CBERT : Let BERT Have Some More Character

Korea University COSE461 Final Project

DongHu Kim
Department of Computer Science
Team 42
2018320120

Abstract

One of the big issues of deep neural networks in general is dealing with adversarial examples. Specifically with BERT language model, using sub-word tokenization method makes the model brittle to character level modifications. CBERT is proposed in focus of alleviating such problem, by accepting separate characters as their input. In our small scale experiment, it is demonstrated that our model is much more robust on random character substitution/insertion/deletion than DistilBERT.

1 Introduction

BERT(Bidirectional Encoder Representations from Transformers) [1] has become a long running trend in natural language processing field, and has proven its effectiveness in various tasks such as question answering [2], machine translation [3], and sentiment analysis [4]. One of the characteristics of BERT is that instead of using separate words as a unit of embedding, BERT has adopted the sub-word tokenization method using WordPiece algorithm [5]. While this is an effective method to give the model improved robustness against unseen examples(OOV), there is a big problem that it cannot retain its strength against character modifications such as mistyping [6]. This could be a serious problem on several tasks like social media generated texts where misspellings commonly occur, or spam-mail filtering where characters are deliberately manipulated to avoid getting filtered.

Sub-word tokenization algorithms try to split each word into sub-word tokens that when gathered will perfectly match the original word. In other words, sub-word tokenization is extremely sensitive to each characters, and a single character change can result in a completely different output. For example, BERT tokenizer tokenizes 'potassium' as a single token, but tokenizes 'potasssium' into four separate tokens 'pot', '##ass', '##si', '##um'. This becomes much harder for tokenizers when trying to recognize character abused words such as 'p0t@ss!um', in which case is tokenized into 'p', '##0', '##t', '@', 'ss', '!', 'um'.

One of the previous methods on dealing with this problem was using GloVe embeddings with character-n gram embeddings on RNN. Although this method was not as powerful as BERT, it was shown to have higher accuracy just after 2 or more character substitutions [6]. In this paper we propose CBERT, a pre-training method inspired by BERT and character-n gram. CBERT's training process is mostly similar to BERT, but uses character representations instead of tokenized sub-words. As a result, CBERT's dictionary only contains single length characters, making it vastly smaller than BERT. Also, it has been shown in our experiments that CBERT is much more robust on character-manipulated inputs.

In this paper, our contributions are the following:

- We propose a character level pretraining method CBERT, which encourages the model to construct a character level understanding of the given corpora.
- We evaluate our method on SST-2 benchmark, and we provide a clear evidence that our method shows highly improved robustness to textual noise compared to baseline model. We use DistilBERT as baseline, which is a compacted version of BERT while retaining most of its decisions. Note that due to lack of computational resources the pre-training and experiments are done in relatively small scale (See Section 4.1).
- The code and pre-trained models are available at `https://github.com/I-AM-PROTO/CBERT`.

2 Related Work

Masked Language Modeling. Masked Language Modeling is one of the two tasks used to pre-train the original BERT. In MLM, several tokens are randomly chosen and replaced with a special token [MASK]. The model's objective is then to predict the masked tokens, based on the remaining tokens. Specifically in BERT, 15% of all tokens are chosen as targets to predict. Of these tokens, 80% are replaced with [MASK], 10% are left unchanged, and 10% are replaced with other random tokens.

Embedding strategy. Early methods such as Word2Vec [7], GloVe [8] embedded texts by separate words, and thus were unable to deal with unseen vocabularies(OOV). BPE(Byte Pair Encoding) [9] and WordPiece [5] were proposed to solve this problem by splitting each word into several tokens. Using this method, OOV's are split into tokens that the model has learnt. However as stated above, this method only enables dealing with OOV's and is still fragile to character modifications. Character language modeling is one effective method to alleviate this problem. This will be further explained below.

Character embedding. Compared to word/sub-word embedding, character embedding strategy is free from OOV problems. Character language modeling have been proven to be an effective training method on multiple NLP tasks such as machine translation [10] and sequence labeling [11]. In addition, character representation method has also been used to construct word representation with character embeddings [12].

3 Approach

In this section, we describe CBERT architecture in further detail, along with its pre-training and fine-tuning process.

3.1 Notations

We denote character embedding as E_{char}. The final hidden layer representation of [CLS] token is denoted as $C \in R^H$ and the final hidden vector for the ith input token as $T_i \in R^H$.

3.2 Model Architecture

We adopt the same architecture from BERT [1], but modify the tokenizer and embedding layer such that the text is embedded character-wise. Instead of sub-word tokenization/embedding using BPE or WordPiece, our model separates the text by characters which are then embedded into independent vectors. Special characters are also considered as independent tokens, including spaces.

