Sacred-text Intelligent Coding and Embedding for Multilingual(SICEM) Neural Machine Translation

AGM Islam, Nishan Karki, Razwan Ahmed Tanvir, Swapnil Saha

12/08/21

1 Abstract

Neural Machine Translation has varieties of applications in various domains and there has been a lot of research performed on this field. In the paper, we propose a multi-lingual model that will be able to translate from any source language to any target language, even though the model has not been trained on the language pair before (zero shot translation). For the model, we used a simple RNN architecture and implemented global attention layer as described in the paper [4]. For training, we used Opus-100 datasets for the model. We will further discuss on how the model was built and the optimization performed on it. For simplicity, we trained our model with five pair of language sets and evaluated the results.

2 Introduction

NMT (Neural Machine Translation) has been widely used for language translation and has produced some state of art results. The encoder/decoder model of NMT makes it a good candidate for the translation and is simple to implement. With the introduction of the attention mechanism, the effectiveness of the NMT model has improved extensively, which encouraged us to deploy the model architecture. The SICEM Neural Machine Translation project implements an RNN model with an additive attention layer. The model is trained using the language from the datasets in multiple languages, to translate to desired language. The input of the model will be a token of the target language and a input sentence, which is a sequence of words. The output will be the translation of the sentence in the desired language as shown below.

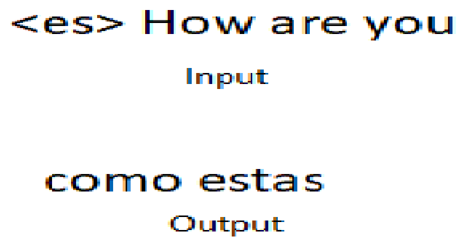


Figure 1: Sample sentence language pair.

3 Related Work

The NLP deploys different implementation of algorithms to draw out the meaningful information from the text source. These algorithms are implemented through machine learning models. RNN and CNN have been a state of art implementation for NLP tasks. Though, RNN has been a easy choice for NLP problem, it has difficulties while predicting longer texts. To alleviate the shortcomings of RNN, LSTM has been introduced. Though, LSTM mitigated the problem with RNN, the initial issues of long term dependencies and parallelization persisted. With the introduction of attention mechanism, the problem with long term dependencies were minimized. Different global and local attention mechanism has since been implemented to generate some improved results in language translation.

4 Dataset Collection

For our training data, we used Opus-100 dataset which is being downloaded from opus website (https://opus.nlpl.eu). The Opus represents more than 100 languages and more than fifty five millions of pair of sentences. For our training purposes, we used the following language pairs.

- English to Portuguese
- English to Spanish
- English to French
- English to Turkish
- English to Italian

We have taken 75,000 sentences from each of the languages. We combined all the files into a single file containing all the five language pairs and 375,000 sentence altogether. This file has been named as "source.txt":

- $< 2es >$ I am going to school
- $< 2pt >$ No one is at home
- $< 2it >$ This is my life
- $< 2tr >$ I like food

Similarly, "target.txt" file has been created which will contain the translation of the source languages.

5 Data pre-processing

For the data-preprocessing, we imported the data from text files into string array. We converted the dataset into a numpy array and the tensorflow text functions were used to pre-process the dataset. The dataset has been converted into lowercase for uniformity. We have also remove whitespace, special characters and duplicate sentences. We added "start" and "end" token to each of the sentences.

