IJCNN 2019. International Joint Conference on Neural Networks. Budapest, Hungary. 14-19 July 2019

Fast and Efficient Text Classification with Class-based Embeddings

Jônatas Wehrmann, Camila Kolling, and Rodrigo C. Barros
Machine Intelligence and Robotics Research Group
School of Technology, Pontifícia Universidade Católica do Rio Grande do Sul
Av. Ipiranga, 6681, 90619-900, Porto Alegre, RS, Brazil
Email: {jonatas.wehrmann,camila.kolling}@acad.pucrs.br, rodrigo.barros@pucrs.br

***Abstract*—Current state-of-the-art approaches for Natural Language Processing tasks such as text classification are either based on Recurrent or Convolutional Neural Networks. Notwithstanding, those approaches often require a long time to train, or large amounts of memory to store the entire trained models. In this paper, we introduce a novel neural network architecture for ultra-fast, memory-efficient text classification. The proposed architecture is based on word embeddings trained directly over the class space, which allows for fast, efficient, and effective text classification. We divide the proposed architecture into four main variations that present distinct capabilities for learning temporal relations. We perform several experiments across four widely-used datasets, in which we achieve results comparable to the state-of-the-art while being much faster and lighter in terms of memory usage. We also present a thorough ablation study to demonstrate the importance of each component within each proposed model. Finally, we show that our model predictions can be visualized and thus easily explained.**

***Index Terms*—Text classification, deep learning, neural networks, natural language processing.**

I. Introduction

Text classification approaches are important components within the Natural Language Processing (NLP) research, and they have been designed for countless application domains such as document classification [1], sentiment analysis [2]–[4], information retrieval [5], [6], hierarchical classification [**?**], [7] and generation of sentence embeddings [8], just to name a few.

A central problem in text classification is feature representation, which has relied for a long time on the well-known bag-of-words (or bag of n-grams) approach. Such a strategy describes the occurrence of words or characters within a document, and basically requires the usage of a vocabulary of known words and the measurement of the occurrence of those known words. Since we need to store the vocabulary, memory requirements are often a practical concern.

Recently, the NLP community has turned to methods that are capable of automatically learning features from raw text, such as Convolutional or Recurrent Neural Networks (CNNs/RNNs). CNNs were originally designed with computer vision applications in mind, but they have shown to be quite effective for a plethora of NLP applications [9], [10]. Indeed, models based on neural networks have outperformed traditional hand-crafted approaches achieving state-of-the-art performance in several NLP tasks [1], [11], [12].

Whereas neural network models often achieve very good performance on text classification, they tend to use a large amount of memory during both training and inference, especially when learning from a given corpus that contains a very large vocabulary. Recent work have tried to change this perspective, e.g., FastText [11]. Such a method represents a document by averaging word vectors from a given sentence, resulting in a bag-of-words-like representation, though allowing the update of word vectors through backpropagation during training as opposed to the static word representation in a standard bag-of-words model.

FastText provides fast learning of word representations and sentence classification. Compared to other systems [9], [13], [14] that are based on either CNNs or RNNs, FastText shows comparable results though much smaller training times. Nevertheless, in spite of being faster to train and to test than traditional techniques based on n-grams, FastText uses a lot of memory to store and process the embeddings. This is an important issue for applications that need to run on systems with limited memory, such as smartphones.

To address the limitation of the current neural network models, we propose four different fast and memory-efficient approaches. The first one, CWE-BCdraws inspiration from FastText [11] and generates word-embedding vectors by directly mapping the word-embedding space to the target class space. Our second method, CWE-SA, replaces traditional pooling functions by a self-attention module, giving different weights for each word and thereby focusing on the most important words of the sentences. CWE-C is the third approach and employs convolutional layers for processing the temporal dimension. Our final approach, CWE-R, is based on recurrent operators: it is designed to learn temporal relations and, unlike traditional RNNs, it does not require additional trainable weight matrices.

We compare our models with previous state-of-the-art approaches. They perform on a par with recently-proposed deep learning methods while being faster and lighter in terms of memory consumption. Our models make use of $\approx 100\times$ less memory while running up to $4\times$ faster when compared to FastText [11]. In addition, they are much easier to visualize and understand since they learn word embeddings that are trained directly on the class space.

978-1-7281-2009-6/$31.00 ©2019 IEEE paper N-19584.pdf

The rest of this paper is organized as follows. In Section II we describe the proposed approach and its variations, while in Section III we detail the setup used for training and evaluating each of the models and the respective baselines. In Section IV we describe the experiments that are performed to quantitatively evaluate our approaches on several text-classification datasets. In Section V we qualitatively assess the proposed methods. Section VI summarizes previous related work, and we finally conclude this paper and discuss future research directions in Section VII .

II. Class-based Word Embedding

In this paper, we introduce Class-based Word Embeddings (CWE), an approach designed to classify text in a fast, light, and effective fashion. Unlike traditional state-of-the-art text classification methods, CWE is developed to work with minimal resources in terms of processing and memory, while achieving solid results across distinct datasets.

