INFO8010: Project:
Machine translation using recurrent neural networks

Alexis Perin[1] and **Adrien Saulas**[2]
[1] *alexis.perin@student.uliege.be (s152394)*
[2] *adriens.saulas@student.uliege.be (s184481)*

I. INTRODUCTION

In this project, our goal was to make use of **recurrent neural networks** in order to build a machine translator to translate sentences from English to French.

For this purpose, we used a dataset [1] of 192341 sentences in English with their translation in French. Note that this dataset has recently been updated and now contains about two thousand additional sentences, however, we used the previous version for this project. This dataset is part of the Tatoeba Project [2], which is a database of sentences with their translations for a large number of pairs of languages.

We had to preprocess the data before we could feed it to the model. More specifically, we had to replace the plain text sentences by sequences of integers. This preprocessing will be explained in sections 3.A and 3.B.

We decided to design our model based on the **Encoder-Decoder Architecture**, with LSTM layers. More details on this will be provided in section 3.E.

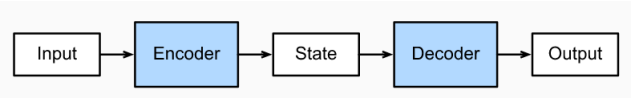

FIG. 1. Encoder-Decoder Architecture [3]

In order to measure the accuracy of our model, we applied the BLEU-score [4], which is an algorithm evaluating the quality of a translation based on the groundtruth reference.

Finally, we were able to translate new English sentences to French. The quality of these translations will be discussed in section 4.

II. RELATED WORK

First, let's take a look at a similar research done in 2014 by a team from Google. Their paper can be found here [5]. (DISCUSS THE PAPER !!!)

III. METHODS

Before we dive into the detailed methodology, note that we did not actually use the entire dataset at our disposal. Indeed, we discarded the duplicates (some sentences have more than one correct translation) and we only kept the first hundred thousand pairs of sentences to train and evaluate our model. Reasons for this will become clearer latter in this report, but it allowed us to reduce the time needed to train the model by a factor of ten.

A. Data Preprocessing

The first step we had to do was to preprocess the data to be able to feed it to the model. To do this, we divided the source and target sentences in separate lists. We then had to clean the data, this is done in four steps:

1. We add spaces between words and punctuation when there is not one already.
2. We convert uppercases to lowercases.
3. We split each sentence word-wise to form a list of words and punctuation marks.
4. We discard every item (punctuation mark, number, etc) that is not an alphabetic word.

As an example, the sentence "*Deep Learning ? It's so fun.*" becomes "*deep learning it s so fun*". We can see that this cleaning procedure removes some information in the data. To make better predictions, we could keep the punctuation marks because they contribute to the sens of a sentence. However, doing so would increase the size of the sequences fed to the model and therefore increase the training time.

B. Tokenization

Since a neural network cannot handle plain text, we convert the sentences from a list of strings to a list of integers. To do so, we created a *Vocabulary* class that can build and handle a vocabulary for a language given a list of sentences.

The *Vocabulary* class creates a set of all the unique words in the data. In this case we simplified it a bit further by considering only the words that appear at least

twice. This will reduce the size of the vocabulary and therefore the size of the model, as will be explained later.

Once we have built the set of unique words, we can use it to assign a unique integer value to each word. Since our vocabulary is built with a limited amount of data, some words might not be in it. This issue is covered by a special token *"unk"* used for unknown words.

Additionally, we append special tokens *"bos"* and *"eos"* at the beginning and the end of each sequence respectively. These will help the model know when to start and end making predictions.

As an example, the (cleaned) sentence "*hey what are you guys talking about*" becomes "*1, 1017, 89, 188, 40, 1020, 593, 943, 2*". With *1* and *2* being special tokens *"bos"* and *"eos"* respectively.

One last step before the data is ready to be fed to the model is to add padding to the sequences. This step is necessary to make all sequences to same length and be able to feed them in batches to the model. This is done by adding special tokens *"pad"* at the end of the sequences in order to make them match the length of the largest sequence in the data. In our implementation, the special token *"pad"* is converted to the integer *0*.

If we go back to the original dataset, we can see that the sentences are in ascending order. Since we used only the first hundred thousand sentences, the maximum length of our sequences is smaller than that of the sentences we discarded. This allows us to reduce the size of all the sequences that we actually use (by reducing the padding) and this contributes to reducing the training time of the model.