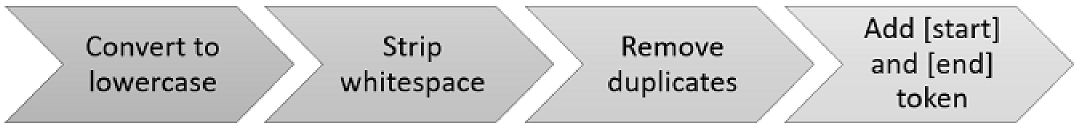
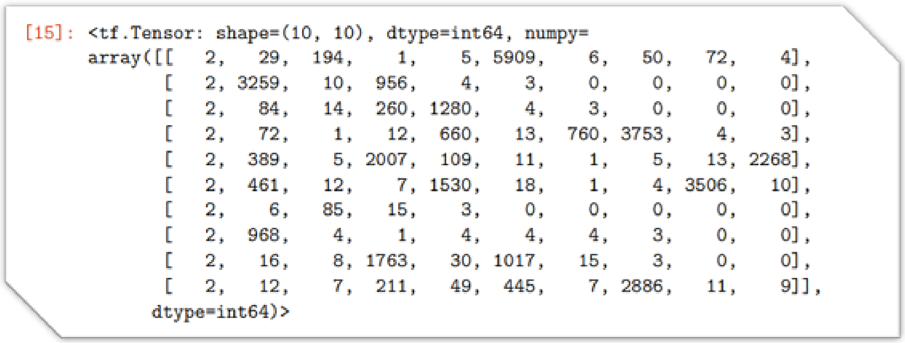


Figure 2: Standardization

After data was standardized, we built a vocabulary for the input and target sentences. We used the textvectorization.getvocabulary() function from tensorflowtext to build the vocabulary. The vocabulary consists of all the unique words of the input and target datasets. We set our vocab size to 20,000. This number was generated using the textvectorization.getvocabularysize() from tensorflow text. We estimated each language to have around 4,000 unique vocab words.

For the vectorization, we used tensorflow.textvectorization to generate token IDs' for each sentence. After Tokenization , data looks like below:



```
[15]: <tf.Tensor: shape=(10, 10), dtype=int64, numpy=
      array([[   2,   29,  194,    1,    5, 5909,    6,   50,   72,    4],
             [   2, 3259,   10,  956,    4,    3,    0,    0,    0,    0],
             [   2,   84,   14,  260, 1280,    4,    3,    0,    0,    0],
             [   2,   72,    1,   12,  660,   13,  760, 3753,    4,    3],
             [   2,  389,    5, 2007,  109,   11,    1,    5,   13, 2268],
             [   2,  461,   12,    7, 1530,   18,    1,    4, 3506,   10],
             [   2,    6,   85,   15,    3,    0,    0,    0,    0,    0],
             [   2,  968,    4,    1,    4,    4,    4,    3,    0,    0],
             [   2,   16,    8, 1763,   30, 1017,   15,    3,    0,    0],
             [   2,   12,    7,  211,   49,  445,    7, 2886,   11,    9]],
            dtype=int64)>
```

Figure 3: After Tokenization

The output vector after tokenization has shape equals to the batch size and length of the longest sentence in the input dataset. For instance: vector shape = (length of the longest sentence, batch size = 128). Every sentence in the dataset was padded with 0 to maintain the vector shape.

```
[18]: Text(0.5, 1.0, 'Mask')
```

