Neural Machine Translation by Jointly Learning to Align and Translate

BENAISSA Tarek, BENBRAHIM Mohammed El Amine, BENSLIMANE Zahra Hafida, BENABID M'hamed Djahid

Editor:

Abstract

As a part of our Advanced Machine Learning project we re-implemented a neural machine translation, the paper introduced a model that translates one sentence from a source language into a sentence in a target language.

This model consists of 3 major parts, an encoder that reads the input sentence and encodes it into a sequence of vectors of variable length, and a decoder that outputs the translated sentence in the target language and the attention, placed in between the previous parts to take in consideration the context of a word in the input sentence depending on its previous and next words.

1. Introduction

Neural machine translation is a task that is becoming more and more essential for many industries of today's world. The overwhelming majority of the machine translation models are of the encoder-decoder type.

Till now, we have known 3 types of models of the machine translation task which are a sequence to sequence based on encoder-decoder model, a sequence to sequence with attention and finally, state of art model known as transformers. Researched were looking for a solution to bypass the fixed length limitation and try to have better results using a variable length context vector.

Our role in this project is to reverse engineer the results obtained in the paper "NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE" by Bahdanau, Cho and Bengio. The paper introduces an innovation to the classic encoder-decoder model by learning to align and translate jointly, using a variable length attention vector while the encoder-decoder based model uses a fixed length attention vector corresponding to the last hidden states of the encoder.

2. Neural Machine Translation

The global of the model proposed by the paper consists of 3 main parts. the encoder, the attention model and the decoder. the graphical illustration of the model is shown below :

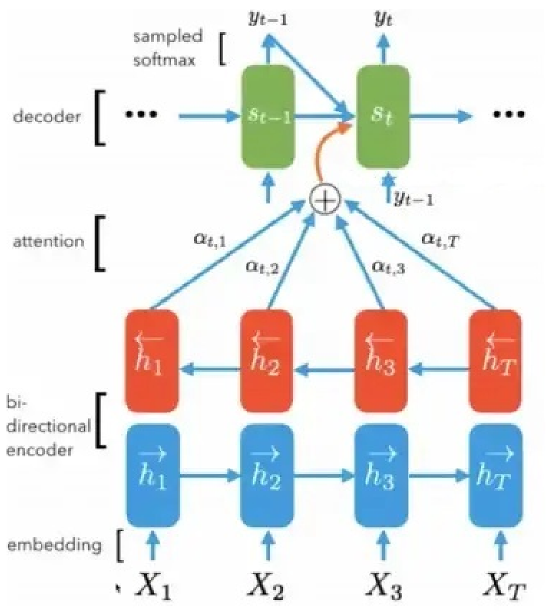

sampled softmax
decoder
attention
bi-directional encoder
embedding
y_{t-1}
y_t
s_{t-1}
s_t
$\alpha_{t,1}$
$\alpha_{t,2}$
$\alpha_{t,3}$
$\alpha_{t,T}$
$\overleftarrow{h}_1$ $\overleftarrow{h}_2$ $\overleftarrow{h}_3$ $\overleftarrow{h}_T$
$\overrightarrow{h}_1$ $\overrightarrow{h}_2$ $\overrightarrow{h}_3$ $\overrightarrow{h}_T$
X_1 X_2 X_3 X_T

FIGURE 1 – Neural Machine Translation with attention overview

2.1 Encoder

The encoder is a recursive neural network (RNN) that reads an input sequence x starting from the first word to the last one.

However, in our model we have used a Bidirectional RNN which is an RNN that once run forwards over the input sequence from left to right and a backward that reads the input from the opposite direction, we could use many types of architecture such as RNN cells, LSTM cells or GRU, in our case we have used GRU cells that returns only the hidden states for all sequence while RNN and LSTM have output where it is unnecessary in our case.

The bidirectional RNN leads to output two hidden states for each time-step,the forward hidden state contain information from earlier in the sequence and the backward from later in the sequence, by concatenating the two hidden states it produces an annotation h_j for each word.
the annotation is expressed by :

$$h_j = \begin{bmatrix} \vec{h}_j \\ \overleftarrow{h}_j \end{bmatrix}$$

Where the forward hidden state is given by :

$$\vec{h}_j = \begin{cases} (1 - \vec{z}_j) \circ \vec{h}_{j-1} + \vec{z}_j \circ \vec{h}_j & , \text{ if } j > 0 \\ 0 & , \text{ if } j = 0 \end{cases}$$

the annotation will be used for alignment and the decoder later.

