Neural Machine Translation with Sequence to Sequence and Attention

Phan Viet Hoang

March 6, 2020

1 Assignment introduction

- Machine translation (MT) is the task to translate a text from a source language to its counterpart in a target language. It is actually a relatively old task. From the 1970s, there were projects to achieve automatic translation.
- In a machine translation task, the input already consists of a sequence of symbols in some language, and the computer program must convert this into a sequence of symbols in another language.
- Over the years, there are three major approaches:
  - Rule-based Machine Translation (RBMT): 1970s-1990s
  - Statistical Machine Translation (SMT): 1990s-2010s
  - Neural Machine Translation (NMT): 2014-Now

In this assignment, my task is to build a simple NMT system with seq2seq which is capable of translating English-Vietnamese

2 Problem overview

2.1 Neural Machine Translation (NMT)

- Unlike the traditional phrase-based translation system which consists of many small sub-components that are tuned separately, neural machine translation attempts to build and train a single, large neural network that reads a sentence and outputs a correct translation
- Therefore, neural machine translation systems are said to be end-to-end systems as only one model is required for the translation, instead of a pipeline of separate tasks.

2.2 Encoder-Decoder (seq2seq) model

2.2.1 Motivation

- Multilayer Perceptron neural network models can be used for machine translation, although the models are limited by a fixed-length input sequence where

the output must be the same length.
- Our model first encode the source text into an internal fixed-length representation called the context vector. We call this summary the "context" C. A second mode, usually an RNN, then reads the context C and generates a sentence in the target language.

2.2.2 Model architecture

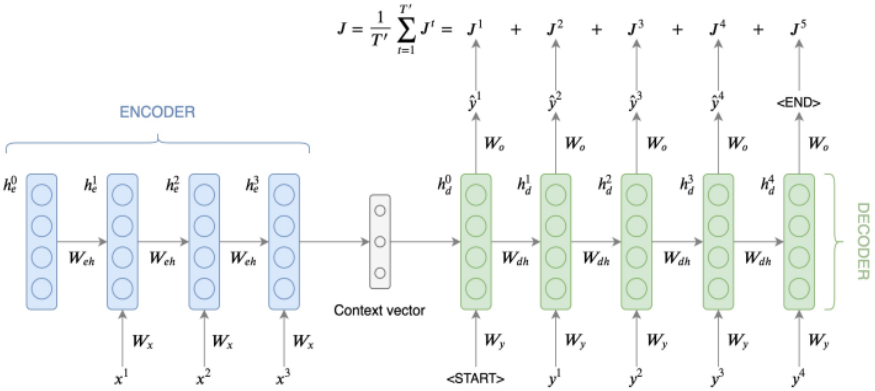
- This architecture is comprised of two models: one for reading the input sequence and encoding it into a fixed-length vector, and a second for decoding the fixed-length vector and outputting the predicted sequence.
- The innovation of this architecture is the use of a fixed-sized internal representation in the heart of the model that input sequences are read to and output sequences are read from. For this reason, the method may be referred to as sequence embedding.
- Encoder is a RNN layer (or a stack of several RNN layers) which takes the input sentence and encode its information into a fixed length context vector. This representation is expected to be a good summary of the whole source sequence's meaning. The most basic form of the encoder-decoder architecture uses the last hidden state of the encoder as the context vector.

• The encoder hidden state at time t hidden states $h_e^t \in R^n$ are computed using the formula

$$h_e^t = \sigma(W_{eh} h_e^{t-1} + W_x x^t)$$

$$h_e^0$$

is randomly initialized



- Encoded vector formula:

$$c = q(h_e^1, ..., h_e^T)$$

where q are some nonlinear functions (

$$q(h_e^1, ..., h_e^T) = h_e^T$$

is used-also not a good choice, the better one will be described below)
- Encoded vector's properties:
  - Final hidden state produced from the encoder part of the model, calculated using the formula above
  - Aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions.
  - It acts as the initial hidden state of the decoder part of the model.

