COMPRESSING ENCODING OF WORDS FOR USE IN CHARACTER-LEVEL CONVOLUTIONAL NETWORKS FOR TEXT CLASSIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper reports the empirical exploration of a novel approach to encode words using compressing inspired techniques for use on convolutional neural networks at character-level. This approach reduces drastically the number of parameters to be optimized using deep learning, resulting in competitive accuracies in only a fraction of the time spent by characters level convolutional neural networks, enabling training in simpler hardware.

1 INTRODUCTION

Classification of documents, that is, assigning a document or extract of a text to a category based on their content is a daily need of modern life. Everyday we manipulate digital documents organizing them into folders, seeking to group characteristics, ideas and subjects for future references. Such organization makes possible to relate other people's thoughts and experiences for feed the development of our own ideas. With the increasing availability of texts, especially through the internet, the need for artificial intelligence tools for automate this function is increasing very fast. Spam detectors, sentiment analysis, news archiving are all technologies based on text classificators.

As a basic task in Natural Language Processing, the literature on text classifiers presents an extensive range of approaches to its realization (Aggarwal & Zhai, 2012; Hotho et al., 2005; Kosala & Blockeel, 2000) . Many are based on the discrimination of the word as an atom of the text structure and the model works through statistics of words occurrence (Zhang & LeCun, 2015). The great variability of words and structures belonging to a language makes this task quite difficult and such models have superior quality mainly in very specific domains, where the vocabulary can be restricted to a much smaller number of words, possibly chosen by a specialist. Moreover, such modeling becomes specific to a language, causing the replication process in another language to be carried out from scratch (Zhang & LeCun, 2015).

With the advent of Deep Learning methodologies (Goodfellow et al., 2016), some techniques have been updated but the clear majority are still based on the tokenization of words and the inference of their statistics. LSTM (Hochreiter & Schmidhuber, 1997) and Word2Vec (Mikolov et al., 2013) are some of the most popular strategies.

The development of convolution techniques (LeCun et al., 1998) coupled with the popularization of parallel computing libraries for using Graphical Processing Units- GPUs (Bergstra et al., 2010; Chollet et al., 2015) has been quite successful in image classification (Krizhevsky et al., 2012). The replication of this success in text classification suffers, among other reasons, by the difficulty of representing one word as a vector. Using the one hot encoding technique, where the vector of the term position is 1 and all other 0, would generate a representation of thousands or even millions of coordinates, making its use impractical both in terms of memory needed to store this vector, as time required to compute its optimizations. Another problem is that typing errors or simple declensions of words in test data could not be codified if they are unseen on train data.

In an audacious paper, Zhang & LeCun (2015) suggest an approach that consider the character as the atomic level, reducing the vocabulary of symbols to only 69 characters and allowing the use of one hot encoding. They considered each of the 1014 characters of a text represented by the line of a matrix where the column referring to the character of the vocabulary is 1 and all the others 0.

With this representation on hand, they applied a model with deep convolutional neural networks and obtained results competitive with other techniques, and in some cases the state of the art.

Building upon the work of Zhang & LeCun (2015), we suggest a novel approach to word representation that encode a word as a sequence of chars in a vector with size order of some hundreds, using the knowledge of character frequency imbalance, and inspired by traditional compression techniques. This procedure made possible to represent larger portions of texts, perform the training in shorter times and applying simpler hardware, preserving the ability to codify any word, even unseen ones.

2 OUR APPROACH

2.1 ENCODING PROCEDURE

In all of ours experiments, we used the following procedure to encode words:

- *Obtaining a character frequency rank*: We read the text database and count the frequency of each character, generating a list sorted by frequency of occurrence. Then we create a rank with only the relative positions of the characters. For a given language, this rank is quite stable since only the order of the rank is used. This means that if all documents are in the same idiom, this procedure can be replaced by a list with the characters rank of frequency for that language.
- *Creating a mapping from character to compress code*: To encode each character, we assign the value 1 to begin and end each char code. Between these values, we insert the digit 0 in the same amount of the rank position of the character. Table 1 have some examples of encoded characters.

