Enhancing BERT for Lexical Normalization

Benjamin Muller **Benoît Sagot** **Djamé Seddah**
Inria
firstname.lastname@inria.fr

Abstract

Language model-based pre-trained representations have become ubiquitous in natural language processing. They have been shown to significantly improve the performance of neural models on a great variety of tasks. However, it remains unclear how useful those general models can be in handling non-canonical text. In this article, focusing on User Generated Content (UGC) in a resource-scarce scenario, we study the ability of BERT (Devlin et al., 2018) to perform lexical normalisation. Our contribution is simple: by framing lexical normalisation as a token prediction task, by enhancing its architecture and by carefully fine-tuning it, we show that BERT can be a competitive lexical normalisation model without the need of any UGC resources aside from 3,000 training sentences. To the best of our knowledge, it is the first work done in adapting and analysing the ability of this model to handle noisy UGC data.[1]

1 Introduction

Pre-trained contextual language models (e.g. ELMo, Peters et al., 2018; BERT, Devlin et al., 2018) have improved the performance of a large number of state-of-the-art models on many Natural Language Processing (NLP) tasks. In this article, we focus on BERT (Bidirectional Encoder Representations from Transformers), the contextual language modelling architecture that recently had the greatest impact.

A major specificity of BERT is that it is trained to jointly predict randomly masked tokens as well as the consecutiveness of two sentences. Moreover, it takes as input *WordPieces* tokens which consists in frequent sub-word units (Schuster and Nakajima, 2012). Finally, available pre-trained models have been trained on the concatenation of the Wikipedia corpus and the BookCorpus, which constitutes a large corpus of canonical (i.e. proper, edited) language.

Putting aside the efficiency of its transformer-based architecture, these three aspects respectively enable BERT to elegantly cope with out-of-vocabulary words and to include contextual information at the token and at the sentence levels, while fully taking advantage of a training corpus containing billions of words.

Without listing all of them, BERT successfully improved the state-of-the-art for a number of tasks such as Name-Entity Recognition, Question Answering (Devlin et al., 2018) and Machine Translation (Lample and Conneau, 2019). Moreover, it has recently been shown to capture a rich set of syntactic information (Hewitt and Manning, 2019; Jawahar et al., 2019), without the added complexity of more complex syntax-based language models.

However, it remains unclear and, to the best of our knowledge, unexplored, how well can BERT be used in handling non-canonical text such as *User-Generated Content* (UGC), especially in a low resource scenario. This question is the focus of this paper.

As described in (Foster, 2010; Seddah et al., 2012; Eisenstein, 2013; Baldwin et al., 2013), UGC is often characterized by the extensive use of abbreviations, slang, internet jargon, emojis, embedded metadata (such as hashtags, URLs or *at* mentions), and non standard syntactic constructions and spelling errors.

This type of non-canonical text, which we characterize as *noisy*, negatively impacts NLP models performances on many tasks as shown in (van der Goot et al., 2017; van der Goot and van Noord, 2018; Moon et al., 2018; Michel and Neubig, 2018) on respectively Part-of-

[1]The code is available in the following repository https://gitlab.inria.fr/bemuller/bert_nomalizer

Proceedings of the 2019 EMNLP Workshop W-NUT: The 5th Workshop on Noisy User-generated Text, pages 297–306

Speech Tagging, Syntactic Parsing, Name-Entity Recognition and Machine Translation.

In this context and as impactful as BERT was shown to be, its ability to handle noisy inputs is still an open question[2]. Indeed, as highlighted above, it was trained on highly edited texts, as expected from Wikipedia and BookCorpus sources, which differ from UGC at many levels of linguistic descriptions, and which, of course, exhibit an important domain gap.

Based on those observations, we take lexical normalisation of UGC as a case study of how BERT can model noisy inputs. Briefly, lexical normalisation is the task of translating non-canonical words into canonical ones. It involves a detection step in assessing if a word is already canonical or not, followed by a normalisation step. All the experiments presented in this paper are carried out on the dataset released by Baldwin et al. (2015), which is the only non-raw resource we use. This is because our goal is to study how well BERT handles noisy UGC by itself, which means that, unlike most previous work (e.g. van der Goot and van Noord, 2017), we cannot make use of external UGC-specific resources such as word embeddings and language models trained on UGC or dedicated lexicons.