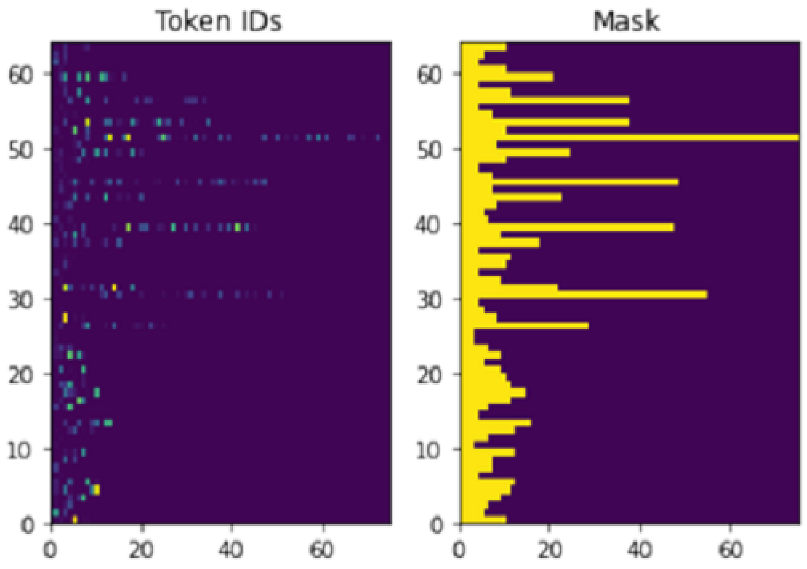


Figure 4: Masking

6 Model Creation

For the model selection, we researched on different sequence to sequence models. We decided to use a RNN model with encoder ad decoder with a global attention layer, as inspired by the paper "Effective Approaches to Attention-based Neural Machine Translation" [4]. The model consists of three parts.

- Encoder layer
- Attention layer
- Decoder layer

We used a RNN architecture as implemented below.