- Decoder is also a RNN to predict an output y^t at each decoding time step t. Each recurrent unit accepts a hidden state from the previous unit and produces output as well as it own hidden state. At the start of the decoding process, the context vector is fed to the first recurrent cell of the decoder.
  - Any hidden state h_d^t is computed using the formula:

$$h_d^t = \sigma(W_{dh} h_{d-1}^t + W_y y^t)$$
$$h_d^0 = h_e^T$$

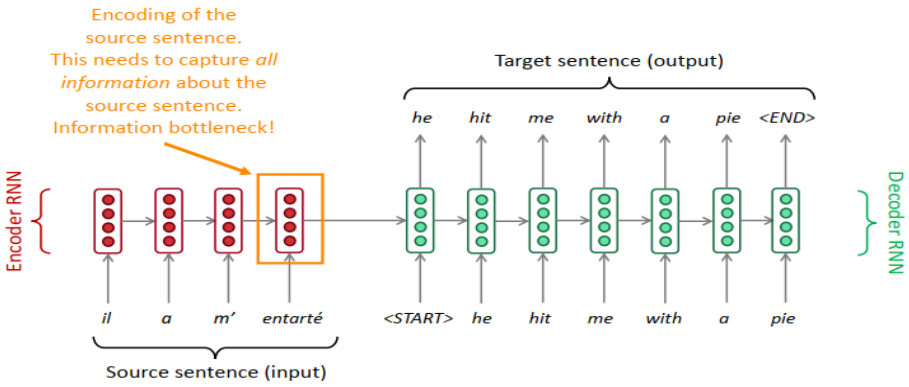
  - The output y^t at time step t is computed using the formula

$$y^t = softmax(W_o h_d^t)$$

2.3 Attention mechanism

2.3.1 Motivation

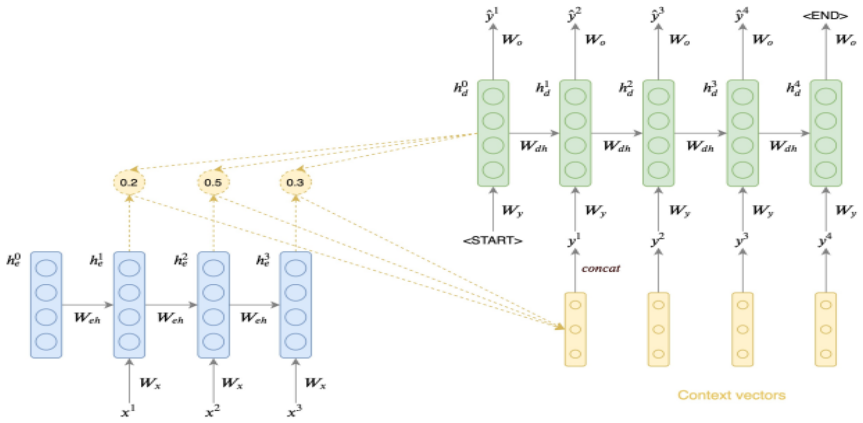
- Although effective, the Encoder-Decoder architecture has problems when trying to encode long sequences of text into a fixed-length internal representation (the bottle neck problem).



- The solution is the use of an attention mechanism that allows the model to learn where to place attention on the input sequence as each word of the output sequence is decoded.

2.3.2 Model architecture

- Core idea: on each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence
Bahdanau style attention:



- With the input sequence $x = \langle x_1, ..., x^T \rangle$
and the output sequence $y = \langle y_1, ..., y^{T'} \rangle$
The attention vector for decoding timestep t is computed as below:

- We already have encoder hidden states $h_e^1, ..., h_e^T$
- We take softmax to get the attention distribution for this step (attention weights)

$$\alpha_{ti} = \frac{exp(score(h_t^d; h_i^e))}{\sum_{i'=1}^{T} exp(score(h_t^d; h_{i'}^e))}$$

where the score(.) function can be modeled with a fully-connected layer with a non-linear activation