Yet, building a lexical normalization model in such a setting is a challenging endeavor. As we will present, blindly fine-tuning BERT on such as task is not possible. It requires architectural and optimization adaptations that constitute the core of our contribution.

In summary, we show that BERT can be adapted to perform lexical normalisation in a low resource setting without external data covering the source UGC domain, aside from 2950 aligned training examples that include only 3928 noisy words. In this purpose, we make three contributions:

- We design a WordPiece tokenizer that enforces alignment between canonical and noisy tokens.
- We enhance the BERT architecture so that the model is able to add extra tokens or remove them when normalisation requires it.
- We fine-tune the overall architecture with a novel noise-specific strategy.

[2]The importance of this research question is further confirmed by the very recent pre-publication of the work by (Gopalakrishnan et al., 2018) who study how BERT is affected by synthetic noise

In a few words, our paper is the first attempt to successfully design a domain transfer model based on BERT in a low resource setting.

2 Related Work

There is an extensive literature on normalizing text from UGC.

The first systematic attempt was Han and Baldwin (2011). They released 549 tweets with their normalized word-aligned counterparts and the first result for a normalization system on tweets. Their model was a Support-Vector-Machine for detecting noisy words. Then a lookup and n-gram based system would pick the best candidate among the closest ones in terms of edit and phonetic distances. Following this work, the literature explored different modelling framework to tackle the task, whether it is Statistical Machine Translation (Li and Liu, 2012), purely unsupervised approach (Yang and Eisenstein, 2013), or syllables level model (Xu et al., 2015).

In 2015, on the occasion of the Workshop on Noisy User-Generated Text, a shared task on lexical normalization of English tweets was organized (Baldwin et al., 2015) for which a collection of annotated tweets for training and evaluation was released. We will refer it as the *lexnorm15* dataset. A wide range of approaches competed. The best approach (Supranovich and Patsepnia, 2015) used a UGC feature-based CRF model for detection and normalization.

In 2016, the MoNoise model (van der Goot and van Noord, 2017) significantly improved the State-of-the-art with a feature-based Random Forest. The model ranks candidates provided by modules such as a spelling checker (aspell), a n-gram based language model and word embeddings trained on millions of tweets.

In summary, two aspects of the past literature on UGC normalization are striking. First, all the past work is based on UGC-specific resources such as lexicons or large UGC corpora. Second, most successful models are modular in the sense that they combine several independent modules that capture different aspects of the problem.

3 Lexical Normalisation

3.1 Task

Lexical normalisation is the task of translating non canonical words into canonical ones.. We illustrate it with the following example (Table 1). Given a

noisy source sentence, our goal is to predict the *gold* canonical sentence.

Noisy	*yea... @beautifulloser8 im abt to type it uuup !!*
Gold	*yeah... @beautifulloser8 i'm about to type it up !*

Table 1: Noisy UGC example and its canonical form (Gold)

We make a few comments on this definition. First, lexical normalisation assumes a certain degree of word level alignment between the non-canonical source text and the canonical one.

Second, language evolves. It varies across domain, communities and time, specifically online (Jurafsky, 2018). There is therefore no universal definition of what is a canonical form and what is not. In the context of NLP, this means that we have to set conventions and define what we consider as canonical. In our case, the task is made less complicated as we are tied to the conventions set by our training data set.