3.3 Character Masked Language Modeling

Modifying MLM, we propose and adopt an unsupervised pre-training task that masks out random characters and predicts the original characters based on remaining information. Instead of BERT's strategy where it is guaranteed that 15% of tokens are targeted, we use a simplified version–primarily for computational efficiency. For each character independently, there is a 12% chance of being replaced with [MASK] token, 1.5% chance of being replaced with any random character, and 1.5%

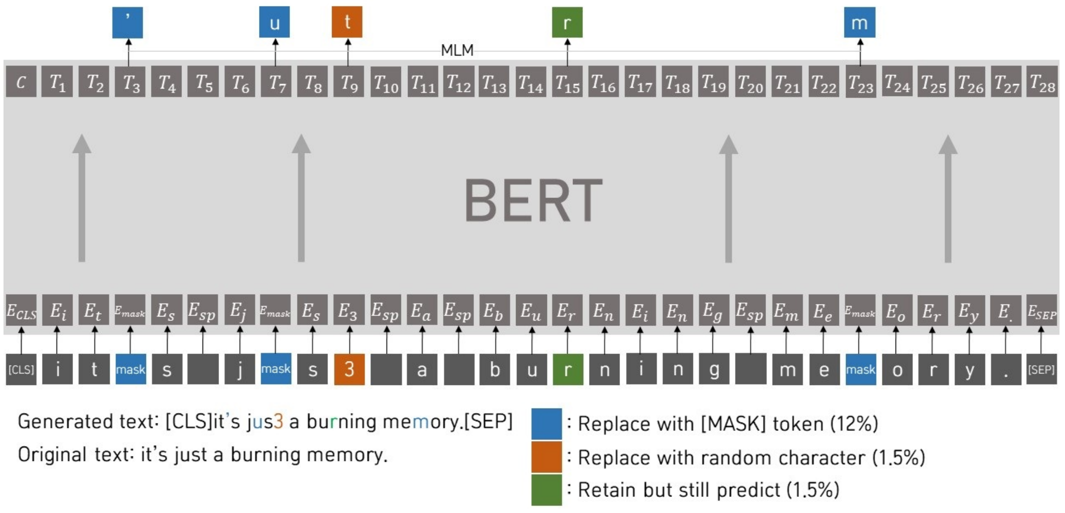
MLM
BERT
Generated text: [CLS]it's jus3 a burning memory.[SEP]
Original text: it's just a burning memory.
: Replace with [MASK] token (12%)
: Replace with random character (1.5%)
: Retain but still predict (1.5%)

Figure 1: A pre-training example of CBERT. All alphabetical characters are uncased. Note that every single token is a candidate, including [CLS] and [SEP].

chance of retaining its character but still targeted for prediction. Otherwise, the character is not predicted by the model. (See figure 1) Note that the probabilities don't overlap, and one and only one of the four operations will be performed on each character.

3.4 Fine-tuning

For token-level tasks like question-answering we use C as an input to the classifier. For sentence classification tasks such as sentiment analysis, we have several choices. We could use C as an input, or average over all final representations C and T_i. Our experiments have found little to zero difference between the two methods, but using C was chosen for a slightly more efficient computation.

4 Experiments

In this section, we describe the experiment settings, their results, and our analysis. Some limitations are first listed in Section 4.1.

4.1 Limited Environment

Due to limited resources including time, some possibly counter-effective choices were made:

- As explained in Section 3.3, our data generation for masked language modeling does not guarantee 15% of tokens as targets. However, we doubt that this was a hinderance to the model's traning, because the percentage of modified characters did not deviate far from 15% on average.
- Unlike BERT, Next sentence prediction(NSP) was not used to pre-train CBERT. Every input contained a single sentence that starts with [CLS] and ends with [SEP]. Note that [SEP] token was still used to indicate the end of each sentence. Embedding-wise, segment information was not used, leaving only character and positional information.
- The dataset used for pre-training was Yelp Review dataset. While there are many other datasets that are much larger are better suited for pre-training, this dataset is one of the datasets with shorter sentences and thus was thought to be more suitable for our environment. The scale of our experiment is also noticeably small: only a subset of the chosen dataset was used for actual training. For precise description, see Section 4.2.

4.2 Datasets and Baseline Models

Yelp Review Dataset. Yelp Review consists of human generated review texts that were uploaded to Yelp. Most sentences are no longer than 256 letters, so very few sentences are lost when we limit the maximum length to 256. We sliced the first 15K reviews from the train dataset and generated approximately 140K sentences, and used them for MLM tasks to pretrain our model. For validation, we used the next 1000 reviews to generate around 10K sentences.

