# A Character-Level Approach to the Text Normalization Problem Based on a New Causal Encoder

Adrián Javaloy Bornás \* Max Planck Institute for Intelligent Systems

Ginés García Mateos \*\* University of Murcia

Text normalization is a ubiquitous process that appears as the first step of many Natural Language Processing problems. However, previous Deep Learning approaches have suffered from so-called silly errors, which are undetectable on unsupervised frameworks, making those models unsuitable for deployment. In this work, we make use of an attention-based encoder-decoder architecture that overcomes these undetectable errors by using a fine-grained character-level approach rather than a word-level one. Furthermore, our new general-purpose encoder based on causal convolutions, called Causal Feature Extractor (CFE), is introduced and compared to other common encoders. The experimental results show the feasibility of this encoder, which leverages the attention mechanisms the most and obtains better results in terms of accuracy, number of parameters and convergence time. While our method results in a slightly worse initial accuracy (92.74%), errors can be automatically detected and, thus, more readily solved, obtaining a more robust model for deployment. Furthermore, there is still plenty of room for future improvements that will push even further these advantages.

### 1. Introduction

The research in natural language processing (NLP) has traditionally focused in the resolution of the *big problems*, such as automatic translation, understanding, summarizing and text generation. However, there are plenty of not so well-known problems that are often overlooked, despite being as hard to grasp as the first ones. In particular, the problem of text normalization is one of such cases. It can be defined as: given an arbitrary text, transform it into its normalized form. This normalized form depends on the context we are working on. For example, in the context of text-to-speech (TTS) systems —which is the objective of this article— normalizing a text means writing it as it should be read, e.g.:

```
I have $20 \to I have twenty dollars  
It happened in 1984 \to It happened in nineteen eighty four He weights 50kg \to He weights fifty kilograms
```

At first glance, this problem might seem trivial and rather unimportant, but nothing could be further from the truth. Normalizing text is an ubiquitous process, present in most of the NLP problems. The reason is that normalizing the input as a first step

<sup>\*</sup> Tübingen, Germany. ajavaloy@tuebingen.mpg.de

<sup>\*\*</sup> Murcia, Spain. ginesgm@um.es

significantly decreases the complexity of those problems, by the fact that equivalent phrases —yet differently written— end up being exactly the same phrase, as shown in Figure 1. WaveNet (van den Oord et al. 2016) is an example of these systems, where a generative model for TTS is trained with normalized text as input.



Figure 1: An example of equivalent phrases.

Despite its apparent simplicity, this problem entails a serious challenge. Data-driven approaches, specifically Deep Learning, deserve a special mention since: (1) there exists a general belief that Deep Learning can solve any kind of problems, and (2) it is the framework used in this article. Text normalization gathers three features that make it challenging for this kind of techniques, as it has been discussed by Sproat and Jaitly (2016). In short, these features are:

- Non-trivial cases (i.e., those whose output and input differ) are sparse.
- It is context-dependent, for example, a normalized date could change depending on the local variant of the language.
- There is no natural reason for building a text normalization database. Everyone knows that 2 means two.

A number of different models have been developed to tackle this problem. The first attempts date back to the times when researches were developing the first full TTS systems, as described by Sproat and Jaitly (2016). The systems based on traditional techniques include finite-state automatons as well as finite state transducers (Sproat 1996). The usage of these models has the advantage of being well-known techniques that work (and fail) as expected; yet, these solutions need to be hand-crafted from scratch for each language, suffering a lack of flexibility (which translates into an increase of production cost).

Nowadays, researchers are moving towards Deep Learning models, that try to learn how to solve the problem from the data itself (Sproat and Jaitly 2016). However, the amount of information these models require to work well can be prohibitive. In cases where the target language is a low-resourced one, that is, a language for which little data is available, rule-based solutions have been attempted (Sodimana et al. 2018), as well as Deep Learning models that make use of data augmentation techniques to compensate the lack of data (Ikeda, Shindo, and Matsumoto 2016). In particular, the model described in this article is quite similar to the one proposed here, except for the encoder and other minor tweaks.

The models proposed by Sproat and Jaitly (2016) require special attention. They were based on Deep Learning techniques and, in each time-step, they read a character and produce an entire word, so they were character-based at the input level, and word-based at the output. These models obtained a really high accuracy performance (one of them achieving a  $99.8\,\%$  on the English test set). Unfortunately, they suffered from undetectable errors, that is, errors that cannot be detected by just looking at the output; for example, transforming I'm 12 into I am thirteen. We suspect that these errors occur as a consequence of using recurrent word-level models.