Finally, to grasp the complexity of such a task, we list and illustrate non exhaustively the sort of linguistic phenomenons that lexical normalisation of UGC involves. Lexical normalisation involves handling the following cases :

- spelling errors : *makeing* in *making*
- internet Slang : *lmfao* in *laughing my f.cking ass off* [3]
- contraction : *lil* for *little*
- abbreviation : *2nite* for *tonight*
- phonetics : *dat* for *that*

It also involves detecting that the following should be untouched : *:), @KhalilBrown, #Beyonce, rt*

3.2 Data

We base all our experiments on the WNUT data released by Baldwin et al. (2015). This dataset includes 2950 noisy tweets for training and 1967 for test. Out of the 44,385 training tokens, 3,928 require normalisation leading to an unbalanced data set. Among those 3,928 noisy tokens, 1043 are 1-to-N (i.e. single noisy words that are normalized as several words) and 10 are N-to-1 cases (i.e. several noisy words that are normalized as single canonical words).

[3]Normalisation found in the lexnorm 2015 dataset

As highlighted before, our framework is more challenging than the standard approach to normalisation, illustrated by the 2015 shared task, that usually authorizes external UGC resources. As our goal is to test the ability of BERT, a model trained on canonical data only, we restrain ourselves to only using the training data as examples of normalisation and nothing more.

Our work is therefore to build a domain transfer model in a low resource setting.

4 Normalisation with BERT

4.1 BERT

We start by presenting the components of BERT that are relevant for our normalisation model. All our work is done on the released *base* version.

4.1.1 WordPiece Tokenization

BERT takes as input sub-word units in the form of *WordPiece* tokens originally introduced in Schuster and Nakajima (2012). The WordPiece vocabulary is computed based on the observed frequency of each sequence of characters of the corpus BERT is pre-trained on: Wikipedia and the BookCorpus. It results in a 30 thousand tokens vocabulary. We will refer to the process of getting WordPiece tokens from word tokens simply as *tokenization* for brievety.

Reusing BERT, in any way, requires to use its original WordPiece vocabulary. In the context of handling non canonical data, this is of primary importance. Indeed, frequent tokens in our non canonical data set might not appear in the vocabulary of BERT and therefore will have to be split. For example, the word *lol* appear more than 222 times in the original lexnorm15 dataset (more than the word *like* that appears 187 times). Still, it is not in BERT-base WordPiece vocabulary. For tokenization of WordPieces, we follow the implementation found in the *huggingface pytorch-pretrained-BERT* project [4]. It is implemented as a greedy matching algorithm. We write it in pseudo-code in Algorithm 1.

4.1.2 Masked Language Model

We now present one crucial aspect of BERT architecture. It was trained jointly on two objectives : On *next sentence* prediction on the one hand. On the other hand, it was trained on

[4]https://github.com/huggingface/pytorch-pretrained-BERT

Algorithm 1: Greedy WordPiece tokenization

```
Vocabulary = Bert WordPiece Vocabulary;
init start=0, string=word,
wordPieceList = list();
while string not empty do
    substring:=string[start:]
    while substring not empty do
        if substring in Vocabulary then
            wordPieceList :=
            wordPieceList U [substring]
            break loop
        else
            substring := substring[:-1]
        end
    end
    start := start + length(substring)
end
Result: wordPieceList
```

Note : Tokenizing words into wordpiece tokens, by matching in an iterative way from left to right, the longest sub-string belonging to the wordpiece vocabulary

Masked Language Model (MLM). As we frame our normalization task very closely to it, we describe MLM briefly.

For each input sequence, 15% of the WordPiece tokens are either replaced with the special token [MASK] (80% of the time), replaced by a random token (10% of the time) or untouched (10% of the time). BERT is trained by predicting this portion of token based on the surrounding context.

4.2 Fine-Tuning BERT for Normalisation

We now present the core of our contribution. How to make BERT a competitive normalisation model? In a nutshell, there are many ways to do lexical normalisation. Neural models have established the state-of-the-art in the related Grammatical Error Correction task using the sequence to sequence paradigm (Sutskever et al., 2014) at the character level. Still, this framework requires a large amount of parallel data. Our preliminary experiments showed that this was unusable for UGC normalisation. Even the use a powerful pre-trained model such as BERT for initializing an encoder-decoder requires the decoder to learn an implicit mapping between noisy words and canonical ones. This is not reachable with only 3000 sentences.