Table 1: Example of coding using English language ranking of characters.

CHARACTER	RANK	COMPRESS CODE
space	0	11
'e'	1	101
'a'	2	1001
't'	3	10001
'i'	4	100001
's'	5	1000001
'n'	7	100000001
'r'	8	1000000001
'd'	10	100000000001
'h'	11	1000000000001
'c'	12	10000000000001
'g'	17	1000000000000000001
	N	'1'+N*'0'+'1'

To encode each word, we just concatenate the codes of each character. As an example, we provide some codified words:

- science:
  10000011000000000000011000011011000000011000000000000001101
- scientist:
  1000001100000000000001100001101100000001100011000011000000110001
- art:
  1001100000000110001
- artist:
  100110000000011000110000110000110001

- tl;dr:
  1000110000000001100000000000000000000000000000011000000000011000000001

- u2:
  10000000000000011000000000000000000000000000001

- e^a (eˆa)
  10110000000000000000000000000000000000000000000000000000000011001

Given a document, we consider that it is composed of words, being those any set of characters limited by the space character. This means "word" could be equations, web site address, latex code, computer commands, etc. In the matrix representation of the document, each row represents a properly encoded word.

The code for each word is then embedded on a matrix of $number\ of\ words \times code\ size$ with its first symbol in column 1. Columns that were not occupied are padded with 0, larger codes are represented up to the chosen limit. Unoccupied lines are filled with 0 and larger documents are represented only up to the chosen maximum number of words, ignoring the last remaining ones. As an example, we represent a document in an 8x65 matrix in Table 2:

Table 2: Matrix encoding of 'Research is an art and a science'

1000000001101100000110110011000000001100000000000011000000000010
1000011000001000000000000000000000000000000000000000000000000000
1001100000001000000000000000000000000000000000000000000000000000
1001100000000110001000000000000000000000000000000000000000000000
1001100000001100000000001000000000000000000000000000000000000000
1001000000000000000000000000000000000000000000000000000000000000
1000001100000000000011000011011000000011000000000000011010000000
0000000000000000000000000000000000000000000000000000000000000000

We use 8x65 (520 elements) to encode the text with a certain amount of slack. At the very least, we would need 7x64 (448 elements). The approach of Zhang & LeCun (2015) would employ at least 69x32 (2208) elements to represent the same sentence.

In our experiments, 256 coordinates were enough to represent 99.5 of the words from one of the databases studied. In all datasets studied in this paper, we choose 128 as a limit of words to represent a document, encoding each text in a 128x256 matrix.

2.2 Training Environment and Network architecture

For all the trainings, we used the environment and parameters settings listed on Table 3:

Table 3: Training environment and parameters

DESCRIPTION	PARAMETERS	OBSERVATION
Neural Net Lib.	Keras 2.0	
Tensor Backend	Theano 0.9	
GPU Interface	Cuda 8	with cuBLAS Patch Update
CNN optimizer	Nvidia Cudnn 5.1	
Program. Lang.	Python 3.6	using Anaconda 4.4.0
Superbatch	10000	Number of matrixes sent to gpu each time
Minibatch	32	Batch to update the network weights
Optimizer	ADAM (Zeiler, 2012)	$lr = 10^{-3},\ \beta_1 = 0.9,\ \beta_2 = 0.999, \epsilon = 10^{-8}$
Epochs	5	
Op. System	Windows 10	
GPU	Nvidia GeForce 1080ti	Nvidia GeForce 930M for time comparison
RAM Memory	16 GB	

We choose a very simple network architecture described in Table 4. Our main concern was to be fast enough to execute tests in all datasets applied by Zhang & LeCun (2015) approach, enabling an unbiased comparison.

Table 4: Network architecture.