C. Analysis of the Dataset

Now that the data is cleaned, we can get some statistics about it, these are shown at table I. As stated above, the number of unique words considers only the words that appear at least twice in the data.

	English	French
Total number of words	749095	807653
Number of unique words	7218	10243
Avg length of a sentence	7.49	8.08
Max length of a sentence	14	20

TABLE I. Some dataset statistics

D. Training and test sets

Since we should never evaluate a model on the same data it was trained on, we split the data into two sets. We decided to use 80% of the data for training and the last 20% for the evaluation.

E. Encoder-Decoder Architecture

The Encoder-Decoder architecture is a classic sequence-to-sequence architecture. It takes as input a source sequence and outputs predictions for the target sequence. The design of our model is shown at figure 2. As the name suggests, this model is divided into two sub-models, an encoder and a decoder.

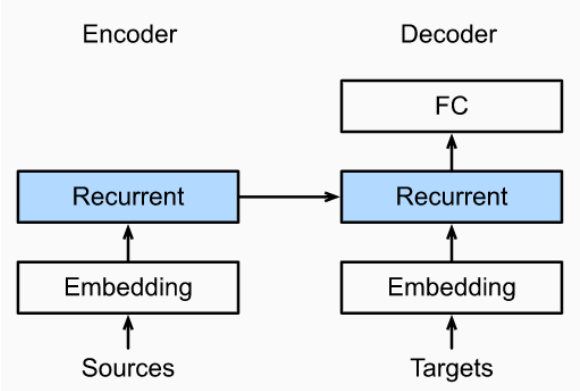

FIG. 2. Layers of the Encoder-Decoder Architecture

The encoder, which gets the source sequence as input, is made of two parts, an embedding layer and a recurrent layer.

The embedding layer converts each element of a sequence into a feature vector. The size of this vector is equal to the size of the embedding layer. All the hyperparameters of the model will be discussed later. However, we can already notice that the input size of the embedding layer must be equal to the size of the source vocabulary. This is the main reason why we didn't use the entire dataset at our disposal, and why we built the vocabularies with words that appear at least twice. Thanks to these simplifications, we were able to reduce the size of the model and save a lot of computation time. This came at the cost of the accuracy of our model.

The feature vector then goes through a recurrent layer. In our case, this layer is actually a 2-layer LSTM. In paper [5], they used a 4-layer LSTM but we decided to limit ourselves to two layers to save computation time. We get two things out of this LSTM layer, an output and the hidden state (actually a hidden state and a cell state) of the LSTM. In this project, we are not interested in the output so we ignore it. The hidden

state however, can be seen as a context vector. And this is what we want to feed to the decoder.

The hidden and the cell states are 3 dimensional tensors of shape $(n\_layers, batch\_size, hidden\_size)$. n_layers and $hidden\_size$ being the number of layers and the size of the LSTM respectively.

The decoder receives as input the hidden and cell states of the encoder as well as the groundtruth target sequence. This sequence is used to apply teacher forcing during training. The decoder is very similar to the encoder in the sens that it has an embedding and a LSTM layer. However, an additional linear layer is needed here in order to make the actual predictions.

The input size of the embedding layer is now equal to the size of the target vocabulary and its output (the feature vector) is of the same size as the one of the encoder. The recurrent part of the decoder is a 2- layer LSTM with the same dimensions as in the encoder.

We can now combine these two components to build a full Encoder-Decoder module. In the forward pass, the source sequence is fed to the encoder to get the hidden and cell states. These we be used as the initial states of the decoder.

Once the encoding is done, we feed the hidden and cell states to the decoder along with the first token of the target sequence. This token will always be *"bos"*. We get an output which is the prediction for the first token and new hidden and cell states.

Then we can feed the decoder with the new hidden and cell states and the predicted token and repeat this operation until the *"eos"* token is predicted.

As mentioned earlier, during training we are using teacher forcing in order to speed up the learning of the model. This is done by randomly feeding the decoder with the actual groundtruth token instead of the predicted one, with some probability p. In this case with decided to set this probability to 0.5, meaning that the groundtruth token is fed to the decoder half of the time. During evaluation this mechanism is deactivated since the goal is to evaluate the predictions of the model.

