Course: COSC 4337 Data Science II

Professor: Ricardo Vilalta

TA: Shaila Zaman

Team: 4

Group members: Hieu Trinh, Thanh Le

Neural Machine Translation English to French

1. Introduction

Machine translation(MT) is an essential task that focused on translating text from one language to another. With the power of deep learning, Neural Machine Translation(NMT) has arisen as the most powerful algorithm to perform this task because we can use thousands of different architectures to train different models. One of the most effective and established version of NMT is the **Encoder-Decoder** structure. Furthermore, we want to dive deeper by using the **Transformer** model, which will put attention in every tokens to solve the long-term dependency problem of regular Recurrent Neural Network(RNN) architecture. We will use these two state-of-art architectures to train models using custom dataset of 176,620 rows to translate English to French. We will also mainly use PyTorch technology for model implementation.

2. Data Preparation and Pre-processing

For getting the data in the best way, I am using SpaCy for vocabulary building, TorchText(text Pre-processing) libraries, and sklearn to divide the original dataset into partitions. Here are the steps of pre-processing data:

i. **Train/Valid/Test Split**: partition your data into a specified train/valid/test set. The ratio is 8:1:1
ii. **File Loading**: load the text corpus from the csv file
iii. **Tokenization**: breaking sentences into a list of words
iv. **Vocab**: Generate a list of vocabulary for each language from the text corpus

v. **Words to Integer Mapper**: Map words into integer numbers for the entire corpus and vice versa

vi. **Word Vector**: Convert a word from a higher dimension to a lower dimension (Word Embedding)

vii. **Batching**: Generate batches of the sample.

Then, we will use Torchtext, a powerful NLP library for making the text data ready for a variety of NLP tasks such as padding, making all the tokens lower case, etc.,. It has all the tools to perform preprocessing on the textual data. Here we are going to use 3 classes under torchtext:

- **Fields**: This is a class under Torchtext, where we specify how the preprocessing should be done on our data corpus. We will add <sos> (start of sentence) and <eos> (end of sentence) token to the whole list for effective model training because these two tokens will help the word embedding process of the encoder and decoder by signalizing where to read and terminate the input as well as the output.
- **TabularDataset**: Using this class, we can actually define the Dataset by making it become tensors to map them into integers
- **BucketIterator**: Using this class, we can perform padding our data for approximation and make batches with our data for model training.

3. Overview of models

a) Sequence to Sequence Model

This architecture is composed of two recurrent neural networks(RNNs) called Encoder and Decoder that they are used together to create a translation model. Moreover, this architecture can achieve impressive results if we add the attention mechanisms to the model. The attention mechanisms was born to help memorize long source sentences in NMT because they could have attention vector for each token and directly translate from those vectors instead of memorizing the position of each token. In this particular translation model, we will be using Long Short-Term Memory(LSTM) model, which is a type of RNN. Here are more descriptions about the model:

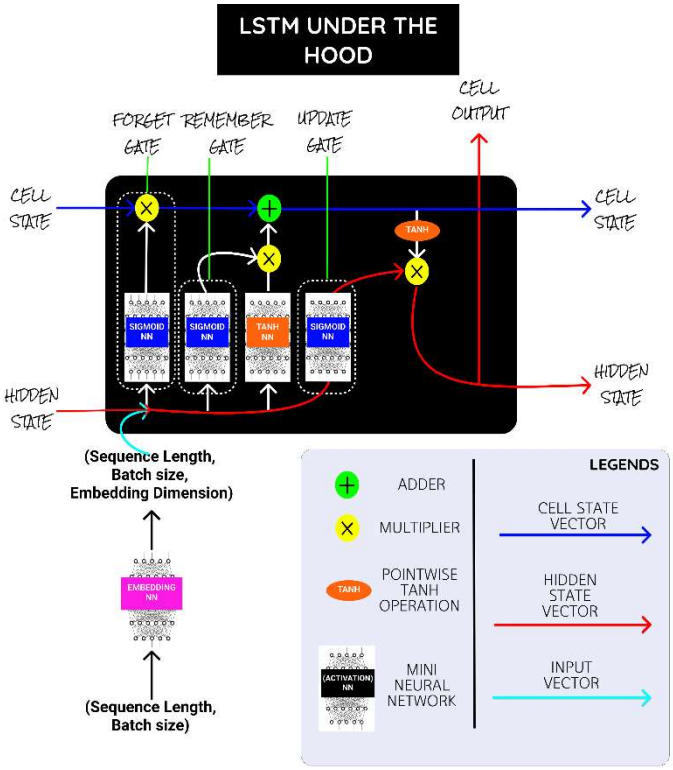


Figure 1: LSTM architecture

i. Inside LSTM cell:

- **Sigmoid NN(activation function):** Squishes the values between 0 and 1. Say a value closer to 0 means to forget and a value closer to 1 means to remember
- **Embedding NN:** Converts the input word indices into word embedding.
- **TanH NN(activation function):** Squishes the value between -1 and 1. It helps to regulate the vector values from either getting exploded to the maximum or shrank to the minimum.

Gates:

- **Forget gate:** Has sigmoid activation in it and range of values between(0-1) and it is multiplied over the cell state to forget some elements.
- **Add gate:** Has TanH activation in it and range of values between(-1 and +1) and it is added over the cell state to remember some elements.
- **Update Hidden:** Updates the Hidden States based on the Cell State.

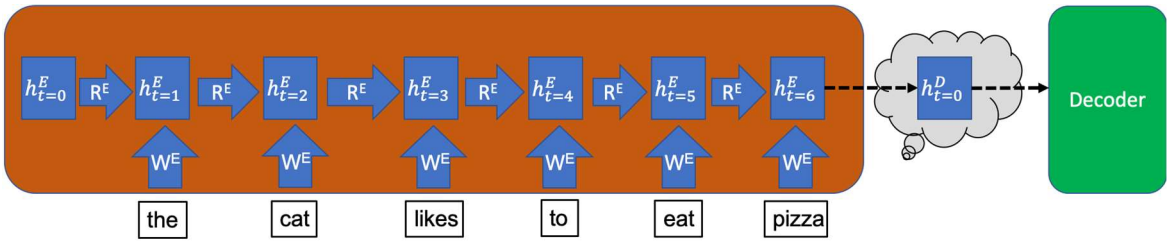


Figure 2: Encoding of the sentence "The cat likes to eat pizza"

ii. **Encoder**: At each time step, the hidden vector takes in information from the inputted word at that time-step, while preserving the information it has already stored from previous time-steps. Thus, at the final time step, the meaning of the whole input sentence is stored in the hidden vector. This hidden vector at the final time-step is the thought vector referred to above, which is then inputted into the Decoder. The process of encoding the English sentence "the cat likes to eat pizza" is represented in Figure 2.

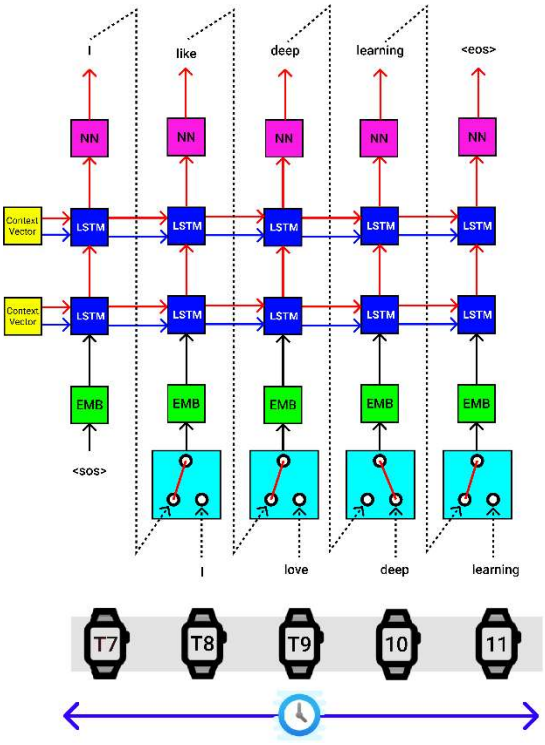


Figure 3: Decoding illustration

iii. **Decoder:** The decoder also does a single step at a time. The Context Vector from the Encoder block is provided as the hidden state(hs) and cell state(cs) for the decoder's first LSTM block. The **start of sentence "SOS"** token is passed to the embedding NN, then passed to the first LSTM cell of the decoder, and finally, it is passed through a linear layer, which provides an output English token prediction probabilities. The output with the highest probability is passed to the next LSTM cell and this process is executed until it reaches the **end of sentences "EOS".**

b) Transformer Model

To solve the problem of parallelization, Transformers try to solve the problem by using Convolutional Neural Networks together with attention models. Attention boosts the speed of how fast the model can translate from one sequence to another.