We therefore adapted BERT in a direct way for normalisation. As described in section 4, BERT Masked Language Model ability allows token prediction. Simply feeding the model with noisy tokens on the input and fine-tuning on canonical token labels transforms BERT into a normalisation model. There are two critical points in doing so successfully. The first is that it requires WordPiece alignment (cf. section 4.2.1). The second is that it requires careful fine-tuning (cf. section 4.2.3).

4.2.1 Wordpiece Alignment

We have in a majority of cases, as described in section 3.2, word level alignment between non canonical and canonical text. Still, the dataset also includes words that are not aligned. For 1-to-N cases we simply remove the spaces. As we work at the WordPiece level this does not bring any issue. For N-to-1 cases (only 10 observations), by considering the special token "|" of the lexnorm15 dataset as any other token, we simply handle source multi-words as a single one, and let the wordpiece tokenization splitting them.

We frame normalization as a 1-to-1 WordPiece token mapping. Based on the word level alignment, we present two methods to get WordPiece alignment : an *Independent Alignment* approach and a *Parallel Alignment* one.

Independent Alignment

We tokenize noisy words and non noisy ones independently (cf. algorithm 1). By doing so, for each word we get non-aligned WordPiece tokens. We handle it in three ways :

- If it is the same number of WordPiece tokens, we keep the alignment as such
- If there are more tokens on the target side, we append the special token [MASK] on the source side. This means that at training time, we force the model to predict a token.
- If there are more tokens on the source side, we introduce a new special token [SPACE].

An alignment example extracted from lexnorm15 can be found in table 2. Briefly, we can point some intuitive pros and cons of such an alignment method. On the one hand, applying tokenization that was used in pre-training BERT means that the sequence of tokens observed during training should be modelled properly by BERT. This should help normalisation. On the other-hand, we understand that learning normalisation in this way requires (as potentially many [MASK] will be introduced) abstracting away from the raw tokens in understanding the surrounding context. This should make normalisation harder. We will see in section 5 that despite its simplicity, such

noisy	canonical
ye	yeah
##a	[SPACE]
im	i
[MASK]	’
[MASK]	m
already	already
knowing	knowing
wa	what
##t	[SPACE]

Table 2: Independent Alignment of *yea im already knowing wat u sayin* normalized as *yeah i’m already knowing what you saying*

an alignment allows our model to reach good performances.

Parallel Alignment

We enhance this first approach with a *parallel alignment* method, described in Algorithm 2.

Our goal is to minimize the number of [MASK] and [SPACE] appended into the source and gold sequences. Therefore, for each word, we start by tokenizing in WordPieces the noisy source word. For each WordPiece met, we start the tokenization on the gold side, starting and ending from the same character positions. As soon as we tokenized the entire gold sub-string, we switch to the next noisy sub-string and so on. By doing so, we ensure a closer alignment at the WordPiece level. We illustrate on the same example this enhanced parallel alignment in Table 3.

We highlight two aspects of our alignment techniques. First, introducing the special token [SPACE] induces an architecture change in the MLM head. We detail this in section 4.2.2-(A). Second, appending the extra token [MASK] on the source side based on the gold sequence induces a discrepancy between training and testing. Indeed, at test time, we do not have the information about whether we need to add an extra token or not. We describe in section 4.2.2-(B) how we extend BERT’s architecture with the addition of an extra classification module to handle this discrepancy.

4.2.2 Architecture Enhancements

(A) Enhancing BERT MLM with [SPACE]

In order to formalize lexical normalisation as a token prediction we introduced in section 4.2.1 the need for a new special token [SPACE]. We want our normalisation model to predict it. We therefore introduce a new label in our output WordPiece vocabulary as well as a new vector in the last softmax layer. We do so in a straightforward way