LAYER TYPE	SHAPE	# PARAM	CONNECTED TO
input_1 (Input)	(None, 128, 256)		
conv1d_1 (3)	(None, 126, 256)	196864	input_1
conv1d_2 (3)	(None, 126, 256)	196864	input_1
conv1d_3 (3)	(None, 126, 256)	196864	input_1
conv1d_4 (3)	(None, 126, 256)	196864	input_1
max_pooling1d_1 (3)	(None, 42, 256)	0	conv1d_1
max_pooling1d_2 (3)	(None, 42, 256)	0	conv1d_2
max_pooling1d_3 (3)	(None, 42, 256)	0	conv1d_3
max_pooling1d_4 (3)	(None, 42, 256)	0	conv1d_4
merge_1 (Merge)	(None, 168, 256)	0	max_pooling1d_1
”	”	”	max_pooling1d_2
”	”	”	max_pooling1d_3
”	”	”	max_pooling1d_4
conv1d_5 (5)	(None, 164, 256)	327936	merge_1
max_pooling1d_5 (3)	(None, 54, 256)	0	conv1d_5
conv1d_6 (5)	(None, 50, 256)	327936	max_pooling1d_5
max_pooling1d_6 (4)	(None, 12, 256)	0	conv1d_6
flatten_1	(None, 3072)	0	max_pooling1d_6
dense_1 (128)	(None, 128)	393344	flatten_1
dense_2 (Output)	(None, #classes)	(128+1)*#classes	dense_1

To perform the training of this architecture with an output of 4 classes, it is necessary to optimize 1,837,188 parameters. As a comparison, in the architecture suggested by Zhang and LeCun it is necessary to optimize 11,337,988 parameters (Zhang et al., 2015).

3 DATASETS AND RESULTS

The databases used were the same as those cited in an article by Zhang et al. (2015), where there is an extensive description of them. In this article, we will only summarize the main descriptions on Table 5.

Table 5: Datasets Description and Time per Epoch on differents NVIDIA GPUs

DATASET	#CLASS	# TRAIN	# TEST	# AVG WORDS	1080TI	930M
ag_news	4	120k	7.6k	45	3 min	25 min
sogou_news	5	450k	60k	578	23 min	93 min
dbpedia	14	560k	70k	55	18 min	116 min
yelp_polarity	2	560k	38k	153	21 min	119 min
yelp_full	5	650k	50k	155	27 min	135 min
yahoo_answers	10	1,400k	60k	112	47 min	290 min
amazon_full	5	3,000k	650k	93	120 min	620 min
amazon_polarity	2	3,600k	400k	91	133 min	744 min

One note is worth recall: the sogou_news dataset is originally in Chinese. Zhang & LeCun (2015) used software packages to produce transcriptions in Pinyin - a Chinese phonetic romanization.

We do not use any preprocessing strategy except the use of lowercase letters. No data enhancement technique was employed.

The comparison results obtained are summarized in the Table 6.

Table 6: Loss Comparison among traditional models, Zhang et al. (2015) and our approach. Bow-Bag of Words, lg.-large, sm-small.

MODEL	AG	SOGOU	DBP	YLP P	YLP F	YAH	AMZ F	AMZ P
Bow	11.19	7.15	3.39	7.76	42.01	31.11	45.36	9.60
Bow TFIDF	10.36	6.55	2.63	6.34	40.14	28.96	44.74	9.00
Ngrams	7.96	2.92	1.37	4.36	43.74	31.53	45.73	7.98
Ngrams TFIDF	7.64	2.81	1.31	4.56	45.20	31.49	47.56	8.46
Bag-of-Means	16.91	10.79	9.55	12.67	47.46	39.45	55.87	18.39
LSTM	13.94	4.82	1.45	5.26	41.83	29.16	40.57	6.10
Lg Conv Zhang	12.82	4.88	1.73	5.89	39.62	29.55	41.31	5.51
Sm Conv Zhang	15.65	8.65	1.98	6.53	40.84	29.84	40.53	5.50
Ours	12.33	4.84	2.07	7.96	42.00	31.90	41.91	6.31

4 DISCUSSION

The main objective of this research was to evaluate the possibility of using a coding approach that contemplated the construction of words using characters as basic tokens. Our main contribution is demonstrate that such approach allow reduce the dimensionality of the encoding matrix, allowing substantial shorter optimization times and the use of devices with lower computational power.