CWE works by learning a text classification function $\psi(\mathcal{T}) = \mathbf{y}$, where $\mathcal{T} \supset \{\omega_j\}_{j=1}^{t}$ is a given text (instance) within the text-classification dataset, and it comprises t words, each encoded as a $\omega_j \in \mathbb{R}^C$ vector, and $\mathbf{y}$ is the respective binary-class vector for a C-class classification problem, so that $\sum_i^C \mathbf{y_i} = 1$. In the following sections we detail several flavors of function $\psi(\cdot)$, which gives CWE distinct capabilities with advantages and disadvantages.

A. CWE-BC: *Bag-of-Classes*

CWE-BC is the first strategy for learning function $\psi(\cdot)$, which somewhat draws inspiration from the FastText [11] model. In the latter, word embeddings $\omega \in \mathbb{R}^d$ are averaged in order to build a d-dimensional sentence feature representation, which is then linearly mapped through a trainable weight-matrix $W \in \mathbb{R}^{d \times h}$ to a hidden feature space with h dimensions, and finally projected onto the C-dimensional class space. Our approach aims to generate word-embedding vectors by exploring direct relations across the word-embedding space and the target class space. In CWE-BC, the direct word-class mapping is achieved by training $\omega \in \mathbb{R}^C$ word embeddings instead of $\omega \in \mathbb{R}^d$, discarding the need for additional transition weight-matrices. The final classification scores are given by processing the input $\mathcal{T} \in \mathbb{R}^{t \times C}$ matrix with a given pooling function. We have tried three distinct pooling functions, namely $\max$, mean, and $\mathrm{selfatt}$, the latter a self-attention-based function. Figure 1 depicts the CWE-BC approach.

Note that by using a pooling function directly over unigram word embeddings, CWE-BC fully discards temporal information, i.e., the order of the words across a sentence is not considered and does not affect the model predictions whatsoever. Therefore, such a method can be regarded either as a linear bag-of-words or as a bag-of-classes approach. Our hypothesis is that despite CWE-BC being very simple, it will be capable of providing good predictive performance across distinct datasets, since there are several words that are directly related to a given class. In sentiment analysis datasets, for instance, it is easy to find specific words that carry strong class-based content, such as *amazing* and *awesome*, which most certainly denote a positive polarity, while *terrible* and *worst* represent mostly the negative polarity.

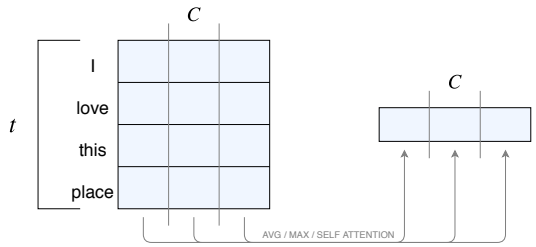

Fig. 1. Model architecture of CWE-BC. Each word is embedded according to the number of classes C and passed to one of the pooling functions.

In addition, word embeddings trained with CWE-BC can be easily visualized when the class space is small. For instance, assume that a sentiment analysis model is trained to classify text as either positive or negative. In that case, one could naturally visualize those embeddings in a bidimensional Euclidean space and fully explain the model predictions, without the need of further employing algorithms to visualize high-dimensional data such as t-SNE [15], which presents a high asymptotic computational complexity (quadratic in the number of instances), limiting its use to roughly 10,000 instances [15]. In theory, high-dimensional word-embedding-based methods project the input data so that the feature space lies on several different, though related, low-dimensional manifolds. In CWE, we directly optimize a low-dimensional manifold which is equivalent to an Euclidean space, making it much easier to visualize.

B. CWE-SA: *Self-Attentive Pooling*

Our second approach replaces traditional global pooling strategies by a learned self-attention module ($\mathrm{selfatt}$), responsible to assign distinct weights for each word so the final sentence representation is given by a weighted mean pooling. We refer to this method hereby as CWE-SA, and it is depicted in Figure 2. The self-attention mechanism was originally introduced in [16], being applied within RNNs.

Several papers have evaluated such an approach to summarize both convolutional and word-based data. In [17], the authors use the self-attention module so that the models can more easily focus on the most important sentence parts. Here we use the same input representation strategy $\mathcal{T} \in \mathbb{R}^{t \times C}$, which is processed by a $\tanh$-based fully-connected layer with n neurons (here we use $n = C$) that is responsible for normalizing activation values in the $(-1,\ 1)$ range, namely $\hat{\mathcal{T}}$. A second fully-connected layer, namely $\phi(\hat{\mathcal{T}})$ generates the annotation vector $\theta \in \mathbb{R}^{t \times 1}$ that will contain all word weights after a $\mathrm{softmax}$ function:

$$\theta_k = \frac{\exp(\phi(\hat{\mathcal{T}})_k)}{\sum_{j=1}^{t} \exp(\phi(\hat{\mathcal{T}})_j)}, \tag{1}$$

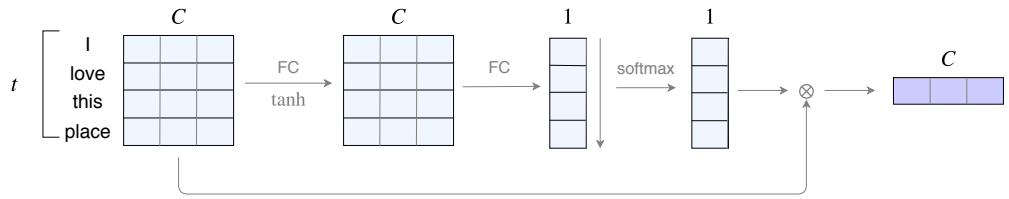

Fig. 2. Model architecture of CWE-SA. In this case, the first fully-connected layer transforms C classes into a $t \times n$ matrix, and the $\tanh$ is used as activation function. This matrix is passed to another fully-connected layer, generating a $t \times 1$ annotation vector that employs a $\mathrm{softmax}$ function so the weights for the words sum to 1. Finally, this matrix is multiplied by the input matrix, generating the final vector of C classes, which will display probabilities after the $\mathrm{softmax}$ function.