The present approach has been designed with two goals in mind. The first one is offering a solution for the text normalization problem that exclusively uses neural networks, taking advantage of the benefits of using data-driven solutions. Furthermore, a secondary goal is to introduce convolutional components in this neural model, substituting its recurrent counterparts and, thus, speeding up the whole process. Moreover, proving the usefulness of such convolutional architecture would help to push even further the idea that Convolutional Neural Networks (CNN) can be used outside of a computer vision framework.

The main contributions of this work are as follows: (1) proposal of a character-based approach for the text normalization problem which does not suffer from unsolvable and undetectable errors; (2) a new general-purpose encoder based on causal convolutions, the Causal Feature Extractor (CFE), is introduced and tested; and (3) a variation of the traditional attention mechanisms is introduced, in which a context matrix is generated, instead of a context vector.

### 2. Materials and Methods

#### 2.1 Dataset

As stated in Section 1, it can be challenging to obtain a valid database of normalized text. Fortunately, a huge database was built and released to the whole Machine Learning community thanks to Sproat and Jaitly (2016).

This database was shaped for their word-level model and, therefore, it requires some preprocessing before being suitable for a character-level approach. Particularly, each entry on the original database is a pair of words (or a special symbol) plus an additional column describing its semiotic class, as shown in Figure 2. In order to use a character-level approach, each row needs to be composed of all the words pertaining to the same phrase, and information regarding each individual word (such as its semiotic class) has to be discarded.

```
"Semiotic Class", "Input Token", "Output Token"
"PLAIN", "Rosemary, "<self>"
"PLAIN", "is", "<self>"
"PLAIN", "a", "<self>"
"PLAIN", "plant", "<self>"
"PUNCT", ".", "sil"
"<eos>", "<eos>", ""
"DATE", "2006", "two thousand six"
"LETTERS", "IUCN", "i u c n"
```

Figure 2: A sample from the original dataset.

As shown in Figure 2, there are special symbols in the original dataset, namely:  $(1) < e \circ s >$ , denoting the end of the current sentence; (2) sil, marking a silence (comma, colon, and so on); and (3) < self >, meaning that the output in that entry is the same as the input. Since these symbols cannot be used in a character-level approach (due to the alignment problem), they are removed in the following way:  $< e \circ s >$  disappears once the sentence has been recomposed; and the remaining symbols are substituted by the input, which is also the output.

Other minor changes have to made on the original dataset to speed up the training process, obtaining a dataset as shown in Figure 3. The process<sup>1</sup> consists of the following steps:

- 1. Concatenation of words belonging to the same phrase and removal of special symbols, as mentioned before.
- 2. Phrases with non-permitted characters are discarded, keeping an alphabet of v=127 characters, including numbers, simple arithmetic symbols, currency, and the English alphabet.
- 3. Entries with an output longer than 177 characters are discarded as well, which represent only  $0.01\,\%$  of the population.
- 4. Entries are sorted in descending order with respect to their output length. This way, the padding introduced in batches is minimized and, as described by Xu et al. (2015), convergence speed is increased without a significant loss in accuracy.

```
"Input Token", "Output Token"
"Rosemary is a plant .", "Rosemary is a plant ."
"2006 IUCN .", "two thousand six i u c n ."
"We all lost .", "We all lost ."
"vol 6 no", "volume six no"
"Rees et al .", "Rees et al ."
```

Figure 3: Sample entries from the preprocessed dataset.

### 2.2 Experimental Setup

After preprocessing the dataset, subsets of the final dataset have to be chosen in order to train and compare the models in a reasonable time. For this purpose, three experiments are prepared, each one having its own dataset. The first two datasets will be used to test and compare different models, whereas the latter will be used to train the final model and compare it with prior results. Figure 1 shows their names, training times, number of training elements, and the way entries have been selected: random means that they have been randomly taken and shortest that the elements with shortest outputs have been selected. In all cases, 1/5 additional entries are taken for the validation and for the test sets.

Name	Duration	Size	Selection
E1	1 h	50 000	shortest
E2	$12\mathrm{h}$	50000	random
E3	$22\mathrm{h}$	1000000	random

Table 1: Description of the sets used in the experiments.

Regarding the actual input and output used in the model, a one-hot encoding has been chosen, i.e., a string  $s = s_0 s_1 \dots s_l$  of size  $l \in \mathbb{N}$  will turn into a matrix  $\mathbf{X} \in \mathbb{M}_{v \times l}$ 

<sup>1</sup> The code used for preprocessing the data is available at: https://github.com/adrianjav/text-normalization-preprocess.

whose *i*-th column  $x_i \in \mathbf{X}$  is set to zero in every position but the one corresponding to the index of the character  $s_i$  according to the model alphabet.

The advantages and disadvantages of using a character-level model have been described by other authors, since it appears as a design question in many NLP problems. Four arguments in favor of character-level approaches are shown, three of them introduced by Chung, Cho, and Bengio (2016), and the last one given by Lee, Cho, and Hofmann (2017):

- Out-of-vocabulary issues do not appear anymore. We could suffer from out-ofalphabet issues, but these are easier to solve.
- Such approaches are able to model rare morphological variants of a word.
- Input segmentation is no longer required.
- By not segmenting, we encourage the models to discover the internal rules and structure of the sentences by themselves.