Algorithm 2: Parallel WordPiece tokenization

```
Vocabulary = Bert WordPiece Vocabulary;
Init start=0; string=canonical word;
string_noisy = noisy word; end_gold=0;
  wordPListNoisy=list(); wordPieceListGold=list();
while string_noisy not empty do
    string_noisy:=string_noisy[start:]
    substr_noisy:=string_noisy
    while substr_noisy not empty do
        breaking:=False
        if substr_noisy in Vocabulary then
            wordPListNoisy :=
            wordPListNoisyU[substr_noisy]
            if start equals length string_noisy then
                end_gold:=length(string)
            else
                end_gold:=
                start+length(substr_noisy)
            end
            while substr_gold not empty do
                substr_gold:=
                string[start:end_gold]
                if substr_gold in Vocabulary then
                    wordPieceListGold:=
                    wordPieceListGold U
                      [substr_gold]
                    break loop
                else
                    end_gold := end_gold -1
                end
            end
        else
            substr_noisy:=substr_noisy[:-1]
        end
        if breaking then
            break loop
        end
    end
    start := start + length(substr_noisy)
end
Result: wordPListNoisy
```

Note : Tokenizing noisy tokens and canonical tokens in wordpieces in parrallel to minimize the number of appended [MASK] and [SPACE]

by appending to the output matrix a vector sampled from a normal distribution[5].

(B) #Next [MASK] predictor

As we have described, alignment requires in some cases the introduction of [MASK] tokens within the source sequence based on the gold sequence. We handle the discrepancy introduced between training and testing in the following way. We add an extra token classification module to BERT architecture. This module takes as input BERT last hidden state of each WordPiece tokens and predict the number of [MASK] to append next

In table 4, we illustrate the training signal of

[5]each dimension $v_d \sim \mathcal{N}(mean_i(x_d), \sigma_i^2(x_d))$ (i indexing the WordPiece vocabulary and d the dense dimension of BERT output layer), $mean_i$ (resp. σ_i^2) means mean (resp. variance) along i dimension

Noisy	Canonical
ye	ye
##a	##ah
im	i
[MASK]	’
[MASK]	m
already	already
knowing	knowing
wa	wh
##t	##at

Table 3: Parallel Alignment of *yea im already knowing wat u sayin* normalized as *yeah i’m already knowing what you saying*

Noisy	Gold	#next mask
ye	ye	0
##a	##ah	0
im	i	2
[MASK]	’	-
[MASK]	m	-
already	already	0
knowing	knowing	0
wa	wh	0
##t	##at	0

Table 4: Parallel Alignment of *yea im already knowing wat u sayin* normalized as *yeah i’m already knowing what you saying* with gold number of next masks for each source token

the overall architecture. It takes noisy WordPiece tokens as input. As gold labels, it takes on the one side the gold WordPiece tokens and on the other side the number of [MASK] to append next to each source WordPiece tokens.

At test time, we first predict the number of next masks to introduce in the noisy sequence. We then predict normalized tokens using the full sequence.

This *#next mask* prediction module exceeds the context of normalisation. Indeed, it provides a straightforward way of performing data augmentation on any Masked Language Model architecture. We leave to future work the investigation of its impact beyond lexical normalisation.

4.2.3 Fine-Tuning

We describe here how we fine-tune our architecture for normalisation. Our goal is to learn lexical normalisation in a general manner. To do so, intuitively, our model needs to: on the one hand, preserve its language model ability that will allow generalization. On the other hand, the MLM needs to adjust itself to learn alignment between noisy tokens and canonical tokens.

Based on those intuitions, we performe fine-tuning in the following way:

(i) Our first approach is to back-propagate on all tokens at each iteration. We also dropout 10% of input tokens by replacing them with the [MASK] as done during BERT pre-training. In this setting, all tokens are considered indifferently whether they require normalisation or not .

(ii) The second approach that happens to perform the best is our *Noise-focus* fine-tuning. The intuition is that it should be much easier for the model to learn to predict already normalized tokens than the ones that require normalization. For this reason, we design the following strategy: For a specific portion of batches noted p_{noise} we only back-propagate through noisy tokens. We found that having an increasing number of noise-specific batch while training provides the best results.

Formally we describe our strategy as follows. For each mini-batch, we sample b following the distribution $b \sim \text{Bernoulli}(p_{noise})$, with $p_{noise} = \min\left(\frac{epoch}{n\_epoch}, 0.5\right)$, $epoch$ being the current number of epoch and n_epoch the total number of epochs.

