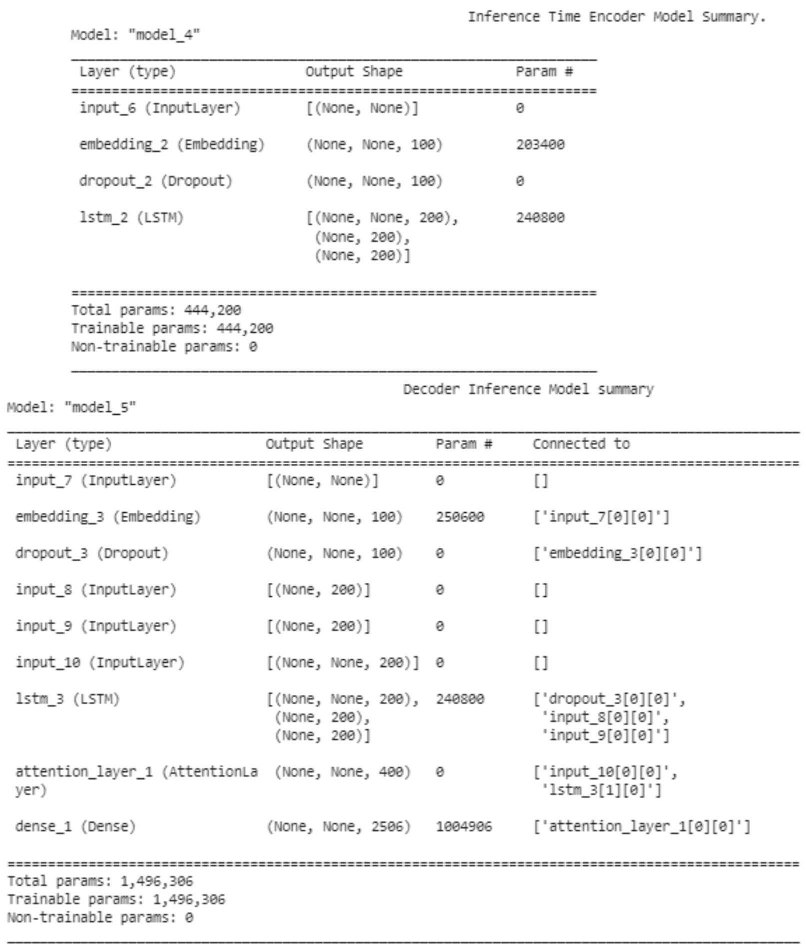
```
#Change shape: from [batch_size,max_target_sent_len, hidden_size] to [batch_size, hidden_size, max_target_sent_len]
decoder_outputs_1 = K.permute_dimensions(decoder_outputs, (0, 2, 1))
#Atention scores: performing dot product and then apply softmax.
luong_score = K.batch_dot(encoder_outputs, decoder_outputs_1)
luong_score = K.softmax(luong_score, axis=1)
# Expand the last dimension of luong_score [batch_size, max_source_sent_len, max_target_sent_len, 1].
luong_score = K.expand_dims(luong_score, axis=-1)
# Expand dimension encoder_ouputs: [batch_size, max_source_sent_len, 1, hidden_size]
encoder_outputs = K.expand_dims(encoder_outputs, axis=2)
#create encoder_vector by doing element-wise multiplication between the encoder_outputs and their attention scores
#and summing max_source_sent_len dimension.
encoder_vector = luong_score * encoder_outputs
encoder_vector = K.sum(encoder_vector, axis=1)
```

Figure 7: Attention code

The next step is to run the code and analyse the results. Before that, in Figure 8 a summary of the model is presented.



```
                                        Train Model Summary.
Model: "model_3"
____________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
====================================================================================================
 input_6 (InputLayer)           [(None, None)]       0           []

 embedding_2 (Embedding)        (None, None, 100)    203400      ['input_6[0][0]']

 input_7 (InputLayer)           [(None, None)]       0           []

 dropout_2 (Dropout)            (None, None, 100)    0           ['embedding_2[0][0]']

 embedding_3 (Embedding)        (None, None, 100)    250600      ['input_7[0][0]']

 lstm_2 (LSTM)                  [(None, None, 200),  240800      ['dropout_2[0][0]']
                                 (None, 200),
                                 (None, 200)]

 dropout_3 (Dropout)            (None, None, 100)    0           ['embedding_3[0][0]']

 lstm_3 (LSTM)                  [(None, None, 200),  240800      ['dropout_3[0][0]',
                                 (None, 200),                     'lstm_2[0][1]',
                                 (None, 200)]                     'lstm_2[0][2]']

 attention_layer (AttentionLaye  (None, None, 400)   0           ['lstm_2[0][0]',
 r)                                                               'lstm_3[0][0]']

 dense_1 (Dense)                (None, None, 2506)   1004906     ['attention_layer[0][0]']

====================================================================================================
Total params: 1,940,506
Trainable params: 1,940,506
Non-trainable params: 0
____________________________________________________________________________________________________
```