• We use to take a weighted sum of the encoder hidden states to get the attention output (context vector)

$$c^t = \sum_i \alpha_{ti} h_i^e$$

• Combine the context vector with the current target hidden state to yield the final attention vector

$$a^t = f(c^t, h_d^t) = tanh(W_c[\, c^t; h_d^c] \,)$$

• The attention vector is fed as an input to the next time step, proceed as in the non-attention seq2seq model

$$y^t = softmax(W_y a^t)$$

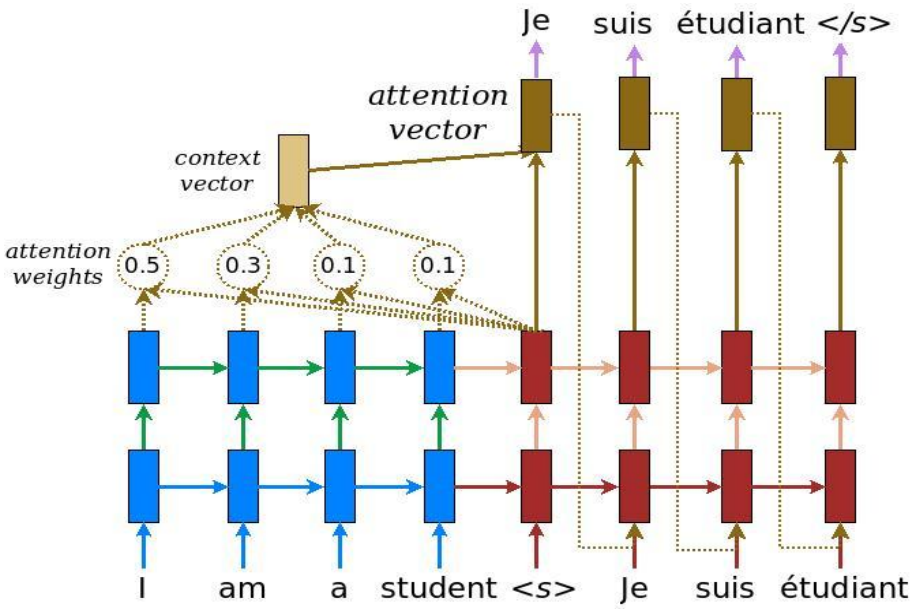
• As discussed above,the score(.) function can be modeled with a fully-connected layer with a non-linear activation, which is still ambiguous. To be more specific, we consider three different choices

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W_a} \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_a}[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$

2.4 Training a seq2seq with Bahdanau style attention model

- In this section I will provide a step-by-step training progress explanation, inherited from Matthew's assignment guide

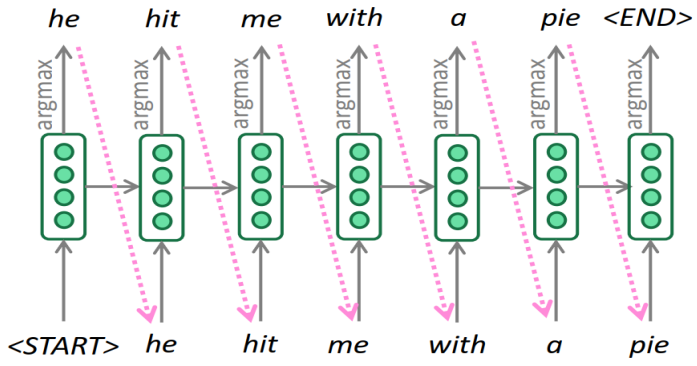
• Pass the input through the encoder which return encoder output and the encoder hidden state.
• The encoder output, encoder hidden state and the decoder input (which is the start (token) is passed to the decoder.
• The decoder returns the predictions and the decoder hidden state.
• The decoder hidden state is then passed back into the model and the predictions are used to calculate the loss.
• Use teacher forcing to decide the next input to the decoder.



• Teacher forcing is the technique where the target word is passed as the next input to the decoder (illustration above)

• The final step is to calculate the gradients and apply it to the optimizer and backpropagate.

