Automatic Text Punctuation with Sequence To Sequence LSTM for Spanish

Oscar Fabián Ñáñez Núñez, Juan Camilo Calero Espinosa
Universidad Nacional de Colombia
Machine Learning

***Abstract*— One of the biggest problem in Automatic Speech Recognition (ASR) recognition is the utility of the generated texts, and that is because all we got from the majority of ASR's is a plain text without punctuations, which is hard to analyze for humans and machines for tasks as Sentiment Analysis. There has been different methods to tackle this problem such as tree-based methods, statistical methods, sequential models, among others. In this paper, we propose a Sequence-To-Sequence (Seq2Seq) method, in which we treat this problem as a machine translation problem, assuming that the source language is the original text (no punctuation) and the goal language is the punctuated text. We apply this method to a spanish dataset in order to apply it in future works to spanish ASR applications.**

I. INTRODUCTION

The automatic punctuation has been an important research work in ASR field, since the generated text is in almost all cases without punctuations making it difficult to perform NLP applications.

There has been lot of research works around this problem, some of them are just applied in the ASR field because they use voice features such as pitch, beat detection and other features related to the time or frequency domain of the audio which then feed some learning model (trees, Neural Networks, statistical, etc) as we can see in [1], however, they can't be used for only text application.

On the other hand, a method that has more extensive application field is the automatic punctuation for just text, which can be more difficult because in this case, we don't have clues of punctuation such as breaks, changes of pitch, among others. In this case, the features are learned directly from the text, and even though these features are not obvious, Deep Learning has shown good results.

Furthermore, there is also some works that propose a hybrid between acoustic and textual features. For example, in [2] we can see a model that uses textual information and acoustic (prosodic) information and based on adaptive boosting.

A novel method that implements sequential models, which is also a hybrid solution is presented in [3], where a two stage LSTM learn textual features in the first stage, and then it combines these textual features with pause durations to adapt these features to the speech domain.

Recently, a method that uses attentional mechanisms has been proposed in [4]. In this work, the text feeds the model, which can utilize long contexts in both directions and direct attention where necessary, thanks to the attentional benefits, to finally complete the unpunctuated segments of text.

In this work, we propose a simpler method than [5], but we also emphasize just in the textual domain. To tackle this problem we treated the punctuation problem has a Machine Translation problem, in which the source language are spanish sentences without punctuation and the target language are the same spanish words with the correspondent punctuation. An LSTM with an architecture Seq2Seq was used to train the dataset, which corresponds to The Wikipedia corpus, because we consider it a big enough corpus and it also bring other benefits which we will mention later.

The next section describes the dataset and our approach in detail. Section 3 describes training strategies, metrics and results. Finally, Section 4 concludes the paper.

II. METHOD

II-A. Dataset

The dataset chosen was The Wikipedia because it has huge amount of information and variety of topics, besides it was a lot of noise and punctuations errors due to Wikis do not have good corrections, allowing the network to deal with theses noises during training time. The download was made thorough Wikimedia dump service [5], and the Spanish archive was selected. But this file is in format xml, not good for our training model, so an extractor had to be used in order to get a single txt file containing all the text of the Wikipedia. The extractor used was WikiExtractor [6], with license GPL-3-0. The extraction of the text, using 16 cores, took about 30 minutes.

Then, some filters had to be used to fit our dataset to our model, all of those filters were implemented using regular expressions with the Linux tool sed [7]. Those filters are:

- **Remove all non printable characters**
- **Remove Wikipedia's Tags and IDs of documents**
- **Convert all Real or Integer numbers into a single <num>symbol, so commas, points and percentages on those numbers are no taken into account as punctuations symbols**
- **Separate all special symbols with spaces, so each one of them can be seen as a separate character and each**

word is not modified by them, for example ¡ahora! would be seen as a different word as ahora, but ¡ ahora ! can be seen as three different words. Those symbols are & % $ # _ { } [ ] - / . « » ¨ ’ “ ” ? ¿ ! ¡ , : = | ` ; ( ) + * ˜ < > − \ ‘