Stanford Sentiment Treebank(SST-2). Stanford Sentiment Treebank [13] is a collection of 11,855 single sentences extracted from movie reviews, each labeled with its sentiment in 5 classes from very negative to very positive. The objective of SST-2 is to make a binary prediction of whether the given sentence is positive or negative. Neutral sentences were excluded from our evaluation. Accented characters such as è, ñ were manually converted into regular characters. For our baseline, we use a fine-tuned DistilBERT model from HuggingFace [14].

4.3 Evaluation method

Prior to evaluation, we generate adversarial examples to test our model's robustness to character manipulation (See Algorithm 1). We implement the algorithm that gives some randomness to how much characters will be taken by each operation. Deletion, replacement, insertion are done in said order such that none of the operations affect the previous, and there are always K guaranteed modified characters for all sentences.

Algorithm 1 Random adversarial example generation

```
**Input**: text, K
**Output**: text with K characters replaced, inserted, or removed
$R_{replace}, R_{insert}, R_{remove}$ = Random(0,1)
$N_{replace} = \lfloor K * R_{replace}/(R_{replace} + R_{insert} + R_{remove}) \rfloor$
$N_{insert} = \lfloor K * R_{insert}/(R_{replace} + R_{insert} + R_{remove}) \rfloor$
$N_{remove} = K - N_{replace} - N_{insert}$
Remove random $R_{remove}$ characters from text
Replace random $R_{replace}$ characters from text with random character
Insert random $R_{insert}$ characters into text in random position
```

When evaluating a model, we go through $K = 0, \cdots, 10$. For each K, we evaluate the metric of the given dataset with 10 predetermined RNG seeds and output its average as a result. The RNG seeds are global across all models and all datasets, so that each model is given the exact same inputs. For SST-2, we compute the average of prediction accuracy.

4.4 Experimental Details

The structure of our experimented model is mostly identical to the original BERT. Some specifications of our experimented model are the following:

- number of characters in vocabulary: 71
- character embedding size: 768
- feedforward dimension size: 2048
- sentence length limit:
- number of attention heads: 12
- number of layers: 6
- total parameters: 33M

Our pre-training strategy is given by the following:

- initialization: Xavier uniform for all weights
- learning rate: 5e-5
- total epochs: 235 (checkpoint every 5 epochs)

K	CBERT		DistilBERT(baseline)	
	val	test	val	test
0	74.0	73.2	91.1	92.3
1	72.7(-1.3)	73.1(-0.1)	89.0(-2.1)	90.5(-1.8)
2	72.9(-1.1)	71.8(-1.4)	86.7(-4.4)	88.5(-3.8)
3	71.7(-2.3)	71.3(-1.9)	84.8(-6.3)	85.7(-6.6)
4	71.8(-2.2)	71.0(-2.2)	83.0(-8.1)	83.4(-8.9)
5	71.2(-2.8)	70.6(-2.6)	80.7(-10.4)	81.1(-11.2)
6	70.8(-3.2)	69.6(-3.6)	78.2(-12.9)	78.4(-13.9)
7	71.1(-2.9)	68.8(-4.4)	75.7(-15.4)	76.2(-16.1)
8	69.4(-3.6)	68.4(-4.8)	73.5(-17.6)	73.5(-18.8)
9	69.4(-3.6)	68.1(-5.1)	71.1(-20.0)	72.0(-20.3)
10	69.1(-3.9)	67.7(-5.5)	69.6(-21.5)	69.5(-22.8)

Table 1: Accuracy result on validation/test set of SST-2 dataset. All results are performed and averaged over the same 10 RNG seeds. Colored numbers indicate the loss of accuracy from $K = 0$. Color schemes are: $0.0 \leq$ pink $< 3.0 \leq$ orange $< 6.0 \leq$ red $< 9.0 \leq$ purple $< 14.0 \leq$ violet

- loss function: NLL loss
- optimizer: AdamW (no weight decay)

Finally, our fine-tuning settings are the following:

- classifier: single linear layer
- classifier initialization: Xavier uniform for weights, zeros for biases
- learning rate: 5e-5
- total epochs: 10-15
- loss function: Cross Entropy Loss
- optimizer: AdamW (no weight decay)

4.5 Results

At pre-training stage, our model was able to reach the performance of original BERT in terms of accuracy, reaching 87.8% on train set and 89.0% on validation set. We believe this was possible with just 235 epochs only because the dataset itself was small enough. It is surprising however, that this was not overfitting at any rate, as we can see on the high validation accuracy.