2.2 Attention

One of the problems in machine translation is the alignment because it identifies which part of the input sequence can match with each word in the output, this problem could be solved with many types of attention such as luong attention or the dot-product attention but in the article the researchers created a new type of attention (Bahdanau) so we have used it.

Bahdanau attention is an interface between the encoder and decoder that provides the decoder with information about the current output state using every encoder hidden state. With this setting, the model is able to selectively focus on useful parts of the input sequence and hence, learn the alignment between the input words and the corresponding words in the target sentence.

As explained in the previous section, the annotations that were made by the encoder are used to compute the context vector c_i which is a weighted sum of the annotations h_i and is given by :

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

the context vector is changing each time we process the next word in the output sequence where c_i is context vector corresponding to the current target word.
a_{ij} is the weight of each hidden state h_j, it defines the importance of the input time-step j in order to predict the right output word, we call it the attention weight, and it is computed by :

$$\alpha_j = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{ik}\right)}$$

e_{ij} is the alignment where :

$$e_{ij} = a\left(s_{i-1}, h_j\right)$$

We can interpret the values of a as it tell us how much attention the model should pay to the word in the input when generating a certain output.

2.3 Decoder

The context vector that is obtained in the previous step is concatenated with the previous decoder output and fed into the Decoder RNN cell to produce a new hidden state. Then,the process repeats itself for each time step of the decoder until an $' < END >'$ token is produced.
The hidden state s_i of the decoder given the annotations from the encoder is computed by

$$s_i = (1 - z_i) \circ s_{i-1} + z_i \circ \tilde{s}_i,$$

where

$$\begin{aligned} \tilde{s}_i &= \tanh\left(W E y_{i-1} + U\left[r_i \circ s_{i-1}\right] + C c_i\right) \\ z_i &= \sigma\left(W_z E y_{i-1} + U_z s_{i-1} + C_z c_i\right) \\ r_i &= \sigma\left(W_r E y_{i-1} + U_r s_{i-1} + C_r c_i\right) \end{aligned}$$

To better the model during the training, we use a teacher forcing which means instead of taking the previous output of the decoder as the new input (our prediction) we use the real word of the target sequence aiming to not propagate the prediction errors as gradually as we go through translation.

We have used a GRU architecture which give us only the hidden state as output, then we obtain the word in the target language for the time step by passing this output through a Linear layer, which acts as a classifier that gives a word from the target language vocabulary based on the outputs which acts like probability.

3. Datasets

The WMT' 14 dataset contains approximately 850M words, during our research the time needed for our model to train on the totallity of the dataset would be closer to 6 days. Evidently, we chose at first to shrink the dataset to a couple hundred thousand words, to facilitate the process and make sure our python code was correct before moving into the full dataset. We chose to concatenate the Europarl dataset with the IWSLT-17 dataset to form our mini-set.

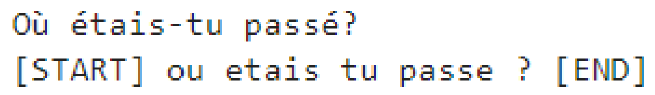
We were later able to treat th entire dataset which totaled a whopping 14GO of english sentences and their translation in french.

4. Implementation and analysis

In this part we demonstrate the re-implementation of each part of and the results obtained.

4.1 Data Processing

Before inputting the data into the encoder it should be processed and manipulated so our model can be efficient. First of all, we mixed the data to avoid having only sentences with the same length, the next step concerns languages that have accents within their alphabets like French for example, using the Unicode normalization that splits the accented characters and replace them by the equivalent ASCII code of them. However, the process was not limited only on accented characters, replacing hyphens with spaces and adding spaces around each punctuation were essential. Finally, so our model can understand the beginning and the end of our sequence we have added *Start* and *End* indicators. In the example below you can notice the difference between before and after processing a sentence in French :



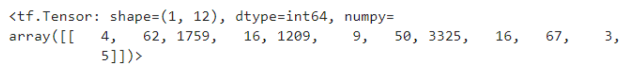
```
Où étais-tu passé?
[START] ou etais tu passe ? [END]
```

22

Figure 2 – Before and after processing a sentence

The next step of the processing is to use TextVectorization to map text features to integer sequences where it transforms a batch of strings into a list of token indices. this example shows how this sentence was tranformed into a list of token indices :

'[START] la conception est primordiale , et si vous partez des donnees numerisees sur ordinateur , il faut quelles soient extremement precises . [END] '



```
<tf.Tensor: shape=(1, 12), dtype=int64, numpy=
array([[   4,   62, 1759,   16, 1209,    9,   50, 3325,   16,   67,    3,
           5]])>
```

Figure 3 – The sentence after tokenization

4.2 Encoder

We have created an inherited class from tensorflows keras layers which has 3 attributes : $inputVocabularSize$ that indicates the number of unique words N of the source language, the embedding dimension is generally equals to 128 or 256 but since we have a big data set we decided to take 256 so the embedding vector can handle all the words and *encunits* is the number of neurons in the GRU hidden layer.

