COMPACT ENCODING OF WORDS FOR EFFICIENT CHARACTER-LEVEL CONVOLUTIONAL NEURAL NETWORK 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 classification accuracy values in only a fraction of the time spent by characters level convolutional neural networks, enabling training in simpler hardware.

1 INTRODUCTION

Documents classification is one of the key tasks addressed in the context of natural language processing (Sebastiani, 2002). It implies associating a document —or a text fragment, for that matter— to a category or label relying on their content. This is an important and daily need of modern life. Everyday, we manipulate 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 classifiers.

Work on document classification encompasses a broad range of approaches to (see (Sebastiani, 2002; Aggarwal & Zhai, 2012; Hotho et al., 2005; Kosala & Blockeel, 2000)). An important portion of those approaches rely on a representation that handles words as the atomic element of text. Consequently, those models carry out their analysis through statistics of words occurrence (Zhang & LeCun, 2015). However, the variability of words and structures belonging to a language makes this task difficult. That is why, such models have superior quality mainly in specific domains, where the vocabulary can be restricted to a relatively small number of words, possibly chosen by a specialist. Furthermore, such modeling becomes specific to a language, causing the replication process in another language to be carried out from scratch (Zhang & LeCun, 2015).

In recent years, we have experienced a revolution in the machine learning with the advent of deep learning methodologies (Goodfellow et al., 2016). The development of convolutional neural networks (CNNs) (LeCun et al., 1998) coupled with the popularization of parallel computing libraries (e. g. Theano (Bergstra et al., 2010), Tensorflow (Abadi et al., 2015), Keras (Chollet et al., 2015), etc.) that simplify general-purpose computing on graphics processing units (GPGPU) (Mittal & Vetter, 2015) has been successful in tackling image classification problem (Krizhevsky et al., 2012) quickly becoming the state of the art of the field.

It was therefore natural that the deep learning principles were to be extended to the document classification domain. Some existing methods have been updated but the clear majority are still based on the tokenization of words and the inference of their statistics. Bag of Words (BOW) (Johnson & Zhang, 2014) and word2vec (Mikolov et al., 2013) are some of the most popular strategies.

The replication of image classification success in the documents domain suffers, among other reasons, by the difficulty of representing one word as a vector. Using the one hot encoding technique (Harris & Harris, 2012), 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 optimization. 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 groundbreaking paper, Zhang & LeCun (2015) suggested 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 PRELIMINARIES

The success in using Convolutional Neural Networks (CNNs) (LeCun et al., 1998) in image classification (Krizhevsky et al., 2012) flourish the development of many libraries (Chetlur et al., 2014; Chollet et al., 2015; Bergstra et al., 2010) , techniques and hardware. The effort to use Neural Convolution Networks for text classification tasks is justified by the possibility of appropriating these tools for obtaining better results and more robust algorithms, facilitating the use of the same approach for several applications.

There are two usual approaches to use CNN with text: Bag of Words (BOW) (Johnson & Zhang, 2014) and word2vec (Mikolov et al., 2013).

In Bag of Words and some of its variants, for representing each word in a vocabulary of size n, a digit 1 is placed in the correspondent position of that word in a $1 \times N$ vector, all others positions remaining with digit 0. Since natural languages usually have a large vocabulary, a limited subset of the vocabulary must be used in order to make viable to perform the necessary computations in terms of memory requirements. The chosen subset of the vocabulary must be representative of the texts. Therefore, in practical problems a great deal of attention is devoted to this matter. In particular, it is common to involve an application domain specialist or use some kind of word frequency statistic or relevance metric, where the most frequent and rare words are excluded.

In word2vec approach, each word is represented in a embedding of fixed size, representing its co-occurrence in a large corpus of text in the same language of the texts of interest. It is possible to use only pretrained vectors or readjust the representation with new words.The main problem with both strategies is that it does not allows to represent words that are not in the training dataset.

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 dimension of the matrices to be computed are still obstacles to the most effective use of the method. Searching for a way to reduce these training time remaining with the flexibility and power of character level convolutional to classify text, we found a way to better encode the text. Our approach , with competitive accuracy, achieve a significant reduction on time execution from hours to min and from days to hours, by epoch.