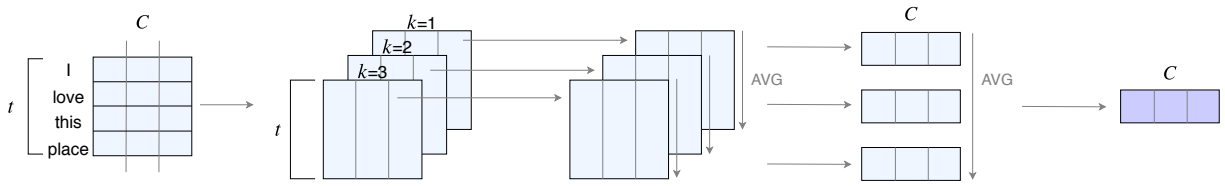

Fig. 3. Model architecture of CWE-C. Each word is embedded according to the number of classes and passed to a convolutional layer. Each convolutional layer has a different kernel size k, which affects the size of the receptive field over the temporal dimension, i.e., the number of words being processed altogether. Each feature map resulting from the convolutional layers are processed with a pooling layer, resulting in n arrays with dimension $1xC$. Finally, all arrays are averaged.

where θ_k provides the importance of the k^{th} word, and $\sum_{j=1}^{t} \theta_j = 1$. Ultimately, model predictions $\hat{y}$ are generated as $\hat{y} = \mathcal{T}^T \theta$, activated once again via $\mathrm{softmax}$.

Similarly to CWE-BC, this approach also discards temporal data, being unable to learn word dependencies and eventually leading to sub-optimal results. To circumvent this issue, we discuss two additional methods that are capable of leveraging temporal data for text classification, presented in Sections II-C and II-D.

C. CWE-C: A Convolutional Approach

CWE-C is a CWE variation that employs at least one convolutional layer for processing the temporal dimension. By applying a convolution with f filters of kernel size k directly over the input data, one can embed k words altogether within an f-sized feature representation. Thus, by processing the input textual data with distinct convolutional layers, one can learn k-gram-like information. Even though such a capability has been previously explored [1], [3], [5], to the best of our knowledge it was never applied over C-dimensional data, which may present additional learning constraints.

Inspired by [1], our models employ parallel convolutional layers over the input text $\mathcal{T}$. Each of these layers use f convolutional filters of length k, where the j^{th} filter in the i^{th} convolutional layer generates feature map $\mathcal{F}_{ij}$ whose x^{th} position is given by:

$$\mathcal{F}_{ij}^{x} = \phi\left(b_{ij} + \sum_{m=0}^{f_{i-1}} \sum_{p=0}^{k-1} w_{ijm}^{p} \mathcal{T}_{(i-1)m}^{(x+p)}\right) \qquad (2)$$

where ϕ is an activation function, b_{ij} is the bias for the respective convolutional filter, m iterates over the feature maps (channels), p indexes the position of the kernel, w_{ijm}^{p} is the filter weight, and $\mathcal{F}_{(i-1)m}^{x+p}$ is the value of the previous feature map (or input). Note that m iterates over the C dimensions of the input text $\mathcal{T}$, while p iterates over the temporal dimension building k-gram-like word embeddings.

The default version of CWE-C, as depicted in Figure 3, employs three parallel convolutional layers that are applied over the input data, containing $\{k = 1, k = 2, k = 3\}$. Each of those layers is padded so the output size remains unchanged, and the resulting feature matrix can be merged for building a consensual representation. Finally, a pooling strategy (mean, max, or $\mathrm{selfatt}$) is used to generate the final predictions.

D. CWE-R: A Recurrent Approach

Our final approach, hereafter called CWE-R, is based on fast and light recurrent operators that are designed to learn temporal relations. Differently from traditional RNNs such as LSTMs [18] and GRUs [19], CWE-R does not require any additional trainable weight matrices, since it applies non-linearity functions to the word embeddings themselves allied to residual connections over distinct time steps.

Formally, a given text $\mathcal{T}$ that contains word-embedding vectors $\{\omega_1, \omega_2, ..., \omega_t\}$ is forwarded through CWE-R by activating the first word embedding $\phi(\omega_1)$, which is then added to the original state of the second word embedding ω_2. This addition, which can also be seen as a temporal residual connection, resembles to some extent the input gate of the LSTM network. Note that by adding the non-linear word embedding to a linear one, distinct word orders would most likely produce distinct results, i.e., $\phi(\omega_1) + \omega_2 \neq \phi(\omega_2) + \omega_1$. Therefore, CWE-R should be able to learn temporal relations across words. Note that although the current CWE-R defi-

TABLE I
OVERVIEW OF THE TEXT CLASSIFICATION DATASETS.