- In the generating output sequences from the probabilities prediction of Seq2seq model progess, one can easily implement the greedy decoding method: take most probable word on each step and try to decode until the model produces a end of string token or when the text has T elements to avoid too long sequence (where T is some pre-defined cutoff threshold):



- But there exist a much more efficient decoding method: Beam search. On each step of decoder,it keeps track of the k most probable candidates and decode until:

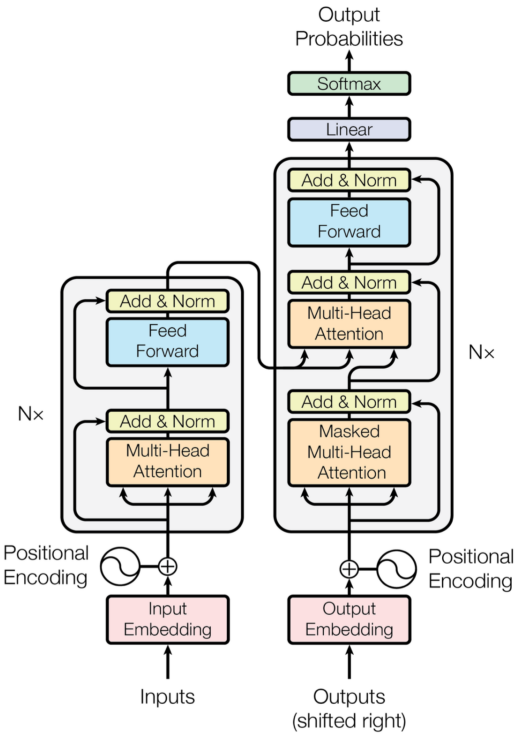
• We reach timestep T (where T is some pre-defined cutoff)

or

- We have at least n completed hypotheses (where n is pre-defined cutoff)