Since text segmentation is known to be problematic and error prone, even for well-known languages like English, getting rid of this step without losing performance is a significant advantage to take into account.

We present an additional argument for character-level approaches. If the model uses attention mechanisms, observing the attention matrices after a particular sample could allow us to gain a better understanding of the system's logic and the language itself. For example, consider the case where the model transforms 2s into two seconds; its attention matrix could potentially show that the last letter was produced by looking at the number.

### 2.3 Encoder-Decoder Architecture

The encoder-decoder architecture is a common design in recent Neural Machine Translation literature, and its architecture is easy to grasp. The model is composed of two parts: (1) an encoder that takes the input  $\mathbf X$  (in this case, a phrase) and produces an intermediate representation  $\mathbf Z$  (or code) that highlights its main features; and (2) a decoder that processes that set of features and produces the required output  $\mathbf Y$  (in this case, a normalized phrase). Figure 4 shows a basic diagram of this model.

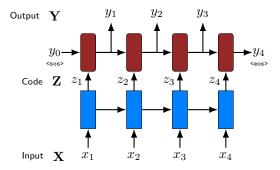


Figure 4: A basic encoder-decoder architecture. Blue: encoder. Red: decoder.

There is a trend in using Long Short-Term Memory (LSTM) neural networks as encoders and decoders (for example, Sutskever, Vinyals, and Le (2014)) due to their ability to capture long dependencies among the elements of a sequence. Our proposed model

will use an LSTM network as decoder. However, different encoders will be analyzed, including the proposed one, and their performance will be tested and compared.

The basic encoder-decoder architecture looks great at first, but some key issues arise when they are put it on practice. Two of them stand out and are worth mentioning: (1) as shown in Figure 4, at each step the decoder works with the code produced at that moment, hindering the usage of long-term dependencies; and (2) output and input need to have the same length, constraining the suitable use cases of the model.

These two setbacks are overcome by the implementation of attention mechanisms (Bahdanau, Cho, and Bengio 2014). The idea behind them, depicted in Figure 5, is quite simple: first, produce the codes of the whole input sequence at once and, in each time step, let the decoder choose the most interesting elements of the input based on the latest output.

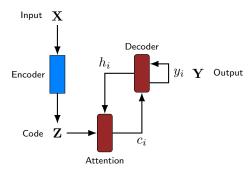


Figure 5: An encoder-decoder architecture with attention mechanisms.

This can be expressed in mathematical terms as follows. Suppose that  $\mathbf{Z} = z_0 z_1 \dots z_k$  is the code at the *i*-th time step; then, the model gets a description of the interesting features  $h_i$  (typically given by the decoder hidden states) and a small neural network produces a vector  $\alpha = \alpha_0 \alpha_1 \dots \alpha_k$  from this description. This vector  $\alpha$  is transformed into a stochastic vector, i.e., a vector such that  $\sum \alpha_i = 1$ , via

$$\alpha_i = \frac{\exp \alpha_i}{\sum_j \exp \alpha_j}$$

Now,  $\alpha_i$  represents the interest of the decoder with respect to the *i*-th element of the code  $z_i$  and a context vector is produced, that is, a vector representing the portion of the input that is actually interesting for the decoder.

Traditionally, this context vector is taken as a weighted sum of the elements of  $z_i$ , weighted by  $\alpha$ ,  $c = \sum_i \alpha_i z_i$ . A different approach is taken in this research. Instead of performing a weighted sum, a hyperparameter d describing the number of context elements is set, and a context matrix c is generated where the i-th column  $c_i$  corresponds to the element  $\alpha_i z_i$  having the i-th greatest value of  $\alpha_i$ , that is:

$$c_i = a_j z_j$$
 where  $j = \operatorname*{arg\,max}_{j=0}^k \left\{ a_j \text{ such that } a_j z_j \neq c_l \text{ for } 0 \leq l < i \right\}$ 

The idea inspiring this modification is that by not averaging the feature vectors, the internal semantic of each individual element is preserved.

### 2.4 The Proposed Causal Feature Encoder

The new encoder proposed in this paper can be described as a two-step modification of a traditional CNN. The first change is that, instead of using regular convolutions, causal convolutions (introduced by van den Oord et al. (2016)) are used. Figures 6a and 6b show a basic representation of a regular and causal neural network, respectively.

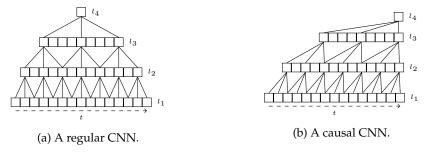


Figure 6: A sample of regular (a) and casual (b) CNNs.