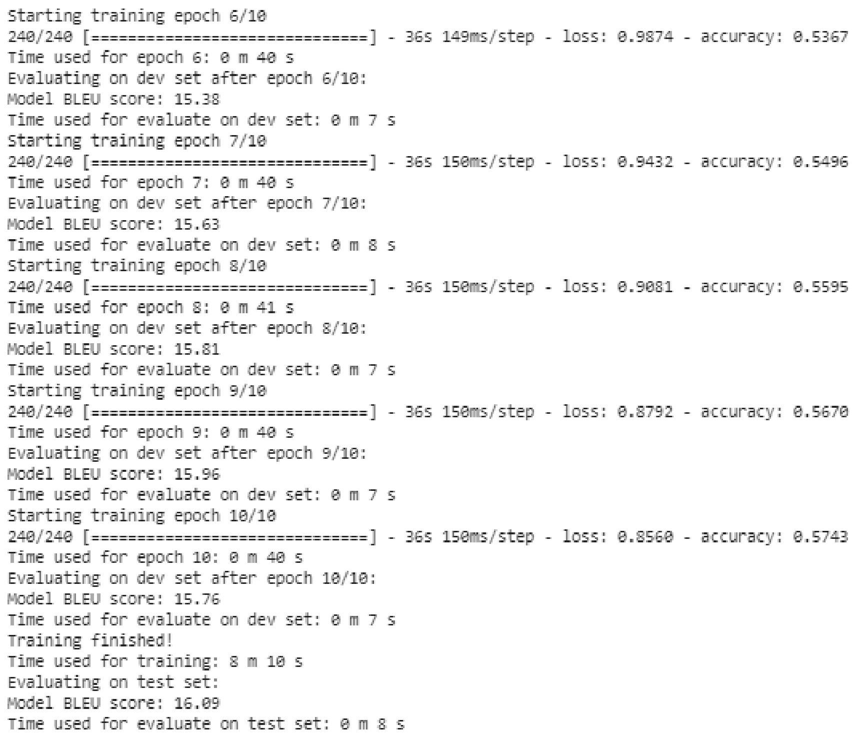
```
                                            Inference Time Encoder Model Summary.
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, None)]            0

 embedding_2 (Embedding)     (None, None, 100)         203400

 dropout_2 (Dropout)         (None, None, 100)         0

 lstm_2 (LSTM)               [(None, None, 200),       240800
                              (None, 200),
                              (None, 200)]

=================================================================
Total params: 444,200
Trainable params: 444,200
Non-trainable params: 0
_________________________________________________________________
```

```
                                          Decoder Inference Model summary
Model: "model_5"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_7 (InputLayer)           [(None, None)]       0           []

 embedding_3 (Embedding)        (None, None, 100)    250600      ['input_7[0][0]']

 dropout_3 (Dropout)            (None, None, 100)    0           ['embedding_3[0][0]']

 input_8 (InputLayer)           [(None, 200)]        0           []

 input_9 (InputLayer)           [(None, 200)]        0           []

 input_10 (InputLayer)          [(None, None, 200)]  0           []

 lstm_3 (LSTM)                  [(None, None, 200),  240800      ['dropout_3[0][0]',
                                 (None, 200),                     'input_8[0][0]',
                                 (None, 200)]                     'input_9[0][0]']

 attention_layer_1 (AttentionLa  (None, None, 400)   0           ['input_10[0][0]',
 yer)                                                             'lstm_3[1][0]']

 dense_1 (Dense)                (None, None, 2506)   1004906     ['attention_layer_1[0][0]']

==================================================================================================
Total params: 1,496,306
Trainable params: 1,496,306
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 8: Summary of the model with attention

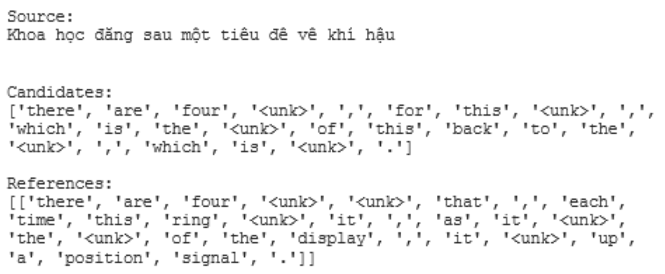
Figure 9, the results obtained for the last training epoch and the testing set are observable.The training set reaches a BLEU score of 15.76 and the test set score is 16.09. In both cases, the BLEU score has increased significantly (5 times) in comparison to the NMT without attention. We can conclude that in seq2seq models, the attention mechanism does make a difference. Therefore, we can say that exploring contextual relations with high semantic significance and creating attention-based scores to filter particular words help to extract the primary weighted features and consequently improve the performance of our model.

A good way of seeing the improvement of the model is Figure 10, where we can see that in comparison to Figure 6, the candidate is more similar to the reference and less words are tagged as unknown. Nevertheless, further improvements need to be done in order to get a more accurate translation.



