Neural Approaches to Text Normalization

Pau Batlle, Miguel Cidrás, Esteve Tarragó

Deep Learning for Speech and Language Course, January 2018

Abstract

Spoken text normalization, this is, transforming text into a spoken form, is both necessary for most text to speech systems and a very interesting problem itself, with many different solutions proposed over time. In our project we glance at the different ways deep learning can approach this problem, concluding that methods that perform better in traditional neural machine translation tasks are capable of transferring this advantage into text normalization, when seen as a pure translation problem.

1. Introduction to the Text Normalization problem

In the most general meaning, text normalization refers to transforming text into a specified canonical form. However, in this text we will focus in a particular case, inspired by Kaggle's Text Normalization Challenge[1], which in a sense is motivated by applications in other deep learning areas.

Indeed, the problem is to transform given text into the canonical *spoken* form, this is, return how would a human read it, which can be particularly tricky in case of numbers, dates, acronyms, adresses... This is obviously needed for generating speech from text as one of the very early stages of the pipeline which converts raw text to an audio file. In other words, it is a needed preprocess of the input text for training text to speech algorithms.

General text normalization and also the specific problem of spoken normalization can and have been addressed in multiple ways, both neural and non-

[1]This former Kaggle competition (https://www.kaggle.com/c/text-normalization-challenge-english-language) has been both the initial motivation of our project and a source of ideas for different approaches

neural[2]. Our intent with this work is not to find the best way to solve the problem nor to explore all of possible solutions, but to use this problem to learn about different aspects of deep learning applied to language such as neural machine translation, word embeddings or sequence2sequence approaches. As we will discuss, this problem can be interpreted in multiple ways, which is something that was really appealing to us and the main motivation that made us choose this topic from the wide list of interesting projects that were proposed during the course.

2. The dataset

For this project we used the exact same datasets that participants of the kaggle competition were provided. The train dataset consisted of around 10M words separated in around 750k sentences. For each word, sentence id, tokenid, class (word type), before (raw word) and after (normalized word) are given. On the other hand, the test connsisted of around 1M words separated in around 70k sentences. The word type and the normalized word are not given in the test dataset. An example of the dataset is shown in Figure 1.

	sentence_id	token_id	class	before	after
0	0	0	PLAIN	Brillantaisia	Brillantaisia
1	0	1	PLAIN	is	is
2	0	2	PLAIN	a	a
3	0	3	PLAIN	genus	genus
4	0	4	PLAIN	of	of

Figure 1: First sentence of the dataser

3. Different approaches to the problem

In this section we explore the problem in different ways. The most common way to address this problem in a neural way is viewing it as a translation problem, and so this has also been our main focus during the project.

[2]In fact, some top seeded solutions of the Kaggle Competitions did not make use of deep learning at all

3.1. Neural Machine Translation

The idea behind this approach is as simple as seeing the canonical form (expected output of the network) as a new language and the job to normalize text as a translation job. One can therefore apply to this problem all of the algorithms and architectures that are used in Neural Machine Translation. Additionally, the fact that the both "languages" are almost identical most of the time could be used in our favor to improve the results, as it is done in the literature. However, due to hard time constraints, in this case we will use the methods "blindly", in the exact same way we would use them to translate between two actual languages. Our goal is to see whether an increase of performance in methods that translate between two actual languages is corresponded to an increase of performance in text normalization task. To do so, we will apply each one of them to the task and later we will compare the results to the neural machine translation results found in the literature when translating english to german. We will take one RNN architecture, one CNN architecture and one attention model for the comparison.

3.1.1. Keras sequence2sequence scheme

Our first experiment was implementing a sequence2sequence encoder decoder scheme in *Keras*, trying to translate each word separately, independently from the context of the sentence it appears on. This architecture aims to take profit from the fact that most of words are normalized the same way independently from their context and that characters are strongly related in their normalization translation. Additionally, self category characters tend to have the same rules in normalization, and thus theoretically it would be possible to train different networks for every category. The input and the output are represented in one hot-encoding sequence. The decoder is fed with its own output until end of sequence is met with a greedy strategy.