The class is constructed of 2 layers : the embedding layer takes as input the tokens given by the data processing function and as output it gives a vector of 256 that characterize each word in a unique manner, this layer will be an input for a bidirectional GRU layer, this last will give us 3 outputs : the whole sequence hidden states that will be used to compute the context vector, the last state forward and the last state backward will be given as input to the decoder.

Our network will take as input a tf.tensor of shape (BatchSize, MaxSequenceLength, tokens) The figure below shows the scheme of the encoder :

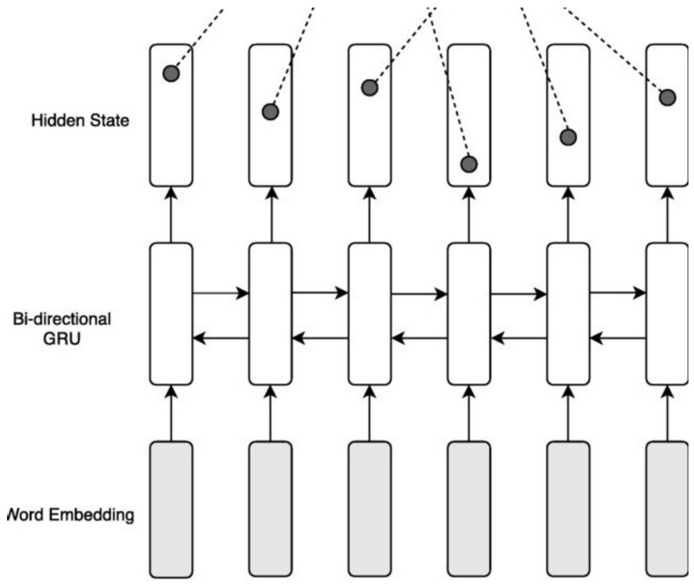


FIGURE 4 – The encoder scheme

4.3 Attention

in order to train the alignment model to obtain the context vector, we created 2 dense layers to calculate e_ij, these 2 layers are used directly in the function AdditiveAttention that was implemented by Bahdanau.

The AdditiveAttention function calculates the attention weight and the context vector. During training, we have done a padding so the sequences of all the words will have the same length as the longest one, we use a mask to take off that padding to calculate the attention only on the words of the sequences. The function takes in input the hidden state h_j of the encoder and the previous hidden state s_{i-1} of the decoder to calculate the alignment e_{ij} and finally pass through a soft max layer to compute the attention weight. these results are used to compute the context vector c_i by applying a weighted sum of the encoder hidden states for each time-step times the attention weights.

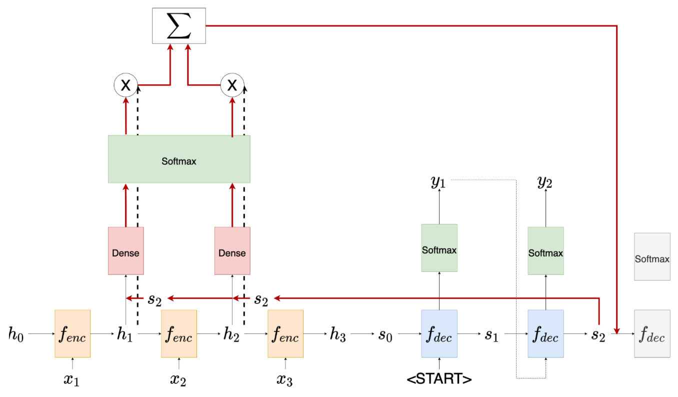


FIGURE 5 – The attention mechanism

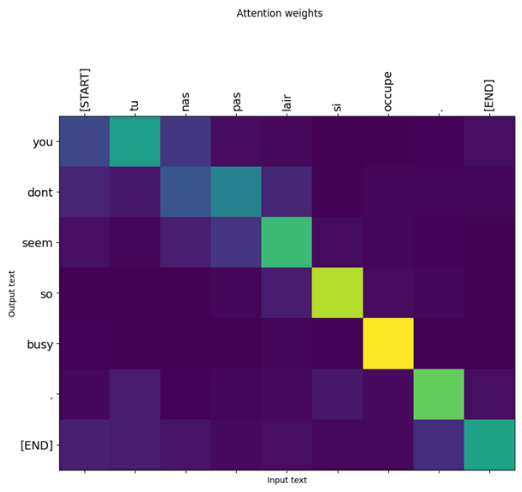
4.4 Decoder

To implement the decoder we have made an inherited class from tensorflows keras layers, the class is constructed of 5 layers : the first layer is an embedding of the target tokens resulting a vector of size 256, secondly, the previous result is inputted with the decoder previous state in an RNN layer with a GRU Architecture, however the context vector and the attention weights are recalculated again by calling the AdditiveAttention function that we have discussed in the previous section. The context vector and the attention weights are concatenated and fed into a dense layer to generate the attention vector.