Datasets	Training	Validation	Test	Total	#Characters/Inst	#Words/Inst	Vocabulary Size	Classes (C)
AGNews	112,400	7,600	7,600	127,600	239.62 ± 66.78	48.01 ± 13.78	35,065	4
DBPedia	490,000	70,000	70,000	630,000	307.00 ± 139.34	57.69 ± 24.42	193,680	14
Twitter	89,720	12,818	25,635	128,173	79.09 ± 37.22	18.37 ± 7.55	24,768	2
Yelp P.	522,000	38,000	38,000	598,000	728.43 ± 670.08	156.66 ± 141.68	26,241	2

nition uses the normalized last non-linear state $\mathrm{softmax}(\hat{\omega}_t)$ for performing the sentence prediction $\hat{y}$, one could use any of the previously-discussed pooling strategies across all the t non-linear states as well.

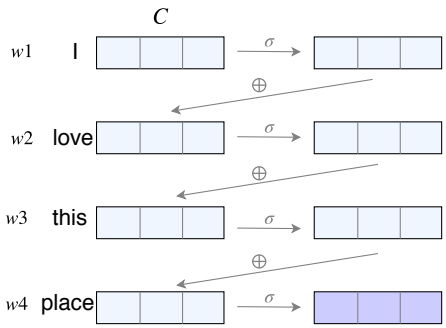

Fig. 4. Model architecture of CWE-R.

III. EXPERIMENTAL SETUP

In this section we detail the setup used for training and evaluating our models and the baseline approaches.

A. Datasets

We train and evaluate our models on four distinct text classification datasets: (i) AGNews, which contains short texts extracted from the AG corpus, and whose texts are classified according to 4 categories, namely Sports, Business, Sci/Tech and World; it comprises 30k training samples and 1.9k test samples per class; (ii) DBPedia is labeled according to an ontology, built with 14 categories from DBPedia 2014; each class has 40k training samples and 5k testing samples; (iii) Twitter [3], [20], a multilingual sentiment analysis dataset comprised of Tweets manually-annotated according to the classes positive and negative; it comprehends Tweets in English, German, Spanish, and Portuguese; and (iv) Yelp Polarity [9], a large scale sentiment analysis dataset that comprises more than 500,000 reviews from several products and places.

Given that most datasets do not provide a public validation set, we randomly selected instances from the training data according to the size of the test set. Table I presents an overview of the four datasets that were used in the experiments.

B. Hyper-parameters

All of our models were trained using Adam for minimizing the Categorical Cross-entropy loss function. We used the default learning rate as suggested in [21], namely 1×10^{-3}. For building the default vocabularies for each dataset, we filtered rare words by keeping words that present minimum frequency of 4 occurrences within their respective corpus. Note that both CWE-BC and CWE-R are free from model hyper-parameters, only requiring the setup of the optimizer. On the other hand, CWE-C does take additional model hyper-parameters: p is the number of parallel convolutional layers, each one containing f filters, with k kernel size. We used $p = 3$, $k = \{1, 2, 3\}$, and $f = c$ when not specified otherwise.

Regarding the pooling strategy, mean pooling is the default option for all methods but CWE-R, which uses the last activation of the hidden-state as the sentence representation. Finally, the hyper-parameter for controlling the size of the word embeddings with respect to the number of classes is by default $\gamma = 1$.

IV. EXPERIMENTS

We first compare our models with the current state-of-the-art. Then, we analyze the impact of the pooling strategies for summarizing the temporal dimension: max, mean, or $\mathrm{selfatt}$. Next, we evaluate the impact of varying γ on CWE-BC, CWE-R, and CWE-C. We also evaluate the impact of the vocabulary size for CWE-BC, CWE-R, and CWE-C.

A. State-of-the-art

In this section, we compare our methods with the state-of-the-art for text classification, depicted in the upper region of Table II. We measure predictive performance in terms of test set accuracy for each model in each dataset. Notably, our simpler approach CWE-BC, which employs only bag-of-class-based word embeddings, was capable of achieving performance comparable to the state-of-the-art models in three out of four datasets, namely AGNews (91.8%), DBPedia (98.5%), and Twitter (72.8%). In addition, it outperformed much more complex and heavier models in at least one dataset, e.g., char-CNN [9], char-CRNN [22], FastText (unigram) [11], and VCDNN [13], the latter being a very deep character-level neural network that requires several hours to train a single epoch [11]. Table II also shows that CWE-BC and CWE-SA achieve comparable performance. Indeed, the use of the self-attention strategy for pooling did not help achieving better predictive performance, which is somehow surprising given that it allows for the model to more easily focus in the most important words across a given text. CWE-R also presented sound results, outperforming all methods but FastText(bigram) on AGNews. Overall, it performed in a similar fashion to CWE-BC for all datasets, which gives us the intuition that

the temporal relations are not that helpful to classify text from the selected data. One also can observe that convolution-based approaches present better performance when trained with large datasets such as Yelp Polarity, in which CWE-C achieves 94.3% accuracy, only 1.4% behind both VCDNN and FastText (bigram) despite being much faster for classification and requiring fewer trainable parameters.