Figure 4: Transformer illustration

i. **Transformer architecture:**

The Transformer has a similar kind of architecture as the previous models above. But the Transformers consists of six encoders and six decoders. Each encoder is very similar to each other. All encoders have the same architecture. Decoders share the same property, i.e. they are also very similar to each other. Each encoder consists of two layers: Self-attention and a Forward Neural Network. Figure 5 will illustrate how tokens go through encoders and decoders.

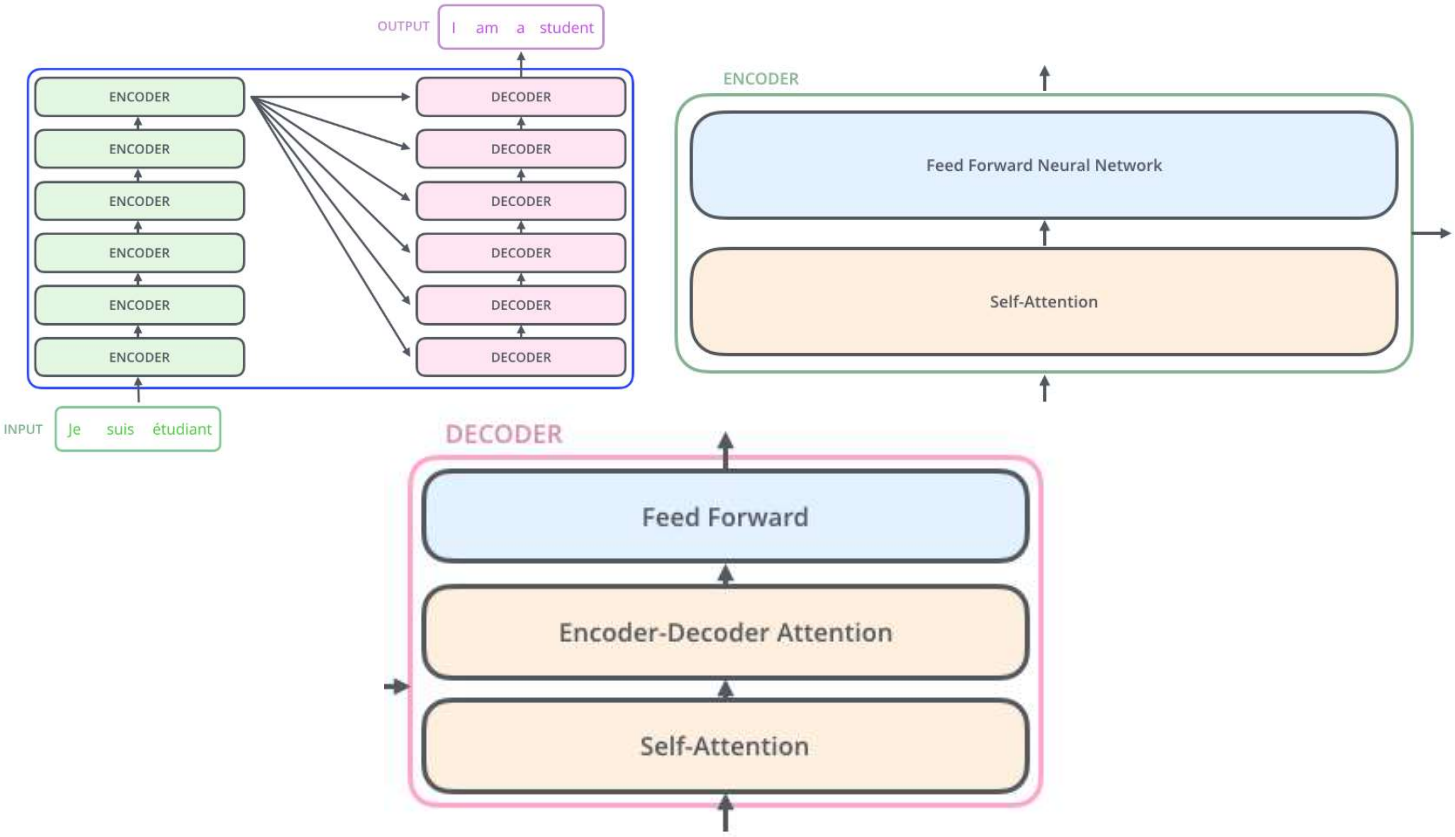


Figure 5: Insight illustration of Transformer Model

ii. Calculating self-attention steps:

- Creating three vectors from each encoder's input vectors(Query vectors, Key vectors, Value vectors)
- Calculating the score by taking dot product of the query vector with the key vector of the respective word we are scoring.
- Divide the score by 8. This leads to having a more **stable gradients**. Then, we pass the result through a **softmax** operation. **Softmax** normalizes the scores so they're all positive and add up to 1.
- Multiply each vector by the softmax score.

- Sum up the weighted value vectors. This produces the output of the self-attention layer at this position.

4. Model Performance

a) Hyperparameter Tuning:

- We use PyTorch nn.Module to create our custom model. Parameters used are vocab_size, optimizer, dropout, embedding_size, etc,…
- We also clip the gradient by setting the maximum to 1 so that it will prevent **exploding gradients.**
- We also use the learning rate with scheduler so that it could adjust the learning rate during training by reducing the learning rate according to a pre-defined schedule. This technique will help the model adapt to the problem.
- We also apply **early stopping** technique to our models so that they could know where to stop if there are no improvements after a certain amount of epochs. This could help us prevent from underfitting/overfitting.

Parameters	**Seq2Seq**	**Transformer**
Vocab_size	10000	10000
Encoder_embedding_size	300	512
Decoder_embedding_size	300	512
Hidden_size	1024	Not used
Num_layers	2	3
Learning_rate	0.001	0.055 or 3e^-4
Num_epochs	50	50
Batch_size	64	32
Encoder_dropout	0.5	0.3
Decoder_dropout	0.5	0.3
Optimizer	Adam, AdamW	Adam, AdamW
Input_size_encoder	176,620	176,620

Input_size_decoder	176,620	176,620
Scheduler	ReduceLROnPlateau	ReduceLROnPlateau
Loss function	CrossEntropyLoss	CrossEntropyLoss
Early stopping	True	True

b) Time taken:

Model and Optimizer	**Time**
Seq2Seq, Adam	3 hours 33 minutes
Seq2Seq, AdamW	3 hours 24 minutes
Transformer, Adam	1 hours 34 minutes
Transformer, AdamW	1 hours 26 minutes

As we can see, Transformer models run faster than the Seq2Seq models because the encoder-decoder system is more effective by calculating self-attention for each token instead of relying on the LSTM in each timestamp.

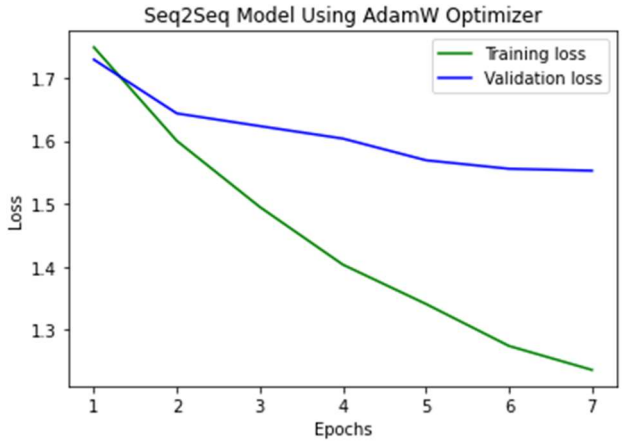
5) Model Evaluation

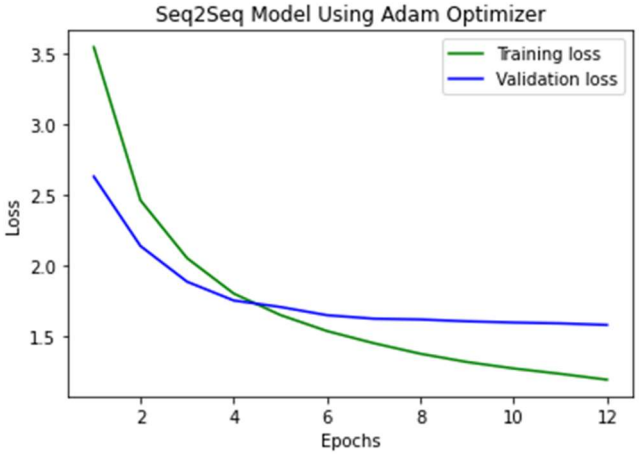
a) Training loss versus valid loss:

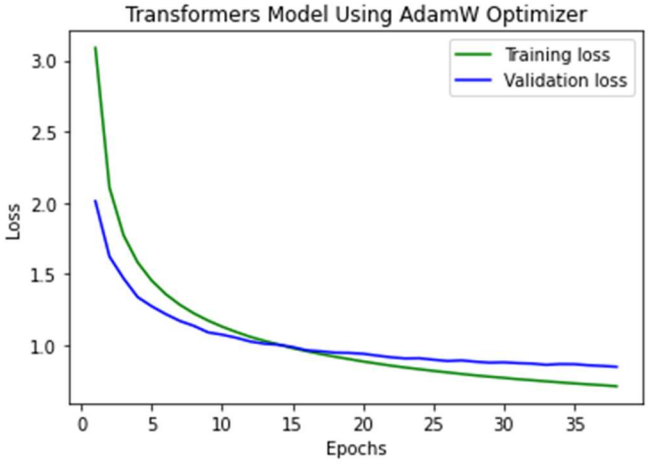
- Instead of comparing to test set, we want to compare the model to the valid test because we want to see how models will learn the new vocabs
- Choosing the right optimizer is so important that it can help you save a lot of time at the beginning of the training process. Looking at the table below as the Max_train_loss is usually the training loss of the start of the training process.
- Once again, there is a huge difference between Seq2Seq and Transformers model if we look at the training loss and validating loss. The minimum training loss and validating loss of Transformer models falls in the range of 0.71-0.73 and 0.84-0.86, way better than 1.19-1.23 and 1.55-1.58 of Seq2Seq models. So far, the Transformer models are running faster and having better results thanks to one of the latest architecture.

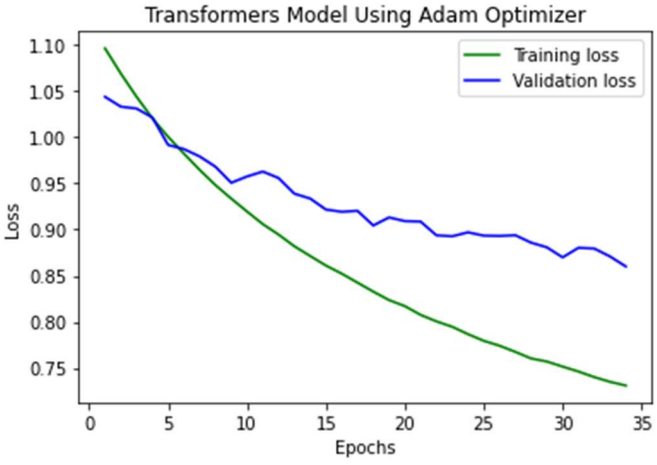
Models and Optimizer	Max_train_loss	Min_train_loss	Max_valid_loss	Min_valid_loss
Seq2Seq,Adam	3.55	1.19	2.63	1.58
Seq2Seq, AdamW	1.74	1.23	1.72	1.55
Transformer, Adam	1.09	0.73	1.04	0.86
Transformer, AdamW	3.08	0.71	2.01	0.84

- Here are some plots of training loss vs. validating loss. The number of epochs could be different since we apply early stopping once the validating loss could not be improved.