Finally the probabilities of each word is calculated using the attention vector.

4.5 Results

We have trained our model on a data set that has 167130 sentence between short sentences of 1-2 words till long sentences, we have limited the vocabulary size to 5000 which means it will considerate only 5000 words that repeats usually. Once the training done, we have saved the model and tested it on some sentences (initiated in train set) and other custom sentences. in the figures below you will find the results after the translation with the attention heat maps :



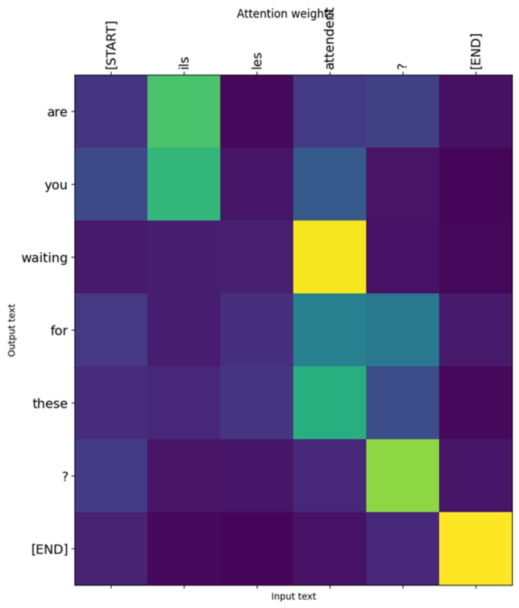


Figure 6 – Results for sentence 1 and 2

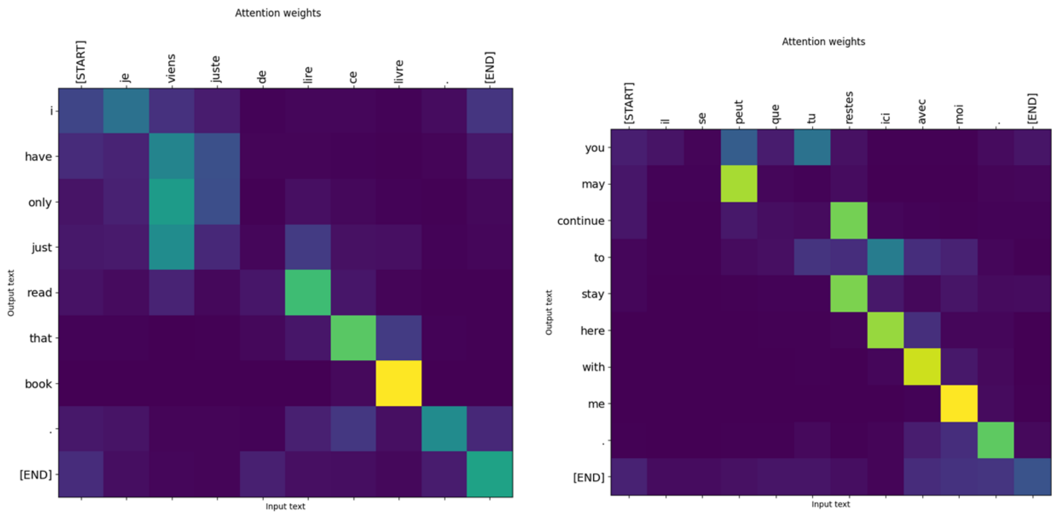


FIGURE 7 – Resulats for sentence 3 et 4

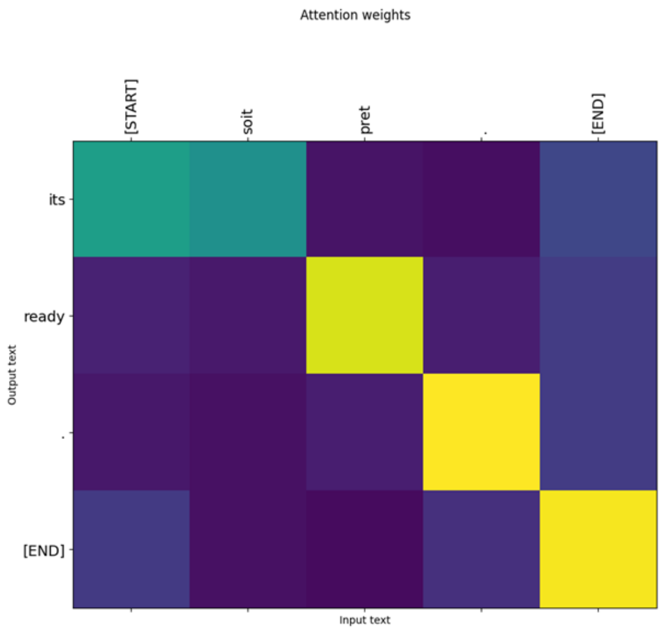
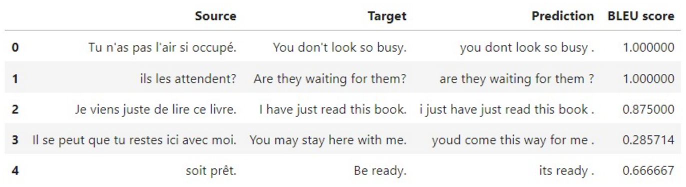


FIGURE 8 – Result for sentence 5

4.6 Discussion

We had excellent results while using these train sets but the one we added lastly had bad results because the model was not well trained.

We have calculated the blue score for few sentences independently, once training on the whole data set we will do a curve of the bleu score in function of the sentence length as implemented in the article even though they did not mention how to do plot the blue score.



	Source	Target	Prediction	BLEU score
0	Tu n'as pas l'air si occupé.	You don't look so busy.	you dont look so busy .	1.000000
1	ils les attendent?	Are they waiting for them?	are they waiting for them ?	1.000000
2	Je viens juste de lire ce livre.	I have just read this book.	i just have just read this book .	0.875000
3	Il se peut que tu restes ici avec moi.	You may stay here with me.	youd come this way for me .	0.285714