3 Implement report

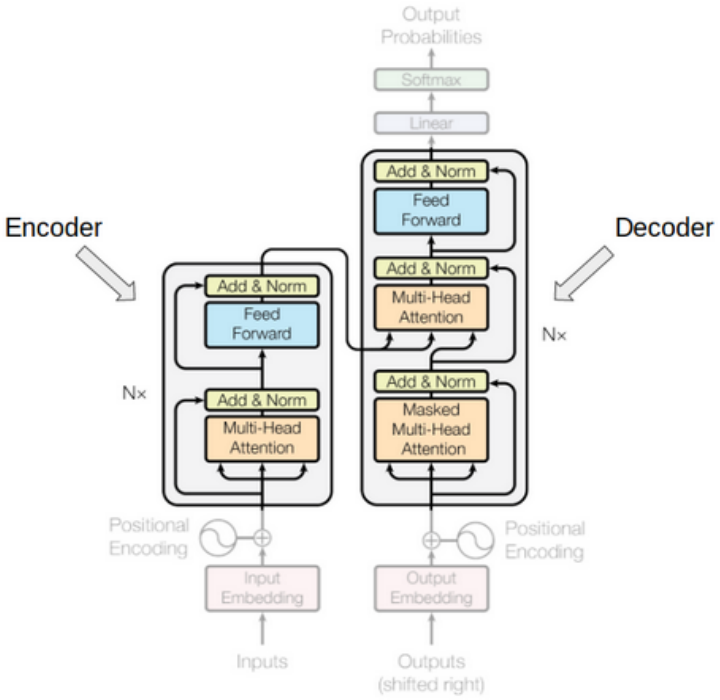
3.1 Transformer architecture explain



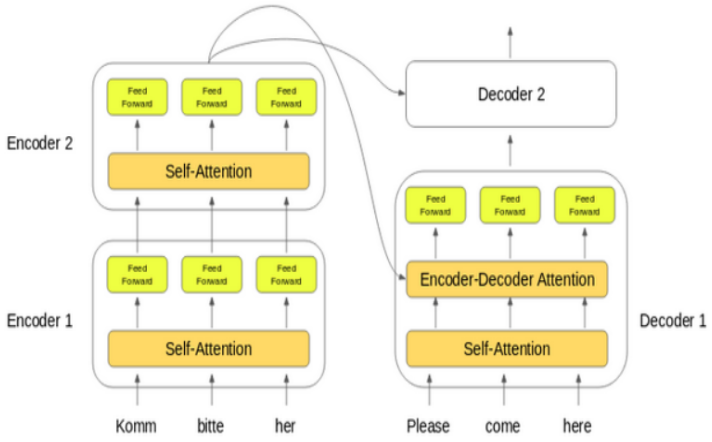
- The Transformer in NLP is a novel architecture that aims to solve sequence-to-sequence tasks while handling long-range dependencies with ease
- "The Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution." - Attention is all you need paper
- Here, "transduction" means the conversion of input sequences into output sequences. The idea behind Transformer is to handle the dependencies between input and output with attention and recurrence completely

3.1.1 Encoder and Decoder Stacks

- Let's first focus on the Encoder and Decoder parts only.



- The encoder and decoder blocks are actually multiple identical encoders and decoders stacked on top of each other.



- The architecture is a little bit similar to the seq2seq above
  - The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. Each one is broken down into two sub-layers

• The encoder's inputs first flow through a self-attention layer
• The outputs of the self-attention layer are fed to a feed-forward neural network. The exact same feed-forward network is independently applied to each position
• The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence

* Notice that since the position of each word is absolutely essential to the translation and on the figure above, a positional encoding layer is added when encoding each word (before feeding to the encoder layer) the detailed explanation will be provided below

3.1.2 Self-Attention

* In this section I will present a clear explanation for the intuition behind self-attention

- Create three vectors from each of the encoder's input vectors (in this case, the embedding of each word). So for each word, we create a Query vector, a Key vector, and a Value vector.

$$q_i, k_i \in R^{d_k}$$
$$v_i \in R^{d_v}$$

- The second step in calculating self-attention is to calculate a score. The score determines how much focus to place on other parts of the input sentence as we encode a word at a certain position.

- The score is calculated by taking the dot product of the query vector with the key vector of the respective word we're scoring (i^{th} word in my example).

$$\text{score}_i n = q_i.k_n^T (n \in \{1, .., l_{\text{sequence}}\})$$

- The third and forth steps are to divide the scores by the square root of the dimension of the key vectors, then pass the result through a softmax operation

$$\text{normalized score}_i = softmax(\frac{\text{score}_i}{\sqrt{d_k}})$$

- The fifth step is to multiply each value vector by the softmax score (in preparation to sum them up). The intuition here is to keep intact the values of the word(s) we want to focus on, and drown-out irrelevant words

$$\text{weight}_{in} = \text{normalized score}_i * v_n$$

- The sixth step is to sum up the weighted value vectors. This produces the output of the self-attention layer at this position

$$\text{Attention output}_i = \sum_{n=1}^{l_{\text{sequence}}} \text{weight}_{in}$$

- The resulting vector is one we can send along to the feed-forward neural network

3.1.3 Multi-Head Attention

- Instead of performing a single attention function with d_{model}-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to $d_k, d_k$and d_v dimensions, respectively
- On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding d_v-dimensional output values. These are concatenated and once again projected, resulting in the final values
- Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

