Machine translation with Recurrent Neural Networks

Vera Neplenbroek veraneplenbroek@icloud.com

Kamiel Fokkink kamielfokkink@gmail.com

Baran İşcanlı barantevitol@gmail.com

Abstract

The goal of this project is to create a functioning English-Dutch translation model. The data set contains English and Dutch translations of the same movie subtitles, which will be connected in the translation model to build the translator. The model consists of two Recurrent Neural Networks, an Encoder and a Decoder, which will transform the English input sentence to a tensor, and then to a Dutch output sentence. It is based on a PyTorch implementation, but extended by reversing the input sentences, and using different data formats. Evaluation will be done using the BLEU measure, giving each translated Dutch sentence/text a score out of 1, based on how much it corresponds to the target sentence/text.

1 Introduction

Translating one word from one language to another is not very difficult. Each word can be mapped to one or various equivalent words in the other language. Indeed, traditional paper dictionaries can easily do the job. However, knowing the individual translation of each word is not enough to understand sentences or documents in the other language. The difficulty comes from language being compositional, thus the combining of words builds up meaning in the sentence. The final meaning of a sentence can not be reduced to individual meanings of the words, but also comes from their combinations. This is where traditional dictionaries fail, and machine translation comes in.

There exist several approaches to machine translation. The rule-based approach needs human made dictionaries and extensive grammar rules. The statistical approach is based on observations of large amounts of text, and generating patterns from it. And there are hybrid forms that combine elements from each of the two approaches. Within each paradigm, there exist numerous different techniques that can be applied to create a translation model. In recent years, the statistical approach has seen great improvements, thanks to developments in the field of machine learning. For our project, the particular technique that we decided to use was Sequence to Sequence translation. This approach was pioneered in 2013, by Kalchbrenner and Blunsom [1], and makes use of recurrent neural networks (RNN's). Our aim is to produce a machine translation model that is based on this architecture. It will be trained on English-Dutch sentence pairs. So the model will be able to take in a sentence in English, and output its Dutch translation.

The dataset that we used for the training of our model consists of pairs of subtitles from movies, in two different languages. The dataset was retrieved from Open Subtitles[2]. The data is structured in such a way that sentence pairs are coupled together, so for each line uttered in the movie there is the English and the Dutch subtitle for that line. There is an important advantage for using this kind of data for training. Namely, the text is well aligned, so for each sentence in English, there is a direct parallel equivalent for that sentence in Dutch. Had we used more freely translated datasets, such as books, articles or blogposts,

[1]N. Kalchbrenner, Phil Blunsom 2013. Recurrent Continuous Translation Models Association for Computational Linguistics

[2]Open Subtitles

the individual sentence pairs would fit less well to each other, making it much more difficult for the model to train on.

2 Preprocessing

The dataset we work with came in the TMX format. It was a dataset of 5.6 GB size. Thus, in order to work with it, we needed to divide the dataset into chunks or use samples we extract from it. Furthermore, we decided that while the TMX format might have its benefits, we did not need most of the perks it has. Therefore, we converted the dataset into a pickle, which enabled us to store it as python objects. Finally, we realized that our model required some normalization on each sentence via punctuation removal and lowercasing each word.

2.1 Format Handling and Dividing the Dataset

As mentioned above, our dataset was too large to read all of it at once. Therefore, after experimenting with a couple of methods, we decided to use basic file-streaming to read, process and save chunks of the dataset. After deciding on an approximate size for each chunk, which was roughly 500 MB's, we created a pipeline. Simply put, the pipeline took a chunk, cleared the irrelevant tags off of it, grabbed each English-Dutch sentence pair with a regex. The pipeline treated possibilities like half of a line being in another chunk or the translation of a line being in another chunk. Furthermore, it paired each sentence with its translation, put the pair in a list, and pickled the list in a file. This way, we acquired clean sentence pairs in different chunks, which we can read one by one from the pickle file. Finally, we created a function to read chunks from a pickle for easy usage.

2.2 Sampling

The developing process of models required small samples from the dataset to see the efficiency and quality of the model in a rough manner, before a proper evaluation process. However, just taking a chunk of the dataset would result with a high variation of low amounts of data. Therefore, we created a sampling function, which uses a preset of rules and extracts sentence pairs fitting that rule, then pickles them in another file for usage. This way, the accuracy of the model can be observed faster without an evaluation model on early stages. Also, we cleared the dataset off of some punctuation and transformed uppercase letters to lowercase. This was done to fit our dataset better for our model and remove irrelevant information.

3 The RNN Encoder-Decoder model

While regular neural networks are a powerful tool to train a model, they have one important limitation: they only take inputs of fixed length, and produce outputs of fixed lengths. For the translation of a sequence of words from English to Dutch this is not very useful, because both the length of the input and output sequence will vary between different sentences. We want to train a model that can take in any sentence of arbitrary length, and produce a corresponding translation. The solution to train this kind of model is provided by Recurrent Neural Networks[3] [4] [5]

3.1 Encoder-Decoder

An RNN can take in a sequence of any length, and produce any desired output. The key element to do this is the hidden state. To work with an input sequence $\mathbf{x} = (x_1, x_2, , x_t)$, the RNN looks at the elements of the input step by step. In our case, the input is a sentence, and the elements are words. Beginning at the first word, the RNN creates a hidden state, and an output vector y. For each next word, the RNN looks at that word and the previous hidden state, and updates to a new hidden state by

[3]D. Bahdanau, K. Cho, Y. Bengio. 2014. Neural machine translation by jointly learning to align and translate. Retrieved from the arXiv database.

[4]I. Sutskever, O. Vinyals, Q. Le. 2014. Sequence to sequence learning with neural networks. Retrieved from the arXiv database.

[5]K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. Retrieved from the arXiv database.

$$\mathbf{h}_{(t)} = f(\mathbf{h}_{(t-1)}, x_t),$$

where f can be any non-linear activation function. After the final word x_t has been evaluated, we have the final output vector y, also called context vector. This is the first half of the model, or the Encoder. In this step, the meaning of the English input sentence gets encoded into a vector, thus we have a numerical representation of the input.

In the second step, we decode the context vector into a sentence in Dutch, again by using an RNN. The approach is very similar, just the other way round. The initial hidden state of the decoder is the context vector, and its first input is the <SOS> (start of sentence) token. At this first step, the decoder also creates its own hidden state. From there, it generates each new word of the output sequence, by looking at its previous output, the context vector, and its own hidden state. Thus the activation function for the decoder looks a bit different:

$$\mathbf{h}_t = f(\mathbf{h}_{(t-1)}, y_{t-1}, \mathbf{c})$$

Finally, it will output a generated sentence in Dutch.

3.2 Training

During the training of the model, both these RNN's are jointly trained via gradient-descent. The goal of the training is to maximize the probability that our model assigns to each Dutch sentence, given each English sentence.

$$max \tfrac{1}{N} \textstyle\sum_{n=1}^{N} log p_\theta(y_n|x_n)$$

The sentences y_n and x_n are those that we have from the data. The probability depends on the parameters theta of our model, and is logarithmic for easier summation and to stay within computational bounds.

We used an implementation from a PyTorch tutorial [6] as the basis for our model. First we spent time understanding the code to get a good general idea of what is going on in each part. Then we looked into adapting the code to our needs, and have it fit to our dataset. We were able to interact with the code in various places as well, to implement some tweaks to the model, as discussed in the next section.

[6]PyTorch, 2017

3.3 Changes to the model

When implementing the changes to the model we used a small sample from the dataset with sentences that are similar in grammar. We looked at the BLEU score computed after evaluating the trained model, which was compared to the one for the base model: 0.01850. We implemented and evaluated the following changes/tweaks to the PyTorch implementation:

Attention

Score: 0.02501
When using a regular decoder, the output vector from the encoder is all the decoder has to learn from and to base its translation on. This means the full input sentence and its meaning is now represented by one vector only.

When using a decoder with attention mechanism, however, for each word that the decoder outputs, it is provided with a set of attention weights. These weights tell the decoder which parts of the input vector are important to translate the word, for example a noun will be important information for translating the article that comes before it. This set of attention weights will be multiplied with the output vector from the encoder, to create a vector of weighted inputs for the decoder. The maximum length of the input sentences, output vectors for the encoder, needs to be specified before training the model, to make sure the set of weights has the right length for the longest sentence or sentences in the data. For shorter sentences not all weights will be used [7].

Teacher Forcing

Score: 0.04614
Teacher forcing is used in training the model, where instead of the last output from the decoder, on which the decoder would usually base its next prediction, the target tensor is fed to the decoder. Teacher forcing helps the model to learn coherent grammar, but if over-exploited, the finalized model will have difficulty creating a grammatical structure by itself. Since the model is used to getting the

[7]Bahdanau, 2014

first few words provided by the teacher, when it does not get these in the evaluation phase it does not know how to starting creating the output sentence[8] [9]. We chose to implement teacher forcing with a ratio of 0.5, so it was only applied on half of the sentences. This way, we could combine the best of both worlds.

Reversing the input

Score: 0.02149
Reversing the input (source) sentences when feeding them to the model to train has proven[10] to decrease the distance between words in the source and in the target language. In other words, long-term dependencies are being replaced by short-term dependencies. This results in a higher probability of two words, each from a different language, which context is depended on each other to be close in distance.

Activation functions

Score: 0.0
At first we tried to implement the log sigmoid and leaky relu activation functions, but in the original log softmax activation function a dimension was specified, which could not be specified for the log sigmoid and leaky relu activation functions. This had as a result that the output sentences did not show any signs of the model having trained, but instead only contained sequences of the word 'ik' and 'ik ben', apparently the only words the model had learned.

Since this did not work out we tried using activation functions for which the dimension could be specified, namely the softmax and softmin activation functions. For these functions, however, the output sentences for each input consisted of similar words, and all output sentences were of the maximum length specified for the input and output sentences. Even training with more iterations did not change this. We think that the fact that no other activation function worked has to do with the way the model from the PyTorch implementation was set up, and that trying different activation functions is justified from a theoretical viewpoint [11].

Different NN units

We considered changing the linear units that make up the RNN in the Pytorch implementation into RNN or LSTM units, but since Pytorch created a custom RNN from the linear unit specific for this implementation, it did not work and did not make sense to change the linear unit.

We have decided to implement all the changes that improved the evaluation score, which were the attention mechanism, teacher forcing, and reversing the input sentences.

4 Evaluation

To evaluate the performances of our model we chose to write our own implementation of the widely used BLEU score. The BLEU score takes all the n-grams up until the specified n and calculates the amount of similarities between two sentences, where for each n-gram each word can only be used once to count towards a match with the other sentence. To calculate the BLEU score over a set of sentences, the geometric mean of the scores for each individual sentence is used [12].

5 Discussion

One of the limitations of our model are imperfections in our dataset. The sentence pairs we used to train the model on were less well aligned than we thought. While each pair in the data both expressed the same line in the movie, there are various cases where they were still written in a different way. For example, the English sentence would say 'we didn't expect to see you here' and the Dutch one would say 'we expected to find John here'. While this essentially expresses the same idea, it is not a one-to-one translation. This could mess up the

[8]PyTorch, 2017
[9]H. Jaeger. 2002. A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach
[10]Sutskever, 2014
[11]PyTorch, 2017
[12]K. Papineni, S. Roukos, T. Ward, W. Zhu. 2002 BLEU: a Method for Automatic Evaluation of Machine Translation Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)

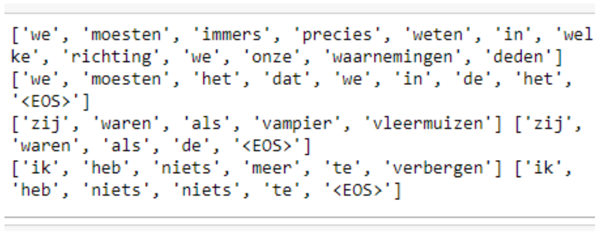
training, and eventually lower its performance. Out of 50 training pairs, 9 were counted to be less well-aligned in this manner. It is difficult to estimate a ratio for the whole data, but at least a significant portion of the pairs are suboptimal. Filtering these pairs out would require human intervention, checking each pair individually if it is a good translation or not. Doing this manually for all our datapairs is beyond the scope of this project. Hence, we just trained on the data that we had, accepting that this fact lowers the performance of the model.

There were a few changes that we tried to implement into our model, to try and improve results, but that didn't work out. One of them was to try and write the attention mechanism ourselves. In the attention decoder, attention weights get inserted into the context vector. While during this course we have focused on the probabilistic approach towards NLP, there is also the symbolic approach, by writing top-down rules. We thought that in the attention step there could be an opportunity to try and experiment with the symbolic approach. Rules could indicate which parts of the context vector to focus on, such as that a determinant will probably be followed by a noun. However, there are a few obstacles to this approach. Firstly, we would need to pos-tag the sentence to implement these rules, which would significantly increase running time. Secondly, the question of finding appropriate weights is a difficult one. It could be solved empirically, by training with several different weightings and compare the results. But this would require us to train multiple models to be able to evaluate the effects of different weightings, which is too costly in terms of computing power and time. Hence, we decided to not implement it, and just rely on training the attention decoder RNN.

6 Conclusion

In the end, because of scale issues, we did not train over the whole dataset. Instead, we filtered sentence pairs on a maximum length of 15 words per sentence, starting with [I, you, we, he, she, they, it], and followed by [is, are, am, have, has, were, had]. This way we limited the size of our training data, while keeping a similar grammatical structure over our sentence pairs, to focus the training. We split this filtered dataset into training and test set in a 80/20 ratio. The total training time was 3 days. Despite this training, our final model still had not learned a large vocabulary. It knows all the pronouns, determinants, and some basic verbs, but can not translate nouns or sentence structures. However, learning this is just a matter of time and computing power. The printout in Figure 1 shows three examples of the target sentence, and the translation that our model came up with. It is clear that the frequent, easy words get translated correctly, but the vocabulary is still small.

When evaluating the translations that our model gave for the test set, we got the following scores between 0 and 1 for the various n-grams. 4-grams: 0.03, 3-grams: 0.09, 2-grams: 0.17, 1-grams: 0.29. So a considerable portion of the words gets correctly translated. We can conclude that the basis of our model works, albeit with a limited vocabulary, and is able to produce a somewhat meaningful translated sentence.



```
['we', 'moesten', 'immers', 'precies', 'weten', 'in', 'wel
ke', 'richting', 'we', 'onze', 'waarnemingen', 'deden']
['we', 'moesten', 'het', 'dat', 'we', 'in', 'de', 'het',
'<EOS>']
['zij', 'waren', 'als', 'vampier', 'vleermuizen'] ['zij',
'waren', 'als', 'de', '<EOS>']
['ik', 'heb', 'niets', 'meer', 'te', 'verbergen'] ['ik',
'heb', 'niets', 'niets', 'te', '<EOS>']
```

References

[Bahdanau et al.2014] D. Bahdanau, K. Cho, Y. Bengio. 2014. *Neural machine translation by jointly learning to align and translate.* Retrieved from the arXiv database.

[Cho et al.2014] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio. 2014. *Learning phrase representations using RNN encoder-decoder for statistical machine translation.* Retrieved from the arXiv database.

[Jaeger2002] H. Jaeger. 2002. *A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach*

[Kalchbrenner et al.2013] N. Kalchbrenner, Phil Blunsom 2013. *Recurrent Continuous Translation Models* Association for Computational Linguistics

[Papineni et al.2002] K. Papineni, S. Roukos, T. Ward, W. Zhu. 2002 *BLEU: a Method for Automatic Evaluation of Machine Translation* Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)

[PyTorch2017] *NLP From Scratch: Translation with a Sequence to Sequence Network and Attention*

[Sutskever et al.2014] I. Sutskever, O. Vinyals, Q. Le. 2014. *Sequence to sequence learning with neural networks.* Retrieved from the arXiv database.