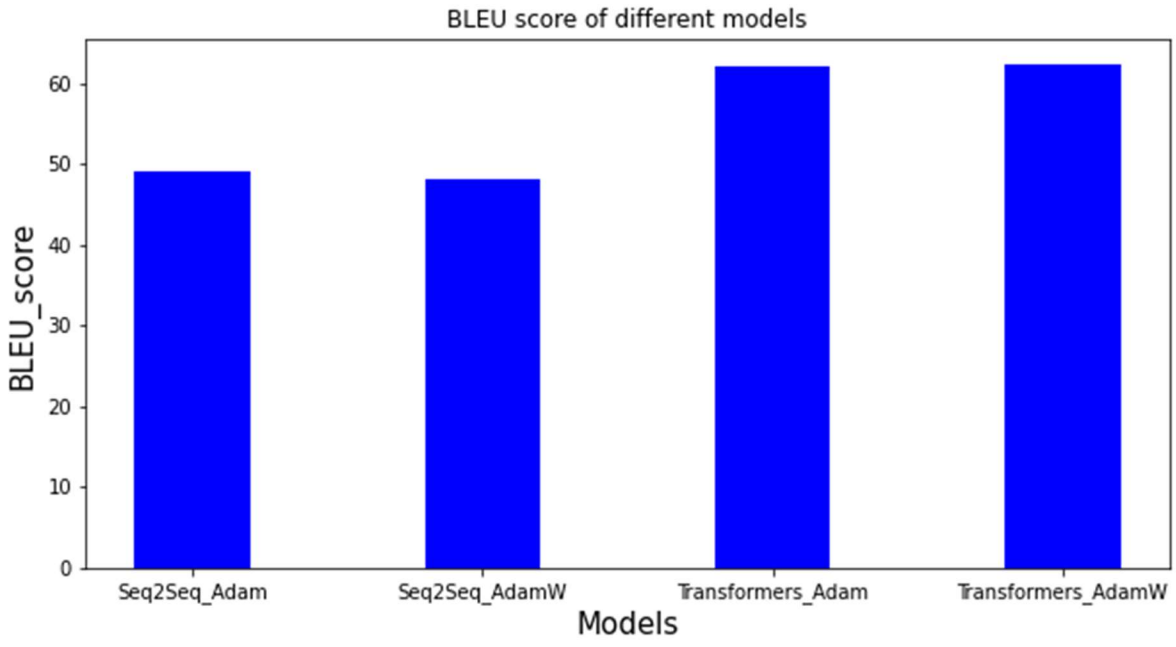






b) BLEU Score

- BLEU, or Bilingual Evaluation Understudy, is a score for comparing a candidate translation of text to one or more referenced translation. The BLEU metric scores a translation from a scale from 0 to 1, in an attempt to measure the adequacy and fluency of the MT output. The closer to 1 the test sentences score, the better the system is deemed to be.
- Some good ranges of BLEU Score:
    - **30 – 40**: Understandable to good translation
    - **40 – 50**: High quality translations
    - **50 – 60**: Very high quality, adequate, and fluent translations
    - **>60**: Quality often better than human
    - The Transformers model using AdamW scores the highest: **62.31**
    - The Seq2Seq model using adamW scores the lowest: **48.17**
    - Here is the bar chart of the BLEU scores:



c) Sentence translation

Here are some sentences that are translated using Transformer model with AdamW optimizer:

Source sentence: ['they', 'refused', '.']
Target sentence: ['elles', 'ont', 'refusé', '.']
Output sentence: ['elles', 'ont', 'refusé', '.']

Source sentence: ['she', 'cried', '.']
Target sentence: ['elle', 'pleurait', '.']
Output sentence: ['elle', 'pleura', '.']

Source sentence: ['they', 'sweated', '.']
Target sentence: ['elles', 'ont', '<unk>', '.']
Output sentence: ['ils', 'ont', '<unk>', '.']

There are some "unk" tokens, meaning that the frequency of these tokens are so low that our machine is not able to learn to translate these tokens to French.

6) Conclusion

- In general, the machine translation predicted most of the tokens correctly which is a positive sign that we are doing it in the right path.
- Although the Transformer models get a pretty high BLEU score, we believe that our machine translation did not validate enough data because the size of the dataset is very small which could lead to a very high BLEU score.
- In the future, we could collect more data for our machine to learn so that the vocabulary size of each language could be increased and the machine could make better predictions.
- We could also use new and advanced pre-trained model for tokenizing and padding to see how our models could be improved. Moreover, we could use pre-trained word embedding to make each token more unique instead of using PyTorch embedding.

Works Cited

Alammar, Jay. "The Illustrated Transformer." *Jay Alammar – Visualizing Machine Learning One Concept at a Time*, jalammar.github.io/illustrated-transformer/.

V, Balakrishnakumar. "A Comprehensive Guide to Neural Machine Translation Using Seq2Sequence Modelling Using PyTorch." *Medium*, 14 Sept. 2020, towardsdatascience.com/a-comprehensive-guide-to-neural-machine-translation-using-seq2sequence-modelling-using-pytorch-41c9b84ba350#bfab.