The second step allows the CFE to solve an important drawback: it can only capture dependencies in one direction. To overcome it, the CFE is made bidirectional as with LSTMs. In this way, it contains two independent models that read the input in each direction and concatenate their outputs to produce the desired output. This is depicted in Figure 7.

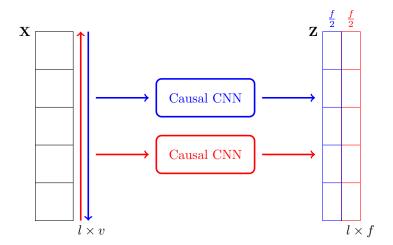


Figure 7: Diagram showing the bidirectionality of the proposed CFE.

An additional change has been made. Because of the long sequences found in text normalization (up to 177 characters), the concept of dilated convolutions has been applied to CFE, as described by van den Oord et al. (2016), doubling the dilatation of each layer as it goes deeper into the structure. By doing that, the actual receptive field of the model is significantly increased without increasing the number of parameters.

This new encoder comes as an attempt to solve a problem that CNNs show with attention mechanisms. In previous experiments, it was observed that CNNs tend to

attend the wrong inputs according to our prior intuition, namely they choose the i + C-th element instead of the i-th element, where C is a constant. Our intuition is that this is caused by the padding introduced in each side. By using causal convolutions, the model is forced to choose the outermost elements if it is interested in those.

### 2.5 Statistical Test

When comparing various models, it is critical to ensure that the differences that can be appreciated are statistically significant. It has to be proved that those differences are actual differences, and not a product of the implicit variance coming from training the models. This is typically performed using some statistical test that will assert that the differences are actual differences up to some probability percentage, usually 95 %.

For this article, we have opted for the approximate randomization test (Riezler and III 2005). This statistical test measures the probability of the outputs of two different models of being indistinguishable, i.e., the probability that, by just looking at the predictions, we cannot tell whether those predictions come from different models. The main reasons for opting for this method are: (1) it is computationally cheap; (2) it is distribution-free, meaning that it does not make any assumptions on the distribution measured; and (3) it is model-free, that is, the only required resources to perform the test are the actual predictions, making it suitable for any kind of conceivable model.

Let us assume that the predictions are two ordered sets, A and B, and that we have a function e that measures the closeness of the predictions with respect to the actual solutions Y, for example, the accuracy. Then, we can define the function:

$$t(A,B) = |e(A,Y) - e(B,Y)|$$

and we are seeking the probability of getting a bigger error than t(A, B), assuming that both sets of predictions are indistinguishable, i.e.,  $P(X \ge t(A, B)|H_0)$ .

The algorithm that approximates this value just repeats many times (typically a thousand) the same process: it randomly swaps each element of the first set with its counterpart in the second set, and counts the number of times that the total error, measured by t, is greater or equal than the original one, that is, t(A,B). Figure 8 shows the pseudocode of this algorithm.

# 3. Results

# 3.1 Proposed Methods and Number of Parameters

As said before, different encoders are considered to test whether CFE entails an actual improvement with respect other encoders. These encoders (and their alias) are the following:

LSTM A simple bidirectional LSTM network.

FCNN A FCNN encoder where the *i*-th element is an embedding of the *i*-th input.

FE A traditional CNN with dilated convolutions.

CFE The Causal Feature Extractor encoder.

Hyperparameters of each model were manually tuned, and the results have been averaged over five exact models trained with different random seeds.

The most basic question comparing multiple neural models concerns the number of trainable parameters. This data is quite easy to obtain, and knowing the numbers of

Figure 8: Pseudocode of the approximate randomization test. R is the number of repetitions selected.

parameters of a model, equivalently, its size —and, to a lesser extent, its complexity—can be a deciding point in case of a tie. The number of parameters of the models are shown in Table 2.

Encoder	LSTM	FCNN	FE	CFE
No. of parameters: encoder (millions)	1.102	0.285	0.111	0.111
No. of parameters: total (millions)	7.380	6.653	6.479	6.479

Table 2: Number of parameters of each model.

# 3.2 First Experiment

The results obtained for the first experiment are shown in Table 3. These results are averaged over five runs and extracted from the test set results, except from the results concerning the training speed, which are taken from the training logs. From left to right, the columns of Figure 3 show:

- Negative Log-Likelihood Loss (NLLLoss). It is the measure optimized by the neural network during training, since it is the usual measure in a classification setting.
- Character Error Rate (CER). It is defined as the Levenshtein distance between the prediction and the expected value, measured in characters.
- Accuracy. It is a basic and well-known measure, defined as the percentage of correct predictions.
- Number of iterations performed during the training phase in the duration of the experiment (in this case 1 hour).
- Rate. Number of iterations per second, on average, achieved during training.