The results on SST-2 are provided in Table 1. As expected, CBERT's baseline performance($K = 0$) was not very competitive. However, it is evident that CBERT is vastly more resilient to character manipulation than DistilBERT. When more than 3 characters are modified, DistilBERT's accuracy suffers more than four times the amount of loss compared to our model. Even though CBERT performs poorly on low K, it retains its performance on high K values to the point where both DistilBERT and CBERT have little difference in accuracy.

We suspect that CBERT's low accuracy in general is caused by the limited pre-training dataset. On the other hand, the resilience of CBERT to our character adversarial examples clearly shows that character level embedding is a powerful method to deal with such adversaries. We believe a more refined model trained with larger corpora would perform far better than what we have achieved.

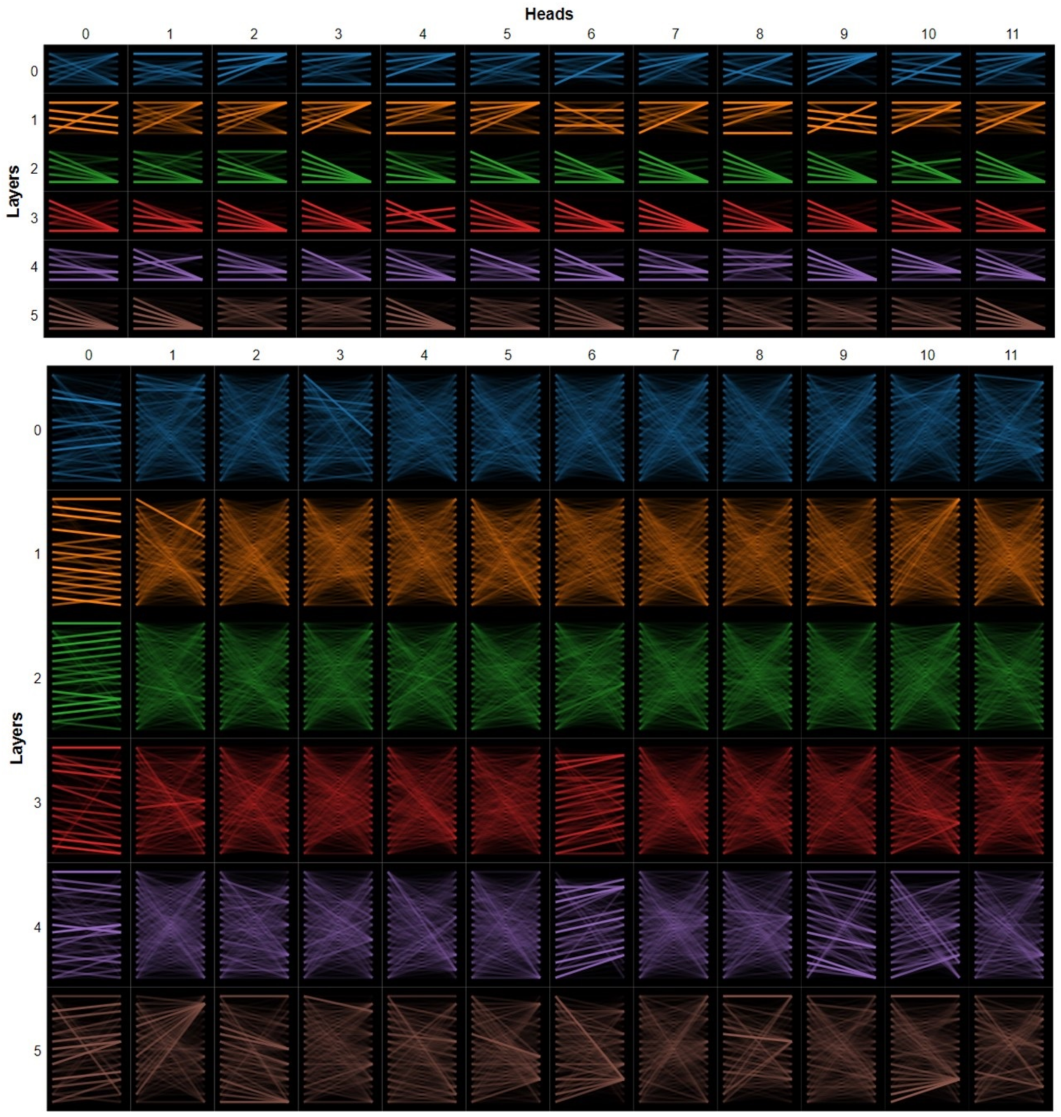

Figure 2: Attention visualization of DistilBERT(above) and CBERT(below) on text "cats are cute". In DistilBERT, the text is tokenized into [[CLS], cats, are, cute, [SEP]]. Both have successfully classified the text's sentiment as positive. DistilBERT has organized focus of attention to a specific token on each layer and head. CBERT on the other hand, shows distributed attention except for some heads like 0.