$$\text{MultiHead(Q, K, V)} = \text{Concat}(head_1, ..., head_h)W^O$$

where

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices

$$W_i^Q \in R^{d_{model}.d_k}$$

$$W_i^K \in R^{d_{model}.d_k}$$

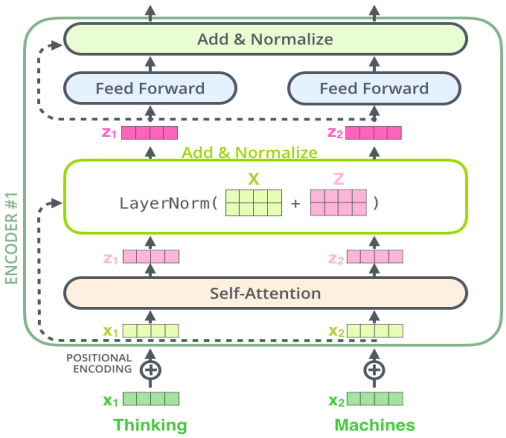
$$W_i^V \in R^{d_{model}.d_v}$$

$$W^O \in R^{hd_v.d_{model}}$$

- Therefore:
  - It expands the model's ability to focus on different positions.
  - It gives the attention layer multiple "representation subspaces

3.1.4 Residual connections and layer normalization

These are 2 simple concepts of training neural network technique, so I will give a brief overview and not dive to deep into the math formula (which will be provided in the implementation section)



- Residual connections is described as a help in avoiding the vanishing gradient problem in deep networks
- The residual connection directly adds the value at the beginning of the block, x, to the end of the block $(F(x) + x)$
- This residual connection doesn't go through activation functions that "squashes" the derivatives, resulting in a higher overall derivative of the block.
- Batch normaliation is a great way to reduce the training time is to normalize the activities of the neurons but it is not obvious how to apply it to recurrent neural networks. That is the reason why we use layer normalization instead
- It is also straightforward to apply to recurrent neural networks by computing the normalization statistics separately at each time step

3.2 Implement explain

3.2.1 Training

- Load and process data from Stanford machine translation dataset, reformat with tensorflow.dataset module
- Positional encoding (positional_encoding function)

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

- Mask all the pad tokens in the batch of sequence. It ensures that the model does not treat padding as the input..
- Scaled dot product attention: the attention function used by the transformer takes three inputs: Q (query), K (key), V (value). The equation used to calculate the attention weights is:

$$\text{Attention}(Q, K, V) = softmax(\frac{Q.K^T}{\sqrt{dk}})V$$

- Multi-head attention consists of four parts:
  - Linear layers and split into heads.
  - Scaled dot-product attention.
  - Concatenation of heads.
  - Final linear layer.
- Each encoder layer consists of sublayers:
  - Multi-head attention (with padding mask)
  - Point wise feed forward networks.
- Each decoder layer consists of sublayers:
  - Masked multi-head attention (with look ahead mask and padding mask)
  - Multi-head attention (with padding mask). V (value) and K (key) receive the encoder output as inputs. Q (query) receives the output from the masked multi-head attention sublayer.
  - Point wise feed forward networks
- Each of the sublayers (both encoder and decoder) has a residual connection around it followed by a layer normalization.
- The Encoder consists of:
  - Input Embedding
  - Positional Encoding
  - N encoder layers
- The Decoder consists of:
  - Output Embedding
  - Positional Encoding
  - N decoder layers
- Optimizer:Use the Adam optimizer with a custom learning rate scheduler according to the formula in the paper.

$$lr = d_{model}^{-0.5}.min(\text{step num}^{-0.5}, \text{step num.warmup steps}^{-1.5})$$

- Loss: cross-entropy
- Metric: accuracy
- The target is divided into Input and Real. Input is passed as an input to the decoder. Real is that same input shifted by 1: At each location in Input, Real contains the next token that should be predicted.

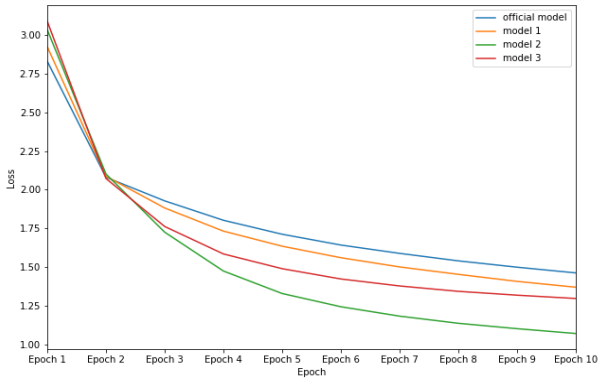
Sentence	Input	Real
(sos) I am Hoang (eos)	(sos) I am Hoang	I am Hoang (eos)