	Test			Validation		
Encoder	NLLLoss	CER (%)	Acc (%)	No. iters	Rate	
LSTM	1.352	3.13	95.87	3 620	1.005	
<b>FCNN</b>	5.035	70.61	28.40	4370	1.214	
FE	1.042	2.52	96.46	6980	1.939	
CFE	0.952	2.24	96.83	6300	1.750	

Table 3: Results obtained for the first experiment (E1).

In order to get an understanding of the differences in the training process, Figure 9 shows the evolution of the NLLLoss over the validation tests of each model during the training process. Table 4 shows the resulting p-values after running the approximate randomization test over each pair of models.

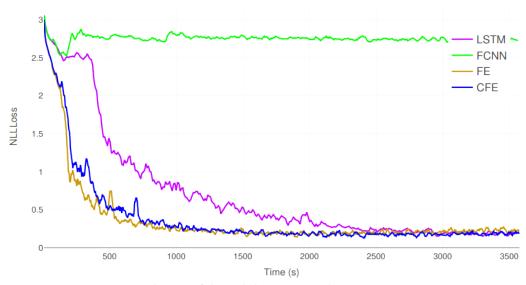


Figure 9: Evolution of the validation error during training on E1.

p-value	LSTM	<b>FCNN</b>	FE	CFE
LSTM		0.001	0.001	0.003
<b>FCNN</b>	0.001		0.001	0.001
FE	0.001	0.001		0.019
CFE	0.003	0.001	0.019	

Table 4: P-values of the first experiment (E1).

# 3.3 Second Experiment

As before, Table 5 shows the same measures, but now regarding the second experiment. Figure 10 and Table 6 show the evolution of the validation error and the results of the statistical test on the second experiment, respectively.

		Test	Validati	ion	
Encoder	NLLLoss	CER (%)	Acc (%)	No. iters	Rate
LSTM	3.310	25.59	71.06	8 900	0.206
FCNN	5.396	82.09	17.90	17750	0.411
FE	2.680	11.93	83.38	36650	0.848
CFE	2.686	12.69	83.45	36200	0.838

Table 5: Results obtained for the second experiment (E2).

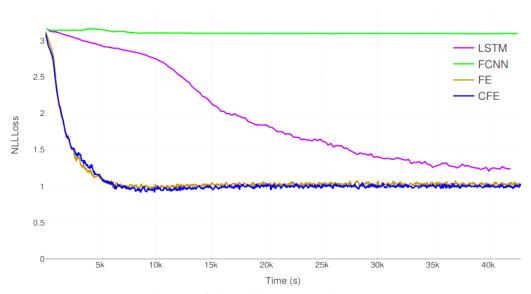


Figure 10: Evolution of the validation error during training on E2.

p-value	LSTM	FCNN	FE	CFE
LSTM		0.001	0.001	0.001
<b>FCNN</b>	0.001		0.001	0.001
FE	0.001	0.001		0.001
CFE	0.001	0.001	0.001	

Table 6: P-values for the second experiment (E2).

# 3.4 Third Experiment

In this subsection, the results of the final model after running the third experiment are shown. The final architecture is identical to the one with the CFE encoder used in the previous experiments. Figure 11 depicts the evolution of the training and validation error during the training phase, and Table 7 shows the results obtained for the test set.

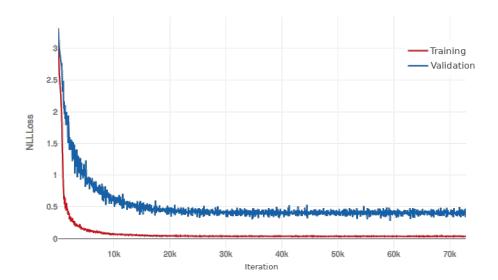


Figure 11: Evolution of the training (red) and validation (blue) errors over E3.

		Test	
Encoder	NLLLoss	CER (%)	Acc (%)
CFE	1.701	5.44	92.74

Table 7: Results on the test set for the third experiment (E3).

### 3.5 Attention Matrices

In order to show whether the CFE encoder makes a better usage of the attention mechanisms than its non-causal counterpart, it is necessary to show some actual examples and the attention matrices that they generate. These matrices are a representation of the decoder focus while it was processing the input: the i-th row represents the i-th character it predicted, and the j-th column is the model focus while predicting that character.

The first case, shown in Table 8, is an example extracted from the test set of the first experiment. The input phrase is 23 Aug 2013. Regarding what it would be expected from the attention matrix to look like, it can expressed in three phases: (1) it writes out the day while focusing on its digits; (2) shifts its attention towards the month; and (3) it finishes by looking at the year. Figure 12 shows the attention matrices.

Table 8: Predictions of the different models for the first example.