		CBERT	
		success	fail
BERT	success	cats are cute a boring movie	cts are cute -
	fail	cats are culte a borin movie	cats are cte a ;oring movie

Table 2: Success and failure examples for two models, on sentiment analysis task. We generated adversarial examples with only one manipulation, which would realistically happen in real life situations.

5 Analysis

We first visualize and compare how each sentence is processed in DistilBERT and CBERT. Shown on Figure 2 is a visualized self-attention of each model on different layers and attention heads. CBERT has mostly distributed and random attention except for some heads in some layers. This is very distinctive to DistilBERT where most attention weights are focused on certain tokens such as [CLS] and [SEP].

We also investigate several successful/unsuccessful sentiment analysis examples on the two models. From our base sentence (e.g."cats are cute"), we try to make false-negative or false-positive examples by modifying a random character, and observe when the model makes wrong predictions. For DistilBERT, 'cute' is the only token that indicates positiveness. Manipulating the model to falsely tokenize such word to meaningless tokens(e.g. 'cte' tokenized to 'ct', '##e') or even opposite-sentimental tokens(e.g. 'culte' tokenized to 'cult', ##e') would easily confuse the model. CBERT on the other hand, has seemingly no meaningful pattern or traits to be investigated. However, it is worth noting that finding false examples for CBERT took up to 30 attempts, whereas DistilBERT didn't take more than 3.

We have analysed that CBERT has no meaningful attention connection, and that there is no clear explanation to when the model makes correct classification and when otherwise. We interpret our findings in two ways. This could be another indication that our model was not trained properly for more general corpora, hence the messiness of attention and the lack of interpretability. This could also mean that using only character representation makes it difficult to make sense of a whole sentence, in which case means that combining with other methods like original BERT could be a great way to improve its performance.

6 Conclusion

In this paper, we address an important weakness of BERT: character sensitivity caused by sub-word tokenization method. We proposed a new pre-trained model CBERT with new tokenization algorithm, that tokenizes each characters rather than sub-words. We then applied a slightly modified version of MLM task for character level language modeling. Our small scale experiments prove that CBERT and character level embedding can improve robustness against character adversaries.

Because we believe our model was not sufficient enough to show CBERT's full potential, further research could start from constructing a more refined model, pre-trained with larger, cleaner, and generally more fitting dataset. CBERT could also be fine-tuned/tested on various other tasks such as question answering and token classification. In a larger picture, CBERT could extend to other domains such as multilingual model or audio extraction, where a continuous sound waves could be split into syllables. Finally, our proposed character level embedding could be combined with other powerful methods such as original BERT to create a much more powerful language model.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[2] Zhiguo Wang, Patrick Ng, Xiaofei Ma, Ramesh Nallapati, and Bing Xiang. Multi-passage bert: A globally normalized bert model for open-domain question answering, 2019.

[3] Haoran Xu, Benjamin Van Durme, and Kenton Murray. Bert, mbert, or bibert? a study on contextualized embeddings for neural machine translation. 2021.

[4] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. SMART: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020.

[5] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *International Conference on Acoustics, Speech and Signal Processing*, pages 5149–5152, 2012.

[6] Lichao Sun, Kazuma Hashimoto, Wenpeng Yin, Akari Asai, Jia Li, Philip Yu, and Caiming Xiong. Adv-bert: Bert is not robust on misspellings! generating nature adversarial samples on bert, 2020.

[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[8] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[9] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units, 2015.

[10] Jason Lee, Kyunghyun Cho, and Thomas Hofmann. Fully character-level neural machine translation without explicit segmentation, 2016.

[11] Gözde Gül Şahin and Mark Steedman. Character-level models versus morphology in semantic role labeling, 2018.

[12] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

[13] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher Manning, Andrew Ng, and Christopher Potts. Parsing With Compositional Vector Grammars. In *EMNLP*. 2013.

[14] distilbert-base-uncased-finetuned-sst-2-english. `https://huggingface.co/distilbert-base-uncased-finetuned-sst-2-english`. Accessed: 2022-05-22.