- **Separate all line breaks with a space and convert them into a <eos>(end of string) symbol**
- **Copy the dataset, but removing some punctuation symbols, those symbols are: . ? ¿ ! ¡ , : ;**

II-B. IMPLEMENTATION

The model consists of a Recurrent Neural Network (RNN) with a Seq2Seq Architecture and LSTM units, which is a typical model used for Machine Translation. In this case, we chose LSTM units because they give us a long dependency thanks to the memory cell implemented in these units, as we can see in figure 1. Due to this memory cells, the model is capable to keep in memory some words that can be important to punctuate some segment. To simplify the model, and because we consider that future words are not such important to choose a punctuation as previous words, we implemented a Unidirectional Seq2Seq LSTM.

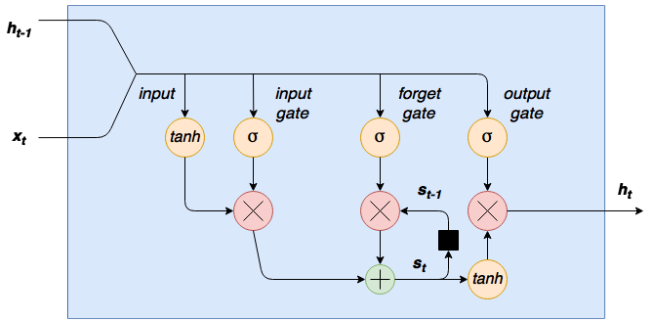

Fig. 1. Intern structure of a LSTM

To represent the words and the punctuation signs, we use word embeddings, particularly, Seq2Seq Word embeddings, which consists in taking a word and finding a vector representation of that word which captures some meaning of the word.

The word vectors are learned separately, then we apply the translation model. We specified the length of the embedding with a value of 500, which correspond to a vector $x = [x_1, x_2, x_3, \ldots, x_{500}]$. In other hand, we considered to take a size of 30 time steps, which means to take 30 words from the text to input the Neural Network, however, the slice between phrases is not also 30, but a $skip\_steps$ as we call it, was implemented, but as punctuated text has different length than the input text, a synchronization had to be done at each slice of text, e.g. if 5 words of the 30 words of a slice of the punctuated text were punctuation symbols, then $skip\_steps$ has to be $skip\_steps$ + 5 for the punctuated text, but it will always be $skip\_steps$ for the non punctuated text. A summary of the implemented model is shown in figure 2.

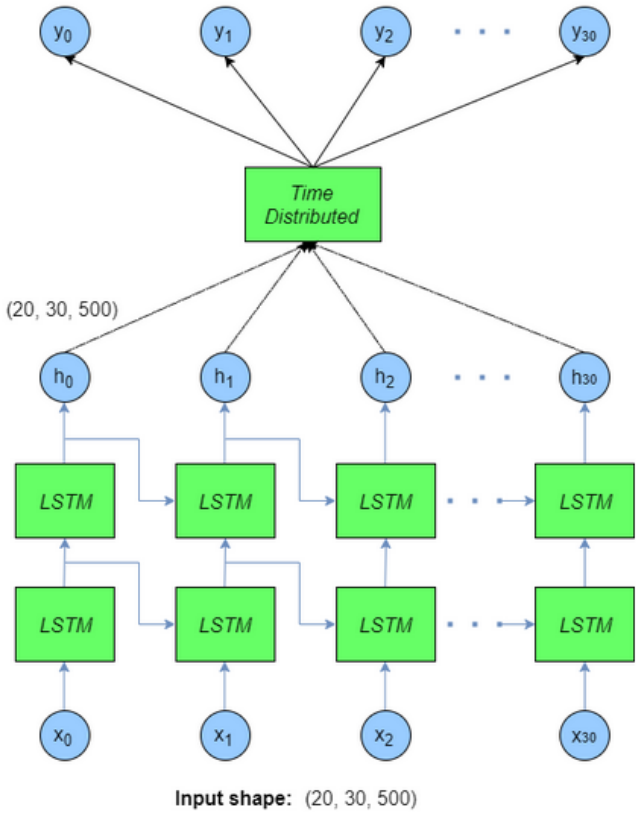

Fig. 2. Model implemented

As we can see in the above picture, what the Seq2Seq embedding model does, is to take all the inputs to compute the current state, and then it computes the loss corresponding to the output, which is the same input but with the punctuation signs. Furthermore, the size of the input, which is the tuple shown in the image corresponds to the following values: (batch size, number of time steps, hidden size), which means that there is a embedding vector correspondent to every input, moreover, the size of the output is similar except for the third term (batch size, number of time steps, alphabet size), this is due to the activation function which is a softmax function with each position corresponding to a word in a one hot representation.

Every input x_i corresponds to a unique integer identifier for that word, and the output vocabulary corresponds to the size of the vector indexing every word. In this case, we don't limit the vocabulary size, since we need every word when we need to replace for the original text, however to decrease the total size, we converted all numbers to a token identifier $< num >$.

The architecture of the network consists of a first layer, which is the embedding layer, this will convert the words into embedding vectors. Then we added two LSTM layers, and after that a dropout layer to avoid overfitting. Finally, we added a time distributed layer to return a sequence at the output layer corresponding to the time steps.

III. EXPERIMENTS

To train this model we used the Keras Framework in a NVIDIA GeForce GTX 1080 Ti. Furthermore, we used Mini-batch Gradient Descent with a mini-batch size of 20.

As optimizer we chose the *Adam optimizer*, which nowadays is commonly used because has shown good results. A cross_entropy loss function was used because it is commonly used with softmax activation functions in the output layer.

Just a small part of the Wikipedia has been used, around 1 % of the total dataset was used for training, validation and tests. Which correspond about 30MB of text, to make an idea, the full bible has a size of around of 4MB, so the network had to learn the punctuation of the equivalent of 8 Bibles 50 times. 50 is the number of epochs chose for training. But training is quite slow, 3 days were required to do this. So a 10 times smaller dataset was trained, but using 100 epochs, 90 times faster and with better results. Figure 3 shows the behavior during training:

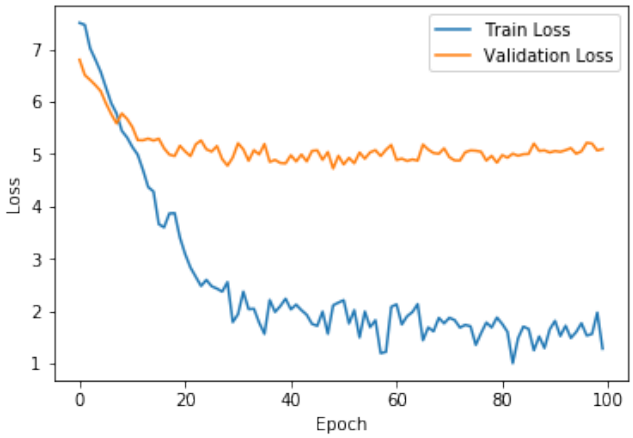

Fig. 3. Loss for training and validation set

And in the below figure we can observe the behavior of the accuracy during training:

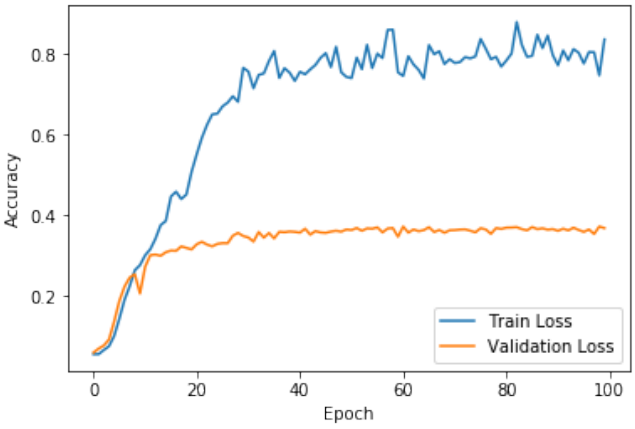

Fig. 4. Accuracy for training and validation set

As we can observe in both pictures, the training was significant until approximately epoch 35. After that, the learning process was stuck in a validation accuracy of about 0.35, which is too low.

Maybe, one of the biggest problem, was the amount of possibilities to consider when the Neural Network tries to predict the next, which could be reduced to the true word and the punctuation signs.

In the below table we find a summary of the accuracy during distinct epochs of training:

Epoch	Training accuracy	Validation Accuracy
20	0.5085	0.3156
40	0.7340	0.3592
60	0.7551	0.3471
80	0.7694	0.3665
100	0.8371	0.3685

Some samples of the generated text can be found in the below pictures, being taken from the validation set:

```
Input text:
petición expresa de los andorranos <eos> Durante el periodo del Mesolítico pequeños grupos de humanos se asentaron en gr
utas próximas al Gran Valira como en la Balma de la Margineda
Expected text:
petición expresa de los andorranos . <eos> Durante el periodo del Mesolítico , pequeños grupos de humanos se asentaron e
n grutas próximas al Gran Valira , como en la Balma
Predicted text:
petición expresa de los andorranos . <eos> Durante el periodo del Mesolítico , pequeños grupos de humanos se asentaron e
n grutas próximas al Gran Valira , como en la Balma
```

Fig. 5. Perfect prediction on training sample

```
Input text:
en movimiento relativo con respecto al aire que la rodea Dos ejemplos de superficies de sustentación son las alas de los
aviones o la hoja del de una hélice <eos>
Expected text:
en movimiento relativo con respecto al aire que la rodea . Dos ejemplos de superficies de sustentación son las alas de l
os aviones o la hoja del de una hélice
Predicted text:
en movimiento relativo con respecto al aire que la rodea , , Dos ejemplos de superficies de sustentación son , las alas
de , los aviones o la hoja del
```

Fig. 6. Some errors in prediction on validation sample

```
Input text:
el artículo segundo de la Constitución Española de <num> que reconoce y garantiza el derecho a la autonomía de las nacio
nalidades y regiones españolas El proceso de autonomía política se
Expected text:
el artículo segundo de la Constitución Española de <num> que reconoce y garantiza el derecho a la autonomía de las nacio
nalidades y regiones españolas . El proceso de autonomía política
Predicted text:
el artículo , segundo de la Constitución Española de <num> que , reconoce y garantiza el derecho a la autonomía de las ,
nacionalidades y . . regiones españolas El
```

Fig. 7. Some errors in prediction on test sample

As we can see, there are some serious mistakes such as putting two punctuation signs followed one by the other, which is hard to explain, because that was not shown in the training set.

IV. CONCLUSIONS

The model performs really well on the training data, but its accuracy is not so high when a text with words that the network has never seen is the input. And with the Wikipedia this probability is high due to the the huge amount of topics and new words that are in a single sample of the dataset, so a lot of effort ant training time is required to have a large vocabulary and a more robust Neural Network.

Furthermore, we consider that a more robust model (such as attention models) and more data, could perform better results, however, it needs more time effort to train a reasonable network which can get better results. On the other hand, some restrictions can be added during training, such as to limit to the next true word and the punctuation signs.

REFERENCES

[1] J. Kolar, J. Svec, and J. Psutka, "Automatic punctuation annotation in czech broadcast news speech." http://www.kky.zcu.cz/en/publications/1/KolarJ_2004_Automaticpunctuation.pdf, 2004.

[2] J. Kolar and L. Lamer, "Development and evaluation of automatic punctuation for french and english speech-to-text." https://www.mde.zcu.cz/data/Kolar12.pdf, 2012.

[3] O. Tilk and T. Alumae, "LSTM for punctuation restoration in speech transcripts." https://phon.ioc.ee/dokuwiki/lib/exe/fetch.php?media=people:tanel:interspeech2015-paper-punct.pdf, 2015.

[4] O. Tilk and T. Alumae, "Bidirectional recurrent neural network with attention mechanism for punctuation restoration." Conference: INTERSPEECH 2016, 2016.

[5] "eswiki dump progress on 20180520." https://dumps.wikimedia.org/eswiki/20180520/. Accessed: 2018-05-28.

[6] attardi, "wikiextractor." https://github.com/attardi/wikiextractor. Accessed: 2018-05-28.

[7] "sed, a stream editor." https://www.gnu.org/software/sed/manual/sed.html. Accessed: 2018-05-30.