F. Hyperparameters

As already stated, the input and output sizes of the model are equal to the source and target vocabulary sizes respectively. We decided to set the size of the embedding layers to 128 and the size of the LSTM layers to 256. Along with that, we set the learning rate to 0.005 and the batch size to 512.

G. Training and Testing

We used the Adam optimizer which is one the default optimizers in deep learning. And the criterion we are using is the Cross Entropy Loss since our problem can be seen as a kind of classification problem.

When testing the model, we record an additional metric which is the BLEU-score. This score gives a value between 0 and 1 that represents the accuracy of the predicted translation with respect to the groundtruth. A BLEU-score of 0 means that the prediction does not make any sens, while a BLEU-score of 1 means that the prediction is perfect.

First we trained the model for 15 epochs. Both training and test losses decreased consistently until epoch 13, when the test loss became slightly worse. However, we assumed this was just a random set back in the training process because we got a better model at epoch 14.

Then at epoch 15, the test loss increased again. To make sure this was not another random set back, we decided to train the model for 5 more epochs. This allowed us to verify that our model was starting to overfit the training data. Indeed, the training loss kept getting lower but the test loss was consistently increasing after epoch 14. We then concluded that the model obtained at epoch 14 was the best one.

We trained our model on GPUs provided by Paperspace.com. In terms of training time, it took about 2 hours and 14 minutes for the first 15 epochs. And an additional 45 minutes for the last 5 epochs. Each epoch lasted around 9 minutes.

IV. RESULTS

The training loss started at about 2.53 and kept decreasing to reach a value around 0.74 after epoch 20.

The test loss started around 2.46 and decreased to a value close to 1.6 at epoch 14. After that, it started increasing until the end of the training, when it reached a value 1.74. A plot of these losses is shown at the figure 3.

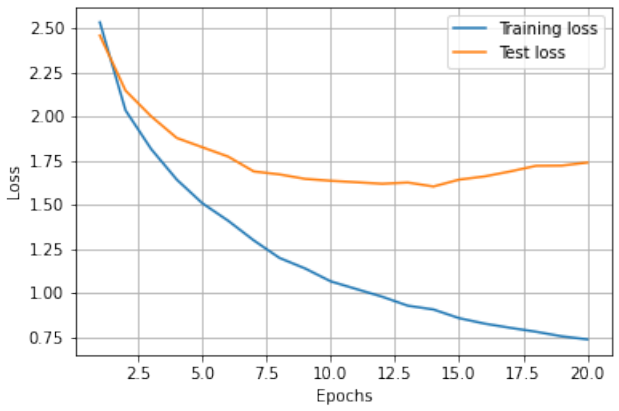

FIG. 3. Training and test losses

During each epoch we also computed the average BLEU-score on the predictions for the test set. The average score started around 0.175, which means that the predicted translations are really bad. Then the score increased steadily until epoch 14, when it was around 0.545. At this point, the BLEU-score reached a plateau and no significant improvements were made during the last 6 epochs. A plot of the average BLEU-score during training is shown at figure 4.

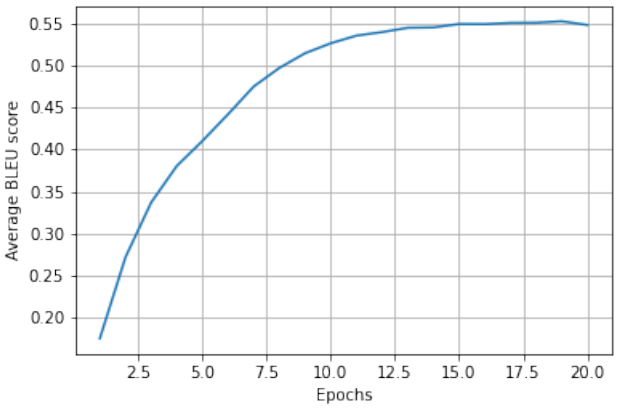

FIG. 4. Average BLEU-score during training

These observations comfort us in the idea that the best model is the one obtained at epoch 14.

After saving this model, we are now able to use it in order to actually translate new sentences. Table II shows four predictions made by our model, along with the translation from Google Translate [6] for comparison.

The first two sentences are intentionally shorter (4 words). We can see that our prediction for the first sentence is perfect. For the second one, our model only got the first half of the sentence right.