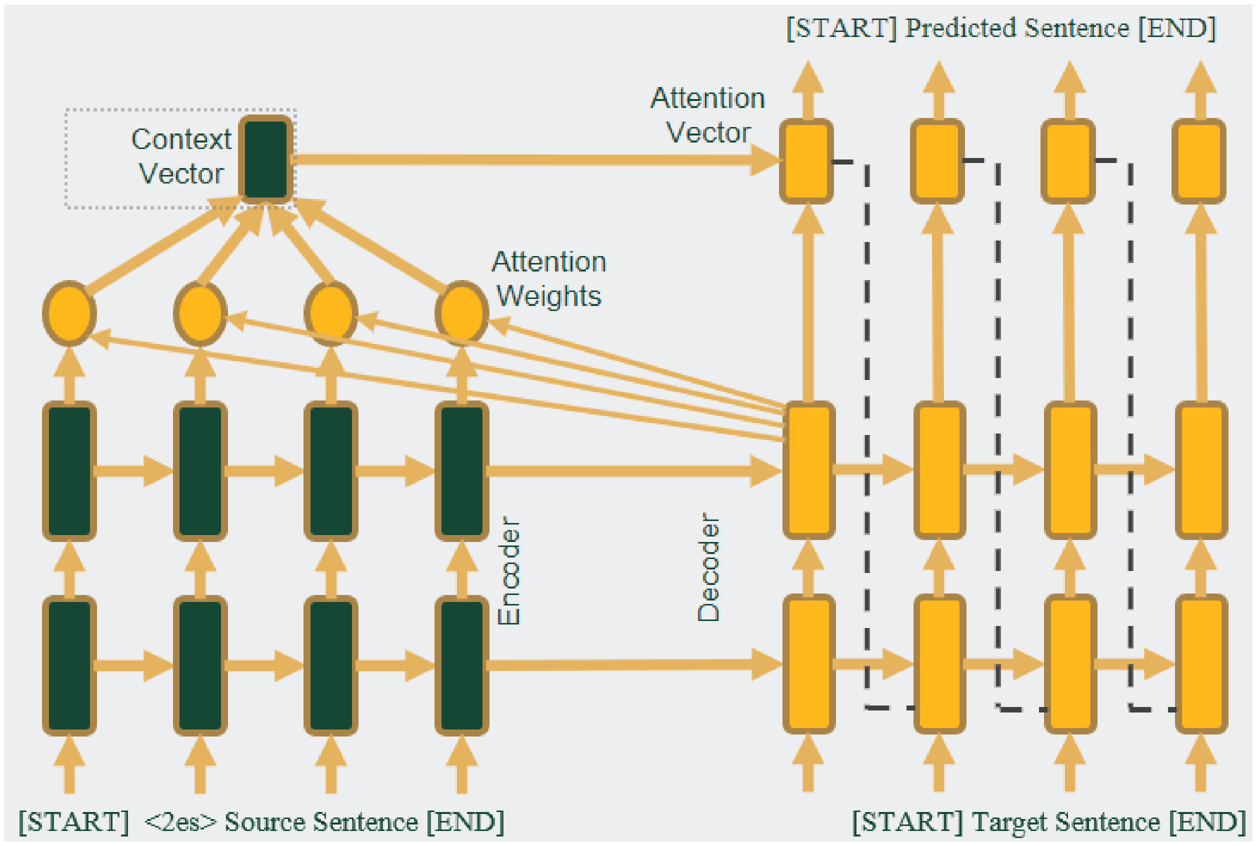

[START] Predicted Sentence [END]
Attention Vector
Context Vector
Attention Weights
Encoder
Decoder
[START] <2es> Source Sentence [END]
[START] Target Sentence [END]

Figure 5: Model Architecture

6.1 Encoder Layer

We have two layers in our encoder

- Embedding layer
- GRU layer

The token that has been fed into the encoder, embedding layer creates an embedding vector using the token. Then the GRU layer outputs the state and the sequence using the output space dimensionality. This is a kind of RNN layer that has activation function ”Tanh” and only holds the memory of the previous state. It trains faster and gives better result than LSTM and therefore, we preferred to use this implementation.

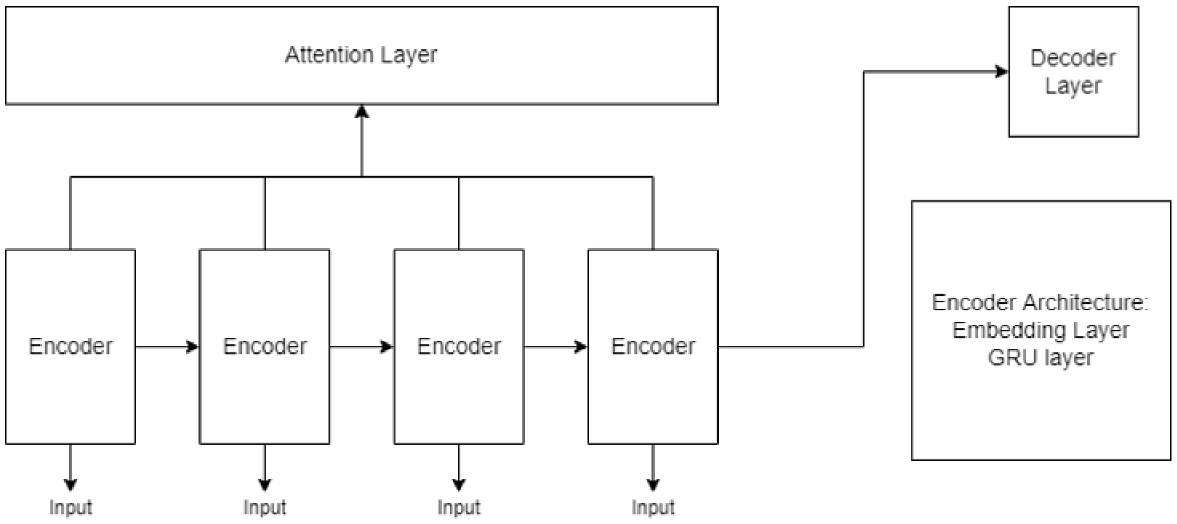


Figure 6: Encoder Architecture

6.2 Attention Layer

The decoder uses the attention weights to focus on parts of the input sentence. Let us consider the following equation to understand how it works.
The following equation is for the attention weights.

$$\alpha = \frac{exp(score(h_t, \overline{h_s}))}{\sum_{s'=1}^{S} exp(score(h_t, \overline{h_{s'}}))} \tag{1}$$

The following equation is for the context vector.

$$c_t = \sum_s \alpha_{ts} \overline{h_s} \tag{2}$$

s and t is the encoder and the decoder index respectively.
attention weights $= \alpha_{ts}$
The key, value pair of the attention key $= h_t$
context vector $= c_t$
Combination of output and the query $= \alpha_t$

α_{ts} is the softmax for the encoder throughout it's output sequence.