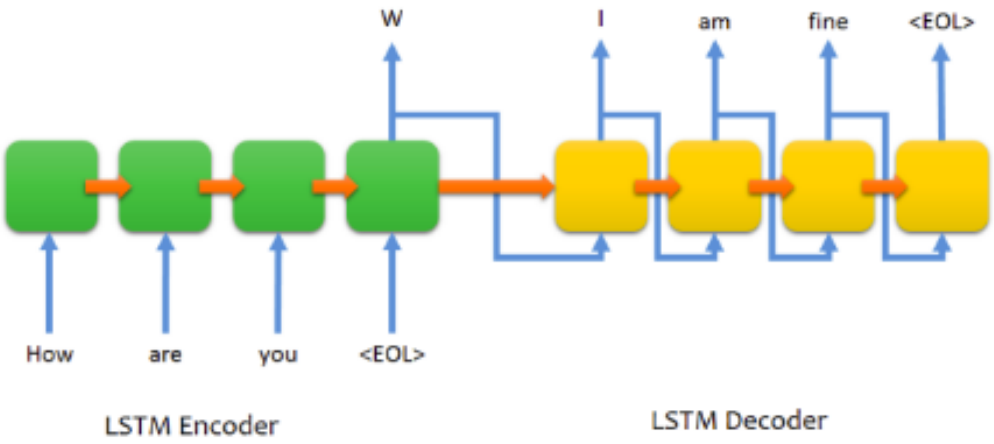

Figure 2: LSTM encoder-decoder we implemented in *Keras*

Given the simplicity of the model, the assumption that context is not an important factor and the low training time at our disposal, the results impressed us. Although validation accuracy was around 90% in all the training sessions we performed (which basically means that is changing unnecessary words since only 7% of words have to be changed), it translated well some non-trivial examples such as the following

- April 10, 2013 ⇒ april tenth twenty thirteen
- 91 ⇒ ninenty one
- ALCS ⇒ a l c s

However, when undertrained the system encountered some classic RNN errors such as repeating words or "inventing" them.

- 2015 ⇒ twenty finty twenty finty twenty finty twenty finty twenty finty twenty finty twenty finty twenty finty twenty finty twenty finty twenty finty twenty finty twenty finty twenty finty twenty finty twenty

It was not easy to select when to stop the training but we finally managed to do so plotting the loss function and accuracy during the actual training.

3.1.2. Using Pytorch and OpenNMT

The next objective of the project was to experiment with other types of Neural Machine, mainly ones involving CNNs. To do so the *Pytorch* implementation of OpenNMT, an open neural machine translation library, was used. Different models were tried to experiment and learn about them, and the final learning was performed using the exact same architecture that was used in the WMT'16 Multimodal Translation task (english to german translation), based mainly in CNNs but with several improvements. A summary of the obtained results can be seen in Figure 3

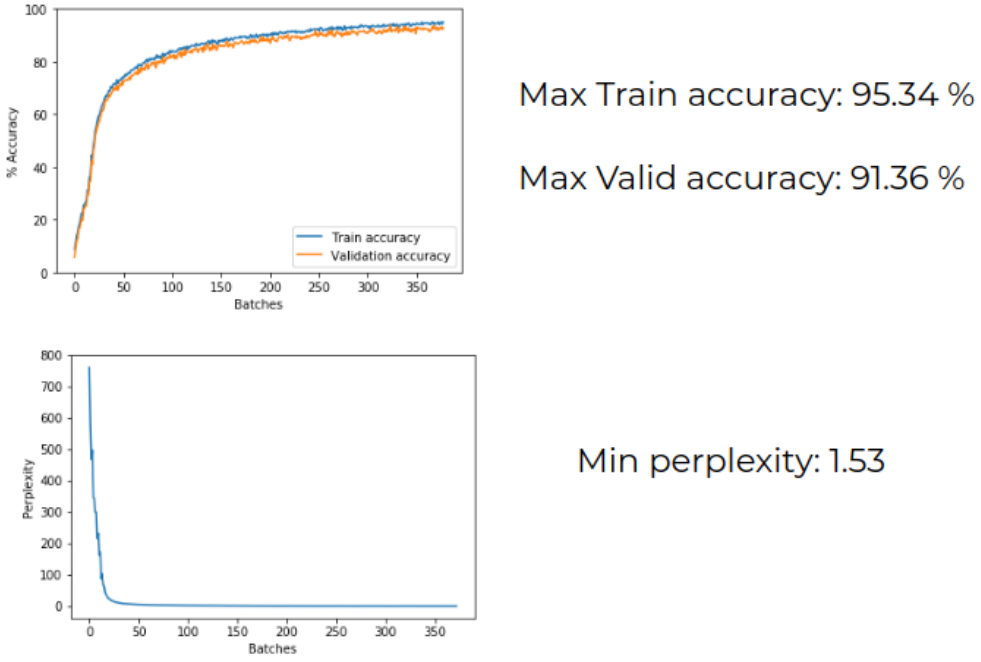

Figure 3: Training results of the model trained with OpenNMT

3.2. Attention-based models: The transformer

The appearance of attention based models had a big impact in the world of NMT, and we wanted to try how well does such model work for normalization. An overview of the transformer can be seen in Figure 4