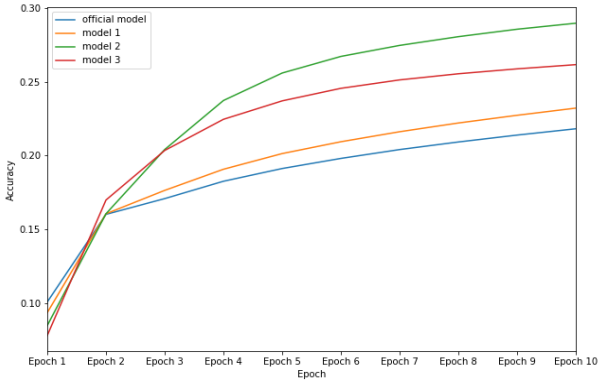
- During training this example uses teacher-forcing technique
- Regularization: drop-out rate = 0.1 (default value for each model)
- Hyper-parameters settings:

Model	No.layers	No.units of dense layer	No.heads	Embedding dim
Official	6	2048	8	512
Model 1	5	1024	8	256
Model 2	4	512	8	128
Model 3	3	256	4	64

3.2.2 Evaluation

- The following steps are used for evaluation:
  - Encode the input sentence using the English tokenizer. Moreover, add the start and end token-this is the encoder input.
  - The decoder input is the start token
  - Calculate the padding masks and the look ahead masks.
  - The decoder then outputs the predictions by looking at the encoder output and its own output (self-attention).
  - Select the last word and calculate the argmax of that.
  - Concatenate the predicted word to the decoder input as pass it to the decoder.





- Metric for the quality of generated text in my assignment is the traditional

BLEU score-which is popular in machine-translation tasks. It can easily explained as follow:

$$p_n = \frac{\sum_{S \in C} \sum_{ngram \in S} Count_{matched}(ngram)}{\sum_{S \in C} \sum_{ngram \in S} Count(ngram)}$$

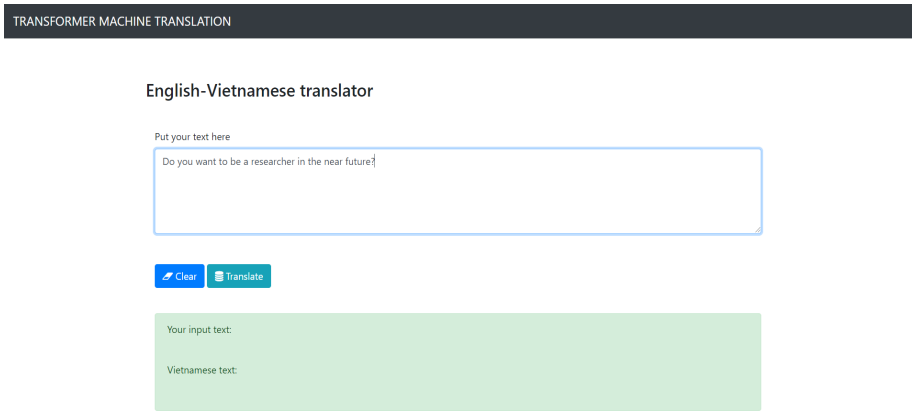
where:
p_n is the n-grams BLEU-precision score taken from every sentence S from corpus C