The effort to use Neural Convolution Networks for text classification tasks is justified by the possibility of appropriating tools from the recent developments of techniques, libraries and hardware used especially in the image classification (LeCun et al., 1998; Bergstra et al., 2010; Chollet et al., 2015; Krizhevsky et al., 2012). Such technologies can generate more general algorithms, facilitating the use of the same approach for several applications, independent of database language.

Establishing the character as the basic unit of word formation provides a better opportunity to be robust to typos, acceptance of neologisms, equations and chemical formulas, abbreviations and idiosyncrasies of written language on the internet, such as emoticons, slang and dialects of the online world. Assuming the word as a base item, much of this ability is lost, especially when models assumes that text producers uses a formal language.

The article by Zhang & LeCun (2015) was a great innovation in this regard. The results obtained by them suggest that language can be treated as a sequence of signals, like any other (Zhang & LeCun, 2015). However, the training times and the dimensionality of the matrices to be computed are still obstacles to the most effective use of the technique, especially by startups that could generate new products and ideas based on this technology. Zhang & LeCun (2015), report that training of Ag News database take 1-3 hours (60-180 min) per epoch, Dbpedia database lasted 5 hours (300 min) per epoch and Amazon 2-5 days (2880-7200 min) per epoch on an Nvidia K40 GPU. As a comparison, our trainings take 3 min, 18 min and 133 min per epoch, respectively on a NVIDIA Geforce 1080ti and 25 min, 116 min and 744 min per epoch respectively on NVIDIA 930M 4GB, a much inferior GPU. The improvement of performance is more impressive if we consider that our results were obtained in 5 epochs and Zhang & LeCun (2015) on 50 epochs!

Large portion of this time is still used to encode and deliver matrices to the GPUs. To calculate the weights in each superbatch of 10000 arrays, a Geforce 1080ti GPU spends only 6 seconds. In a more modest notebook with 16 Gb of RAM and a Nvidia Geforce 930M card, the optimization time for these same arrays is about 100 seconds, a reasonable timeframe even to motivate students and small startups who do not have access to a GPU as modern as GeForce 1080ti to use this approach.

The major bottleneck for analyzing this gigantic amount of matrixed text is the need for intensive use of RAM. Our approach generates a 128x256 matrix, smaller than 1014x69 generated by Zhang & LeCun (2015), but a large set of them quickly occupy the available memory on the computer. On the machine we used, there were 16 GB available, which is not uncommon in modern personal computers. Therefore, the use of generators to control the number of matrices generated and sent to GPU is an important detail in the implementation of this optimization algorithm. If your computer has only 8 GB of RAM or less, you should reduce the number of superbatch to fit the memory.

The encoding used in this article was inspired by the Tagged Huffman Code, developed by Silva de Moura et al. (2000), differing only that in the original conception, 2 digits 0 is added for each position in the rank of the characters. As the goal was to use the smallest possible code to represent words in a vector of the document matrix, we chose to use only one 0 digit.

As in the Silva de Moura et al. (2000) approach, the coding we employ has the following advantages (Brisaboa et al., 2003):

1. No character code is a prefix of another, that is, the match is one-to-one.
2. It allows direct search, that is, to search for a word in a codified document, just encode the word and use traditional methods of comparing strings with the encoded document.

This encoding approach is a compression technique, so it also allows saving already encoded text documents permanently using a binary system, requiring less storage space. These advantages become attractive especially if the goal is to extract knowledge about files in a repository to perform various classifications on different dimensions with the same files.