```
Starting training epoch 6/10
240/240 [==============================] - 36s 149ms/step - loss: 0.9874 - accuracy: 0.5367
Time used for epoch 6: 0 m 40 s
Evaluating on dev set after epoch 6/10:
Model BLEU score: 15.38
Time used for evaluate on dev set: 0 m 7 s
Starting training epoch 7/10
240/240 [==============================] - 36s 150ms/step - loss: 0.9432 - accuracy: 0.5496
Time used for epoch 7: 0 m 40 s
Evaluating on dev set after epoch 7/10:
Model BLEU score: 15.63
Time used for evaluate on dev set: 0 m 8 s
Starting training epoch 8/10
240/240 [==============================] - 36s 150ms/step - loss: 0.9081 - accuracy: 0.5595
Time used for epoch 8: 0 m 41 s
Evaluating on dev set after epoch 8/10:
Model BLEU score: 15.81
Time used for evaluate on dev set: 0 m 7 s
Starting training epoch 9/10
240/240 [==============================] - 36s 150ms/step - loss: 0.8792 - accuracy: 0.5670
Time used for epoch 9: 0 m 40 s
Evaluating on dev set after epoch 9/10:
Model BLEU score: 15.96
Time used for evaluate on dev set: 0 m 7 s
Starting training epoch 10/10
240/240 [==============================] - 36s 150ms/step - loss: 0.8560 - accuracy: 0.5743
Time used for epoch 10: 0 m 40 s
Evaluating on dev set after epoch 10/10:
Model BLEU score: 15.76
Time used for evaluate on dev set: 0 m 7 s
Training finished!
Time used for training: 8 m 10 s
Evaluating on test set:
Model BLEU score: 16.09
Time used for evaluate on test set: 0 m 8 s
```

Figure 9: Attention code



```
Source:
Khoa học đăng sau một tiêu đê vê khí hậu


Candidates:
['there', 'are', 'four', '<unk>', ',', 'for', 'this', '<unk>', ',',
'which', 'is', 'the', '<unk>', 'of', 'this', 'back', 'to', 'the',
'<unk>', ',', 'which', 'is', '<unk>', '.']

References:
[['there', 'are', 'four', '<unk>', '<unk>', 'that', ',', 'each',
'time', 'this', 'ring', '<unk>', 'it', ',', 'as', 'it', '<unk>',
'the', '<unk>', 'of', 'the', 'display', ',', 'it', '<unk>', 'up',
'a', 'position', 'signal', '.']]
```

Figure 10: Output sample with attention.

5 Bibliography.

1 I. Sutskever, O. Vinyals and Q. V. Le, ”Sequence to sequence learning with neural networks.,” Proceedings of the 27th International Conference on Neural Information Processing Systems, vol. 2 (NIPS’14), p. 3104–3112, 2014.

2 K. Cho, B. Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk and Y. Bengio, ”Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.,” Conference on Empirical Methods in Natural Language Processing, 2014.