If b equals 1 we back-propagate through noisy tokens, otherwise we back-propagate in the standard way on all the tokens. In other words, while training, for an increasing portion of batches, we train on tokens that require normalization. We found that this dynamic strategy was much more efficient than applying a static p_{noise}. Moreover, we highlight that the portion of noise specific update is capped at 50% (0.5 in the equation). Above this value, we observed that the performances degraded in predicting non-noisy tokens.

4.2.4 Optimization Details

Note that, excluding the fine-tuning strategy and the alignment algorithm, the optimization hyper-parameters are shared to all the experiments we present next. Generally speaking, we found that optimizing BERT for lexical normalisation with WordPiece alignment is extremely sensitive to hyper-parameters. We managed to reach values that work in all our following experiments. For the optimization, we use the Adam algorithm (Kingma and Ba, 2014). We found that 1e-5 provides the most stable and consistent convergence across all experiments as evaluated on validation set. We found that a mini-batch of dimension 4 brings

the best performance also across all experiments. Finally, we kept a dropout value of 0.1 within the entire BERT model. We train the model for up to 10 epochs and used performance as measured with the F1-score (detailed in the next section) on the validation set as our early-stopping metric.

5 Experiments

All our experiments are run on the lexnorm15 dataset. We do not use any other resources making our problem falling under a low resource domain transfer framework. As only pre-processing, we lower-case all tokens whether they are on the noisy source side or on the canonical side.

We first present our analysis on the validation set that corresponds to the last 450 sentences of the original training set of lexnorm15.

We define the three evaluation metrics on which we make our analysis. We distinguish between *need_norm* words, words that require to be normalized and *need_no_norm* words that have to be "copied" by the model. We refer the words normalized by our model (i.e our model gave a prediction different from the source word) as *pred_need_norm*. We refer to the number of True Prediction of *need_norm* words as *TP*. We then define recall and precision as:

$$recall = \frac{TP}{\#need\_norm}$$

$$precision = \frac{TP}{\#pred\_need\_norm}$$

Following previous works, we will focus on the F1 score as our main evaluation metric. F1 is simply the harmonic mean of the recall and precision. For more fine grained analysis we also report the recall on sub-sample of the evaluated dataset. Particularly, we distinguish between Out-of-Vocabulary (OOV) and In-Vocabulary words (InV) and report the recall on those subsets. We define it formally as:

$$recall\_sample = \frac{TP \cap sample}{\#need\_norm \cap sample}$$

5.1 Alignment algorithm

Does enforcing alignment in a greedy way as described in Algorithm 2 help normalisation ?

As we compare in figure 1, our parallel alignment method provides a +0.5 F1 improvement

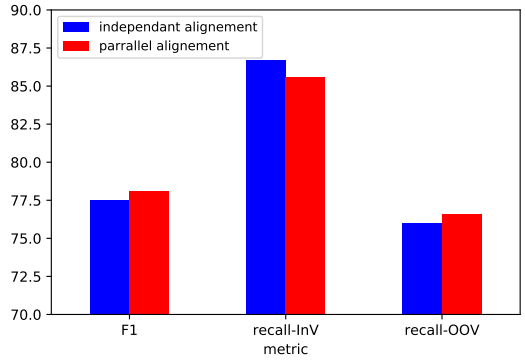

Figure 1: Impact of noisy/canonical alignment method with a focus on generalization by comparing Out-of-Vocabulary (OOV) and In-Vocabulary (InV) performance (development set)

Standard	Noise-focused	Gain
78.1	79.28	+1.18

Table 5: Impact of our noise-specific strategy on the F1 score (development set) reported with best alignment setting

(78.1 vs 77.6 F1). We also compare the performance of our two models on OOV and InV words. Indeed, normalising a seen word is much easier than a word unseen during training. As we observe, the gain coming from our our alignment technique come from a better generalization. We gain +0.6 in recall on OOV thanks to this parallel alignment.