To achieve this performance in execution, our approach consisted of two elements:

- *Obtaining a shorter representation*: At first we tough of using some form of encoding each character, and use the same approach of Zhang & LeCun (2015) but we realize that using a variable length encoding for each word could be more efficient. To do this we need

a way to encode each char, generating distinct concatenated code for representing each word and that words with the same prefix was near each other, specially to respond well to declensions.

- *Obtaining a sparse representation*: To take full advantage of libraries of tensor multiplications like NVidia CuDNN (Chetlur et al., 2014), we need a representation that was sparse, so our code should be composed of many 0 and a few 1.

Though Huffman encoding (Huffman, 1952) is the short possible, it does not generate unique representations once we concatenate char codes to form a word. We investigate encodings of Brisaboa et al. (2003) and found promising alternatives. Our approach is based on Tagged Huffman (Silva de Moura et al., 2000), where a pair of 0 digits is the signal and the digit 1 is the tag of begin and end code, the only difference is that for our approach, we need a shorter version to reduce the size of input matrix, so we choose to use only one digit 0 instead of two for each char, marking the beginning and the end of each char code with a digit 1 the same way. 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.
3. 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 over others strategies that use a word as atomic representation of text is 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 are 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 characters will generate bigger codes, consequently, bigger encoding matrix . If your database have a lot of them, you could use a higher word code size.

Our main contribution is demonstrate that such approach reduce the dimensionality of the encoding matrix, substantially reduce optimization time and allow the use of devices with lower computational power, remaining with competitive accuracy.

3 Compressed Encoding for CNN-based Text Classification

3.1 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 insert the digit 0 in the same amount of the rank position of the character, between a 1 digit to signal the begin and another 1 digit to signal the end of code. Table 1 have some examples of encoded characters.

Table 1: Example of coding using English language ranking of characters. Characters shown are the ones used in the subsequent examples.

CHARACTER	FREQ. RANK	COMPRESSED ENCODING
␣	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	$'1' + n \times '0' + '1'$

To encode each word, we just concatenate the codes of each character. As an example, we provide in Table 2 some examples of plain text words and their corresponding encoding.

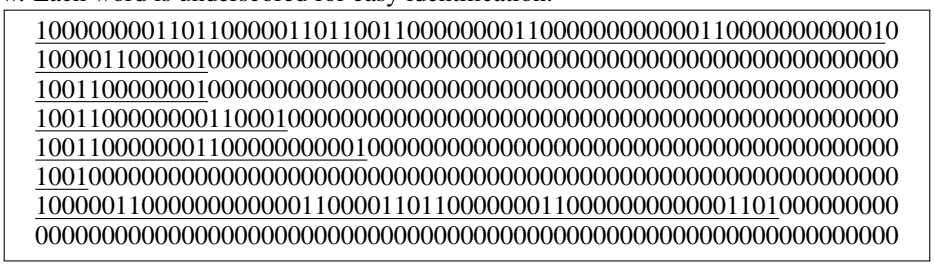
Table 2: Example of coding using English language ranking of characters. Prefix common to more than an word are underscored

TEXT	ENCODED TEXT
science	1000001100000000000011000011011000000011000000000000001101
scientist	10000011000000000000110000110110000000110001100001100000110001
art	1001100000000110001
artist	10011000000001100011000011000000110001
tl;dr	1000110000000011000000000000000000000000000000011000000000011000000001
u2	1000000000000011000000000000000000000000000000001
e^a	10110000000000000000000000000000000000000000000000000000000000011001

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 math equations, web addresses, LaTeX code, computer programming commands, etc. In Table 2 we can see that this encoding could represent words with the same prefix in vectors that have same initial coordinates. Slangs like *tl;dr : too long, did not read* and *u2 : you too* as well mathematical expressions like e^a could be manipulated. 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 Figure 1:

Figure 1: Matrix encoding of sentence 'Research is an art and a science'. Text is encoded one word per row. Each word is underscored for easy identification.