TABLE II
TEXT CLASSIFICATION RESULTS.

Model	AGNews	DBPedia	Yelp P	Twitter
char-CNN [9]	87.2	98.3	94.7	70.6
char-CRNN [23]	91.4	98.6	94.5	-
VDCNN [12]	91.3	98.7	95.7	-
Conv [1]	-	-	-	71.8
Conv-Char-S [3]	-	-	-	72.0
fastText $h = 10$ [11]	91.5	98.1	93.8	71.3
fastText $h = 10$ bigram [11]	92.5	98.6	95.7	-
CHAIN-v1 [5]	91.5	98.6	-	73.5
CWE-BC	91.8	98.5	93.8	72.8
CWE-SA	91.9	98.3	93.5	72.2
CWE-R	92.1	98.4	93.7	72.5
CWE-C	91.2	98.0	94.3	71.1

B. Ablation Study

Validation performance during training. We first compare the training steps for each approach. Figure 5 shows this comparison on AGNews, DBPedia, and Yelp Polarity datasets. Results show that CWE-BC takes more steps to converge compared to CWE-C and CWE-R, since it is a linear model. CWE-C and CWE-R have similar training convergence, except for the Yelp Polarity dataset, in which CWE-R converges faster than CWE-C. Finally, CWE-R and CWE-C are up to $9\times$ faster then CWE-BC.

TABLE III
IMPACT OF THE POOLING STRATEGY.

Dataset	Method	max	mean	selfatt	hs
AGNews	CWE-BC	88.0	91.9	91.9	-
	CWE-R	92.3	92.1	92.2	92.1
	CWE-C	90.4	91.5	91.2	-
Twitter	CWE-BC	67.0	72.3	72.2	-
	CWE-R	72.5	72.4	72.3	72.5
	CWE-C	71.2	72.2	72.2	-
DBPedia	CWE-BC	96.7	98.5	98.3	-
	CWE-R	98.4	98.3	98.4	98.4
	CWE-C	98.1	98.4	97.8	-

Pooling strategy. We analyze the impact of the pooling strategy for summarizing the temporal dimension. We provide results by using $\max$, mean, and $\mathrm{selfatt}$ pooling layers in all of our methods. For CWE-R, since it is based on a recurrent formulation, we also provide experiments using the last activation of the hidden-state as the sentence representation instead of summarizing the temporal dimension via pooling, namely hs. Table III shows the impact of distinct pooling strategies on AGNews, Twitter, and DBPedia. Results show

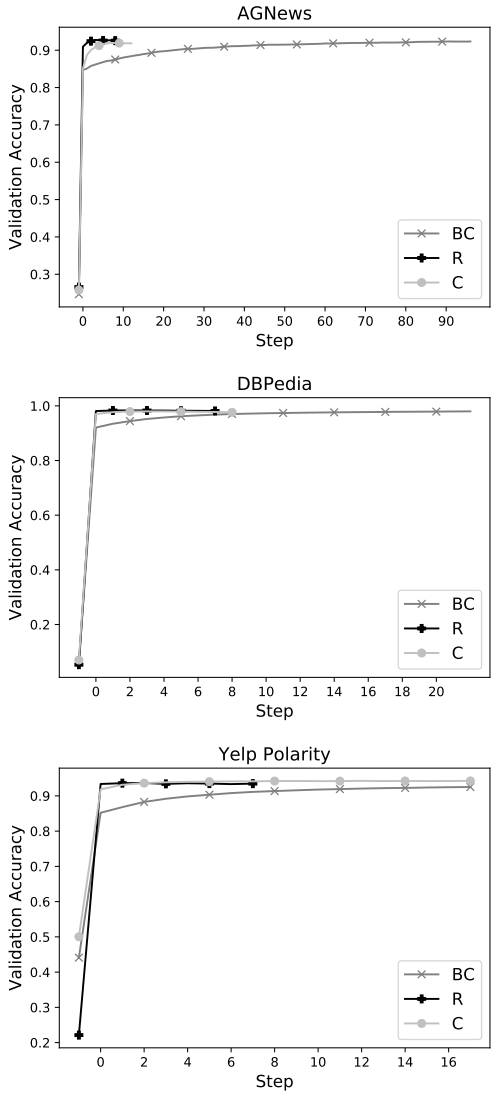

Fig. 5. Validation accuracy during training of CWE.

that all approaches but CWE-BCseem to be quite robust to the pooling strategy. For instance, CWE-R and CWE-C present only a slight variation when changing the pooling strategy. We do believe that such a robustness is due to the fact that differently from CWE-BC, both CWE-R and CWE-Care capable of approximating non-linear functions, which makes it easier for those models to approximate complex functions regardless of the pooling strategy. Finally, CWE-BC underperforms in all datasets with $\max$-pooling while presenting virtually the same results with mean and $\mathrm{selfatt}$.