5.2 Fine-Tuning Strategy

As observed in table 5, our fine-tuning strategy focused on noisy tokens improves with a large margin the performance of our system. We interpret it in the following way: lexical normalisation is imbalanced. As seen in 3.2 there are around 9 times more *need_no_norm* than *need_norm* tokens. By specifically training on noisy tokens we successfully manage to alleviate this aspect of the data.

In conclusion, our best model is BERT trained on parallel tokenized data with the noise-focus fine-tuning strategy. We reach 79.28 in F1 score. The following table illustrates how our model performs normalization on a typical example:

Noisy	*@aijaee i hear you u knw betta to cross mine tho*
Norm	*@aijaee i hear you you know better to cross mine though*

Model	Accuracy
BERT noise-focused	97.5
MoNoise	97.6

Table 6: Comparing our systems to the State-of-the-art system MoNoise (we report on same development dataset reported in MoNoise original paper (last 950 sentences))

Model	F1
Supranovich and Patsepnia, 2015	82.72
Berend and Tasnádi, 2015	80.52
our best model	**79.28**
Beckley, 2015	75.71
GIGO	72.64
Ruiz et al., 2014	53.1

Table 7: Comparing our systems to WNUT 2015 shared task that allowed UGC resources

6 Discussion

We now compare our system to previous works.

As we see in Table 8, our non-UGC system is far from the State-of-the-Art model MoNoise (van der Goot and van Noord, 2017) in terms of F1 score. In order to take into account detection in our metric, we also report the overall accuracy of the system in table 6. We are therefore 6.7 points below in terms of F1 score and 0.2 point below in terms of overall accuracy on lexnorm15 dataset.

However, we emphasize that MoNoise is a feature-based Random Forest based on external modules. Among others, it makes use of a skip-gram model trained on 5 millions tweets, the Aspell tool and a n-gram model trained on more than 700 millions tweets.

In order to have a more balanced comparison, we compare our system to the MoNoise model after removing the feature that has the most impact, according to the original paper: the n-gram module (referred as *MoNoise no n-gram*). In this setting, we significantly outperform the MoNoise model (+1.78 improvement) (Table 8).

Moreover, we based all our work on the lexnorm15 dataset released for the W-NUT 2015 shared task (Baldwin et al., 2015). We compare our model to the competing systems (cf. table 7). Briefly, the second best model (Berend and Tasnádi, 2015) use a n-gram model trained on a English tweet corpus. The best competing system (Supranovich and Patsepnia, 2015) is based on a lexicon extracted from tweets. Still, we see that our model is able to outperform models ranked 3, 4 and 5 that are all built using UGC resources.

Model	F1	UGC resources	speed
MoNoise	86.39	lex15+700Mtweets	57s
our best model	**79.28**	lexnorm15	9.5s
MoNoise NNG	77.5	lex15+5Mtweets	-

Table 8: Comparing our systems to the State-of-the-art system MoNoise on lexnorm15 test. Speed is reported as time to predict 1000 tokens (includes model loading). *MoNoise No-Ngrams* or MoNoise NNG is the score reported in the original paper without the use of UGC-n-grams but with a UGC word2vec

Finally, the state-of-the-art models we presented are modular. They require features from external modules. This makes them extremely slow at test time. We compare it in Table 8, demonstrating another practical interest for our approach. Our model is 6 times faster than MoNoise at prediction time.

Following those observations, we claim that BERT, enhanced to handle token introduction and token removal, fine-tuned in a precise way toward noisy words, is a competitive lexical normalisation model.

This result exceeds the context of lexical normalization of noisy User Generated Content. Indeed, the success of BERT in improving NLP models on a diversity of tasks was, until now, restricted to canonical edited texts. In our work, we showed that it was possible to adapt such a general model to the extreme case of normalising noisy UGC in a low resource setting. We let for future work the adaptation of BERT to other tasks in out-of-domain non canonical context.

7 Conclusion

General pre-trained language model have demonstrated their ability to improve Natural Language Processing systems for most tasks on canonical data. In our work, we demonstrated that they can also be useful in non-canonical noisy text in low resource setting. We hope that this work will pave the way for future research in modelling non-canonical textual data.