English	It is hot today
Our prediction	il fait chaud aujourd hui
Google Translate	Il fait chaud aujourd'hui
English	The sun is shining
Our prediction	le soleil se lève
Google Translate	Le soleil brille
English	Today was a long and tiring day
Our prediction	aujourd hui la journée était une long
Google Translate	Aujourd'hui a été une journée longue et fatigante
English	Today was a long and tiring day, I need some sleep
Our prediction	aujourd hui fut chanceux de plus d temps
Google Translate	Aujourd'hui a été une journée longue et fatigante, j'ai besoin de sommeil

TABLE II. Comparison of some of our predictions

The third sentence is a little longer (7 words), the beginning of our prediction is quite good. However, the last part is completely wrong.

Finally, on a long sentence (11 words) our prediction goes completely off tracks and only the first (and arguably the second) words are correct.

A last observation we can make is that our model does not have the ability to predict uppercases our any punctuation mark. This is obviously due to the simplifications we made on the data during preprocessing.

V. POSSIBLE IMPROVEMENTS

In this project we had limited time and resources at our disposal, so we were not able to dig as deep as we would have wanted in the possibilities of the Encoder-Decoder design. If we had the opportunity to spend more time on this work, here are some ideas we could explore:

1. We could add an attention mechanism to our model, more specifically Bahdanau attention. Our model as is, compresses the input sequence into a single context vector and assumes that this vector contains enough information to generate the output sequence. This can become challenging for long sequences. Instead, an attention mechanism is able to dynamically select only the relevant part of the input in order to generate a specific output token.

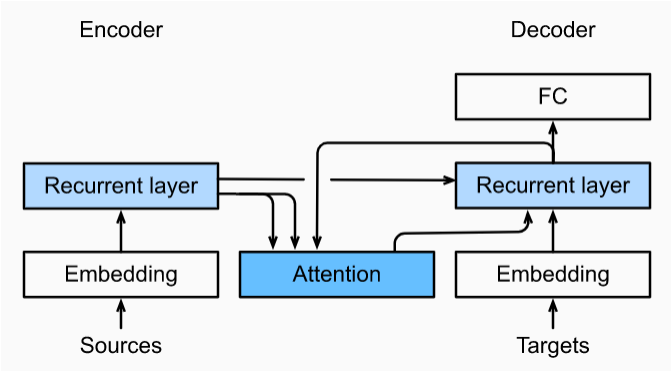

FIG. 5. Encoder-Decoder with Attention

2. We could also make use of the entire dataset at our disposal (and potentially other datasets) to train and test the model. This way the model would be able to better capture the intended message of a sentence (thanks to the duplicates), as well as being able to translate a larger number of words by increasing the size of the vocabularies.

3. It is stated in [5] that reversing the source sequence before feeding it to the encoder yields to better performances of the LSTM layer, hence a better overall prediction. If time had allow it, we would have liked to explore this idea.

4. A nice experiment could also be to build the model using GRU instead of LSTM, and compare the results both for the training time and the accuracy of the predictions. In theory, the performance of GRU deteriorates much faster than that of LSTM when increasing the size of the sequences. This experiment could be a good opportunity to confirm that empirically.

5. Finally, we could explore architectures other than the Encoder-Decoder. The field of deep learning offers a large variety of methods to tackle a problem like the one we have here.

VI. CONCLUSION

In this project, we were able to build a machine translator using recurrent neural networks. Even though the accuracy of our model is nowhere close to what modern translators can do, we were actually surprised by the results we were able to get. Indeed, even with a very limited dataset and some simplifications concerning the vocabulary, were are able to produce decent translations on short sentences.

These results lead us to believe that by adding some features to our model and with a bigger dataset, we could actually end up with a model able to match or even beat the performances of statistical machine translators.

REFERENCES

[1] *Dataset*. URL: `http://www.manythings.org/anki/`.

[2] *Tatoeba Project*. URL: `https://tatoeba.org/en`.

[3] *Encoder-Decoder Architecture*. URL: `https://d2l.ai/chapter_recurrent-modern/encoder-decoder.html`.

[4] *BLEU-score*. URL: `https://en.wikipedia.org/wiki/BLEU`.

[5] I. Sutskever, O. Vinyals, Q. V. Le. "Sequence to Sequence Learning with Neural Networks". In: (2014). URL: `https://arxiv.org/pdf/1409.3215.pdf`.

[6] *Google Translate*. URL: `https://translate.google.be/`.