Impact of γ. Table IV depicts the predictive performance for CWE-BC, CWE-R, and CWE-C when varying $\gamma \in \{10, 50, 100, 150\}$, which is the number of dimensions per class (default $= 1$). Hence, each model is trained with word embeddings of $C \times \gamma$ dimensions in order to evaluate the impact of the embedding size. Specially for this set of experiments, given that the original word-space is larger than

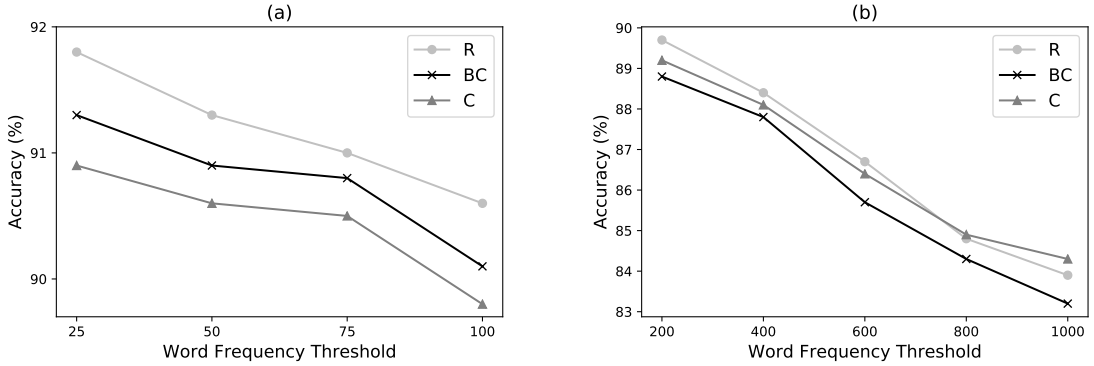

Fig. 6. Impact of the vocabulary size on the validation set accuracy for CWE-BC, CWE-R, and CWE-C. (a) The impact of filtering rare words; and (b) The impact of removing more frequent ones.

the class space, we use a linear layer of shape $(C \times \gamma) \times C$ to project the word-space onto the class space. Clearly CWE-C is the method that better leverages higher-dimensional input data, achieving 95.4% accuracy on Yelp Polarity when $\gamma = 150$, which is only 0.3% behind the state-of-the-art approaches, namely VCDNN and FastText(bigram). Results also show that both CWE-BC and CWE-R in their original incarnations are not affected by larger values of γ, presenting virtually the same results in all experiments.

TABLE IV
IMPACT OF THE CLASS-EMBEDDING SIZE γ.

γ	CWE-BC	CWE-R	CWE-C
1	93.8	93.7	94.3
10	93.9	94.1	94.8
50	93.9	94.4	95.1
100	93.8	94.2	95.3
150	93.8	94.4	95.4

Impact of the vocabulary size. Figure 6 depicts the impact of the vocabulary size on the AGNews dataset for three different approaches, namely CWE-BC, CWE-R, and CWE-C. Figure 6-(a) shows the performance by filtering relatively rare words, i.e., whose frequency ranges from 25 to 100; Figure 6-(b) depicts the effect of pruning the vocabulary by filtering more frequent words, i.e., whose frequency ranges between 200 and 1000. We defined those thresholds according to the word distribution in the AGNews training set. Results show, in all cases, a large performance drop as the word frequency threshold increases. One can observe that such a drop presents a linear correlation to the word frequency threshold. We believe that when limiting the vocabulary, trained models may suffer from underfitting.

C. Time Analysis

Table V shows the time (in seconds) each model takes to train a single epoch. For providing a fair comparison, we reimplemented all methods in PyTorch and trained them on the same hardware, which generated slightly different results when compared to the original FastText report [11]. All time-related experiments were processed on a MacBook Pro, 2 GHz Intel Core i5, 16 GB 1867 MHz LPDDR3 RAM, 512GB SSD.

Results show that CWE-BC is the fastest method to train, taking only 14.7 seconds to train an epoch on Yelp Polarity. Even CWE-C was capable of outperforming FastText in some datasets. As expected, CWE-R is slower when compared to the other CWE incarnations, mostly due to its recurrent nature. Moreover, our methods train up to three orders of magnitude faster than classic convolutional approaches such as VCDNN [12] and ConvChar [9].

TABLE V
TIME ANALYSIS (IN SECONDS).

Method	AGNews	DBPedia	Twitter	Yelp P.
VCDNN (d=29) [12]	3×10^3	3.6×10^3	-	4.1×10^4
ConvChar [9]	1.1×10^4	1.8×10^4	-	-
FastText [11]	3.4	17.8	2.4	60.02
CWE-BC	2.6	12.4	1.9	14.7
CWE-C	5.1	26.9	2.2	54.6
CWE-R	6.2	45.3	2.8	129.6

V. QUALITATIVE ANALYSIS

Our models are easier to visualize and understand given that they learn word embeddings that are trained directly over the class space. In this section, we show examples of word-embedding visualizations without using costly techniques such as t-SNE. In addition, we also depict plots of word-by-word predictions in the learned Euclidean class space. All the following analysis were generated using the first approach, namely CWE-BC, trained on Yelp Polarity. Since it was trained for sentiment analysis, we can directly visualize the trained word embeddings within a bidimensional chart.

A. Embedding Visualization

Our first qualitative analysis is regarding the visualization of the trained word embeddings. Figure 7 shows two distinct charts: (a) the spatial organization of the ten words with the largest magnitude for the positive class; and (b) the top ten

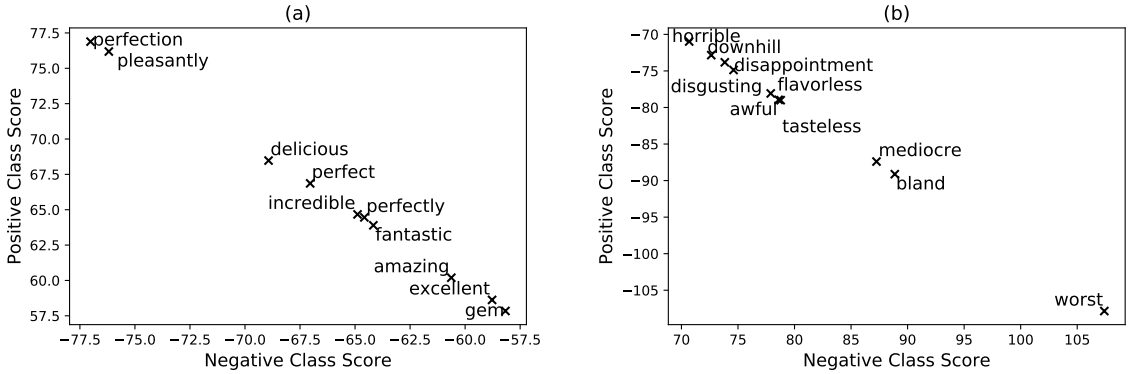

Fig. 7. Visualization of the learned word embeddings in the class space. (a) The ten words with largest score for the positive class. (b) The ten words with largest score for the negative class.

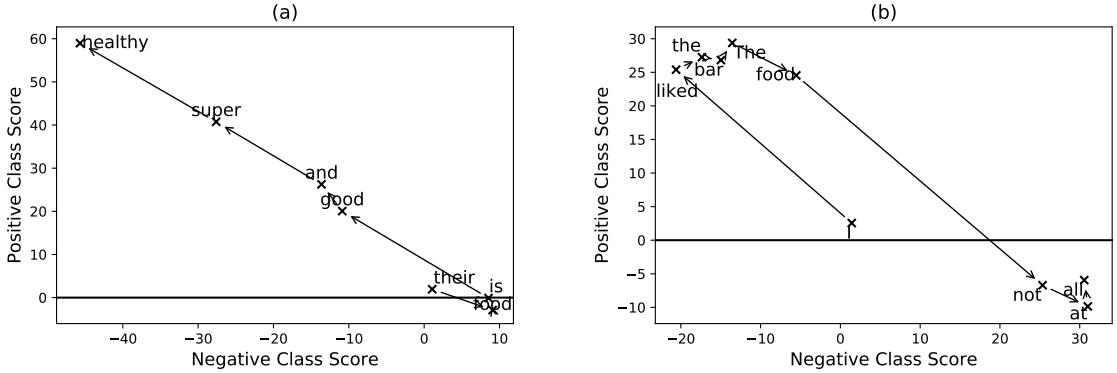

Fig. 8. Visualization of word-by-word prediction generated by CWE-BC. The horizontal line depicts the decision boundary of positive and negative classes, so the positive values of the x-axis generate positive-class predictions whilst negative values at the x-axis result in negative-class predictions. (a) The detailed prediction of a positive class review. (b) The detailed prediction of a negative class review.

words with largest magnitude for the negative class. In this visualization, we can see that the word with largest positive score is *perfection*, while for the negative class the word is *worst*. In addition, we can also observe that the classification of the sentence *"pleasantly disappointment"* should be neutral, i.e., ≈ 0 for both classes, given that words *pleasantly* and *disappointment* present almost perfect opposite weights.

B. Prediction Visualization

Figure 8 shows detailed word-by-word prediction for two excerpts of Yelp Polarity validation reviews, one for each class. Each marker on the charts depicts an intermediate prediction after all previous words. Figure 8-(a) explains the complete prediction for the text *"their food is good and super healthy"*, in which it is easy to see that words in $\{$*their, food, is*$\}$ present neutral content (near-zero values for both classes), while words in $\{$*good, super, healthy*$\}$ largely increase the prediction score for the positive class. Figure 8-(b) details the prediction for review *"I liked the bar. The food not at all."*. In this case, despite the fact that this review contains the positive-biased word *liked*, the following sequence $\{$*not, at, all*$\}$ is responsible for increasing negative scores, ultimately causing the review to be assigned to the negative class. Once again we can observe that there are neutral words that present slight biases that do not largely affect predictions, e.g., those in $\{$*I,the,bar,food,at*$\}$.

VI. Related Work

Traditional approaches for text classification represent documents with sparse lexical features such as n-grams, and then use a linear model or kernel methods over that representation [24], [25]. Recently, models based on neural networks have gained space. For instance, the architecture from [1] is based on convolutional layers, which was not used at that time in applications other than computer vision.

C-LSTM [22] is an architecture that combines both LSTM and convolutional layers for both sentence representation and text classification This model uses a CNN to extract a sequence of higher-level phrase representations, which are fed to a LSTM responsible for encoding the sentence. C-LSTM is said to be capable of capturing both local, global, and temporal sentence semantics.

[9] introduce a CNN that learns directly from raw characters. This architecture achieved state-of-the-art performance without requiring additional pre-processing strategies. VCDNN [13], in turn, is a very deep character-level CNN for text processing that uses several convolutional layers (up

to 29) with filter sizes of 3, allied to pooling operations for reducing the temporal dimension of the sentences.

The authors of [26] propose a hierarchical attention network for text classification. They empirically verify whether better representations can be obtained by incorporating knowledge of document structure into the model architecture. The work of [16], in turn, introduces the self-attention mechanism along with a regularization term for regulating the importance of diversity within the self-attention activation map.

FastText [11] is a model based solely on averaging word embeddings for building bag-of-words-like representations of sentences. The default incarnation of FastText uses only unigram information, fully discarding temporal information. The authors also propose the use of n-gram-based embeddings, which led to state-of-the-art text classification results while training up to two orders of magnitude faster than the baseline approaches. On the other hand, the use of additional n-gram information requires much larger amounts of memory to store all the trained embeddings. Note that our architecture in this paper is orthogonal to the work that perform quantization and compression of the models [27]. Our models are much lighter and faster due to their original design, which does not prevent them to be compressed or quantized.

VII. Conclusion

In this paper, we present four efficient, light, and very fast methods for text classification. All of them follow the same principle, which is to generate word-embedding vectors by exploring direct relations across the word-embedding space and the target class space. By doing so, our approaches are quite fast at both training and inference while using less memory to store and process information. We evaluate our approaches on widely-used datasets and results show that our methods perform on a par with state-of-the-art approaches while being much faster and memory-efficient. Finally, our models are easier to visualize and understand since they learn word embeddings that are trained directly over the class space. For future work, we intend to verify the feasibility of using our approaches in more complex NLP tasks.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nivel Superior – Brasil (CAPES) – Finance Code 001. We also would like to thank Google and FAPERGS for funding this research. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the graphics cards used for this research.

References

[1] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[2] S. Rosenthal, N. Farra, and P. Nakov, "Semeval-2017 task 4: Sentiment analysis in twitter," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, 2017, pp. 502–518.

[3] J. Wehrmann, W. Becker, H. E. Cagnini, and R. C. Barros, "A character-based convolutional neural network for language-agnostic twitter sentiment analysis," in *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE, 2017, pp. 2384–2391.

[4] W. Becker, J. Wehrmann, H. E. L. Cagnini, and R. C. Barros, "An efficient deep neural architecture for multilingual sentiment analysis in twitter," in *Proceedings of the Thirtieth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2017, Marco Island, Florida, USA, May 22-24*, 2017, pp. 246–251.

[5] J. Wehrmann and R. C. Barros, "Bidirectional retrieval made simple," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[6] J. Wehrmann, A. Mattjie, and R. C. Barros, "Order embeddings and character-level convolutions for multimodal alignment," *Pattern Recognition Letters*, vol. 102, pp. 15–22, 2018.

[7] J. Wehrmann, R. Cerri, and R. Barros, "Hierarchical multi-label classification networks," in *International Conference on Machine Learning*, 2018, pp. 5225–5234.

[8] H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, and R. Ward, "Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval," *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, vol. 24, no. 4, pp. 694–707, 2016.

[9] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, 2015, pp. 649–657.

[10] J. Wehrmann, W. E. Becker, and R. C. Barros, "A multi-task neural network for multilingual sentiment classification and language detection on twitter," *Symposium on Applied Computing*, vol. 2, no. 32, p. 37, 2018.

[11] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759*, 2016.

[12] A. Conneau and H. Schwenk, "Very deep convolutional networks for natural language processing," 2016.

[13] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun, "Very deep convolutional networks for text classification," *arXiv preprint arXiv:1606.01781*, 2016.

[14] D. Tang, B. Qin, and T. Liu, "Document modeling with gated recurrent neural network for sentiment classification," in *Proceedings of the 2015 conference on empirical methods in natural language processing*, 2015, pp. 1422–1432.

[15] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[16] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, "A structured self-attentive sentence embedding," *arXiv preprint arXiv:1703.03130*, 2017.

[17] J. Wehrmann, M. A. Lopes, M. D. More, and R. C. Barros, "Fast self-attentive multimodal retrieval," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 1871–1878.

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[19] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[20] I. Mozetič, M. Grčar, and J. Smailović, "Multilingual twitter sentiment classification: The role of human annotators," *PloS one*, vol. 11, no. 5, p. e0155036, 2016.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[22] C. Zhou, C. Sun, Z. Liu, and F. Lau, "A c-lstm neural network for text classification," *arXiv preprint arXiv:1511.08630*, 2015.

[23] Y. Xiao and K. Cho, "Efficient character-level document classification by combining convolution and recurrent layers," *arXiv preprint arXiv:1602.00367*, 2016.

[24] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *European conference on machine learning*. Springer, 1998, pp. 137–142.

[25] S. Wang and C. D. Manning, "Baselines and bigrams: Simple, good sentiment and topic classification," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*. Association for Computational Linguistics, 2012, pp. 90–94.

[26] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," pp. 1480–1489, 2016.

[27] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext.zip: Compressing text classification models," *CoRR*, vol. abs/1612.03651, 2016. [Online]. Available: http://arxiv.org/abs/1612.03651