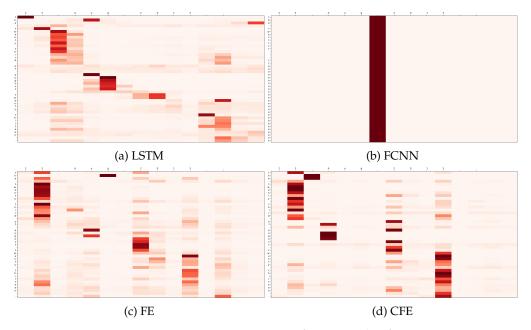


Figure 12: Attention matrices of an example of E1.

The second example has been taken from the test set of the second experiment. Table 9 shows the predictions, whereas Figure 13 shows the attention matrices. This is an example where the input and output are identical, and so, the expected attention matrices should resemble an identity matrix.

Input		Belpiela is a community in Tamale Metropolitan District in the
		Northern Region of Ghana .
Output		Belpiela is a community in Tamale Metropolitan District in the
		Northern Region of Ghana .
LSTM	X	Belpiela is a community in Tamale Metropolitan Disire
		egion te i e
<b>FCNN</b>	X	"
FE	X	Belpiela is a community in Tamale Metropolitan District
		in the Northern Region Region of Ghana .
CFE	✓	Belpiela is a community in Tamale Metropolitan
		District in the Northern Region of Ghana .

Table 9: Predictions of the different models for the second example.

# 3.6 Error analysis

In order to get a better understanding of the type of errors of the proposed model, those produced on the test set by the model of the third experiment has been dumped and analyzed by hand. Based on these observations, the taxonomy of the errors has been defined as follows:

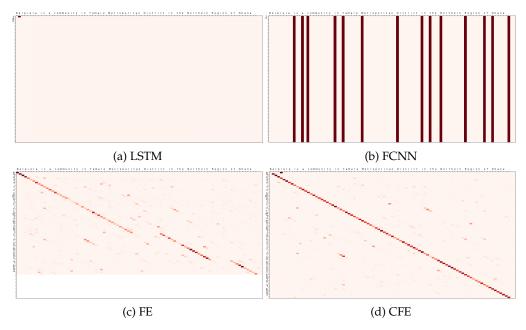


Figure 13: Attention matrices of an example of E2.

T1 Infinite loop errors. The attention system of the model gets stuck and the maximum number of printed characters is reached. For example:

T2 Coincidental errors. Predictions where only a few isolated characters are wrongly printed. For example:

```
Input The income was $11,091 . Output The income was eleven thousand ninety one dollars . Prediction The income was fleven thousand ninety one dollars .
```

T3 Early stop errors. Errors where the model finishes before processing the whole input. For example:

```
Input Parmentier , Bruno ( 2000-05-01 ) .
Output Parmentier , Bruno ( the first of may two thousand ) .
Prediction Parmentier , Bruno ( .
```

T4 (Finite) jumps. The attention model finds the same pattern in the entry and repeats/oversees part of it. For example:

```
Input According to the 2011 census of India , Bhisenagar has
    818 households .
```

Output According to the twenty eleven census of India ,

Bhisenagar has eight hundred eighteen households .

Prediction According to the twenty eleven census of India ,

Bhisenagar has eighteen households .

A simple classification tool has been implemented in order to (approximately) quantify the errors according to their type. Results are shown in Table 10 where *Others* refers to errors unclassified by the tool. Note that errors produced by jumps are not detected by the tool, but represent a big portion of the unclassified errors.

Туре	T1	T2	Т3	Others	Total
Quantity	23381	7159	50	10696	41 286
Percentage (%)	56.63	17.34	0.12	25.9	100

Table 10: Errors distribution from the test set of E3.

Besides these types of errors, it is worth-mentioning those caused by the dataset itself. These come from different sources, for example, from inconsistent rules for normalizing text among different entries, e.g.,

by providing a few entries for rare cases that resemble too much to others, e.g.,

```
Input 1980 A engine added to Transporter ( T 3 ) . Output one nine eight o A engine added to Transporter ( T three ) . Prediction nineteen eighty A engine added to Transporter ( T three ) .
```

by inconsistency in the entries (e.g. American vs British) text, e.g.,

```
Input The mobilisation was announced by the mayor.

Output The mobilization was announced by the mayor.

Prediction The mobilisation was announced by the mayor.

Input The Robinsons are a family in the soap opera Neighbours.

Output The Robinsons are a family in the soap opera neighbors.

Prediction The Robinsons are a family in the soapera Neighbors.
```

The model specially struggles deciding whether it should maintain capital letters on the predictions. This last example also contains an example of overseeing parts of the input, probably because the similarities between the words <code>soap</code> and <code>opera</code>. Another example of such jumps, in this case going backward in the input and thus repeating words, is the following:

```
Input The primary east west highway passing through Belmont is
    interstate 85 .
Output The primary east west highway passing through Belmont is
    interstate eighty five .
Prediction The primary east west west highway passing through
    Belmont is interstate eighty five .
```

which happened because the model confounds the suffix of west with the one of east as it can be seen on Figure 14. Attention matrices can be displayed for all these errors shedding light on the attention-related issue underlying, except for the coincidental errors.