4	soit prêt.	Be ready.	its ready .	0.666667

Figure 9 – Bleu score

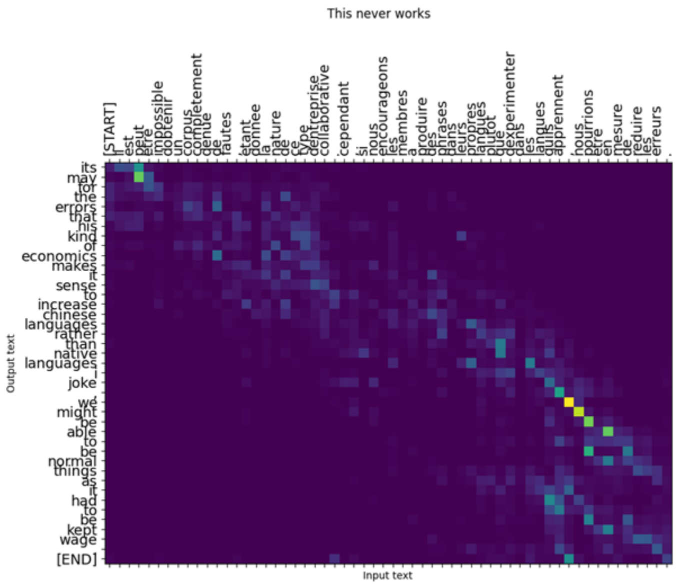


Figure 10 – The attention mechanism

In the figure above we can notice that the results were bad because the sentence has many contexts so it's not possible to do the translation unless we divide the input into small parts.

One of the problems we had is the data sets, we could not find the ones he mentioned in the article, also we did not write the code of the internal architecture of the GRU but we have used the pre-built function directly, the same for the attention mechanism we have used the function built by Bahdanau.

5. Conclusion

Adding an attention cell to a traditional encoder decoder translation model produces far superior results and permits the correct translation of longer sentences. This is acheived by letting the model search for input words or their annotations produced by an encoder when producing a target word, ultimately freeing the model from having to construct a sentence into a fixed length vector and subsequently erasing information when dealing with long sentences.

The results performed on an English to French translation task proved the efficiency of said method, largely outperforming traditional models.

6. Bibliographie

Neural Machine Translation by jointly learning to align and translate : https ://arxiv.org/abs/1409.0473.

Neural Machine translation with attention : https ://www.tensorflow.org/text/tutorials/nmt$_{with_attenti}$

Meduim review of the article : https ://sh-tsang.medium.com/review-neural-machine-translation-by-jointly-learning-to-align-and-translate-3b381fc032e3

Références