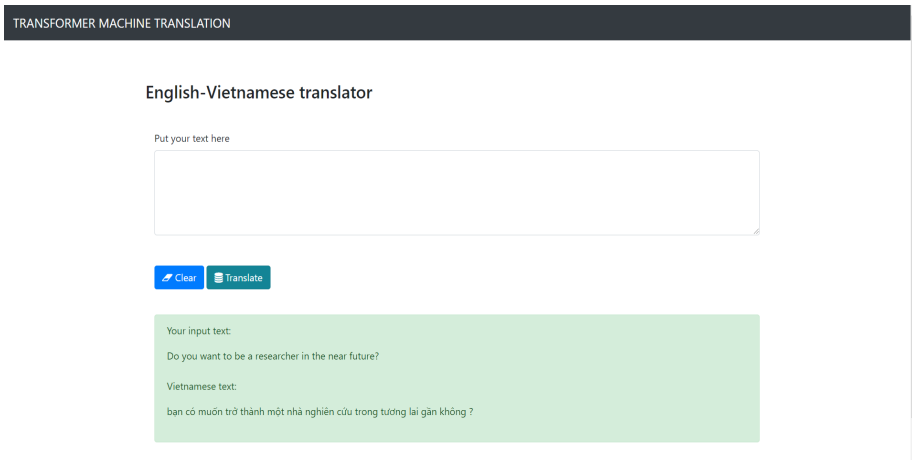
Official	Model 1	Model 2	Model 3
17.37	13.76	21.74	16.24

3.3 Web deployment

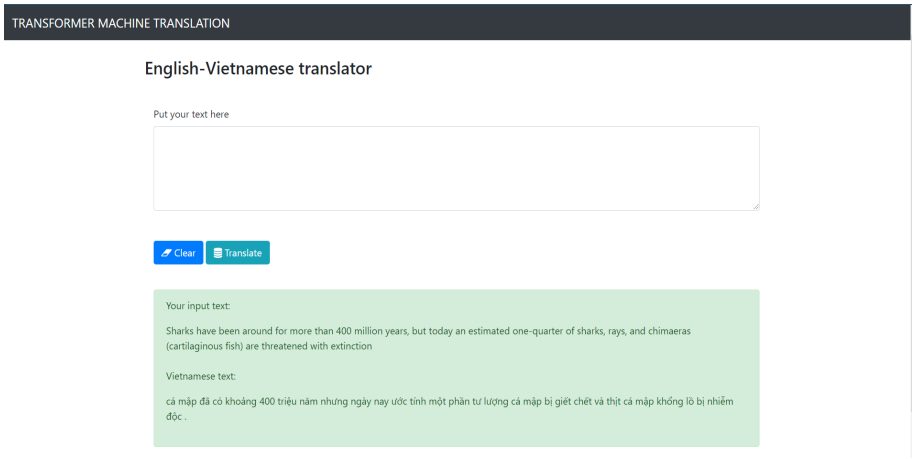
- I've tried several ways to deploy deploy my machine translator to some cloud services for easy usage and illustration but it exceeds the heroku's maximum slug size limit. Even worse, Docker Desktop requires Windows 10 Pro or Enterprise version 15063 to run. Other versions (including my laptop configuration) will have a bunch of steps for installation.
- Anyway, let have some observations with the model's translation:

- It doesn't find difficult in translating some simple sentences:





• But some document containing rare words won't produce expected predictions:



- This is a significant weakness in Neural Machine Translation systems. There are several possible solution (described in papers without code):

• Train an NMT system on data that is augmented by the output of a word alignment algorithm, allowing the NMT system to emit, for each OOV word in the target sentence, the position of its corresponding word in the source sentence

• Encoding rare and unknown words as sequences of subword units. This via smaller units than words

• Annotate the training corpus with information about what do different OOV words (in the target sentence) correspond to in the source sentence.

• NMT learns to track the alignment of rare words across source and target sentences and emits such alignments for the test sentences.

• As a post-processing step, use a dictionary to map rare words from the source language to target language.

References

[1] CS224n: Natural Language Processing with Deep Learning

[2] A Gentle Introduction to Neural Machine Translation

[3] Machine Translation: A Short Overview

[4] Encoder-Decoder Long Short-Term Memory Networks

[5] Understanding Encoder-Decoder Sequence to Sequence Model

[6] Attention in Long Short-Term Memory Recurrent Neural Networks

[7] Effective Approaches to Attention-based Neural Machine Translation

[8] Bài 4 - Attention is all you need

[9] The Illustrated Transformer

[10] Layer Normalization