Let's look at the score function now.

$$score(h_t, \overline{h_s}) = v_a^T tanh(W_1 h_t + W_2 \overline{h_s}) \tag{3}$$

The attention layer predicts one word at a time for the word it is being fed by processing all the hidden state and the encoder output.

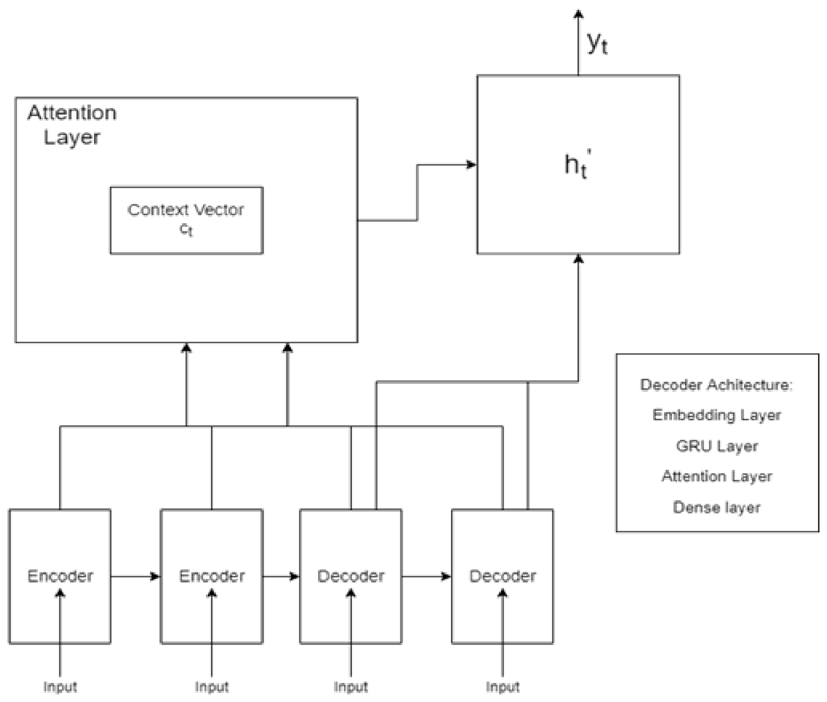


Figure 7: Attention layer Architecture

6.3 Decoder Layer

This layer is responsible for predicting the output tokens. The decoder runs in a loop in this model rather than calling it once. The encoder output is fed into the decoder. RNN keeps track of what has been generated this far. RNN output is used by the decoder to produce the context vector. Then it combines encoder output and the context vector to product attention vector which is then used to predict the next token.

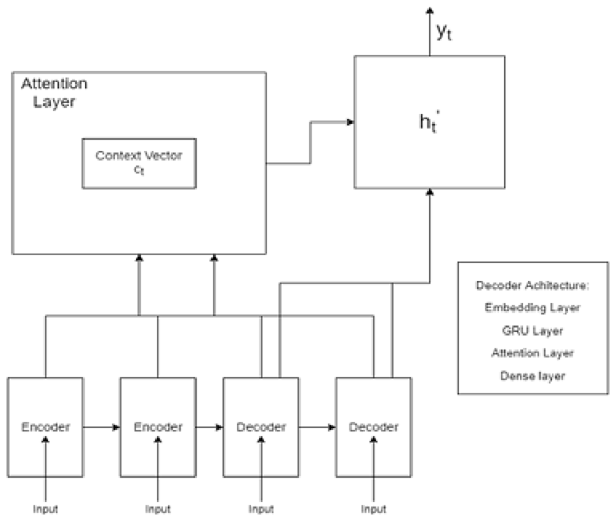


Figure 8: Decoder layer Architecture

7 Train the Model

For training our model, we trained our model with 375,000 pairs of languages where five languages are represented with 75,000 pairs each. We trained our model with a batch size of 128 and model was trained for 3 epochs. We used Categorical Cross entropy as our loss function as our data is sparse. For, the optimizer, we have used adam. And the batch size is 128. Here is the summary of the training of the model.