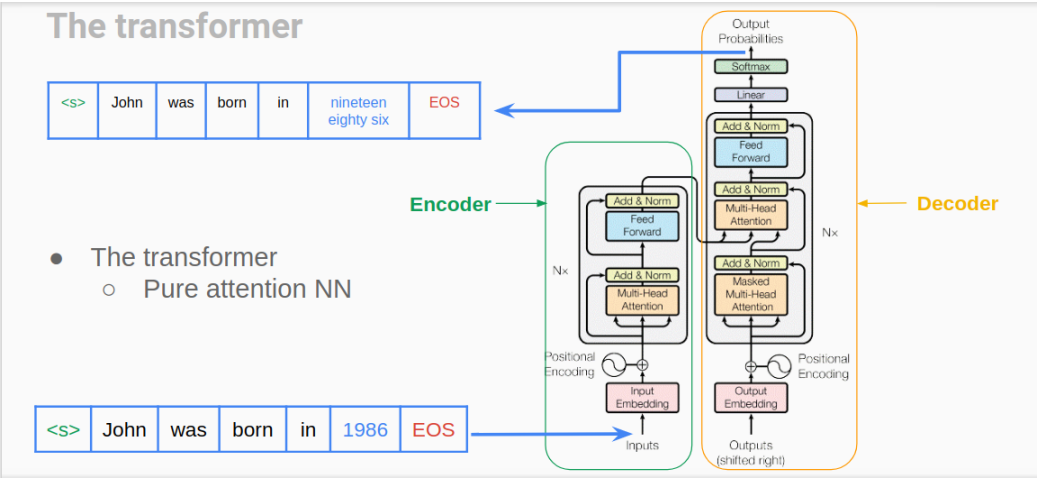

Figure 4: Overview of the transformer

Thanks to the work performed by former students of the UPC course "Deep Learning for Artificial Intelligence", who ported the code of the article "Attention is all you need" to Pytorch, attention-based models could also be compared for this particular task. Adapting this code, we trained such network with our dataset. The results obtained are shown in Figure 5

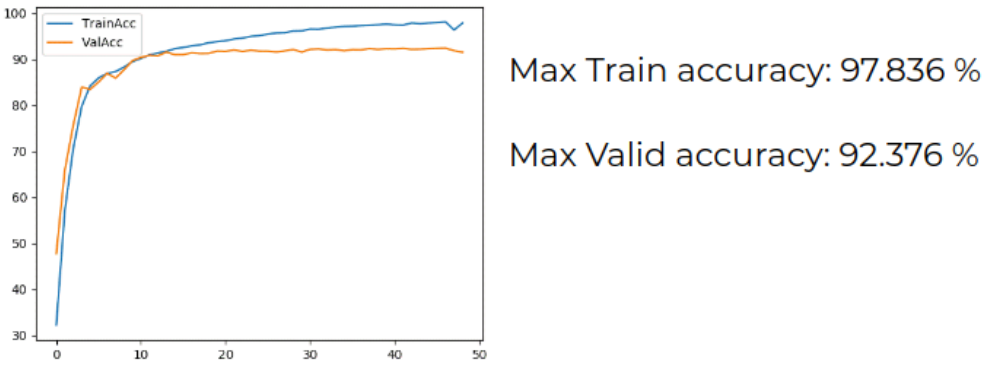

Figure 5: Training results of the Transformer model

3.3. Classification using Word Embeddings for later use in mixed approaches

We wanted to try to work with word embeddings because they are a core topic in almost all language-related deep learning. However, it is not easy

to figure out how to use them for this problem. The lack of generalization to new, non-seen examples of Word2Vec would imply also a lack of generalization in general in case we implemented a model based *entirely* on these embeddings. Therefore, they have to be part of a mixed model: a model where word embeddings can give us useful information for words already seen, which could then be incorporated to other end2end approaches like the ones discussed before. We leave here some of the ideas we had regarding the use in word embeddings, which responded more to educational purposes than actual performance improving puproses.

To normalize a word, we follow the following steps:

1. Check if the word is in the vocabulary of the first embedding.
2. If it is included, we find the embedding and the word in the second embedding that have the most similar embedding with our word. This word will be our normalization.
3. It it is not included, we label as "UNKNOWN".

We tried this method with the first 10000 words of the test data set, and we obtain 93.49 % of accuracy.

Another approach is using the word embedding for classifying in one of the 16 classes. So, if we want to classify a word, we follow the following steps:

1. Check if the word is in the vocabulary of the embedding.
2. If it is included, we find the embedding and we use a trained neural network to classify it.
3. If it is not included, we label as "UNKNOWN".