We used 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.

4 Experimental Study

4.1 Datasets

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. We shared a link [1] where the reader could download it. In this article, we will only summarize the main descriptions

- **Ag_news**: categorized news articles from more than 2000 news sources. Four classes (World, Sports, Business, SciTech). Dataset contains 120k train samples and 7.6k test samples equally distributed (Zhang et al., 2015).
- **Sogou_news**: categorized news articles originally in Chinese. Zhang et al. (2015) applied a *pypinyin* package combined with *jieba* Chinese segmentation system to produce Pinyin – a phonetic romanization of Chinese. Five classes (sports, finance, entertainment, automobile and technology). Dataset contains 450k train samples and 60k test samples equally distributed (Zhang et al., 2015).
- **DBpedia**: title and abstract from Wikipedia articles available in DBpedia crowd-sourced community (Lehmann et al., 2015). Fourteen 14 nonoverlapping classes from DBpedia 2014 Dataset contains 560k train samples and 70k test samples equally distributed (Zhang et al., 2015).
- **Yelp_full**: sentiment analysis from the Yelp Dataset Challenge in 2015[2]. Five classes representing the number of stars a user has given. Dataset contains 560k train samples and 38k test samples equally distributed. (Zhang et al., 2015).
- **Yelp_full**: sentiment analysis from the Yelp Dataset Challenge in 2015[3]. Rating of 1 and 2 stars are represented as Bad and 4 and 5 as Good. Dataset contains 560k train samples and 50k test samples equally distributed. (Zhang et al., 2015).
- **Yahoo_answers**: questions and their answers from Yahoo Answers. Ten classes (Society & Culture, Science & Mathematics, Health, Education & Reference, Computers & Internet, Sports, Business & Finance, Entertainment & Music, Family & Relationships, Politics & Government). Each sample contains question title, question content and best answer. Dataset contains 1400k train samples and 60k test samples (Zhang et al., 2015).