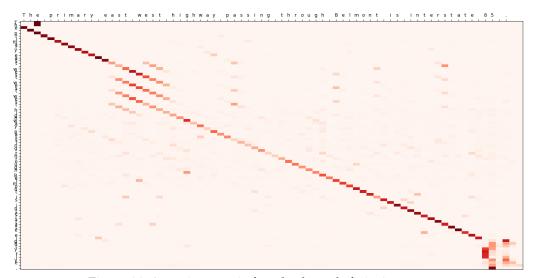


Figure 14: Attention matrix for a backwards finite jump error.

Finally, is it worth-noting the role of undetectable errors as they were a problem in prior work. Among the analyzed test errors it can be found some like those, such as the one shown in the example for error T4. However, in all cases they are a realization of another type of error that happens to look like that by chance. The errors source can be explained and solved more readily and patterns in attention matrices could be leveraged to detect them. For example, the aforementioned error occurs as a realization of a jump error where the model confounds in 818 the first 8 with the third one when processing the input.

# 4. Discussion

This section discusses the results presented in Section 3. Specifically, the main questions stated in Section 1 were:

- 1. Can the problem of text normalization be solved just by making use of neural networks?
- 2. Is it viable such a solution using convolutional components? And which decoder is better?

Answering the first question, the most obvious result we can extract, based on any of the results from E1 and E2 (for example, Figure 10), is that the FCNN encoder does not work at all. Most probably, this erratic behavior comes from the differentiating feature of FCNN, that is, it extracts information of a single character of the input (instead of a neighborhood of it). This serves as a proof of an unsurprising result: in order to work properly, the decoder cannot act on its own, the work of extracting high level features from the characters surroundings is essential.

Let us drop FCNN out of the equation. By looking at the second experiment, we can observe a significant difference between the LSTM and its convolutional counterparts. Specifically, Table 5 shows the accuracy of the convolutional encoders is about  $12\,\%$  higher than the one of the LSTM encoder. Nevertheless, this could have a simple explanation: it could be the case that the only thing that the LSTM needs is time. This leads us to the biggest differences between them: number of parameters, convergence time, and iteration time. Three points strengthen this argument:

- Table 2 shows the number of parameters of each model. The models only differ in the number parameters of the encoder (the rest of the model has 6 368 million parameters), as expected. Thus, the LSTM encoder has ten times more parameters than the convolutional encoders, making it harder to train and more expensive to use.
- Figures 9 and 10 illustrate the convergence time differences. In particular, the LSTM encoder started to converge in E2 after 2 h45 min of training, whereas the convolutional encoders were close to their minimum at the mark of 1 h23 min.
- Regarding the iteration speed, Tables 3 and 5 show that, besides being more accurate, the convolutional encoders operate around two and four times faster than the LSTM encoder, respectively.

We present three arguments to explain this phenomenon: (1) the aforementioned difference in the number of parameters; (2) the existence of recursive connections in the LSTM, making it harder to optimize; and (3) the fact that convolutional networks run quite fast on GPUs. Anyway, this ensures that convolution-based encoders are viable, significantly faster, and statistically distinguishable from recurrent encoders (as shown in Table 6).

After solving one part of the second question, we just need to decide between the convolutional encoders. As show in Table 5, quantitatively both encoders are very similar, even though CFE obtains better results and is distinguishable from FE. Qualitatively, it looks quite brighter for the CFE encoder, for example:

- The first example, Figure 12, shows that the three encoders behave in a similar fashion. However, CFE seems cleaner and more localized, since it knows better where to focus, to the point where it is easy to see three distinguished phases: day, month, and year.
- The second example, Figure 13, is clear. The LSTM encoder did not converge yet, so its prediction is way off the mark. Regarding the convolutional encoders, CFE gets the example right, its attention matrix seems quite clean, and it resembles a lot to an identity matrix; whereas the FE encoder struggles to maintain the focus (many non-diagonal elements have taken attention) and makes erratic leaps (which manifest in missing words in the prediction, see Table 9).

Thus, it can be concluded that, in this case, CFE is preferable to FE due to its qualitative benefits and, to a lesser extent, its quantitative results. Regarding the undetectable errors reported in Sproat and Jaitly (2016), it can be firmly confirmed that they are not an issue in these models as they appear by chance due to solvable errors. Specifically, these errors are highly related with the attention mechanism as, like Table 10 shows, the most common error is getting stuck in an infinite loop. These problems make the model lost its focus and jump around when confounding similar parts of the entry. Therefore, this could be greatly improved by using more sophisticated attention models that, for example, focus on local environments, take into account the index, or force the model to put more focus in the next character of the input.