Acknowledgments

We thank the reviewers for their valuable feedbacks. This work was funded by the ANR projects ParSiTi (ANR-16-CE33-0021), SoSweet (ANR15-CE38-0011-01) and the French-Israeli PHC Maimonide program.

References

Timothy Baldwin, Paul Cook, Marco Lui, Andrew MacKinlay, and Li Wang. 2013. How noisy social media text, how diffrnt social media sources? In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 356–364.

Timothy Baldwin, Marie-Catherine de Marneffe, Bo Han, Young-Bum Kim, Alan Ritter, and Wei Xu. 2015. Shared tasks of the 2015 workshop on noisy user-generated text: Twitter lexical normalization and named entity recognition. In *Proceedings of the Workshop on Noisy User-generated Text*, pages 126–135.

Russell Beckley. 2015. Bekli: A simple approach to twitter text normalization. In *Proceedings of the Workshop on Noisy User-generated Text*, pages 82–86.

Gábor Berend and Ervin Tasnádi. 2015. Uszeged: correction type-sensitive normalization of english tweets using efficiently indexed n-gram statistics. In *Proceedings of the Workshop on Noisy User-generated Text*, pages 120–125.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jacob Eisenstein. 2013. What to do about bad language on the internet. In *HLT-NAACL*, Atlanta, USA.

Jennifer Foster. 2010. "cba to check the spelling": Investigating parser performance on discussion forum posts. In *NAACL*, Los Angeles, California.

Rob van der Goot and Gertjan van Noord. 2017. Monoise: modeling noise using a modular normalization system. *arXiv preprint arXiv:1710.03476*.

Rob van der Goot and Gertjan van Noord. 2018. Modeling input uncertainty in neural network dependency parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4984–4991.

Rob van der Goot, Barbara Plank, and Malvina Nissim. 2017. To normalize, or not to normalize: The impact of normalization on part-of-speech tagging. *arXiv preprint arXiv:1707.05116*.

Soorya Gopalakrishnan, Zhinus Marzi, Upamanyu Madhow, and Ramtin Pedarsani. 2018. Combating adversarial attacks using sparse representations.

Bo Han and Timothy Baldwin. 2011. Lexical normalisation of short text messages: Makn sens a# twitter. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 368–378. Association for Computational Linguistics.

John Hewitt and Christopher D. Manning. 2019. A Structural Probe for Finding Syntax in Word Representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *in proc. of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, Florence, Italy.

Dan Jurafsky. 2018. *Speech & language processing*, 3rd edition. Currently in draft.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Guillaume Lample and Alexis Conneau. 2019. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*.

Chen Li and Yang Liu. 2012. Improving text normalization using character-blocks based models and system combination. *Proceedings of COLING 2012*, pages 1587–1602.

Paul Michel and Graham Neubig. 2018. Mtnt: A testbed for machine translation of noisy text. *arXiv preprint arXiv:1809.00388*.

Seungwhan Moon, Leonardo Neves, and Vitor Carvalho. 2018. Multimodal named entity recognition for short social media posts. *arXiv preprint arXiv:1802.07862*.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Pablo Ruiz, Montse Cuadros, and Thierry Etchegoyhen. 2014. Lexical normalization of spanish tweets with rule-based components and language models. *Procesamiento del Lenguaje Natural*, page 8.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152. IEEE.

Djamé Seddah, Benoît Sagot, Marie Candito, Virginie Mouilleron, and Vanessa Combet. 2012. The French Social Media Bank: a Treebank of Noisy User Generated Content. In *CoLing*, Mumbai, India.

Dmitry Supranovich and Viachaslau Patsepnia. 2015. Ihs_rd: Lexical normalization for english tweets. In *Proceedings of the Workshop on Noisy User-generated Text*, pages 78–81.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Ke Xu, Yunqing Xia, and Chin-Hui Lee. 2015. Tweet normalization with syllables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 920–928.

Yi Yang and Jacob Eisenstein. 2013. A log-linear model for unsupervised text normalization. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 61–72.