A possible advantage of this encoding is to better respond to unseen words on training data, once that the network at least have some prefix to guess the meaning of the word, the same way we humans do. This is especially interesting in languages where there is a lot of declensions like portuguese, spanish, italian, french, russian, german, arabic and turkish, for example.

The coding procedure is not restricted to any size of vocabulary, the only problem is that less frequent characteres will generate bigger codes. If your database have a lot of this characters, you could use a higher word code size.

The results obtained strongly indicate that the use of this coding is a possibility. We emphasize that little effort was made to obtain a better network architecture. The one we use is a fairly simple network, enough to demonstrate the feasibility of the approach with the time and computational resources that we have. Possibly there are several architectures that can generate better results.

In addition, the dimensionality reduction provided enables several architectures to be verified in a reasonable time.

5 CONCLUSION

The main conclusion is that using a compressing technique is a very convenient possibility to represent words for use in Convolutional Neural Networks to classify text. Codifying words using character could be relevant specially on less curated texts datasets, being robust to typos, slangs and others usual characteristics of internet texts. With our results, we reinforce Zhang et al. (2015) conclusions, which states that language could be treated as a signal, no different of any other. We demonstrated that using the approach suggested reduce the dimensionality of matrix representation of texts and allow the use of simpler hardware than character level one hot encoding, allowing a drastic reduction of training time. We performed a comparative test with the approach of Zhang et al. (2015) and others traditional techniques and found competitive results. We speculate that new efforts should best employed in obtaining others architectures that may further favor this strategy.

ACKNOWLEDGMENTS

We gratefully acknowledge the support of NVIDIA Corporation with the donation of a GPU used for this research.

The first author is grateful to the CAPES for the award of a research scholarship.

REFERENCES

CharuC. Aggarwal and ChengXiang Zhai. A survey of text classification algorithms. In Charu C. Aggarwal and ChengXiang Zhai (eds.), *Mining Text Data*, pp. 163–222. Springer US, 2012. ISBN 978-1-4614-3222-7. doi: $10.1007/978-1-4614-3223-4_6$. URL http://dx.doi.org/10.1007/978-1-4614-3223-4_6.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

Nieves Brisaboa, Eva Iglesias, Gonzalo Navarro, and José Paramá. An efficient compression code for text databases. *Advances in Information Retrieval*, pp. 78–78, 2003.

François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Andreas Hotho, Andreas Nrnberger, and Gerhard Paa. A brief survey of text mining. *LDV Forum - GLDV Journal for Computational Linguistics and Language Technology*, 20(1):19–62, May 2005. ISSN 0175-1336. URL http://www.kde.cs.uni-kassel.de/hotho/pub/2005/hotho05TextMining.pdf.

Raymond Kosala and Hendrik Blockeel. Web mining research: a survey. *SIGKDD Explor. Newsl.*, 2(1):1–15, June 2000. doi: http://dx.doi.org/10.1145/360402.360406. URL http://dx.doi.org/10.1145/360402.360406.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1097–1105, 2012.

Yann LeCun, Lon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pp. 2278–2324, 1998.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 3111–3119, 2013.

Edleno Silva de Moura, Gonzalo Navarro, Nivio Ziviani, and Ricardo Baeza-Yates. Fast and flexible word searching on compressed text. *ACM Trans. Inf. Syst.*, 18(2):113–139, April 2000. ISSN 1046-8188. doi: 10.1145/348751.348754. URL http://doi.acm.org/10.1145/348751.348754.

Matthew D. Zeiler. Adadelta: An adaptive learning rate method. *CoRR*, abs/1212.5701, 2012. URL http://dblp.uni-trier.de/db/journals/corr/corr1212.html#abs-1212-5701.

Xiang Zhang and Yann LeCun. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*, 2015.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pp. 649–657, 2015.