[1] https://drive.google.com/open?id=1o5CNT0UHuFfHBxC-Mz4ImFpN2-Lcllmx

[2] https://www.yelp.com/dataset/challenge

[3] https://www.yelp.com/dataset/challenge

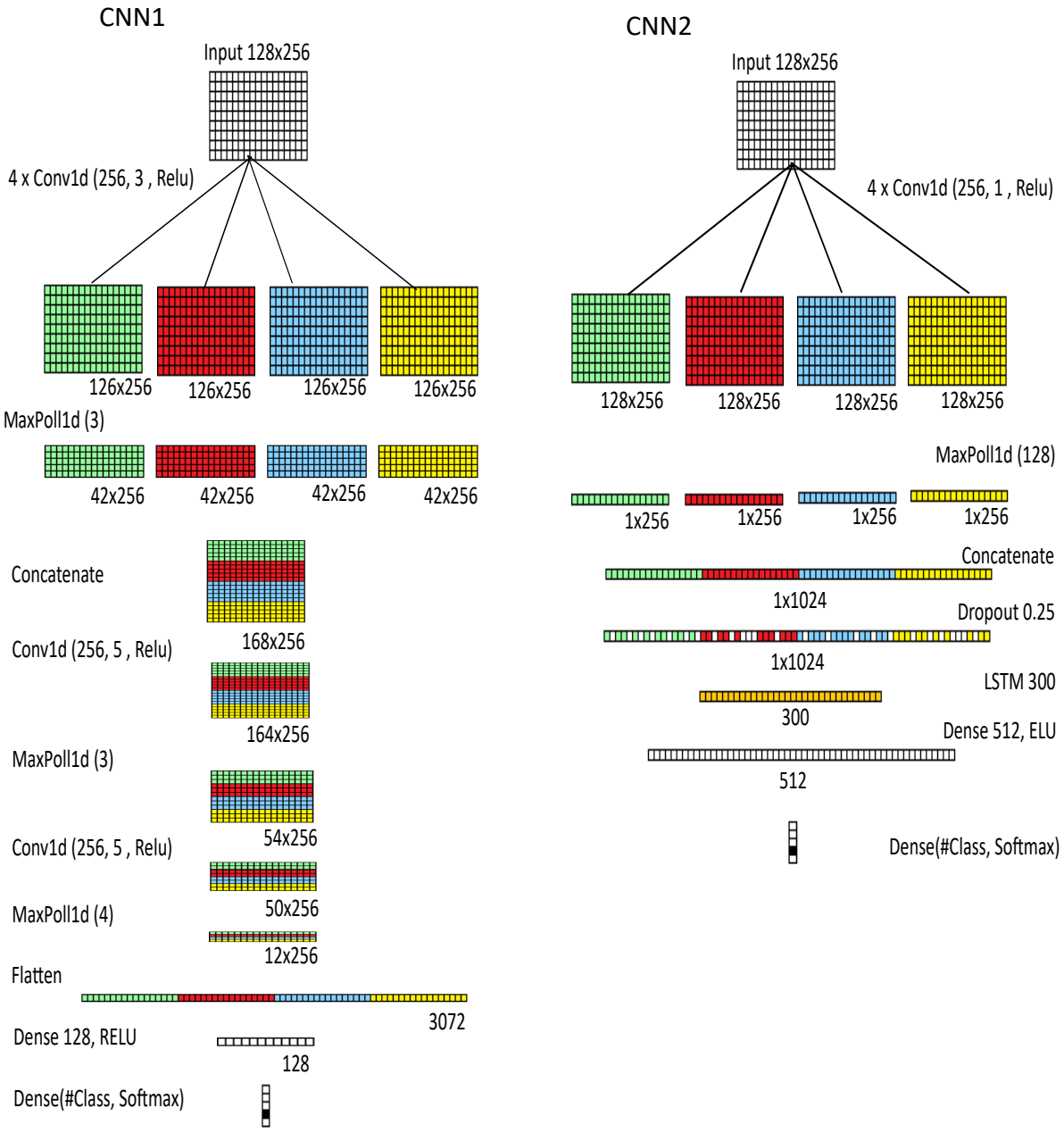


Figure 2: Network architecture

- **Amazon_full**: sentiment analysis from Amazon review dataset from the Stanford Network Analysis Project (SNAP) (McAuley & Leskovec, 2013). Five classes representing the number of stars a user has given. Dataset contains 3000k train samples and 650k test samples.(Zhang et al., 2015).

- **Amazon_polarity**: sentiment analysis from Amazon review dataset from the Stanford Network Analysis Project (SNAP) (McAuley & Leskovec, 2013). Two classes, rating of 1 and 2 stars are represented as Bad and 4 and 5 as Good. Dataset contains 3000k train samples and 650k test samples.(Zhang et al., 2015). Dataset contains 3600k train samples and 400k test samples equally distributed.(Zhang et al., 2015).

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

4.2 Comparison Models

The baseline comparison models are the same of Zhang et al. (2015) where there is an extensive description of them, we just reproduce theirs results, the only difference is that they report loss error and for better comprehension, we translated it to accuracy. In Zhang et al. (2015) there is an extensive description of them. In this paper, we just summarize the main information:

4.2.1 Baseline Models

- **Bag of Words (BOW) and its term-frequency inverse-document-frequency (Bow TFIDF)**: For each dataset, they selected 50,000 most frequent words from the training subset. For the normal bag-of-words, they used the counts of each word as the features and for the TFIDF they used the counts as the term-frequency (Zhang et al., 2015).

- **Bag-of-ngrams (Ngrams) and its TFIDF (Ngrams TFIDF)**: The bag-of-ngrams models were constructed by selecting the 500,000 most frequent n-grams (up to 5-grams) from the training subset for each dataset. The feature values were computed the same way as in the bag-of-words model (Zhang et al., 2015).

- **Bag-of-means on word embedding**: Experimental model that uses k-means on word2vec (Mikolov et al., 2013) learnt from the training subset of each dataset, and then used these learnt means as representatives of the clustered words. Took into consideration all the words that appeared more than 5 times in the training subset. The dimension of the embedding is 300. The bag-of-means features are computed the same way as in the bag-of-words model. The number of means is 5000 (Zhang et al., 2015).

- **Long Short Term Memory - LSTM**:LSTM (Mikolov et al., 2013) model used in our case is word-based, using pretrained word2vec embedding of size 300. The model is formed by taking mean of the outputs of all LSTM cells to form a feature vector, and then using multinomial logistic regression on this feature vector. The output dimension is 512 (Zhang et al., 2015).

4.2.2 Character Level Convolutional Network Model

The model that this paper builds upon is Zhang et al. (2015). It encodes each char as an one-hot encoding in a vocabulary of 69. The non-space characters are letters, numbers and punctuation. The model is composed of 9 layers, 6 of convolutions and 3 fully connected. Theirs architecture are described in Table 3. They used stochastic gradient descent (SGD) with a minibatch of size 128, using momentum 0.9 and initial step size 0.01 which is halved every 3 epochs for 10 times. So, their results were obtained in at least 30 epochs.

Table 3: Architecture of Zhang et al. (2015). Lg-Large, Sm- Small

CONVOLUTIONS				
Layer	**Larg Feature**	**Small Feature**	**Kernel**	**Poll**
1	1024	256	7	3
2	1024	256	7	3
3	1024	256	3	-
4	1024	256	3	-
5	1024	256	3	-
6	1024	256	3	3
FULLY CONNECTED				
Layer	**Lg Feature Out**	**Sm Feature Out**	**Dropout**	
7	2048	1014	.5	
8	2048	1014	.5	
9	Depends on the problem			

4.3 Neural Network Architecture

To verify the efficiency of the encoding procedure, we realized 2 experiments, named CNN1 and CNN2:

CNN1: At first we choose a network architecture that we used to classify text using an embedding created by word2vec, the only difference is that instead of 300 features, we reduce the input size to 256. This architecture we named CNN1. It is based on concatenation of convolutions in a shallow way, inspired on work of Kim (2014), who achieve state of the art results for some databases.

We trained this model for 5 epochs. The neural network architecture is described in Figure 2.

CNN2: Enthusiasmed by the results of the first experiment, we decided to investigate others possible architectures. We created another shallow but wide convolution architecture following the recommendations of Zhang & Wallace (2015) for choosing parameters, executing training on dataset ag_news, for being the shortest:

- *Convolution width filter*: a combination of region sizes near the optimal single best region size outperforms using multiple region sizes far from the optimal single region size (Zhang & Wallace, 2015). We scan the width from 1 to 7 comparing accuracy performance. In these evaluations, convolution of width 1 was a better option.

- *Pooling size*: max pooling consistently performs better than alternative strategies for the task of sentence classification (Zhang & Wallace, 2015).

The architecture CNN2 is illustrated in Figure 2. We trained this model for 12 epochs.

4.4 Results

For all the experiments, we used the environment and parameters settings listed on Table 4.

Table 4: 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	

Table 5: Accuracy comparison among traditional models and main model as found in 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		88.81	92.85	96.61	92.24	57.99	68.89	54.64	90.40
Bow TFIDF		89.64	93.45	97.37	93.66	59.86	71.04	55.26	91.00
Ngrams		92.04	97.08	98.63	95.64	56.26	68.47	54.27	92.02
Ngrams TFIDF		92.36	97.19	98.69	95.44	54.80	68.51	52.44	91.54
Bag-of-Means		83.09	89.21	90.45	87.33	52.54	60.55	44.13	81.61
LSTM		86.06	95.18	98.55	94.74	58.17	70.84	59.43	93.90
Lg Conv Zhang		87.18	95.12	98.27	94.11	60.38	70.45	58.69	94.49
Sm Conv Zhang		84.35	91.35	98.02	93.47	59.16	70.16	59.47	94.50
Ours	CNN1	87.67	95.16	97.93	92.04	58,00	68.10	58.09	93.69
	CNN2	91.43	93.96	98.03	91.53	57.03	70.24	55.72	91.23

All the results and comparison with traditional models and the approach of Zhang et al. (2015) is show in Table 5. On Table 6, we did a time execution comparison, based on reports available in Zhang & LeCun (2015).

Table 6: Time per Epoch comparison between Zhang & LeCun (2015) and our **CNN1** architecture on different NVidia GeForce GPUs.Times for **CNN2** architecture are very close to these.

DATASET	# TRAIN	# TEST	Zhang & LeCun (2015)		Ours	
			large	small	GF 1080ti	GF 930M
ag_news	120k	7.6k	1h	3h	3 min	25 min
sogou_news	450k	60k	-	-	23 min	1h33 min
dbpedia	560k	70k	2h	5h	18 min	1h56 min
yelp_polarity	560k	38k	-	-	21 min	1h59 min
yelp_full	650k	50k	-	-	27 min	2h15 min
yahoo_answers	1,400k	60k	8h	1d	47 min	4h50 min
amazon_full	3,000k	650k	2d	5d	2h	10h20 min
amazon_polarity	3,600k	400k	2d	5d	2h13 min	12h24 min

5 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. Some datasets of text have peculiarities that was not addressed by word frequency methods (BOW, Word2vec). The article of Zhang & LeCun (2015) was a great innovation in this regard. However, the training times are still obstacles to the most effective use of the technique. We though of a better representation should be a solution.

To make a comparison, perform the training of architecture **CNN1** with an output of 4 classes make 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).

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 results that we obtained are very competitive with the approach o Zhang et al. (2015) and traditional techniques. Observing our results, we see even that we could find parameters that results in excellent performance on ag_news dataset, following the suggestions of Zhang & Wallace (2015). One of advantage of a faster algorithm is that if your dataset is not so big, you could scan the feature width to find a solution that optimize accuracy. Another advantage is the possibility to realize k-folds validation, to have a better perspective on how well it will perform for your specific dataset on real life.

We are certain that our algorithm implementation could be even faster. For our inability, we could not find a way to make CPU and GPU working in parallel. Using a GPU Geforce 1080ti, each of the superbatch of 10000 arrays have its weights updated in 30 seconds. Only 6 seconds is consumed by GPU, another 24 seconds is spent in codifying all the matrix and delivery it on GPU. Using a multithread strategy could help in this regard.

The results obtained strongly indicate that the use of this coding is a possibility. We emphasize that we used is a fairly simple network, enough to demonstrate the feasibility of the encoding 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.

6 Final Remarks

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 characters 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.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Watten-

berg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

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.

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014. URL http://arxiv.org/abs/1410.0759.

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.

David Harris and Sarah Harris. *Digital design and computer architecture*. Morgan Kaufmann, San Francisco, CA, USA, 2nd edition, 2012.

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.

David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098–1101, September 1952.

Rie Johnson and Tong Zhang. Effective use of word order for text categorization with convolutional neural networks. *CoRR*, abs/1412.1058, 2014. URL http://arxiv.org/abs/1412.1058.

Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014. URL http://arxiv.org/abs/1408.5882.

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

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 6(2):167–195, 2015. URL http://jens-lehmann.org/files/2015/swj_dbpedia.pdf.

Julian McAuley and Jure Leskovec. Hidden factors and hidden topics: Understanding rating dimensions with review text. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pp. 165–172, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2409-0. doi: 10.1145/2507157.2507163. URL http://doi.acm.org/10.1145/2507157.2507163.

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.

Sparsh Mittal and Jeffrey S. Vetter. A survey of CPU-GPU heterogeneous computing techniques. *ACM Computing Surveys*, 47(4):69:1–69:35, July 2015. ISSN 0360-0300. doi: 10.1145/2788396. URL http://doi.acm.org/10.1145/2788396.

Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, March 2002. ISSN 0360-0300. doi: 10.1145/505282.505283. URL http://doi.acm.org/10.1145/505282.505283.

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.

Ye Zhang and Byron C. Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *CoRR*, abs/1510.03820, 2015. URL http://arxiv.org/abs/1510.03820.