- Take inputs
- Vectorize the text and convert into mask
- Feed them into the encoder in batch
- Loop through each of the target token
  - Decoder will run at one step at a time
  - SparseCategoricalCrossentropy loss will be measured at each of the step
  - Calculate the average loss
- Using the adam optimizer, we try to minimize the overall loss.

To run for 3 epochs it took almost 17.8 hours to get trained. Following is the graph of convergence.

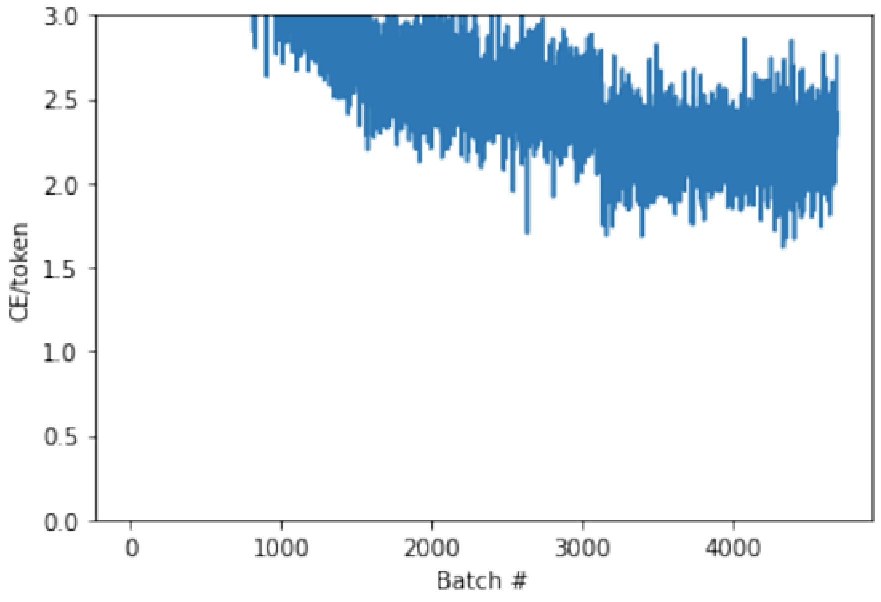


Figure 9: Training

The final batch loss is 2.8135. Due to hardware limitations, we could not run in for more epochs. We could have further minimized the loss running the model for greater epochs. Once the model is finished training, we saved it for further use, so that we no longer need to train it again unless new data is required to be fed.

8 Experiments and Results

After training the model, we can do some experiments with some inputs. From the experiment, we can see that for the small sentences, the model worked reasonably well but for the long sentences the model's focus is distributed and the prediction is not that good. Following is a sample prediction for a small sentence.

