CentraleSupélec

Big Data Research Project

Buse OZER
Eugen PATRASCU

Machine Translation

Advisor: Nacéra BENNACER - CentraleSupelec nacera.bennacer@centralesupelec.fr

Contents

List of Figures

Chapter 1

Introduction

Machine Translation is one of the subdivisions of computational linguistics, aiming to provide end to end automatic translation of a speech or text from one language to another. Many studies either intend to improve the quality of translations or the robustness of these methods and subsequently evaluate their models on different benchmarks. It has been an active research area in the past decades as the demand of machine translation has escalated drastically.

Machine translation is a challenging research area considering that various languages owning different rules and exceptions exist in the word. All these challenges mean that machine translation is still not perfect despite all the years of research and the very complex models that have been created for translation. Some of the challenges were presented in [Garg 2018]. To begin with, different languages may have completely different structures. A word in a language can have multiple meanings and not all words in a language have equivalent words in another language. Moreover, a tense that exists in one language may not exist in another language. The meaning of a word, phrase or expression requires a good level of understanding of the text based on context and culture. Also, there is no unique way of a word, phrase or sentence to be translated, with multiple translations being possible.

There are many machine translation models that have been proposed over the years, each with different architectural characteristics. In this project we develop a machine translation prototype system, with the main objective being to analyse which characteristics or strategies could bring the highest improvements on the quality of machine translation, considering the constraint of having limited computational resources. In chapter 2 we present a literature review of machine translation, covering evaluation methods and the evolution of the topic with a focus on neural machine translation. In chapter 3 we discuss the base model presented above and its Keras implementation. Moreover, we detail the potential adjustments of the base model, in preparation for the following experiments. The datasets used and their preprocessing are also presented. Chapter 4 shows the results of each experiment and concludes which strategies would bring better scores.

Chapter 2

Related Work

This chapter will start by providing an overview of the current evaluation metrics for machine translation techniques, with a focus on the BLEU score as being the currently most widely used metric. The next section will provide a summary of the history of machine translation, mentioning the three main types of models: rule-based, statistical-based and neural-based. We have then decided to detail more the neural based machine translation techniques, as they have been providing the highest accuracy in machine translation since their introduction in 2015. This chapter will end by detailing the current highest accuracy models in machine translation.

2.1 Evaluation of Machine Translation

Evaluation of machine translation is not a straightforward task because, as we discussed previously, there is no unique way to translate a word, phrase or sentence. There are several methodologies for the evaluation of translation performance. Some of them are BLEU (Bilingual Evaluation Understudy), F-measure, NIST, Meteor, Rouge and Word Error Rate. The most widely used metric is BLEU which is an algorithm for evaluating the quality of a machine-translated output. The metric is considered as the correspondence between machine translation and human translation, which is that "the closer a machine translation is to a professional human translation, the better it is". Among other metrics, BLEU is the most correlated one with human judgments of quality. The intuition behind BLEU score is checking if words are generated by the machine that appears at least once in human-generated references. BLEU score is always a number between 0 and 1 and it computes based on N-gram metric. There are some disagreements about the assessments of blue score because there are only comparing a machine-translated output with a few references in a data set. However, sentences can be translated in many different ways and some combinations may not exist in a data set which leads to a bad score in BLEU metric [Li 2014]. Despite these arguments, BLEU has been reported as correlating well with human judgment and remains a benchmark for the assessment of any new methodology and evaluation metric. Moreover, WMT2014 English-German, WMT2014 English-French [WMT2014] is the most popular benchmarks hence most of the time, researches test their methodologies with BLEU metric against these benchmarks in order to evaluate. They are data sets that provide millions of sentence pairs of two given languages (36 million for English to French and 5 million for English to German).

BLEU score is calculated according to the formula based on N-gram as follows [BLE].

$$P(i) = \frac{Matched(i)}{H(i)}$$

Figure 2.1: Percentage of N-gram

In order to calculate the bleu score,the predicted/hypothesis sentence is compared against reference sentences. The percentage of i-gram tuples in the hypothesis that also occur in the references is computed where H(i) is the number of i-gram tuples in the predicted sentence and the Matched(i) is calculated as follows.

$$Matched(i) = \sum_{t_i} \min\{C_h(t_i), \max_j C_{hj}(t_i)\}$$

Figure 2.2: The formulation of Matched(i)

Ch(ti) is the number of times ti occurs in the hypothesis; Chj(ti) is the number of times ti occurs in reference j of this hypothesis. A hypothesis may have multiple references therefore we get the maximum number of times ti occurs among all the references. For instance in 2.3, the predicted/hypothesis sentence of the machine learning model is shown as candidate and there are 2 reference sentences in order to show the accurate translation of the corresponding sentence. First, start by calculating when N=1 which means we check word by word.

Candidate	the	the	the	the	the	the	the
Reference 1	the	cat	is	on	the	mat	
Reference 2	there	is	a	cat	on	the	mat

Figure 2.3: Example from wikipedia

In the example above, word "the" occurs 2 times in reference 1 and occurs 1 time in reference 2. We get the maximum number of times "the" occurs in all references which is 2. And "the" apperars 7 times in the hypothesis sentence thus $P = 2/7$.

After that we calculate for each i-gram where i=1,2,3..,N and get the average and return as bleu score.

2.2 History of Machine Translation

2.2.1 Rule based and Statistical based Machine Translation

There are four major methodologies when we examine the evolution of machine translation [Garg 2018]. Early researches were focused on Rule-based Machine Translation

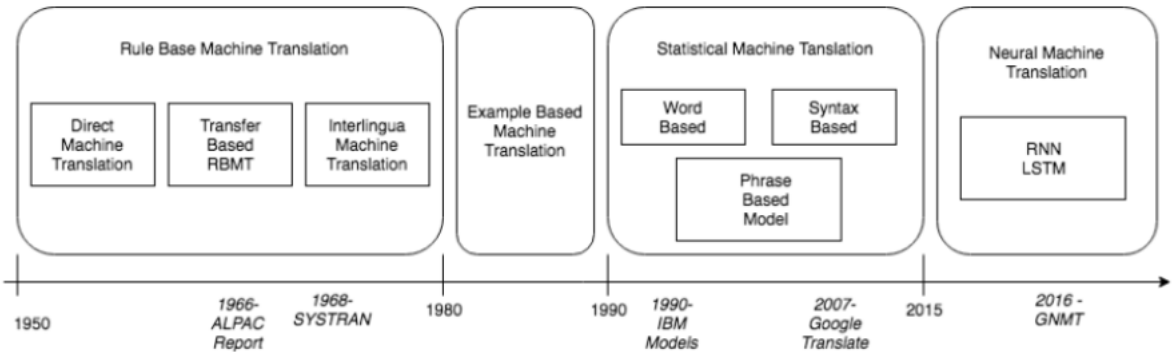

Figure 2.4: History of machine translation

(RBMT). All rules and exceptions of a language were asserted to RBMT system. It falls into three categories which are Direct systems, Transfer RBMT systems, and Interlingual RBMT systems [Carbonell 1978]. These systems are based on mapping an input sentence directly to an output sentence. However, this approach is not compatible for some particular languages where possess different structures such as Japanese and English. In 1980, Example-based Machine Translation approach was proposed by Nagao et al [Nagao 1984] which relies on a big data set of sentence examples and their translations to learn the correspondence between two languages and their structure. It was a revolutionary study by that time because it helped further studies build systems that do not rely on manual rules and exceptions.

In the 1990s, Statistical Machine Translation (SMT) was introduced by Brown et al [Brown 1990]. It falls into three categories which are word-based, syntax-based and phrased based model. Phrased based model was the most widely used approach before the current state-of-the-art. STM is a paradigm where generates translations based on a statistical model and it relies on parameters that are derived from a distribution of a set of bilingual pairs of sentences. This method is based on conditional probability which calculates the probability of a target language phrase of a given source language phrase for each pair of source and target language phrases. However, there are some disadvantages of this approach because STM ignores phrase similarities which can be denoted by parameters and some features. Ignoring phrase similarities leads to sparsity problem. After 2014, neural network approaches have gained popularity owing to their state-of-the-art results across multiple major languages. We elaborate different approaches of neural network in detail.

2.2.2 Neural Machine Translation

One of the first papers that used Neural Networks to improve Machine Translation was [Cho 2014]. They proposed an encoder-decoder architecture, that used Recurrent Neural Networks in both the encoder and the decoder as seen in figure 2.5. This encoder-decoder idea has constituted the base of many machine translation models over the years. The main idea behind this architecture is that it learns the conditional distribution over the

words in the translated sentence, given the input sequence of words (the sentence in the source language). The first component, the encoder, reads in the input sequence word by word until it finds an "end of sequence" character, that signalises that the input phrase has finished. This is done using an RNN, which at the end outputs an encoding c of the input sentence (a summary of the sentence). The encoding c will be passed next to the decoder, which is also an RNN, and which will translate the source sentence word by word. At each step, the decoder translates a word considering the previous translated words and the encoding of the original sentence.

The hidden state of the decoder at time t is computed with:

$$h_t = f(h_{t-1}, y_{t-1}, c) \tag{2.1}$$

And the conditional distribution over the next word to be translated is computed as follows:

$$p(y_t|y_{t-1}, \ldots, y_1) = g(h_t, y_{t-1}, c) \tag{2.2}$$

The architecture trains the two components jointly to maximise the conditional likelihood.

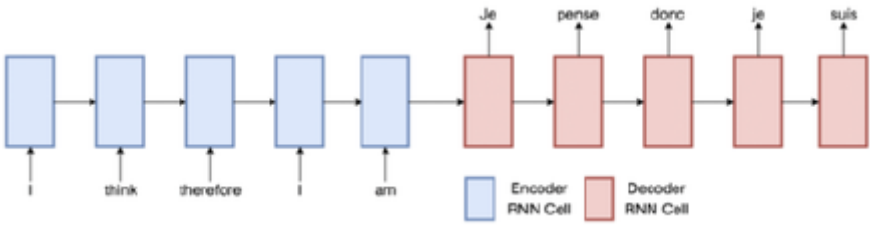

Figure 2.5: Encoder-decoder architecture example for machine translation, from [Garg 2018]

[Cho 2014] used this architecture to learn phrase translation probabilities, to further enrich an existing statistical based translation system. They used the WMT 2014 English-French corpus and the new features obtained through the encoder-decoder architecture and showed this improved the baseline Statistical Machine Translation system.

[Sutskever 2014] built on top of the of the neural network architecture from [Cho 2014] investigating the reliability of deep recurrent neural networks in the domain of machine translation. The initial model had some drawbacks, such as not being able to capture long range dependencies in long sentences due to using basic RNN units in the model. In order to solve some of these issue, the new model brought the following changes to the initial architecture:

- they used LSTM units in both the encoder and the decoder, instead of the basic RNN units
- they trained the model on full sentences not just phrases
- they also used deep LSTMs (4-6 layers)

Another strategy they applied was to reverse the source sentences in the training corpus (while keeping the translated ones unreversed), which showed to have achieved a higher BLEU score. Informally, this achieved a similar functionality to bidirectional RNNs allowing to capture word dependencies in both directions. Even though this system did not achieve the state of the art accuracy at the time, it had a close performance. In 2014 the state of the art was still a statistical machine translation model.

One of the limitations of the [Sutskever 2014] model is that it was not able to translate correctly long sentences. It was shown that for sentences over 20+ words the accuracy started to decrease [Bahdanau 2014]. The reason for that was that one single fixed-length vector embedding of a sentence cannot easily capture the whole information contained, in the case of long sentences. In order to deal with this issue, [Bahdanau 2014] introduced the idea of an attention mechanism. In this case, the encoder does not embed the sequence of input words into only one vector, but into a sequence of vectors. The decoder will only look at a subset of these vectors at any time that it generates a token of the translation. Informally, this is based on the idea that in order to translate a certain word from the source language, it is not needed to look at the whole input sentence, but it is enough to look at a certain sequence of words. The conditional probability of the translated word at time t is defined as:

$$p(y_t|y_1, ..., y_{t-1}, x) = g(y_{t-1}, s_t, c_t) \tag{2.3}$$

where s_i is a hidden state of the decoder generated by:

$$s_t = f(s_{t-1}, y_{t-1}, c_t) \tag{2.4}$$

The context vector c_t is generated as a weighted sum of the input sequence encodings.

$$c_t = \sum_{i=1}^{T^x} \alpha_{ti} h_i \tag{2.5}$$

where a_{ti} are the weights of the input sequence encodings h_i at time t. The weights a_{ti} are computed using a feed-forward neural network. They are based on an alignment model that evaluate how well the positions of the inputs and outputs match. In this approach, for the encoder they used a bidirectional RNN and for the decoder they used a simple RNN. The evaluation was done on WMT 2014 English-French corpus.

The results of the evaluations of the three influential papers were as follows.

Approach	BLEU Score (WMT 2014 En-Fr)
[Cho 2014]	34.54
[Sutskever 2014]	36.5
[Bahdanau 2014]	28.45
State of the art	37

Table 2.1: The BLEU scores of the presented approaches versus 2014 state of the art

As per table 2.1 the state of the art in 2014 was still a Statistical Machine Translation approach. However, the papers presented were very influential and they paved the way for the further Neural Machine Translation approaches that became state of the art. We

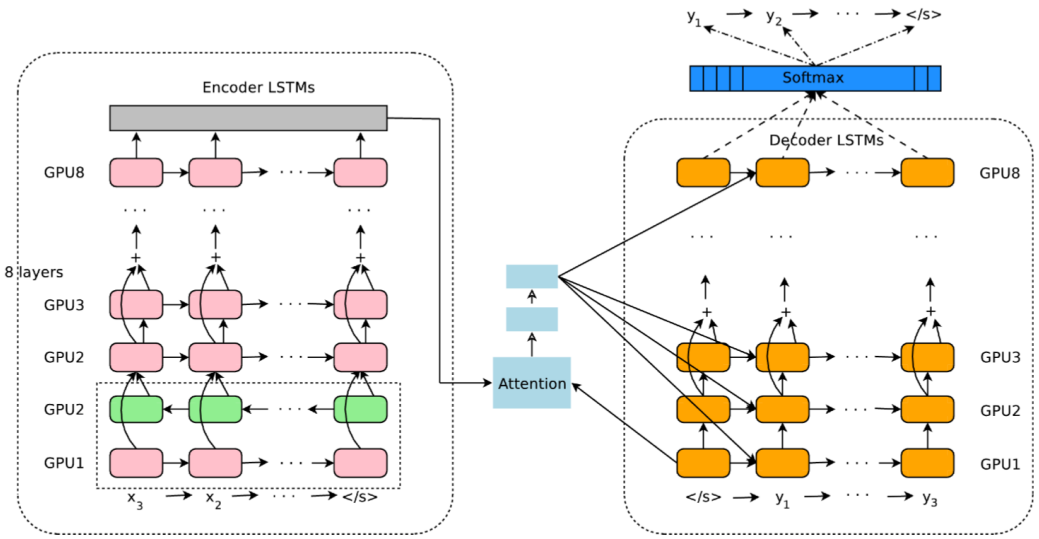

Figure 2.6: Google's Machine Translation Model

can observe that the approach of [Sutskever 2014] gave results close to the state of the art, showing that deep neural networks with LSTM units were an effective approach, better than the combination of neural networks and statistical machine translation presented by [Cho 2014]. [Bahdanau 2014] introduced the idea of attention mechanism, but the reason why their score was lower was that they used a simple one layer encoder-decoder approach with RNN units, which under-performed compared to more sophisticated approaches.

While there had been significant innovation in machine translation, there were still some challenges that needed to be overcome. Some of them were as follows:

- It was hard to build deep RNNs due to the vanishing and exploding gradient problem.
- The training process was very slow and it needed to be optimized.
- There were difficulties in learning dependencies in a sentence as a compact vector could not store all information for long sentences.

Google developed a machine translation system in order to address the problems that are discussed above. They published Google's Neural Machine Translation System (GNMT) [Wu 2016] and it was state-of-the-art in 2016.

As can be seen in figure 2.6, there is an encoder, decoder and attention mechanism that sits between encoder and decoder. Encoder consists of 8 layers of LSTM and the bottom encoder is bi-directional which allows the network to gather information from left to right and right to left of a sentence. The remaining layers of encoder are uni-directional. During training, the first layer of encoder which is called bi-directional layer can be processed in parallel which allows maximum parallelization. After they finish, remaining layers process in sequence, there is no parallelism for this part of the encoder. In addition, decoder consists of 8 LSTM layers which are bi-directional. In this architecture, one model is partitioned on 8 different GPUs and each layer is on separate GPU. Softmax layer is also partitioned on different GPUs to retain the parallelism as much as possible.

Moreover, there is an attention mechanism that connects the top layer of encoder and the bottom layer of decoder. When decoder generates a word, this mechanism allows decoder to process only relevant parts of a sentence. They wanted to use more layers since Deep LSTMs gives better accuracy than standard LSTMs. However, standard stacked LSTM can work well up to 4 layers due to vanishing and exploding gradient problem. In this architecture, they were able to achieve having 8 LSTM layers by using residual connections that enable to simplifies the network by skipping some connections as there are fewer layers to propagate through. After training the model for 3 days, it gives 39.92 blue scores against WMT 14 English-French benchmark, whereas it scores 24.60 against WMT 14 English-German in the evaluation.

However, there are some weaknesses of Google's model because the sequential nature of RNNs makes training very slow and difficulties on learning long-range dependencies still remain. Therefore Facebook Research group proposed Convolutional Sequence Model [Gehring 2017].

They built an architecture based entirely on convolutional neural networks. This approach is faster because the computation of a current time step does not depend on the computations of previous time steps which demonstrates us maximum parallelization. Unlike one attention model of Google, their approach is based on gating and multiple attention steps, since they use attention mechanism in every decoder layer. In addition, representations are built hierarchically hence it is easier to discover the compositional structure and capturing dependencies for long sentences. After training the model for 6-7.5 days on a single GPU, it gives 41.44 blue scores against WMT 14 English-French benchmark, whereas it scores 26.43 against WMT 14 English-German. These results show us that Facebook's model outperforms Google's method on both benchmarks. After this paper, it paved the way for further research in building machine translation systems based on convolutional neural networks.

2.3 Transformer architecture: currently highest BLEU score

Even though the convolutional approaches solved some of the issues encountered by previous machine translation models, they still had challenges. Two of the main problems posed by them were: the high complexity and not being able to capture all dependencies between words in a phrase. An architecture that managed to solve both of those issues was the Transformer architecture presented in [Vaswani 2017]. It managed to capture all three types of dependencies present in machine translation: dependencies between input and output, between input tokens, and between output tokens.

The architecture of the Transformer model can be observed in figure 2.7. It does not contain any RNN, CNN or LSTM units, as it mainly consists of a mechanism called self-attention combined with feed forward neural networks. It contains 6 encoder and 6 decoder layers. The encoder works as described next. It starts by generating initial embeddings for each word, considering the input embeddings and positional encodings. The positional encodings are necessary to capture the order of the words in the input sentence. Using self-attention, it aggregates information from all of the other words, generating a new representation per word. It uses a multi-head attention which means that there are multiple representations created for each word which are then aggregated

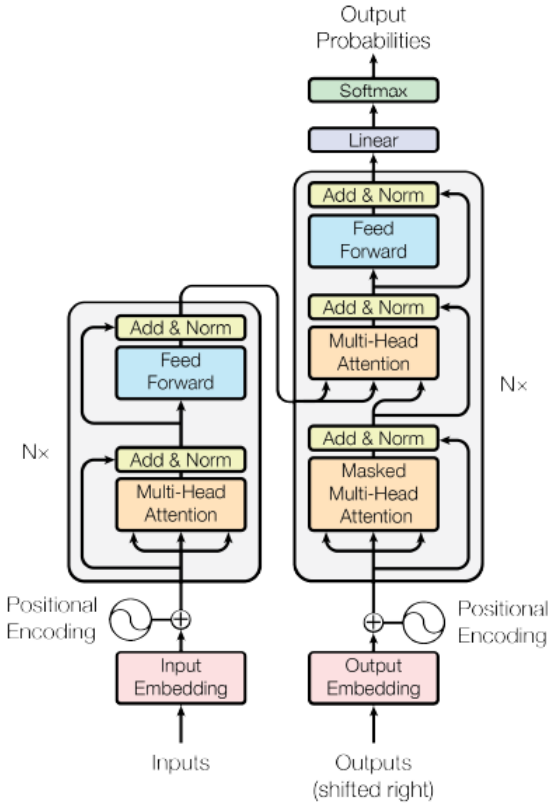

Figure 2.7: Transformed architecture, from [Vaswani 2017]

together. These embeddings are independently fed into the feed-forward network, which is useful for the parallelization of the model. Between each sub-layer, there is a residual connection followed by a layer normalization. The same operations are repeated through the 6 layers and ultimately it outputs the encodings of the words.

The decoder works as follows. It starts with the inputs which are formed from the encoder outputs and the positional encoding. It uses 2 attention mechanisms: within the decoder to capture the context and encoder-decoder attention so that the model knows on what input words to focus when it translates a certain word. The self-attention layer is only allowed to model dependencies with earlier positions in the output sequence, the further (not translated) words being masked. The output of the encoder is transformed into a set of attention vectors to be used by each decoder in its Multi-Head Attention layer which helps the decoder focus on appropriate places in the input sequence. Similarly to the encoder, there are: residual connections, layer norm and feed forward networks. The output of each step is fed to the bottom decoder in the next step and the process is repeated until a special symbol is reached indicating the transformer decoder has completed its output. The Linear layer and Softmax are used to retrieve the translated words from the final output vectors.

As presented in the architecture, the Transformer model's main feature is the use of the self-attention mechanism, which brings the following benefits:

- Lower total computational complexity per layer

- Larger amount of computation that can be parallelized
- Improved dealing with long-range dependencies
- As side benefit, self-attention could yield more interpretable models

The Transformer models has been used in many machine translation approaches since, such as [So 2019], [Shaw 2018], [Kong 2019], including the current state of the art [Edunov 2018]. Facebook AI Research team achieved state of the art results by enriching the parallel training corpus with hundreds of millions of back-translated sentences. Their paper describe various strategies to create synthetic source sentences for the training data, concluding that sampling and noised beam signal provided the best results. The transformer architecture with enriched data achieves state-of-the-art results on WMT'14 English-French and WMT'14 English-German datasets with 45.6 BLEU and 35 BLEU respectively.

This literature review serves as a good reference work for the implementation of the machine translation system. The main elements of our approach are based on the basic encoder decoder architecture from [Cho 2014]'s work. Similarly, we considered [Sutskever 2014]'s approach of using LSTM rather than RNN in order to capture long range dependencies in long sentences. However, we used fewer layers than [Sutskever 2014]'s approach due to the speed and computational issues. Later, we got inspired by some additional features and attention mechanism of [Bahdanau 2014]'s paper. However, we could not go beyond more than [Sutskever 2014]'s approach such as Google's Neural Machine Translation System (GNMT) [Wu 2016] due to the time constraints.

Chapter 3

Methodology

This chapter starts by providing the problem definition of exploring what strategies bring higher improvements in machine translation quality. It describes the data that has been used to train and test the models, and present the pre-processing steps that have been applied. Next, it details the base neural network encoder-decoder model that we are using, together with its implementation using Keras[1]. Next, it explains the implementation of the strategies that we are exploring: architectural changes in the neural network and word embeddings.

3.1 Problem Definition

Achieving highly accurate translations is still a big challenge today. Therefore, there is much research going on in order to improve the quality of machine translation. Our aim is analyzing the impact of different improvement strategies on the quality of machine translation, considering the constraint of having limited computational resources. We will start by a base encoder-decoder model, on top of which we will explore the following strategies to check their effects on the translation accuracy:

- Using different amounts of training data
- Using a bidirectional LSTM encoder versus an unidirectional LSTM encoder
- Using a shallow network (1 layer encoder-decoder) versus a deep neural network (multiple layer encoder decoder)
- Using different numbers of nodes within a LSTM unit
- Using pre-trained word embeddings in the encoder

3.2 Data

We used a French-English sentence corpus provided by Tatoeba project which is a large database that consists of bilingual sentence pairs for a wide range of languages. It is an open source project that grows incrementally by voluntary contribution of its members. In the French-English sentence corpus, each line contains a single English sentence and its corresponding French translation, separated by a tab character from each other. The corpus is general purpose, not domain-specific, and it consists of 170651 pairs of sentences for both languages. Initially, we used 10000 sentence pairs in total as we have limited computational resources. 9000 sentence pairs were used for training the model and the

[1] https://keras.io/

remaining 1000 for the evaluation of the model. French was used as a source language and English as a target language, meaning that a French sentence was given as an input and the model predicted an English translation of the given sentence. There is some information given below about the data set of 10000 sentence pairs:

- The vocabulary size of English sentences is 2123.
- The minimum length of English sentences is 1.
- The maximum length of English sentences is 5.
- The vocabulary size of French sentences is 4374.
- The minimum length of French sentences is 1.
- The maximum length of French sentences is 10.

For some of the experiments we used 20000 and 30000 sentence pairs, keeping the same ratio of training to test of 9 to 1. For another experiment we used a dataset of German to English sentences to test how the translation accuracy changed.

3.2.1 Pre-processing

We performed some initial pre-processing operations in order to prepare the data for the translation. The French-English data set includes punctuation such as in the words “weren’t” and “pre-processing“, different letter cases, different French translations of an English word, and special characters of French language such as in the word "préfère". We will perform the following operations:

- tokenise all phrases word by word
- convert all words to lowercase for consistency in the dataset
- normalise unicode characters
- removing all the non-alphabetic tokens
- removing punctuation

All the pre-processing operations were done in python. On top of these, we performed other preprocessing steps in order to use word-embeddings and one hot encoding, as explained in the sections above. Similar pre-processing was done in the case of German to English dataset.

3.3 Initial model

Our starting architecture is a standard encoder-decoder model, whose theoretical background is explained in section 2.2.2. The encoder is a recurrent neural network with LSTM units. It reads the input sentence word by word sequentially, where each word is represented by a vector, which could be created by different strategies such as using one

hot encodings or word embeddings. The output is an vectorised encoding of the initial sentence.

The decoder is also a recurrent neural network with LSTM units, which takes as an input the mentioned encoding. It outputs the translation of the sentence word by word, considering the distribution of the possible translations and picking the word with the highest probability. The choice of LSTM units is motivated by their effectiveness in translating longer phrases. Basic RNN units have the problem of vanishing gradient, and thus they would not provide accurate translations of long sentences, as described in the 2.

For the optimisation of the model we used the Adam optimiser. Adam is a very popular optimisation algorithm, and the main idea is that it employs an individual learning rate for each weight in the network. It is one of the current best algorithms for optimisation and it is easy to configure. Our problem is a multi-class classification problem, therefore we have decided to use the categorical cross-entropy loss function.

On top of this initial encoder-decoder architecture, we have experimented with some improvement approaches as explained at the beginning of the chapter.

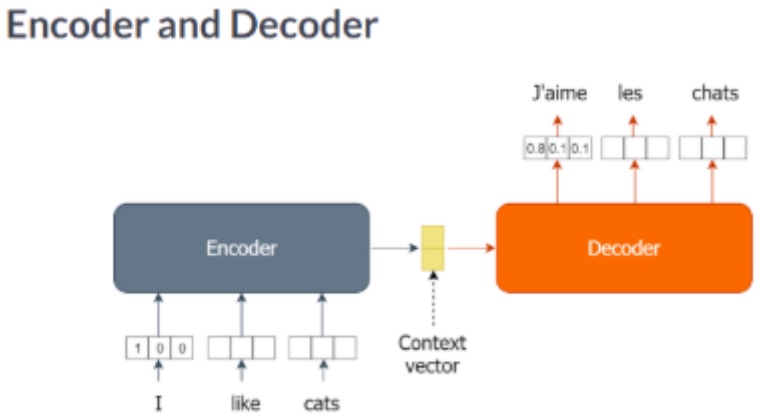

Figure 3.1: Encoder-decoder architecture example for machine translation, from https://www.datacamp.com/home

3.4 Keras implementation

For the implementation of the encoder-decoder model we are using Python with the Keras library. We used the following modules from the library: preprocessing, utils and layers. As part of the *preprocessing* module, we used *tokenizer* which helps us map words to integers and define the dictionary size of associated language. This mapping is necessary to construct word embeddings of the input words later on. After that, we used the *utils* module in order to have one-hot-encoded words of a phrase. We need one-hot-encoded vector for output sequences because we will predict the probability of each word in the vocabulary in order to produce the translated phrase.

Moreover, for the development of the deep learning model, we defined a model object

specifying the various layers used from the input until the output of the model. In the implementation, we used French sentences as input and English sentences as output. First of all, since machine translation is a sequence to sequence model, we used a *Sequential* object. The model contains the following layers as presented in figure 3.2:

- Embedding layer (for the source sentence)
- LSTM layer (for the encoder)
- RepeatVector layer (link between the encoder and the decoder)
- LSTM layer (for the decoder)
- TimeDistributed-Dense layer (for the target sentence)

```
def create_model(vocab_size_src, vocab_size_trg, seq_size_src, seq_size_trg, n_nodes):
    nn = Sequential()
    nn.add(Embedding(vocab_size_src, n_nodes, input_length=seq_size_src, mask_zero=True))
    nn.add(LSTM(n_nodes))
    nn.add(RepeatVector(seq_size_trg))
    nn.add(LSTM(n_nodes, return_sequences=True))
    nn.add(TimeDistributed(Dense(vocab_size_trg, activation='softmax')))
    return nn
```

Figure 3.2: Code of the machine translation system in Keras

We will explain each layer one by one in detail with a visual representation of the implementation represented in figure 3.3.

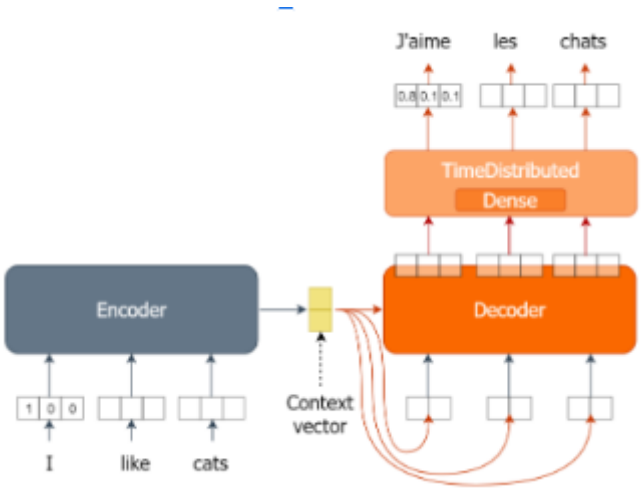

Figure 3.3: Implementation of the machine translation system in Keras, from https://www.datacamp.com/home

Firstly, we used a *model.add.Embedding()* in order to represent the input words as vectors to the encoder (i.e. word embeddings). The most important advantage of using word embeddings is allowing the model to identify words with similar meanings as they have a similar representation. It requires the vocabulary size of the source language, the

number of features of the word vector to be constructed and maximum length of a source sentence as inputs (in order to pad all the input sentences to the same length).

For the implementation of the encoder, we added one layer of LSTM units taking the number of nodes within an LSTM unit as an input. This number is in practice the same size as the dimensionality of the input word embeddings. The choice of LSTM units is motivated by their effectiveness in translating longer phrases. The output of the encoder is a context vector representation of the input sentence.

Moreover, we required a *RepeatedVector* layer for the model. It was required in order to link together the output of the encoder to the input of the decoder. It adds an extra dimension to our dataset, which is the size of the maximum length of the target sentences. It repeats the context vector the specified number of times, so as to be an input to each LSTM unit in the decoder layer.

For the implementation of the decoder, we added one layer of LSTM units taking the number of nodes within an LSTM unit as a parameter. The inputs to the decoder LSTM units are the context vector of the input phrase, the outputs of the previous LSTM units as well as the hidden states of the previous LSTM units. The decoder will produce a vectorized output at each sequence step, of the dimension equal to the dictionary size of the target language. We needed to transform the output to a valid probability distribution, therefore we introduced an additional layer on top of the decoder to produce the probability of target words by adding a *Dense* layer, as in image 3.4. We applied a softmax activation function in order to return a probability distribution over the classes. In this context, the number of classes equals the dictionary size of the target language. Since the decoder layer outputs a time series, we wrapped the Dense layer with a *TimeDistributed* layer allowing the Dense layer to consume time-series inputs.

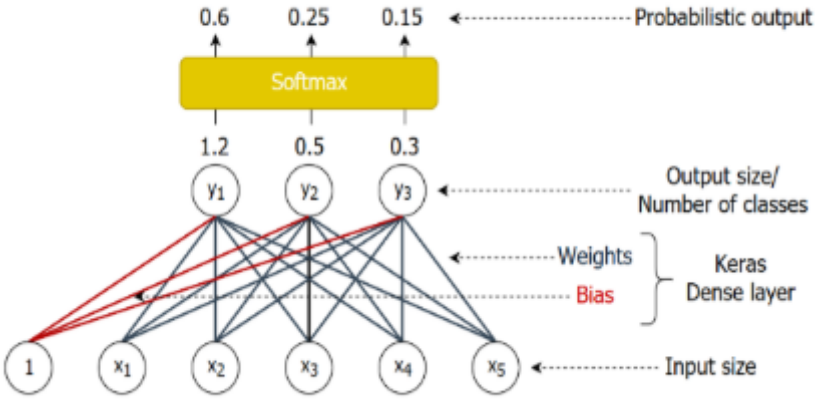

Figure 3.4: Dense layer in Keras for machine translation, from https://www.datacamp.com/home

After defining the model, we compiled the model with *Adam* optimizer and *categorical crossentropy* loss because of the presence of a multi-class classification problem.

3.5 Improvements of the Model

3.5.1 Architectural changes in the neural network

As mentioned at the beginning of the chapter, we consider the following 3 architectural changes to be explored for improved accuracy of the translation model: using a bidirectional layer in the encoder, using deeper networks and using different a different number of hidden units in an LSTM layer. After the implementation of the base encoder-decoder model in Keras, additional changed can be easily added due to the ease of use of the library. Making an LSTM layer bidirectional can be done by applying the *Bidirectional* module to an already existing layer. In order to add more layers to make the network deeper, we can use the *LSTM* multiple times, in both the encoder and the decoder, as needed. However, in the case of the encoder, the LSTM layers have to be connected by setting the lower-level layers to have an output at every sequential unit, instead of only at the last one, as in the case of having just one layer. Similarly, changing the number of units within an LSTM layer can be done by means of changing one parameter.

3.5.2 Word Embedding

Word Embedding is a widely used technique in Natural language processing (NLP). Text classification, sentiment analysis, and machine translation are the typical usage of word embeddings in NLP. In our project, it is crucial to have informative input representation in order have high level of model performance therefore we decided to try a well chosen vector representation for input phrases. This representation should capture the linguistic and semantic meaning of a word as much as possible. In the beginning of the project we used the simplest way of representing each word numerically rather than word embedding. We built a word-to-integer mapping by assigning an integer index to each word therefore we could represent a word as a vector of numbers as follows.

Each word is represented as an n-dimensional vector, where n is the vocabulary size. "1" will be assigned in the position of corresponding word index and "0" will be assigned for the rest of the vector. This process is called one-hot vector encoding. However, there were some problems with one-hot vector encoding. The first problem is the similarity issue. Each word has the same distance from each other in the vector space which means the model is not able capture the similarities between words. The second problem is the vocabulary size. As we increase the vocabulary size, the size of feature vector will increase by the same amount because the dimensionality of one-hot vector is the same as the number of words in the dictionary which might cause a explosion of the feature vector. The last problem is the sparsity of the feature vector. Each word's one-hot vector consists of "0"s except only one "1". Therefore we applied word embedding technique in order to capture the context of a word, semantic and syntactic similarity with other words. A word will have closer spacial position with the words that have similar context unlike one-hot encoding 3.6. The algorithm that is used in order to calculated the distance between words is cosine similarity. The smaller distance yields to a higher similarity in the word embedding space. Word embedding introduces dependence of a word on all the other words, although all the words are independent in one-hot encoding approach. Word embedding is a vector representations of a particular word.

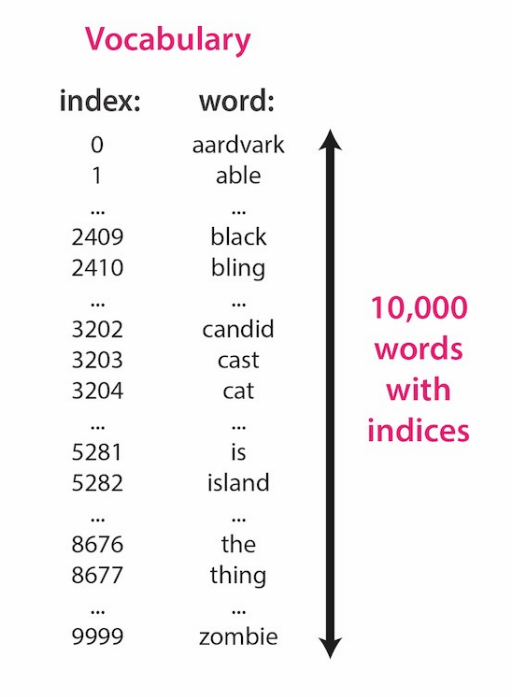

Figure 3.5: Word to index representation, taken by https://towardsdatascience.com

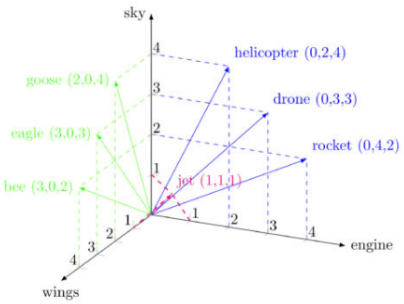

Figure 3.6: Word Embedding, taken by https://corpling.hypotheses.org/495

For instance, word "car" and "bear" are the words semantically quite different than each other therefore embedding space should represent them as vectors which are far from each other. However, word "good" and "great" should be represented as vectors that are quite close each other in the word embedding space. There are some popular pre-trained word embedding models in use as follows.

- Word2Vec (by Google)
- GloVe (by Stanford)
- fastText (by Facebook)

We decided to Word2Vec model which is provided by Google because it provides well-structured documentation and rich resource. Gensim library is used to perform Word2Vec

approach. In the documentation of their website Word2Vec object expalined as follows:

Word2Vec object represents the inner shallow neural network used to train the embeddings. The semantics of the network differ slightly in the two available training modes (CBOW or SG) but you can think of it as a NN with a single projection and hidden layer which we train on the corpus. The weights are then used as our embeddings (which means that the size of the hidden layer is equal to the number of features self.size) [Radimrehurek]

We used pre-trained word embedding for the input sentences therefore we found a data set that consists of french sentences in order to pre-train our model and performed some pre-processing to put them in a desired format and passed all the sentences, the dimensionality of word vectors, the training algorithm as skipgram(sg) into Word2Vec instance and saved the model for future use. It can be seen the implementation as follows.

```
import gensim

model_w2v = gensim.models.Word2Vec(sentences=all_sentences, size=300, sg=1, hs=1)
if not os.path.exists('./WE_models'):
    os.mkdir('./WE_models')
model_w2v.save('WE_models/w2v_sg_300D')
```

Figure 3.7: Word2Vec implementation

After saving the model, we are able to load the pre-trained model and use the word embeddings for the new phrases. We used an example of the word "cours" in order to demonstrate all the similar words in embedding space as follows.

```
#Load the model
model_w2v = gensim.models.Word2Vec.load('WE_models/w2v_sg_300D')
#See the most similar words to cours
model_w2v.wv.most_similar('cours')
```

```
[('turbulences', 0.4291991591453552),
 ('dintro', 0.42504751682281494),
 ('modifications', 0.41741570830345154),
 ('repetitions', 0.41544803977012634),
 ('visites', 0.41166946291923523),
 ('semestre', 0.4116436243057251),
 ('lindependance', 0.39758729934692383),
 ('retouches', 0.39716506004333496),
 ('phases', 0.3952268660068512),
 ('usines', 0.39489084482192993)]
```

Figure 3.8: Word2Vec implementation 2

Chapter 4

Experiments and evaluation

In this chapter, we discuss the results of the different experiments that have been run to evaluate the accuracy of the considered models and strategies. As we discussed previously in Evaluation section of 2, the most widely used metric in machine translation is the BLEU score and we are using it in order to evaluate the translation quality of our model against French-English and German-English corpora. The BLEU score always return a value between 0 and 1 and computes a score based on N-grams. We used a library for evaluating a bleu score that is already implemented by The Natural Language Toolkit which was written for the Python language. We compare different deep learning configurations and make conclusions based on the BLEU score improvement.

4.1 Experiments

We have computed several experiments to check if the results of the translation would improve over the results of a simple encoder-decoder model. We compared all the experiments against the initial model which consists of 1 layer of encoder, 1 layer of decoder with 10000 sentences. We divided all the experiments into 4 main categories as follows.

- Changing the data size
- Changing the architecture of the network, configuring the parameters of the networks
- Considering a different set of language pairs
- Performing pre-trained word embeddings

4.1.1 Impact of Data Size

The first experiment is changing the data size in order to see the effect on the machine translation accuracy. We performed experiments with 1 layer of encoder and 1 layer of decoder with 10000 sentences and 20000 sentences. It can be seen the results of this experiment in 4.1.

	1 LSTM layer of encoder, 1 LSTM layer of decoder	
Metric	10000 phrases	20000 phrases
BLEU-1	0.451	0.576
BLEU-2	0.326	0.462
BLEU-3	0.265	0.402
BLEU-4	0.135	0.238

Figure 4.1: Results of 10k and 20k

As a conclusion, as we increased the data size, it increased the bleu scores of each n-gram therefore we can see the benefit of increasing the data size in order to have better results. However, it requires time and computational resource.

4.1.2 Impact of the Architecture of the Model

Regarding the architecture of the model, we have performed several experiments to check the effects of different characteristics on the translation accuracy.

4.1.2.1 Impact of Bidirectional Layer

A bidirectional layer brings the benefit of capturing the dependencies between the words in the phrase in both directions. This increased the slightly the BLEU scores on each of the n-grams, as we can see in figure 4.2.

Metric	Base model (10K)	Bidirectional encoder (10K)
BLEU-1	0.54732	0.56076
BLEU-2	0.44148	0.44965
BLEU-3	0.35848	0.38341
BLEU-4	0.15688	0.18903

Figure 4.2: Results of unidirectional vs bidirectional LSTM layer in the encoder for 10K

4.1.2.2 Comparison of Shallow and Deep Network

We observe in figure 4.3 that the initial results are worse when using a deep network rather than a shallow one. Normally, this would not be expected. The reason for this is that there is too little training data for the complexity of a deeper network, which has more parameters that need to learnt than a shallow 1-layer network. Indeed, we observe that by adding more training data (using 20,000 rows, then 30,000) the results are improving. However, we can see that we get better results by using more data with the shallow architecture than with the deeper one. It is expected that the deeper network should perform better, but this would probably happen for amounts of data of a magnitude higher, which cannot be tested given our limited computational resources.

Metric	Base model (10K)	Base model (20K)	Deep network (10 K)	Deep network (20K)	Deep network (30K)
BLEU-1	0.54732	0.5766	0.41228	0.53371	0.55356
BLEU-2	0.44148	0.46271	0.27795	0.4151	0.43459
BLEU-3	0.35848	0.40283	0.19157	0.34934	0.37877
BLEU-4	0.15688	0.2384	0.0629	0.18943	0.23749

Figure 4.3: Results of shallow vs deep network for different dataset sizes

4.1.2.3 Changing Number of Nodes

We performed some experiments regarding the changing the number of nodes within layer in order to see the effect on machine translation performance. We tested with 128, 256

and 512 units within a layer with 10k and 20k data. It can be seen the results of this experiment in 4.4.

	1 LSTM layer of encoder, 1 LSTM layer of decoder		
	10000 phrases; 9000 for training, 1000 for testing		
Metric	128 units	256 units	512 units
BLEU-1	0.369	0.451	0.563
BLEU-2	0.228	0.326	0.463
BLEU-3	0.145	0.265	0.394
BLEU-4	0.041	0.135	0.201
	1 LSTM layer of encoder, 1 LSTM layer of decoder		
	20000 phrases; 18000 for training, 2000 for testing		
Metric	128 units	256 units	512 units
BLEU-1	0.515	0.576	0.628
BLEU-2	0.389	0.462	0.526
BLEU-3	0.324	0.402	0.475
BLEU-4	0.16373	0.238	0.308

Figure 4.4: Results of 10k and 20k with 128, 256, 512 units

We observed that as we increased the number of units in a layer, it helped increase the Bleu score of each n-gram. This shows that complex networks are better at capturing the patterns for machine translation like we expected because machine translation is a complex task to perform. Also, the results with 20k for different nodes are better than the results with 10k.However, if we increase more than 512 nodes in a layer, it will cause over-fitting due to the high complexity of the architecture and not having sufficient data. We should have an architecture with 512 nodes within a layer and have more data in order have better accuracy.

4.1.3 Impact of the Word Embedding

The model was compared based on Word2Vec and pre-tarined Word2Vec approach with 20k. It can be seen the results of this experiment in 4.5.

	1 LSTM layer of encoder, 1 LSTM layer of decoder	
Metric	Word2Vec	pre-trained Word2Vec
BLEU-1	0.55	0.561
BLEU-2	0.428	0.438
BLEU-3	0.364	0.371
BLEU-4	0.205	0.216

Figure 4.5: Results of Word2Vec versus pre-trained Word2Vec with 20k

We observed that pre-trained Word2Vec gives slightly better results than Word2Vec because we used our data set in order to train our Word2Vec model. However, in the

pre-trained Word2Vec, we used a generalized data set that we found online independently from our data set. The reason of getting slightly better results is the data size of pre-trained Word2Vec is slightly bigger than Word2Vec therefore it shows us the importance of using generalized data set and the size of pre-trained model. In order to get better accuracy we should use a bigger data set to train the Word2Vec model.

4.1.4 Different language pairs: German - English

The model was evaluated on 10,000 rows of German-English sentence pairs in order to compare the results with the French-English sentence pairs. As we can see in the figure 4.6, the results are similar, slightly worse in the case of German to English. This corresponds to the usual case in research on the two main benchmarks of German to English and French to English. The reasons for this could be that French is a language that is more similar to English than German, which has more complex rules, different word order, etc.

Metric	Base model (10K) for FR-EN	Base model (10K) for DE-EN
BLEU-1	0.54732	0.53537
BLEU-2	0.44148	0.40294
BLEU-3	0.35848	0.31645
BLEU-4	0.15688	0.14515

Figure 4.6: Results of running the base model on 10K datasets from French to English and from German to English

4.2 Conclusions of the results

As a conclusion, it is crucial to have a good amount of high-quality training data. It can bring a high improvement in the total BLEU score, more than variations in the architecture of the model. However, different configurations of the architecture can bring significant improvements over a base model, as we have seen in the case of using a bidirectional layer or using a higher number of LSTM units. Using a deep architecture in this case drove down the BLEU score, most likely due to using too little data to train. Moreover, there are differences in the translation quality considering the language pairs used, depending on how similar the two language are. More similar languages facilitate the translation, whereas more different ones would require more training data and more complex models to give translations of the same quality.

There are some examples of our machine translation model below.

```
source = attache tes lacets, target = tie your shoes, predicted = tie your shoes
source = je paie mes propres dettes, target = i pay my own way, predicted = i pay my own
source = je suis tres serieuse, target = im quite serious, predicted = im very fat
source = les mecs sont stupides, target = guys are stupid, predicted = guys are stupid
source = il lui faut trouver du travail, target = he must find work, predicted = he must find work
```

Figure 4.7: Some examples of our machine translation model

Chapter 5

Conclusions and future work

In this project we implemented a base encoder-decoder machine translation system and we evaluated different strategies to improve its BLEU score accuracy. The report started by presenting an introduction to machine translation, mentioning evaluation methods, history of the topic, and detailing the main neural machine translation models that have revolutionised the way automatic translation is done. It described the data sets used for the experiments, represented by French to English parallel corpuses of 10,000 to 30,000 phrases and an additional German to English corpus of 10,000 phrases. The necessary pre-processing steps of the textual data were also described in detail: tokenisation, lowercase transformation, removing punctuation etc. As a base model we chose a shallow LSTM encoder-decoder machine translation system, which was first detailed theoretically and then implemented using the Keras library. Various experiments were run to check their impact on the translation quality: varying the amount of training data, changing the model architecture (considering bidirectional LSTM, more layers and more LSTM units) and considering different language pairs. It was shown that having a large amount of good quality data is essential for the performance of the model. Considering the computational resources limitation, a shallow model with bidirectional encoder and 512 LSTM units and also having a well chosen pre-trained Word2Vec would give the best accuracy. Changes could be observed in the choice of language pairs as well, considering the differences in syntax and semantics between languages. Thus, the translation from French to English had a slightly higher quality than the translation from German to English.

As future work, there are various strategies that can bring significant improvements. To begin with, deeper networks of 4 to 6 layers should give better results but they require much more data and more computational power to be trained. In order to have high quality results it would be advised to use millions of sentence pairs for training which would need to be run in clusters of computers. Moreover, the implementation of an attention mechanism would improve the translation of sentences of over 20 words. In our project, we tried to use an attention mechanism but we run into some technical issues, due to the library implementation. It required having the same length of the source and target sentences, and even then, the results would bring other issues linked to the maximum length of a sentence in the target language. Considering the time constraints, we decided to drop this from our current project. Another idea of improvement would be to have a pre-trained Word2Vec model that is trained on millions of phrases, not only on thousands can give better results. However, it is hard to find pre-trained model on French corpora therefore we have to train our Word2Vec model and save it for future use.

Bibliography

[Bahdanau 2014] Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014. (Cited on pages 7, 8 and 11.)

[BLE] BLEU. https://en.wikipedia.org/wiki/BLEU. (Accessed on 02/06/2020). (Cited on page 4.)

[Brown 1990] Peter F Brown, John Cocke, Stephen A Della Pietra, Vincent J Della Pietra, Frederik Jelinek, John D Lafferty, Robert L Mercer and Paul S Roossin. A statistical approach to machine translation. Computational linguistics, vol. 16, no. 2, pages 79–85, 1990. (Cited on page 5.)

[Carbonell 1978] Jaime G Carbonell, Richard E Cullinford and Anatole V Gershman. Knowledge-Based Machine Translation. Rapport technique, YALE UNIV NEW HAVEN CONN DEPT OF COMPUTER SCIENCE, 1978. (Cited on page 5.)

[Cho 2014] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014. (Cited on pages 5, 6, 7, 8 and 11.)

[Edunov 2018] Sergey Edunov, Myle Ott, Michael Auli and David Grangier. Understanding back-translation at scale. arXiv preprint arXiv:1808.09381, 2018. (Cited on page 11.)

[Garg 2018] Ankush Garg and Mayank Agarwal. Machine Translation: A Literature Review. arXiv preprint arXiv:1901.01122, 2018. (Cited on pages iii, 1, 4 and 6.)

[Gehring 2017] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats and Yann N Dauphin. Convolutional sequence to sequence learning. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 1243–1252. JMLR. org, 2017. (Cited on page 9.)

[Kong 2019] Xiang Kong, Qizhe Xie, Zihang Dai and Eduard Hovy. Fast and simple mixture of softmaxes with bpe and hybrid-lightrnn for language generation. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, pages 6626–6633, 2019. (Cited on page 11.)

[Li 2014] Haiying Li, Arthur C Graesser and Zhiqiang Cai. Comparison of Google translation with human translation. In The Twenty-Seventh International Flairs Conference, 2014. (Cited on page 3.)

[Nagao 1984] Makoto Nagao. A framework of a mechanical translation between Japanese and English by analogy principle. Artificial and human intelligence, pages 351–354, 1984. (Cited on page 5.)

[Radimrehurek] Radimrehurek. gensim: models.word2vec – Word2vec embeddings. https://radimrehurek.com/gensim/models/word2vec.html. (Accessed on 02/06/2020). (Cited on page 20.)

[Shaw 2018] Peter Shaw, Jakob Uszkoreit and Ashish Vaswani. Self-attention with relative position representations. arXiv preprint arXiv:1803.02155, 2018. (Cited on page 11.)

[So 2019] David R So, Chen Liang and Quoc V Le. The evolved transformer. arXiv preprint arXiv:1901.11117, 2019. (Cited on page 11.)

[Sutskever 2014] Ilya Sutskever, Oriol Vinyals and Quoc V Le. Sequence to sequence learning with neural networks. In Advances in neural information processing systems, pages 3104–3112, 2014. (Cited on pages 6, 7, 8 and 11.)

[Vaswani 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser and Illia Polosukhin. Attention is all you need. In Advances in neural information processing systems, pages 5998–6008, 2017. (Cited on pages iii, 9 and 10.)

[WMT2014] WMT2014. The Stanford Natural Language Processing Group. https://nlp.stanford.edu/projects/nmt/. (Accessed on 02/06/2020). (Cited on page 3.)

[Wu 2016] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey et al. Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144, 2016. (Cited on pages 8 and 11.)

Abstract: Machine translation is an active and challenging research area with important use in the industry. There are many machine translation models that have been proposed over the years, each with different architectural characteristics. In this project we develop a machine translation prototype system, with the main objective being to analyse which characteristics or strategies could bring the highest improvements on the quality of machine translation, considering the constraint of having limited computational resources. The project will use a corpus of parallel phrases from French to English, and as a base model we will consider a shallow LSTM encoder-decoder. The report will show that it is crucial to have a good amount of high-quality training data, which can improve the translations more than variations in the architecture of the model. Considering our context a shallow model with bidirectional encoder and 512 LSTM units and also having a well chosen pre-trained Word2Vec will give the best accuracy. Using a deep architecture in this case drove down the BLEU score, due to computational limitations that prevented us from using large amounts of training data. Changes will be observed in the choice of language pairs as well, considering the differences in syntax and semantics between languages. Thus, a translation from French to English will be shown to have a slightly higher quality than the translation from German to English.

Keywords: machine translation, neural networks, LSTM, encoder-decoder, BLEU score