We obtain a multiclass classification model (using logistic regression) with 96.87 % of accuracy. As each class follows a few fixed rules for its normalization, it could be a good method to solve this problem.

We could also use a similar neuronal network for classifying in two classes, depending if the word needs normalization. Similarly, we follow the following steps:

1. Check if the word is in the vocabulary of the embedding.
2. If it is included, we find the embedding and we use a trained neural network to classify it.
3. It it is not included, we label as "UNKNOWN"

We obtain a binary classificator with 97.07% of accurancy. As each class follows a few fixed rules, they can be normalized individually with no machine learning algorithms. This could be another method to solve this problem .

We have to take into account, however, that we still would have to process the "UNKNOWNS" in some other way. As we've mentioned, the system by itself can not generalize, and so, it is not more intelligent that just looking at the dataset once we find a known example. However, the fact that the classification accuracies are so high just using the vectors made us realize how embeddings really can capture semantic meaning of words, which was a very positive learning experience overall.

4. Furthering our work

In this section we discuss experiments we thought relevant in the context in this project that we could not perform basically because of time constraints. As mentioned earlier, LSTM approaches could be done in every single category. More parameters could also have been taken into account when training the models with OpenNMT, and we would also have liked to do a bit of research of attention models to further increase the performance in the given task. Finally, since the dataset is not balanced, a precision recall analysis of each method would have given us the chance to compare in a more fair manner the different methods.

5. Results and Conclusions

The results we obtained can be summarized in the following table:

Method	**Validation accuracy (%)**
RNN	$\sim$90
CNN	91.36
Attention	92.376

When comparing these results to the BLEU performance indicator for english french translation, which can be seen in Figure 6, we see that the

relative performance order of the three approaches is conserved, at least as far as validation accuracy is concerned.

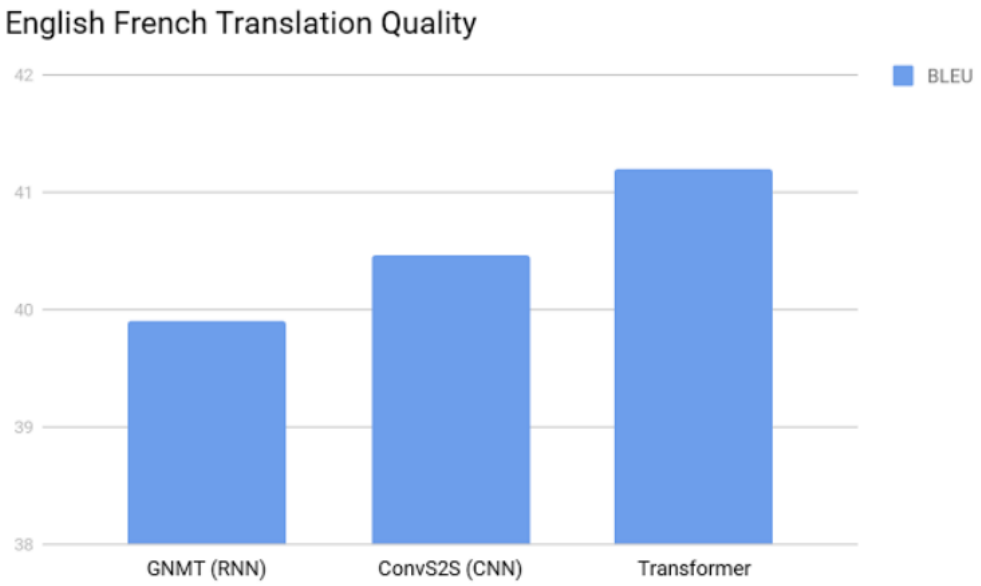

Figure 6: BLEU Performance index

Finally, our conclusion is that text normalization behaves similarly to other translation tasks when applied the different methods we have discussed in this project. Although we have not discussed it here, it is fair to notice that overall, feeding the data to a NMT system without making any changes to it does not produce state-of-the-art results, but still high quality results for a task which is far from trivial.

6. Bibliography

- Natural Language Understanding with Distributed Representation, Kyunghyun Cho, Chapter 6, 2015
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention is all you need"
- Alex Kendall, Yaring Gal, Roberto Cipolla. "Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics"
- https://www.kaggle.com/c/text-normalization-challenge-english-language
- Sequence to Sequence Learning with Neural Networks https://arxiv.org/abs/1409.321
- Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation https://arxiv.org/abs/1406.1078
- Neural Machine Translation (seq2seq) Tutorial https://github.com/tensorflow/nm
- OpenNMT http://opennmt.net/