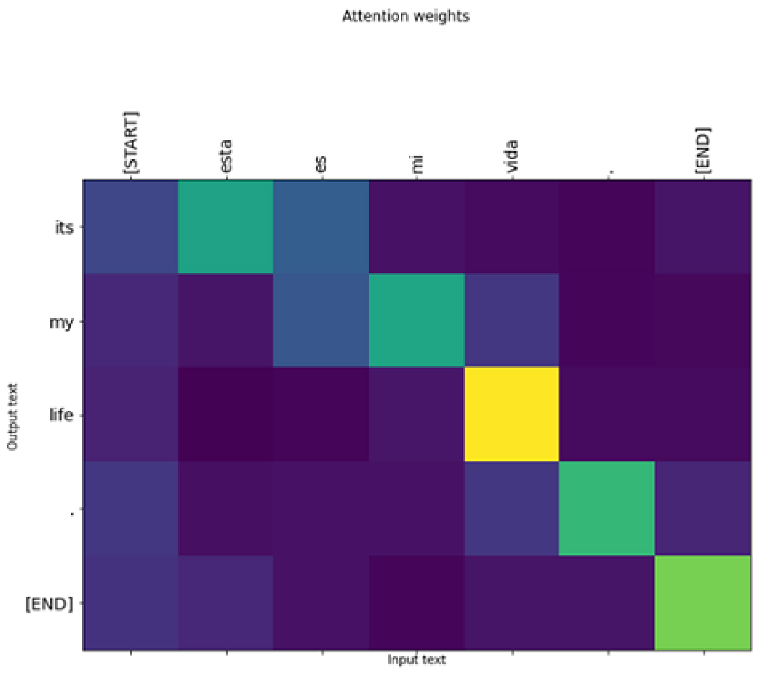


Figure 10: Small Sentence

Following is a sample prediction for long sentence.

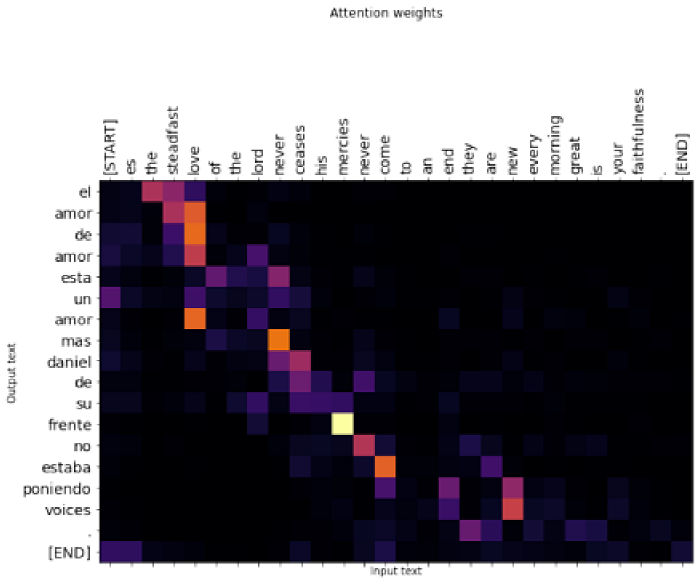
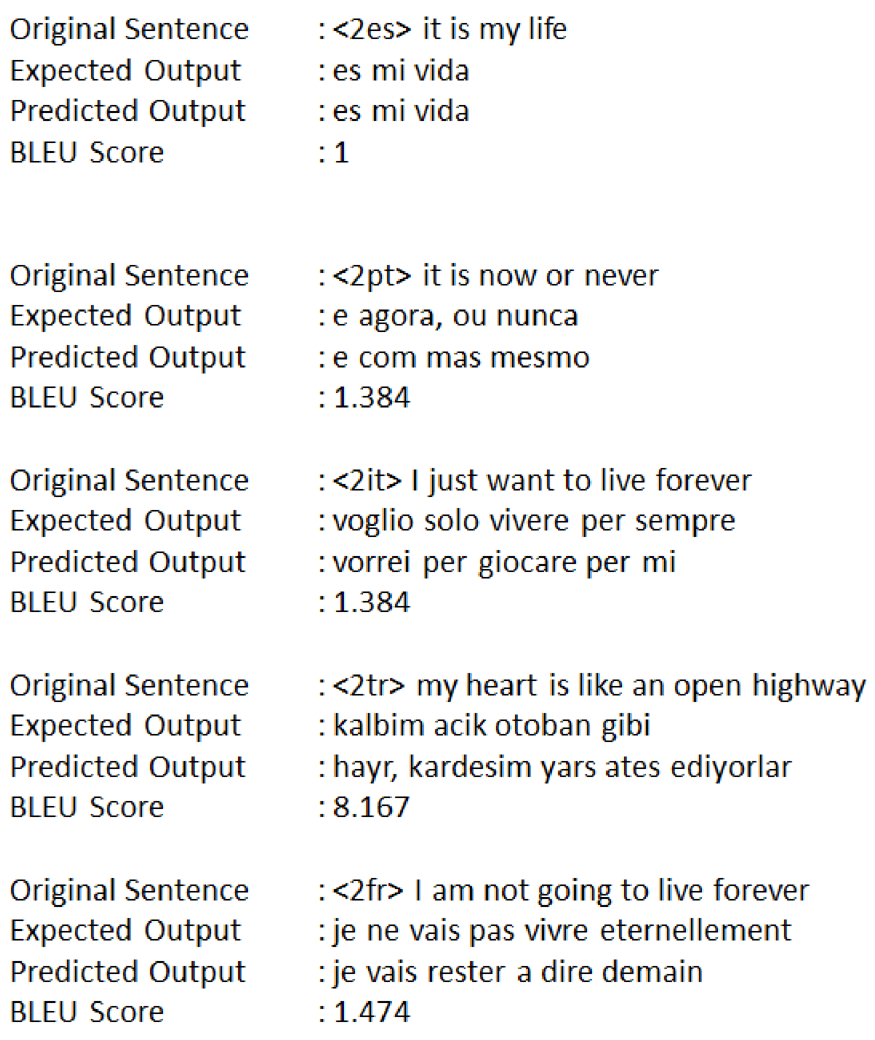