Finally, we are going to discuss the results obtained on the third experiment. Figure 11 shows that, during training, the model quickly converged and there seems to be a gap between training and generalization error that the model has not been able to resolve. However, the results obtained on the test set are quite promising: it has obtained 92.74 % accuracy and 5.44 % CER, against the 99.8 % accuracy and 13.43 % CER obtained by the models of Sproat and Jaitly (2016) and Ikeda, Shindo, and Matsumoto (2016), respectively.

Leaving out the obvious differences (datasets, training time, and so on) that make comparing these models cumbersome, it is clear that these first results are promising and point out into a viable direction to solve the problem of text normalization in a data-driven fashion without the existence of undetectable errors being an unsolvable error, thus answering the first question made at the beginning of this section.

#### 5. Conclusions

In this paper, a new encoder-decoder architecture with attention mechanisms has been proposed for the problem of text normalization, using a character-level approach and introducing a new type of encoder. This encoder, called Causal Feature Extractor, is a brand new technique designed to work well in cooperation with the attention mechanisms. In the experiments, it has empirically proved to achieve good results, using the attention matrices more like someone would expect to use them by hand. Besides, it is able to work at least as good as the best of the compared encoders, and it brings all the benefits of using convolutional neural networks to the table. The last thing that distinguishes this encoder to the traditional recurrent encoders is its simplicity to be adapted to other input layouts (for example, matrices of pixels).

With respect to prior work, the initial results have shown to be quite close to the state-of-the-art, with plenty of room to future improvement. Despite getting a worse accuracy result (92.74% vs. 99.8%), it does not critically suffer from undetectable errors (meaning that it can be viable for commercial use), nor it seems to concentrate its errors on any particular semiotic class since the errors are attention-based.

Finally, some other interesting results can be extracted, like the introduction of a new variation of the attention mechanisms that uses a context matrix instead of a vector, or the empirical proof that empowers the role of encoders in the encoder-decoder architectures by showing that the system does not work if we just take features of single elements (without their neighborhood).

Future research lines could focus on some of the following:

- The usage of the CFE encoder as a general-purpose encoder.
- A deeper training and hyperparameters selection to obtain better results.
- Exploring the errors made by the model, like the leap errors mentioned in Section 4.
- Conditioning the model to external factors, for example, to distinguish between British and American English.

### 6. Acknowledgments

We would like to express our gratitude to Richard Sproat for his useful feedback on this article. Besides, Adrián acknowledges support from the Max Planck Institute for Intelligent Systems.

### References

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Chung, Junyoung, Kyunghyun Cho, and Yoshua Bengio. 2016. A character-level decoder without explicit segmentation for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL* 2016, *August* 7-12, 2016, *Berlin, Germany, Volume* 1: Long Papers.

Ikeda, Taishi, Hiroyuki Shindo, and Yuji Matsumoto. 2016. Japanese text normalization with encoder-decoder model. In *Proceedings of the 2nd Workshop on Noisy User-generated Text*, NUT@COLING 2016, Osaka, Japan, December 11, 2016, pages 129–137.

Lee, Jason, Kyunghyun Cho, and Thomas Hofmann. 2017. Fully character-level neural machine translation without explicit segmentation. *TACL*, 5:365–378.

van den Oord, Aäron, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. In *The 9th* ISCA Speech Synthesis Workshop, Sunnyvale, CA, USA, 13-15 September 2016, page 125.

Riezler, Stefan and John T. Maxwell III. 2005. On some pitfalls in automatic evaluation and significance testing for MT. In Proceedings of the Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization@ACL 2005, Ann Arbor, Michigan, USA, June 29, 2005, pages 57–64.

Sodimana, Keshan, Pasindu De Silva, Richard Sproat, A Theeraphol, Chen Fang Li, Alexander Gutkin, Supheakmungkol Sarin, and Knot Pipatsrisawat. 2018. Text normalization for bangla, khmer, nepali, javanese, sinhala, and sundanese tts systems. In 6th International Workshop on Spoken Language Technologies for Under-Resourced Languages (SLTU-2018), pages 147–151, 29-31 August 2018, Gurugram, India.

Sproat, Řichard. 1996. Multilingual text analysis for text-to-speech synthesis. *Natural Language Engineering*, 2(4):369–380.

Sproat, Richard and Navdeep Jaitly. 2016. RNN approaches to text normalization: A challenge. *CoRR*, abs/1611.00068.

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada, pages 3104–3112.

Xu, Kelvin, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the* 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015, pages 2048–2057.