Figure 11: Long Sentence

Following are the BLEU score for translation of different languages.



Original Sentence	: <2es> it is my life
Expected Output	: es mi vida
Predicted Output	: es mi vida
BLEU Score	: 1

Original Sentence	: <2pt> it is now or never
Expected Output	: e agora, ou nunca
Predicted Output	: e com mas mesmo
BLEU Score	: 1.384

Original Sentence	: <2it> I just want to live forever
Expected Output	: voglio solo vivere per sempre
Predicted Output	: vorrei per giocare per mi
BLEU Score	: 1.384

Original Sentence	: <2tr> my heart is like an open highway
Expected Output	: kalbim acik otoban gibi
Predicted Output	: hayr, kardesim yars ates ediyorlar
BLEU Score	: 8.167

Original Sentence	: <2fr> I am not going to live forever
Expected Output	: je ne vais pas vivre eternellement
Predicted Output	: je vais rester a dire demain
BLEU Score	: 1.474

Figure 12: BLEU Score

9 Conclusion

In conclusion, we presented a model which has been trained with five datasets. We can feed more datasets into the model for improved translations. Moreover, increased number of epochs could bring better results. For the intensive learning of our model, powerful hardware could be used so that the training time could be reduced when the model will be trained with lots of data. Furthermore, we can also improve the model by incorporating the self-attention in the encoder and decoder. Currently, the model has access to its previous RNN state output. Finally, we want to build a website where users will be able to read the Bible in any language as per the user's preferences and the translation will be generated with our SICEM Neural Machine Translation.

10 Running the model

The model was created in the Jupyter Notebook.
To run the model:

1. Copy the attached folder to the local PC: DM Final-Team Bear Grills
2. Open Jupyter Notebook
3. Open "DMFinalProjectTeamBearGrills.ipyn"b in Jupyter Notebook

To load the saved model

1. Go to the "Loading a saved model" section at end of the .ipynb file
2. Run "reloadedtranslator" to load the translator folder
3. All files are included in the submitted folder.

Colab Links:

1. Colab Link
2. Dataset Link
3. Writeup Link

References

[1] Aharoni, Roee, Melvin Johnson, and Orhan Firat. "Massively multilingual neural machine translation." arXiv preprintarXiv:1903.00089 (2019).

[2] Johnson, Melvin, et al. "Google's multilingual neural machine translation system: Enabling zero-shot translation." Trans-actions of the Association for Computational Linguistics 5 (2017): 339-351.

[3] arXiv:1706.03762 [cs.CL]

[4] Minh-Thang Luong, Hieu Pham, Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation"