Data Mining and Knowledge Discovery manuscript No.
(will be inserted by the editor)

Word-Class Embeddings for Multiclass Text Classification

Alejandro Moreo · Andrea Esuli · Fabrizio Sebastiani

Received: August 2020 / Accepted: date

Abstract Pre-trained word embeddings encode general word semantics and lexical regularities of natural language, and have proven useful across many NLP tasks, including word sense disambiguation, machine translation, and sentiment analysis, to name a few. In supervised tasks such as multiclass text classification (the focus of this article) it seems appealing to enhance word representations with ad-hoc embeddings that encode task-specific information. We propose (supervised) *word-class embeddings* (WCEs), and show that, when concatenated to (unsupervised) pre-trained word embeddings, they substantially facilitate the training of deep-learning models in multiclass classification by topic. We show empirical evidence that WCEs yield a consistent improvement in multiclass classification accuracy, using six popular neural architectures and six widely used and publicly available datasets for multiclass text classification. One further advantage of this method is that it is conceptually simple and straightforward to implement. Our code that implements WCEs is publicly available at `https://github.com/AlexMoreo/word-class-embeddings`.

Keywords Word-Class Embeddings · Word embeddings · Distributional hypothesis · Multiclass text classification · Neural text classification

1 Introduction

Recent advances in deep learning have led to important improvements in many NLP tasks that deal with the semantic analysis of text, including word sense disambiguation, machine translation, summarization, question answering, and sentiment analysis (see (Collobert et al., 2011; LeCun et al., 2015), for an overview). At the heart of the neural approach to the semantics of text lies

All authors are at Istituto di Scienza e Tecnologie dell'Informazione, Consiglio Nazionale delle Ricerche, 56124 Pisa, Italy
E-mail: firstname.lastname@isti.cnr.it

the concept of *word embedding* (a.k.a. *continuous* or *distributed representation* (Bengio et al., 2003; Mikolov et al., 2013b)), a dense representation of a word's meaning in a vector space where the semantic similarity of words is embodied in the notion of distance between vectors.

Word embeddings can either be initialized randomly and allowed to evolve along the rest of the model parameters, or be initialized from pre-trained word embeddings obtained offline by scanning massive amounts of textual data. This latter approach is generally preferred, since pre-trained embeddings encode an effective prior that embodies our general-purpose knowledge of the semantics of words, and that can be successfully transferred to (and eventually fine-tuned for) specific application contexts and downstream tasks (Erhan et al., 2010).

Approaches to generate word embeddings typically rely on the *distributional hypothesis*, according to which words that tend to occur in similar contexts tend to have similar meanings (Harris, 1954). Different realizations of this hypothesis were initially based on *context-counting* approaches (Blei et al., 2003; Bullinaria and Levy, 2007; Deerwester et al., 1990; Sahlgren, 2005) and later based on *context-predicting* approaches (Grave et al., 2017; Mikolov et al., 2013b; Pennington et al., 2014). Context-counting approaches collect frequencies of word co-occurrence and typically involve some form of matrix factorization to obtain the final word representations (Deerwester et al., 1990). Conversely, in context-predicting approaches the word representations constitute the parameters of a model trained to predict some distributional property of the data. As an example, `word2vec`'s *skip-gram with negative sampling* method (SGNS) (Mikolov et al., 2013b) tries to guess the surrounding words from the observation of the central word in a sliding context window.

While the relative desirability of one paradigm over the other was once the subject of debate (Baroni et al., 2014), it has later been argued that the two approaches simply embody different ways of pursuing what is essentially the same objective (Levy and Goldberg, 2014), and that differences in performance are mainly explainable in terms of hyperparameter settings and design choices (Levy et al., 2015). It has been proven that the optimum of the objective function that SGNS (Mikolov et al., 2013b) and *noise-contrasting estimation* (NCE) (Mnih and Kavukcuoglu, 2013), two context-predicting methods, seek to optimize, can directly be attained by a context-counting method called *shifted pointwise mutual information* (SPMI) (Levy and Goldberg, 2014). This seems to suggest that SPMI (and context-counting approaches in general) should be preferred to SGNS or NCE (and to context-predicting approaches in general). However, there are practical reasons why the opposite is the case. The main drawback of context-counting methods is the fact that they need to work with the entire co-occurrence matrix, something that becomes impractical when large quantities of text are involved. This problem does not harm neural supervised learning approaches, though, which are inherently incremental when adopting stochastic optimization (a standard practice nowadays). For this reason, the neural approach is currently the dominant one in modern distributional semantics.

Through the lens of the downstream task, the pre-trained word embeddings that all these methods generate are unsupervised, in the sense that they capture how words are distributed in general language use, in a way which is completely independent of (and thus not optimized for) the downstream task. However, in supervised tasks such as text classification (the focus of this work), it seems reasonable to imbue the word representations with supervised information that is available during training. In this article we propose *word-class embeddings* (WCEs), a form of supervised embeddings of words specifically designed for multiclass text classification,[1] that directly model the interactions between words and class labels.

A related intuition has been explored before in the context of text classification (Bojanowski et al., 2017; Grave et al., 2017; Tang et al., 2015; Wang et al., 2018) by jointly modelling word embeddings and *label embeddings* in a common vector space as part of the optimization procedure. Arguably, the best-known among the methods based on this intuition is `fastText` (Bojanowski et al., 2017; Grave et al., 2017), a variant of `word2vec`'s *continuous bag-of-words* method (CBOW – see Section 2.1) that substitutes the target central word that CBOW seeks to predict, with a token representing one of the document's labels. The result is a method that jointly models words[2] and labels as vectors, by recasting labels as new words and simply applying the distributional hypothesis anew.

We follow a different approach from those explored before by confining the supervised embeddings in a dedicated vector space, so that they can then be concatenated with any unsupervised pre-trained representations. The resulting embedding matrix can be used as the building block of any neural architecture. Our method does not involve any optimization procedure but operates directly on the co-occurrence counters. In a way, the method we propose might be regarded as the context-counting counterpart of (the context-predicting) `fastText` for word-class distributions, just like SPMI stands to SGNS for word-word distributions (Levy and Goldberg, 2014). Note that the disadvantage of context-counting approaches with respect to context-predicting ones that we have discussed before (i.e., the need to work with the entire, potentially huge co-occurrence matrix) does not arise here, since the amount of labelled documents one typically has in text classification applications is limited, and working with the co-occurrence matrix is thus unproblematic.

One advantage of this method is that it is conceptually straightforward (or, echoing the words of Daumé (2007), "frustratingly easy"), and very simple to implement, since it comes down to one matrix multiplication, followed by a pass of standardization and optionally by the application of dropout (Srivastava et al., 2014). Yet, as we empirically show, extending the pre-trained

[1] Given a set of classes (a.k.a. a *codeframe*) $\mathcal{C} = \{c_1, \ldots, c_m\}$, a classification problem is said to be *multiclass* if $m > 2$; it is said to be *single-label* if each item always belongs to exactly one class; it is said to be *multilabel* if each item can belong to any number (i.e., 0, 1, or more than 1) of classes in $\mathcal{C}$.

[2] `fastText` can consider not only unigrams but also n-grams and subwords as the surface forms of input.

unsupervised word embeddings with our task-specific supervised WCEs substantially facilitates the training of neural classifiers, and yields consistent improvements in multiclass classification performance across six widely used and publicly available text classification datasets, and six popular neural architectures (including `fastText` and BERT). Experiments also show that our word-class embeddings can be computed very quickly.

The rest of this article is structured as follows. In Section 2 we thoroughly review related work. We explain the method in Section 3, while Section 4 reports the experimental evaluation we have conducted. Section 5 tackles a few advanced topics related to WCEs, while Section 6 concludes, pointing at possible avenues for future work.

2 Related Work

In this section we turn to review relevant related work on word embeddings (Section 2.1) and neural approaches to text classification that exploit either word or label embeddings (Section 2.2).

2.1 Word Embeddings

Although the term *word embedding* owns its popularity to the neural approach, the very first attempts to generate distributed representations arose in the realm of context-counting approaches. Arguably, the best-known one is *Latent Semantic Analysis* (LSA) (Deerwester et al., 1990), a method that obtains r-dimensional representations of words by factoring (via singular value decomposition – SVD) a word-by-context co-occurrence matrix, and retaining the r eigenvectors with the highest eigenvalue. *Positive Pointwise Mutual Information* (PPMI) (Levy and Goldberg, 2014) takes the positive part of PMI as applied to the counters of the matrix, before decomposing it.[3] We explore PPMI as an alternative to our method in Section 4.8.

The neural approach to distributional semantics started with (Bengio et al., 2003), and gathered momentum with `word2vec` (Mikolov et al., 2013b), a method based on a two-layer neural network trained to predict the words in the context of a central word (*skip-gram* – SG) or the center word from the words in a (sliding) context window (*continuous bag-of-words* – CBOW). Input and output words are represented as one-hot vectors, and the first layer acts as a lookup table indexing the word embeddings (the layer parameters). `word2vec` owns part of its success to hierarchical softmax and negative sampling (Mikolov et al., 2013a), that permitted to dramatically speed up their

[3] *Pointwise Mutual Information* (PMI) is defined as $\text{PMI}(w_i, c_j) = \log \frac{\Pr(w_i, c_j)}{\Pr(w_i)\Pr(c_j)}$, where $\Pr(w_i, c_j)$ is the joint probability of word w_i and context c_j, and $\Pr(w_i)$ and $\Pr(c_j)$ are the marginal probabilities of the word and context, respectively. PPMI takes the positive part of PMI, i.e., $\text{PPMI}(w_i, c_j) = \max\{0, \text{PMI}(w_i, c_j)\}$.

computation, thus allowing the method to scale to massive amounts of textual data. `GloVe` is another popular method for generating embeddings, that has proven superior to `word2vec` in various tasks (Pennington et al., 2014). `GloVe` learns the word vectors that better reconstruct the probabilities of co-occurrence between pairs of words as estimated via their dot product. Similarly, `fastText` can be used in "unsupervised" mode to act as a method for the generation of word embeddings. When doing so, `fastText` implements SG or CBOW, but adds further tricks, such as the possibility to operate with subword information in order to cope with out-of-vocabulary words (Grave et al., 2017). `word2vec`, `GloVe`, and `fastText` have been used to generate large sets of embeddings that have later been made publicly available. We use all these sets of pre-trained vectors in the experiments of Section 4.

Despite the good performance these sets of pre-trained embeddings have delivered across many NLP tasks, they all fail to provide contextualized representations of words. The recent trend in embedding-generation research focuses on devising ways for conditioning the representation of the word on its position and context in the sentence. Some renowned examples along these lines include *Embeddings from Language Models* (ELMo) (Peters et al., 2018), *Bidirectional Encoder Representations from Transformers* (BERT) (Devlin et al., 2019), or the generalized autoregressive pre-training method based on Transformer-XL (XLNet) (Yang et al., 2019b). Pre-trained versions of these models have been made available to the community, and are known to enable state-of-the-art performance in many NLP tasks by simply adding one or few layers on top of the chosen neural architecture and fine-tuning on task-dependent data. In Section 4 we consider BERT-generated embeddings as a representative example of contextualized embeddings, and show how WCEs improve classification performance when concatenated to them.

2.2 Neural Text Classification

Text classification (TC) is a supervised learning task in which a model is trained to predict labels for unseen documents from the observation of labelled documents. Unlike traditional machine learning approaches to TC which represented documents through sparse vectors of lexical features (Joachims, 1998; Wang and Manning, 2012), the neural approach to TC builds on top of distributed representations for words and documents. Popular architectures routinely adopted in neural TC include convolutional neural networks (CNNs) (Collobert et al., 2011; Kim, 2014; Le et al., 2018), recurrent neural networks (RNNs) (Hochreiter and Schmidhuber, 1997; Lai et al., 2015; Rumelhart et al., 1986), recursive deep models (RDMs) (Socher et al., 2013), and attention models (ATTNs) (Luong et al., 2015; Vaswani et al., 2017). The training strategy is common across all these architectures: they first generate a document representation and then connect it directly to the labels during training. Typically, classifiers built on top of these networks generate a document embedding that is then connected, thorough one or more feed-forward

layers, to the target labels. The supervised information contributes only indirectly to representing the documents, as resulting from iterations of *backward* passes during training. Put it another way, words and labels are disconnected when generating the document representation in the forward pass.

Different models have been proposed that instead leverage the training labels directly at the word level. Tang et al. (2015) proposed *Predictive Text Embeddings* (PTEs), a type of embeddings that rely on a heterogeneous text network consisting of three bipartite graphs, each of which models one particular type of co-occurrence: word-word, word-document, and word-label. An embedding is then generated for each vertex in the graph, and documents are represented by averaging the embeddings of the vertices corresponding to the words they contain. However, the embeddings of the vertices corresponding to the labels are not used directly by the classifier, but only concur in the generation of the word embeddings.

Grave et al. (2017) and Bojanowski et al. (2017) proposed `fastText`, a variant of the CBOW architecture for text classification that models both word embeddings and label embeddings. `fastText` seeks to predict one of the document's labels (instead of the central word) and incorporates further tricks (e.g., n-gram features, sub-word information) to further improve efficiency. As a variant of CBOW, and similarly to PTEs, `fastText` represents a document as a simple average of word embeddings (although `fastText` implements a full set of heuristics –or "a bag of tricks", as the authors put it– to boost performance).

Yet another attempt to incorporate class label information into distributed word representations is to be found in (Jin et al., 2016). The proposed method, called *Bag-of-Embeddings* (BoE), is an extension of SGNS that, differently from SGNS, generates one distributional representation for each pair (word, class), i.e., the target word embeddings are class-conditional (the context embeddings are, as in SGNS, class-independent). The document class is then inferred via maximum likelihood by considering the (class-conditional) word embeddings of the words in the document. Note that, while our WCEs build on top of class-conditional distributions as well, they are meant to embed words, and not word-class pairs. This means that, if $\mathcal{V}$ is the vocabulary and $\mathcal{C}$ is the codeframe (i.e., the set of classes of interest), our WCE method computes $|\mathcal{V}| \times |\mathcal{C}|$ parameters (though this can be controlled for large codeframes – Section 4.6) while BoE computes instead $|\mathcal{V}| \times r + |\mathcal{V}| \times |\mathcal{C}| \times r$ parameters for the context vectors and the word-class embeddings, respectively, where r is the dimensionality of the embeddings. This can rapidly become unfeasible for large codeframes. Furthermore, our WCEs do not require optimization and are meant to be used in any neural classifier, while it is not clear how, if at all, BoE could be used outside the scope of the maximum likelihood classification model they were optimized for.

Pappas and Henderson (2019) incorporate a model of *compatibility* between a document embedding (generated by a combination of a word-level attention model and a sentence-level attention model) and a label embedding (generated as the average of the embeddings of the words in a brief description of the

class semantics) by means of a nonlinear feedforward model[4] whose number of parameters is independent of the number of input labels.

Wang et al. (2018) instead model the compatibility between words and labels (and not between documents and labels as in (Pappas and Henderson, 2019)) in order to define an attention model. They propose a model called *Label Embedding Attentive Model* (LEAM), which jointly embeds words and labels in the same latent space. Once words and labels are embedded in a common vector space, word-label compatibility is measured via cosine similarity. The label-embedding attentive model allows LEAM to go beyond simply averaging word embeddings (as, e.g., PTE and `fastText` do), and to weight differently the contribution of the word embeddings in a supervised fashion. An interesting variant of LEAM is the *Weighted Word Embedding Model* (WWEM) (Ren et al., 2019). WWEM replaces the attention model that LEAM computes with a simpler weighting criterion that uses Supervised Term Weighting (STW) scores for the weighted embedding average. WWEM also models bi-grams (weights and embeddings) in order to account for local word order information. In a similar vein, Gupta et al. (2019) propose *OptEm* (standing for *Optimal Embeddings*), a method that similarly relies on weighted averaging as the composition function. Word weights are obtained in OptEm as part of a SVM optimization that aims at separating the positive examples from the negative examples in binary text classification. In contrast to these methods, our WCEs method models the word-class compatibilities directly by observing the co-occurrences of words and labels of the entire training set, without generating intermediate embeddings for words and labels, and without performing any optimization. The word-label correlations compose our WCEs, while they are instead used, via weighted averaging, to scale the relative importance of the input words in LEAM, WWEM, and OptEm. A differentiating aspect of our method is that it keeps the modelling of word-class interactions separate from the original word embedding. Word-class correlations are confined in a dedicated vector space, whose vectors enhance (by concatenation) the unsupervised representations. The net effect is an embedding matrix that is better suited to classification, and imposes no restriction to the network architecture using it.

Dong et al. (2020) propose P-LSIAM (standing for *Pre-trained Labels embedding and Self-Interaction Attention based text classification Model*), a model that expands the ideas of LEAM. P-LSIAN relies on BERT (instead of `GloVe` pre-trained vectors as LEAM does) to transform the text into contextualized word embeddings and sentence embeddings. Word-label compatibility is then computed as in LEAM; however, in this case, the resulting attention weights are used not only to rescale the importance of the input, but also the contribution of each label embedding. The second important difference with respect to LEAM concerns the way P-LSIAM embeds documents. P-LSIAM replaces

[4] The compatibility between a label embedding matrix E and a document embedding h is defined to be proportional to $\sigma(EU + b_u)\sigma(Vh + b_v)$, and this is in contrast to what is customarily done in previous related literature that relied instead on bi-linear models of the form EWh for the same purpose (U, b_u, V, b_v, W are learnable parameters).

the weighted average of word embeddings with a weighted average of *interaction embeddings*; these are generated via a "self-interaction" attention model that models how the different sentences of the text (as embedded by BERT) interact with each other.

The main difference between our method and the ones described above lies in the fact that ours is a method for learning word representations for text classification, and is not a classifier *per se*. Incidentally, this means that WCEs can be used within any classifier that relies on pre-trained embeddings, by simply expanding the initial embedding matrix with WCEs, and without requiring any modification in the architecture of the classifier. We show examples of this in Section 4.4, in which we show how WCEs can lead to substantial improvements when used within different models, including `fastText` and LEAM among other general architectures.

The rationale behind WCE is similar in spirit to the method of (Lei et al., 2019). The authors propose *Task-Oriented Representation* (TOR), that models word-class distributional vectors based on a direct measure of the correlation between the word and each of the classes in the codeframe. However, there are important differences between TOR and WCEs. TOR relies on $P(w_i|c_j)$ (that the authors call *Word Probability* – WP) or on $P(c_j|w_i)$ (that the authors call *Class Probability* – CP) as the correlation measures. However, these metrics are inevitably biased towards the prevalence of word w_i (in the case of WP) or towards the prevalence of class c_j (in the case of CP), while, as will be shown in Section 3, these biases are explicitly factored out in the computation of WCEs. TOR has been tested on small codeframes (with no more than 6 classes) and thus the method does not implement any mechanism to deal with the computational burden that dealing with large codeframes entails. WCEs do instead cater for large codeframes (see Section 3.1), which allows our experiments to be run on datasets containing up to 2,706 classes. Finally, and differently from our WCEs, TOR does not implement any regularization to prevent overfitting the supervised signal; in Section 4.6 we show how properly regularizing the supervised embeddings (see Section 3.3) is fundamental to obtain good results. We compare our WCEs against TOR in the experiments of Section 4.4 and show that these differences bring about significantly higher performance for our method.

3 Method

Let $\mathcal{C} = \{c_1, \ldots, c_m\}$ be the classification scheme (a.k.a. *codeframe*). We consider multiclass classifiers $h : \mathcal{D} \rightarrow \{0, 1\}^m$, mapping documents from a domain $\mathcal{D}$ into vectors of m binary class labels. (The label vector contains a single 1 in single-label multiclass classification ($m > 2$) and in binary classification ($m = 2$), and any combination of 0's and 1's in multilabel multiclass classification.) We are interested in equipping the classifiers with continuous distributed representations of words, i.e., with an embedding function $E : \mathcal{V} \rightarrow \mathbb{R}^r$ (sometimes called the *lookup table*, and often presented simply as a matrix $\mathbf{E}$)

mapping words in the vocabulary $\mathcal{V}$ (the set containing any desired textual feature, e.g., words, stems, word n-grams, or any such surface form of interest) into a dense r-dimensional vector space. Most methods based on distributional semantics learn the mapping E on the basis of how words are distributed in an external corpus $\mathcal{D}'$ of textual data (sometimes huge, and sometimes unrelated to the domain $\mathcal{D}$). We instead investigate task-specific mappings, i.e., mappings specific to the domain $\mathcal{D}$ and codeframe $\mathcal{C}$, based on how words are distributed across classes.

We define the word-class embedding $E(w_i) \in \mathbb{R}^r$ of word w_i with respect to codeframe $\mathcal{C}$ as

$$E(w_i) = \psi(\eta(w_i, c_1), \ldots, \eta(w_i, c_m)) \in \mathbb{R}^r \tag{1}$$

where $\eta : \mathcal{V} \times \mathcal{C} \rightarrow \mathbb{R}$ is a real-valued function that quantifies the correlation between word $w_i \in \mathcal{V}$ and class $c_j \in \mathcal{C}$, and where $\psi : \mathbb{R}^m \rightarrow \mathbb{R}^r$ is any projection function mapping vectors of class-conditional priors into an r-dimensional embedding space. (More details on both η and ψ later on.) The value of $\eta(w_i, c_j)$ can be estimated from a training set of labelled documents $L = \{(x_k, \mathbf{y}_k)\}_{k=1}^{n}$, with x_k the k-th training document and $\mathbf{y}_k \in \{0,1\}^m$ the binary vector indicating the class labels attributed to x_k. We make the default assumption that the same training set L is also used by the supervised learning algorithm for generating the classifier h.

We now detail the embedding generation process using matrix notation. First, we map set L into a matrix $\mathbf{X} \in \mathbb{R}^{n \times |\mathcal{V}|}$, where $\mathbf{X}$ consists of the vectorial representations of the documents in L according to a weighted (e.g., tfidf, or BM25) "bag-of-words" feature model. Note that the step of mapping L into $\mathbf{X}$ is specific to the WCE generation procedure, and imposes no restrictions on the supervised learning method to be used, which may instead rely on a different mechanism for representing documents. Similarly, we create a document-class binary matrix $\mathbf{Y} \in \{0,1\}^{n \times m}$, consisting of the n binary vectors $\mathbf{y}_k \in \{0,1\}^m$. We generate a word-class matrix $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times m}$ as

$$\mathbf{A} = \mathbf{X}_1^\top \mathbf{Y} \tag{2}$$

where $\mathbf{X}_1$ denotes the matrix $\mathbf{X}$ with L1-normalized columns. L1 normalization serves the purpose of factoring out term prevalence (see below).[5] Element a_{ij} of matrix $\mathbf{A}$ thus represents the correlation between the i-th feature and the j-th class across the n labelled documents, as quantified by the dot product.[6]

We may expect a randomly chosen word to show no a priori significant correlation with a randomly chosen class label. We thus want to choose as our η function one that is centered at the expected correlation value (i.e., the value

[5] Put it another way, L1 normalization fixes a "budget" of mass 1 to the score a term can deliver for any class, irrespectively of its prevalence in language or in the corpus.

[6] It is worth recalling that the bag-of-words model tends to produce matrices that are highly sparse. Many software packages take advantage of this sparsity in order to compute matrix multiplication efficiently, at a cost that, in practice, falls far below the asymptotic bound $O(|\mathcal{V}|nm)$. We discuss empirical computational complexity issues in Section 4.7.

of correlation that may simply be explained by chance). In other words, we want η to return positive (resp., negative) values whenever the presence of w_i brings stronger (resp., weaker) evidence that the document is in c_j than this expected value, and close to 0 when the presence of w_i brings no significant evidence about the presence of c_j. A natural way to fulfill this requirement is through *standardizing*. We thus (independently) standardize each of the m dimensions of $\mathbf{A}$ so that the resulting matrix $\mathbf{S}$ is such that the distribution of the elements in its columns has zero mean and unit variance, i.e.,

$$s_{ij} \leftarrow z_j(a_{ij}) = \frac{a_{ij} - \overline{\mu}_j}{\overline{\sigma}_j} \tag{3}$$

where z_j denotes the function that returns standard scores (a.k.a. z-scores) for column j (i.e., for the random variable which takes on values $\{a_{1j}, \ldots, a_{|\mathcal{V}|j}\}$), with sample mean

$$\overline{\mu}_j = \frac{1}{|\mathcal{V}|} \sum_{i=1}^{|\mathcal{V}|} a_{ij} \tag{4}$$

and sample standard deviation

$$\overline{\sigma}_j = \sqrt{\frac{1}{|\mathcal{V}| - 1} \sum_{i=1}^{|\mathcal{V}|} (a_{ij} - \overline{\mu}_j)^2} \tag{5}$$

Note that the resulting random variable which takes on values $\{s_{1j}, \ldots, s_{vj}\}$ is unbiased with respect to the feature prevalence of w_i and the prevalence of class c_j. The reason is that the feature prevalence has been factored out after the L1 normalization of the columns of $\mathbf{X}$, while the class prevalence has become a constant factor for each column in $\mathbf{A}$, and is thus implicitly factored out during standardizing.

Function η is thus

$$\eta(w_i, c_j) = z_j(\mathbf{w}_i^\top \mathbf{c}_j) \tag{6}$$

where $\mathbf{w}_i \in \mathbb{R}^n$ is the L1-normalized column vector of weighted values for word w_i in $\mathbf{X}$ and $\mathbf{c}_j \in \mathbb{R}^n$ is the binary column vector of class c_j in $\mathbf{Y}$. In Section 4.8 we experiment with functions alternative to the dot product as the instantiation of function η, including ones that, unlike the dot product, pay equal attention to positive and negative correlation.

There are additional motivations behind the use of standardizing. On one hand, the zero-mean property establishes the zero-vector as a natural choice for any possible future word not encountered at training time, since the zero-vector would indicate that the word shows no a priori correlation to any of the classes. (Further considerations regarding the treatment of out-of-vocabulary words are discussed in Section 4.11.) On the other hand, unit variance guarantees that all classes contribute approximately equally to the representation, which reinforces the possibility that the downstream classifier performs well on all classes.

For the moment being, let us simply define the projector ψ in Equation 1 to be the identity function (thus forcing r to be equal to m; we will come back to this in Section 3.1); then $\mathbf{S}$ is the resulting WCE matrix. Arranged in rows are the WCEs, that encode how each word is distributed across the classes in the codeframe. The WCE matrix $\mathbf{S}$ can finally be concatenated with any other pre-trained word embedding matrix $\mathbf{U}$ (as those produced by, e.g., `GloVe` or `word2vec`) to define the embedding matrix $\mathbf{E}$.

3.1 Large Codeframes

The necessity of dealing with large codeframes could easily cause the optimization of neural models relying on WCEs to become intractable. The reason is that, in many applications of text classification, hundreds of thousands of features are generated, and the newly added WCEs lie on a (dense) vector space with as many dimensions as classes in the codeframe. In such cases we might want ψ to implement a dimensionality reduction technique, thus mapping m-dimensional vectors into an r-dimensional space, with $r < m$.

In this work we assume ψ to be implemented via principal component analysis (PCA), in order to replace $\mathbf{S}$ with a low-rank approximation of it. In the experiments of Section 4, when dealing with codeframes with $m > 300$ we choose to retain only the 300 principal components with the largest eigenvalues (i.e., those explaining the largest variance); in the literature, 300 is indeed a popular choice for the size of word embeddings. (Somehow abusing notation, and when clear from context, we will use symbol $\mathbf{S}$ to either denote $\mathbf{S}$ *or* its low-rank approximation, assuming the application of PCA to be implicit whenever $m > 300$.)

Alternative ways for implementing ψ might be found in the class of *label-embedding* approaches from the *extreme multilabel text classification* literature (Bhatia et al., 2015; Hsu et al., 2009; Yu et al., 2014) , or more generally in dimensionality reduction techniques (Baldi, 2011; van der Maaten and Hinton, 2008).

3.2 Theoretical Framework

Levy and Goldberg (2014) proved that the optimum that `word2vec`'s SGNS searches for is attained by a variant of a well-known information-theoretic measure, the *pointwise mutual information* (PMI), shifted by a constant factor. In particular, they proved that the minimum of the SGNS's loss function is achieved for word-embeddings $\mathbf{w}_i$ and context-embeddings $\mathbf{q}_j$ that, when stacked in matrix form $\mathbf{W}$ and $\mathbf{Q}$, respectively, define a word-by-context matrix

$$\mathbf{M} = \mathbf{W}\mathbf{Q}^\top \tag{7}$$

whose elements satisfy

$$\mathbf{M}_{ij} = \mathbf{w}_i \cdot \mathbf{q}_j = \mathrm{SPMI}_k(w_i, q_j) = \mathrm{PMI}(w_i, q_j) - \log k \tag{8}$$

for some constant k. That is, SGNS is implicitly factoring a PMI matrix which is *shifted* (SPMI) by a constant factor $\log k$.

Both $\mathbf{M}$ and $\mathbf{W}$ contain distributed representations of words as their rows. Levy and Goldberg (2014) also show that, since word vectors in $\mathbf{M}$ are high-dimensional and dense, it is useful to substitute the correlation function SPMI by one that replaces all negative values with zeros, called *shifted positive* PMI (SPPMI), thus making the matrix become sparse, and then applying PCA to reduce the number of dimensions.[7]

Accordingly, the word embedding $E(w_i)$ for any word w_i can be expressed as

$$E(w_i) = \text{PCA}_r^i(\text{SPPMI}_k(w_i, q_1), \ldots, \text{SPPMI}_k(w_i, q_u)) \tag{9}$$

where u is the number of contexts, SPPMI_k is the correlation function between words and contexts, and PCA_r^i is the function returning the ith row from the r-dimensional decomposition $\mathbf{U}_r\mathbf{\Sigma}_r$ of $\mathbf{M}$.

Note that WCEs smoothly fit within this framework. The word-class matrix $\mathbf{A}$ from Equation 2 is analogous to the word-by-context matrix $\mathbf{M}$ from Equation 7, with the contexts being defined by all the documents that share a label. Matrices $\mathbf{A}$ and $\mathbf{M}$ can both be expressed as the product of a word matrix by a context matrix. In this respect, SGNS learns dense and latent representations for words and contexts in matrices $\mathbf{W}$ and $\mathbf{Q}$, respectively, while WCE directly uses the sparse columns of $\mathbf{X}_1$ and $\mathbf{Y}$ as the distributional vectors.[8]

The analogy between Equations 1 and 9 is immediate when considering the contexts to be defined by the labels, PCA_r^i to be an instantiation of a generalized dimensionality reduction function ψ, and SPPMI_k to be an instantiation of a generalized correlation function η.

The variant we propose in Equation 6 has one further advantage with respect to SPPMI, i.e., that it takes into consideration the relative importance of the word in the document (as quantified by the tfidf weights of matrix $\mathbf{X}$) instead of simply looking at the presence/absence of the word in the document. Note also that the number of classes in a codeframe is expected to be much smaller than the number of possible contexts typically allocated in the $\mathbf{M}$ matrix, and thus imposing sparsity is not necessary in our case. In Section 4.8 we compare our correlation measures with others, including PPMI.

Summing up, while unsupervised word embeddings capture distributional information about how words relate to other words by mining their contexts

[7] PCA is based on (truncated) Singular Value Decomposition (SVD). The SVD of a matrix $\mathbf{M}$ is a factorization of the form $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, in which $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices containing the left- and right-eigenvectors of $\mathbf{M}$ as their columns, respectively, and $\mathbf{\Sigma}$ is a diagonal matrix containing the eigenvalues of $\mathbf{M}$. That is, PCA is an alternative way of factoring $\mathbf{M}$ w.r.t. Equation 7. The dimensionality reduction is achieved by ordering the components by decreasing order of eigenvalues, and truncating the matrices. The optimal rank r approximation of $\mathbf{M}$ is thus given by $\mathbf{U}_r\Sigma_r$ which only accounts for the r largest eigenvalues and their corresponding r left-eigenvectors.

[8] Note that the columns of $\mathbf{Y}$ are binary, indicating the presence or absence of the label for each document. It is interesting to look at $\mathbf{Y}$'s binary columns as indicator functions that decide which elements from $\mathbf{X}_1^\top$ rows contribute to the summation in the dot product.

of usage, WCEs capture distributional information about how words relate to the classes by leveraging the data labels. The former is useful for capturing general word semantics, while the latter is useful for capturing task-dependent word semantics. Adding WCEs makes thus explicit to the model information which is relevant for the classification task, so that the model is released from discovering it by itself during the learning process.

3.3 Regularization

WCEs inject a task-specific pressure into the representation mechanism that might compromise the data-generating process of training and test documents, since, unlike when using pre-trained embeddings, words from the training documents have played a role in the generation of WCEs. Indeed, during preliminary experiments we observed that models operating with a concatenation of WCEs and pre-trained word-embeddings sometimes incur a much lower training loss than those using pre-trained word-embeddings only, but the former tend to perform substantially worse on unseen data (more details on this in Section 4.6). This case of overfitting makes evident the need for properly regularizing the model.

In order to perform regularization, we apply a variant of dropout (Srivastava et al., 2014) to the embedding layer. Dropout consists of zeroing random activations in order to prevent nodes from co-adapting. Since dropout is only applied in the training phase, the activation values are scaled by $(1-p)^{-1}$ during training, with p the drop probability, in order to keep the expected activation consistent with the test phase.

Supervised dropout applies only to the WCEs. (Cases in which we do not apply supervised dropout, or in which we instead apply dropout to the entire embedding layer, are discussed later on as well.) Let $\mathbf{E} = [\mathbf{U} \oplus \mathbf{S}] \in \mathbb{R}^{|\mathcal{V}| \times (q+r)}$ represent the entire embedding layer, consisting of the concatenation (here denoted by the $\oplus$ operator) of the unsupervised q-dimensional matrix $\mathbf{U}$ and the supervised r-dimensional matrix $\mathbf{S}$. In order to bring to bear correct expected activations during test, we compute the scaling at training time as[9]

$$D(\mathbf{E}) = \frac{[\mathbf{U} \oplus (1-p) \cdot d(\mathbf{S})]}{1 - \frac{pr}{q+r}} \tag{10}$$

where d indicates *dropout* and D indicates *supervised dropout*.

Finally, note that, as can be appreciated from the description above, our method for generating WCEs is conceptually straightforward, since it comes down to the matrix multiplication of Equation 2, followed by the pass of standardization described in Equation 3, optionally followed by dimensionality reduction (described in Section 3.1), optionally followed by the application of dropout that we have just described. Yet, as we will empirically show in Section 4, it is surprisingly effective.

[9] Since we undertake a stochastic optimization, this actually applies to *batches* of data.

4 Experiments

In this section we describe the experiments that we have carried out in order to quantify the contribution of WCEs to multiclass text classification. In order to make all the experiments discussed in this paper fully reproducible, we make available at `https://github.com/AlexMoreo/word-class-embeddings` the code that implements our method and all the baselines used in this work.

4.1 Datasets

In our experiments we use the following six publicly available datasets:

- Reuters-21578 is a popular multilabel dataset which consists of a set of 12,902 news stories, partitioned (according to the "ModApté" split we adopt) into a training set of 9,603 documents and a test set of 3,299 documents.[10] In our experiments we restrict our attention to the 115 classes with at least one positive training example. This dataset presents cases of severe imbalance, with many classes containing fewer than 5 positive examples.
- 20Newsgroups is a single-label test collection of approximately 20,000 posts on Usenet discussion groups, nearly evenly partitioned across 20 different newsgroups (classes).[11] In this article we use the "harder" version of the dataset, i.e., the one from which all metadata (headers, footers, and quotes) have been removed.[12]
- Ohsumed (Hersh et al., 1994) is a dataset consisting of a set of MEDLINE documents spanning the years from 1987 to 1991.[13] Each entry consists of summary information relative to a paper published on one of 270 medical journals. The available fields are title, abstract, MeSH indexing terms, author, source, and publication type. Following (Joachims, 1998), we restrict our experiments to the set of 23 cardiovascular disease classes, and we use the 34,389 documents of year 1991 that have at least one of these 23 classes. Since no standard training/test split has been proposed in the literature we randomly partition the set into a part to be used for training (70% of the documents) and a part to be used for testing (the other 30%).

[10] `http://www.daviddlewis.com/resources/testcollections/reuters21578/`

[11] `http://qwone.com/~jason/20Newsgroups/`. Note that this version of 20Newsgroups is indeed single-label: while a previous version contained a small set of document with more than one label (corresponding to posts that had been cross-posted to more than one newsgroup), that set is not present in this version we use.

[12] While some previous papers (e.g., (Tang et al., 2015)) have reported substantially higher scores for this dataset, it is worth noticing that we use a harder, more realistic version of the dataset than has been used in those papers. Following (Moreo et al., 2020), in our version we remove all headers, footers, and quotes, since these fields contain words that are highly correlated with the target labels, thus making the classification task unrealistically easy; see `http://scikit-learn.org/stable/datasets/twenty_newsgroups.html` for further details. Our results are indeed consistent with other papers following the same policy.

[13] `http://disi.unitn.it/moschitti/corpora.htm`

- RCV1-v2 is a dataset comprising 804,414 news stories published by Reuters from Aug 20, 1996, to Aug 19, 1997; for text classification purposes it is traditionally split into a training set consisting of the (chronologically) first 23,149 documents (the ones written in Aug 1996), and a test set consisting of the last 781,265 documents (the ones written from Sep 1996 onwards).[14] In our experiments we use this dataset in its entirety, and stick to the standard training/test split described above. RCV1-v2 is multilabel, i.e., a document may belong to several classes at the same time. Of the 103 classes of which its "Topic" hierarchy consists, in our experiments we restrict our attention to the 101 classes with at least one positive training example. This dataset is the one with the largest test set in our experiments.
- JRC-Acquis (version 3.0) is a collection of legislative texts of European Union law written between the 1950s and 2006 (Steinberger et al., 2006).[15] JRC-Acquis is publicly available for research purposes, and covers 22 official European languages. We restrict our attention to the English subset, which consists of 20,370 documents. For our experiments, we consider the 13,137 documents written in the [1950, 2005] interval as the training set, and leave the remaining 7,233 documents written in 2006 as the test set. The dataset is multilabel and is labelled according to the EuroVoc thesaurus. We focus on the 2,706 classes with at least one positive element in the training set. This dataset is the one with the largest codeframe in our experiments.
- WIPO-gamma is a test collection of patent documents.[16] Documents are labelled according to the International Patent Classification (IPC) taxonomy, covering patents and patent applications in all areas of technology. We focus on the single-label version labelled at the subclass level in the IPC hierarchy. For our experiments, we extract the abstract field of the documents (thus discarding the list of inventors, list of applicant companies, claims, and the long description), and follow the train/test split made available by the WIPO organization. The dataset contains a total of 1,118,299 documents, of which 896,363 (80%) is used as the training set, and the remaining 221,936 (20%) are used for test. This dataset is the one with the largest training set in our experiments.

Details of these datasets are given in Table 1. Note that the datasets chosen cover a broad spectrum of experimental conditions, including single-label and multilabel scenarios, a number of classes ranging from tens (20Newsgroups) to thousands (JRC-Acquis), a number of documents from small (Reuters-21578) to very large (WIPO-gamma), from well balanced datasets (20Newsgroups) to severely imbalanced ones (e.g., RCV1-v2), etc. Note also that the

[14] Available from `http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm`

[15] `https://ec.europa.eu/jrc/en/language-technologies/jrc-acquis`

[16] `https://www.wipo.int/classifications/ipc/en/ITsupport/Categorization/dataset/`

Dataset	Type	# Classes	# Train docs	# Test docs	Total # of docs	Vocabulary	OOV	# Words	Prev(mean)	Prev(std)	Prev(min)	Prev(max)
Reuters-21578	ML	115	9,603	3,299	12,902	8,250	24,094	1.7M	83.9	314.3	1	2,877
20Newsgroups	SL	20	11,314	7,532	18,846	17,184	112,594	3.5M	565.7	56.8	377	600
Ohsumed	ML	23	24,061	10,328	34,389	18,238	44,062	6.2M	1,734.3	1,523.6	301	6,729
RCV1-v2	ML	101	23,149	781,265	804,414	24,816	384,327	188.1M	709.9	1,417.2	1	10,282
JRC-Acquis	ML	2,706	13,137	7,233	20,370	21,109	141,800	28.3M	25.7	58.6	2	1,151
WIPO-gamma	SL	613	896,363	221,936	1,118,299	114,802	395,570	417.8M	1,462.3	4,354.6	1	63,465

Table 1 Details of the datasets we use in this research. Column "Type" indicates whether the classification is multilabel (ML) or single-label (SL). Column "Vocabulary" shows the number of words occurring at least 5 times in the training set, while Column "OOV" shows the number of words occurring in fewer than 5 training documents or exclusively in test documents. Columns with prefix "Prev" indicate a few statistics about class prevalence in the training set.

first four datasets (Reuters-21578, 20Newsgroups, Ohsumed, RCV1-v2) are probably the most popular datasets in text classification research; together with the fact that they are all publicly available, this guarantees a high level of comparability (and interpretability) to our results. In our experiments we concentrate our attention on the classification by topic of long texts, and leave other dimensions of text classification (e.g., classification by sentiment, sentence classification, classification of short texts) for future work (see also the discussion in Section 5.3).

We pre-process text via the default analyzer available in the `scikit-learn` framework[17] (which applies lowercasing, stop word removal, punctuation removal), and by masking numbers with a dedicated token. For the computation of the WCEs, we retain all words (unigrams) appearing at least 5 times in the training set. However, in experiments involving pre-trained embeddings we also consider those out-of-vocabulary (OOV) words for which a pre-trained embedding exists; these words are represented by the zero vector in the WCE space (see Section 3). This is a major advantage of also using pre-trained embeddings, which allow neural models to also make sense (at testing time) of words unseen at training time if they have anyway been encountered during the pre-training phase (see Section 4.11 for more on this).

4.2 Evaluation Measures

As the effectiveness measure we use F_1, the harmonic mean of precision (π) and recall (ρ), defined as $F_1 = (2\pi\rho)/(\pi+\rho) = (2\text{TP})/(2\text{TP}+\text{FP}+\text{FN})$, where TP, FP, FN, are the numbers of true positives, false positives, false negatives, from the binary contingency table. We take $F_1 = 1$ when $\text{TP} = \text{FP} = \text{FN} = 0$, since the classifier has correctly classified all examples as negative.

As defined above, F_1 is a measure for binary classification only. For multiclass classification, we average F_1 across all the classes of a given codeframe

[17] `http://scikit-learn.org/`

by computing both micro-averaged F_1 (denoted by F_1^μ) and macro-averaged F_1 (denoted by F_1^M). F_1^μ is obtained by (i) computing the class-specific values TP_j, FP_j, and FN_j, (ii) obtaining TP as the sum of the TP_j's (same for FP and FN), and then applying the F_1 formula. F_1^M is obtained by first computing the class-specific F_1 values and then averaging them across the classes.

Regarding these evaluation measures, we will often prefer to report the relative improvement (expressed as a percentage) that a method equipped with WCEs obtains with respect to the same method not using WCEs. The relative improvement of method A over method B in terms of evaluation metric E is obtained as

$$\text{RI}(A, B, E) = \frac{E(A) - E(B)}{E(B)} \tag{11}$$

4.3 Supervised Learners for Classifier Training

We test the contribution of WCEs to text classification using a "traditional" (i.e., non-neural), high-performance learner (support vector machines), two supervised deep learning classifiers (LEAM and `fastText`), and three popular architectures based on deep neural networks (convolutional neural networks, long-short term memory networks, and attention models). [18]

For each such system we explore different variants, corresponding to different ways of instantiating the embedding matrix $\mathbf{E}$. As the pre-trained embeddings we use the biggest set of `GloVe` vectors made available, and consisting of 2.2M 300-dimensional word embeddings generated from a text corpus of 840 billion tokens.[19] (In Section 4.9 we report results of using differently characterized embeddings beyond `GloVe` embeddings.) The variants we explore are the following:

`Random`: *randomly* initialized *trainable* embeddings (their dimensionality is optimized from the range $\{50, 200, 300\}$ on a validation set).

`GloVe(static)`: *static* pre-trained `GloVe` embeddings.

`GloVe(trainable)`: *trainable* vectors initialized with pre-trained `GloVe` embeddings.

`GloVe+Random`: *trainable* embeddings initialized as the concatenation of pre-trained `GloVe` embeddings and *random* embeddings. The random embeddings are chosen to have the same dimensionality as WCEs. This configuration serves for control purposes, in order to ensure that any possible relative improvement brought about by the use of WCEs cannot merely be attributed to the presence of more parameters in the embedding layer. For the sake of comparability we thus apply supervised dropout to the randomly initialized part of the embedding. This serves the purpose of

[18] Note that these deep models are here not meant to be used as baselines, but to serve as vehicles on which to test WCEs. In other words, the actual baseline for any model equipped with WCEs is the same model not using WCEs.

[19] `http://nlp.stanford.edu/data/glove.840B.300d.zip`

rejecting the possibility that any improvement we measure for WCEs is actually explainable in terms of regularization only.

`GloVe+WCEs(static)`: *static* concatenations of pre-trained `GloVe` embeddings and WCEs.

`GloVe+WCEs(trainable)`: *trainable* embeddings initialized with the concatenation of pre-trained `GloVe` embeddings and WCEs.

We perform hyperparameter search via grid-search on the validation set, independently for each combination of type ⟨dataset, architecture, variant⟩.[20] The hyperparameters to be optimized are dependent on the architecture, and are explained in the sections below.

4.3.1 Support Vector Machines with asymmetric costs

The problem of training a classifier via SVMs with asymmetric costs is stated as the empirical risk minimization problem (Cortes and Vapnik, 1995; Morik et al., 1999)[21]

$$\begin{aligned} \text{minimize:} \quad & \frac{1}{2}||\mathbf{w}||^2 + C_+ \sum_{k=1}^{n} \xi_k[y_k = +1] + C_- \sum_{k=1}^{n} \xi_k[y_k = -1] \\ \text{over:} \quad & \mathbf{w}, b, \xi_1, \ldots, \xi_n \\ \text{subject to:} \quad & \forall_{k=1}^{n} : y_k(\mathbf{w} \cdot \mathbf{x}_k + b) \geq 1 - \xi_k \\ & \forall_{k=1}^{n} : \xi_k > 0 \end{aligned}$$

where $\mathbf{w}$ and b are the parameters (hyperplane and bias) of the separation functional, ξ_k are the slack variables for the labelled examples (x_k, y_k), $[\cdot]$ is the indicator function that returns 1 if its argument is true and 0 if it is false, and C_+ and C_- are two hyperparameters that control the trade-off between training error and margin for positive and negative examples, respectively (Morik et al., 1999). It is useful to factor C_+ and C_- as $C_+ = CJ_+$ and $C_- = CJ_-$, so that the asymmetric cost factors are confined to two dedicated hyperparameters, with J_+ (resp., J_-) controlling the amount by which training error on positive examples (resp., negative examples) outweighs error on the negatives (resp., positives). We follow (Morik et al., 1999) and set the cost factors so that the ratio J_+/J_- equals the ratio N/P between the number N of negative training examples and the number P of positive training examples.[22] The trade-off between training error and margin, now confined to C, becomes the only hyperparameter we tune. Note that, while in other application fields

[20] We generate the validation set by randomly sampling 20% of the training set, with a maximum of 20,000 documents; the rest is taken to be the training set proper. We keep the training/validation split consistent across all methods.

[21] Note that, consistently with (Cortes and Vapnik, 1995; Morik et al., 1999), in this formulation we assume the class labels y_k to be in $\{-1, +1\}$, while in Section 3 we had assumed them to be in $\{0, 1\}$; the difference is, of course, unproblematic.

[22] In `scikit-learn` this is achieved by setting $J_+ = n/(mP)$ and $J_- = n/(mN)$, and corresponds to setting the parameter `class_weight` to "balanced".

the kernel to employ is considered an important parameter to optimize, in text classification it is customary to employ the linear kernel, since there are theoretical arguments for its optimality in these contexts (Joachims, 2001); we thus use the linear kernel without further ado.

Although the application of SVMs to text classification dates back to (Dumais et al., 1998; Joachims, 1998), SVMs are still considered among the strongest baselines for text classification.[23] Having set the asymmetric cost factors, the most influential hyperparameter for SVMs is C. We choose the best value for C from the set $\{10^{-3}, 10^{-2}, \ldots, 10^{+3}\}$ by performing 5-fold cross-validation on the full set of labelled documents (i.e., the training set proper plus the validation set); we perform the optimization of C independently for each class.[24] In the experiments of this paper we use the implementation of SVMs available in `scikit-learn`.[25] We leave the rest of the parameters set to their default values.

The field of supervised word embeddings in deep learning has some connections with supervised term weighting (STW) functions in bag-of-words models (Debole and Sebastiani, 2003) (see Section 5.1 for a broader discussion). Although STW has not shown consistent improvements over unsupervised functions such as tfidf, we include them in our experimentation for completeness. In brief, STW schemes replace the "idf" factor of tfidf by a factor based on the class-conditional distribution of the word, and apply some aggregation function to combine the different scores obtained for each class. We report experiments using ConfWeight (Soucy and Mineau, 2005) as our STW function, with MaxPooling as the aggregation function, since (in experiments that we do not report for the sake of brevity) this turned out to be the top-performing strategy from a pool of 6 popular STW functions[26] and 3 aggregation functions[27].

Since SVMs do not cater for word embedding fine-tuning, we only report experiments involving sets of static embeddings. Specifically, for SVMs we report the following experiments:

[23] Somehow surprisingly, though, several relevant related works where SVMs are used as baselines (see, e.g., (Grave et al., 2017; Jiang et al., 2018; Zhang et al., 2015)) do not report the details of how, if at all, they tune the SVM hyperparameters.

[24] Using k-fold cross-validation (k-FCV) on the full set of labelled documents is a more expensive, but stronger, way of doing parameter optimization than using a single split between a training set and a validation set, because k-FCV performs k such splits. We here use k-FCV for SVMs and single-split optimization for all the other deep learning -based architectures because it is realistic to do so, i.e., because SVMs are computationally cheap enough for us to be able to afford k-FCV, while neural architectures are not.

[25] This implementation relies on `liblinear`. See `https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html` for further details.

[26] The STW functions we have considered include chi-square, information gain, gain ratio, pointwise mutual information (Debole and Sebastiani, 2003), ConfWeight (Soucy and Mineau, 2005), and relevance frequency (Lan et al., 2009).

[27] Given a word w, a codeframe $\mathcal{C} = \{c_1, \ldots, c_m\}$, and a STW functions f that generates a list of scores $S = (f(w, c_1), \ldots, f(w, c_m))$, we consider the following aggregation functions: averaging $\left(\frac{1}{m}\sum_{c\in\mathcal{C}} f(w,c)\right)$, averaging weighted by class prevalence $\left(\frac{\sum_{c\in\mathcal{C}} f(w,c)p(c)}{\sum_{c\in\mathcal{C}} p(c)}\right)$ where $p(c)$ is the prevalence of class c, and max-pooling $(\max_{c\in\mathcal{C}}\{f(w,c)\})$.

`SVM-tfidf`: a SVM-based classifier trained on the tfidf matrix $\mathbf{X}$ defined in Section 3;
`SVM-tfcw`: a SVM-based classifier trained on the STW matrix generated via ConfWeight and MaxPooling (as discussed above);
`SVM-GloVe`: a SVM-based classifier trained on the projection $\mathbf{XU}$, with $\mathbf{U}$ the pre-trained `GloVe` embeddings;
`SVM-GloVe+WCEs`: a SVM-based classifier trained on the projection $\mathbf{XE}$, with $\mathbf{E} = [\mathbf{U} \oplus \mathbf{S}]$ the concatenation of pre-trained `GloVe` embeddings and WCEs.
`SVM-BERT-finetuned`: a SVM-based classifier trained on the document embeddings produced by fine-tuned BERT (more on this in Section 4.3.2).
`SVM-BERT-finetuned+WCEs`: a SVM-based classifier trained on the concatenation of document embeddings produced by fine-tuned BERT and the projection $\mathbf{XS}$, where $\mathbf{S}$ is the WCE matrix introduced in Section 3.

4.3.2 BERT fine-tuning

The paper that introduces the BERT language model (Devlin et al., 2019) describes its use in supervised tasks as a two-step process. The first step consists of the pre-training phase, in which the actual language model is fit on very large amounts of text via a costly training process. The second step consists of the fine-tuning phase, in which the pretrained BERT model is extended via a few task-specific parameters. The entire network is trained, via a much faster process than the one used in pre-training, on a task-specific training set, thereby *fine-tuning* all the BERT parameters, plus the task-specific ones, on the task of interest. In the case of classification the additional parameters consist of a single dense layer that takes as input the embedding returned by BERT for "[CLS]", a token that is always appended at the beginning of any input text with the specific purpose of serving as a representation of the content of the entire text. This layer returns a probability distribution over the labels.

The fine-tuning process is computationally cheaper than the pre-training process, because most of the model has already been fit, and the computation required to adapt it to the supervised task is usually limited. This fact allows to add the rich amount of general knowledge about language captured during the pre-training process (ideally to be run once for all), to many downstream supervised tasks. Note that the computational cost of fine-tuning the model is still orders of magnitude higher that any other learning approach used in our study, since the number of parameters involved in the fitting process is very large (110 millions for the $\text{BERT}_{\text{base}}$ model we used).

The BERT fine-tuning approach has obtained state-of-the-art results on many supervised tasks; for this reason we include it in our experiments. We use `simpletransformers`[28] and the `BERT-base-uncased model`[29], which has been pre-trained by Devlin et al. (2019) on a collection of English text (con-

[28] `https://github.com/ThilinaRajapakse/simpletransformers`
[29] `https://huggingface.co/bert-base-uncased`

sisting of Wikipedia and a large collection of books) for a total of 3,300M word occurrences, requiring four days to be fit on 16 TPUs.

We also use BERT in a different set of experiments (see Section 4.9), in which we experiment with the contextualized word embeddings that BERT generates for each token in the document (in contrast to only using the "[CLS]" token).

4.3.3 LEAM

We report experiments for LEAM (Wang et al., 2018) (discussed in Section 2.2).

LEAM works by computing a compatibility matrix $\mathbf{G}$ in which element $\mathbf{G}_{ij}$ represents the compatibility between the ith label and the jth word in the input sequence, as quantified via the cosine similarity between the corresponding label embedding and word embedding.

The compatibility scores are then convolved so as to bring to bear word order dependencies into the model. A max-pooling and a softmax operators are later used to convert the compatibility scores into an array of attention coefficients that scale each word in the input. The document embedding is obtained by averaging the (scaled) word embeddings, and the classification scores are then computed by a final feed-forward layer.

For our experiments we use LEAM's official implementation[30], that we modify to adopt early-stopping as the convergence criterion. We leave all the hyperparameters set to their default values. We explore all combinations involving training vectors (`Random`, `GloVe(trainable)`, `GloVe+Random`, and `GloVe+WCEs(trainable)`), since LEAM requires that the embedding matrix be trainable.

4.3.4 fastText

We report experiments for `fastText`[31] (Bojanowski et al., 2017; Grave et al., 2017), a system which we have already discussed in Sections 1 and 2.2.

We select hyperparameters via grid-search optimization on the validation set. Following (Bojanowski et al., 2017; Grave et al., 2017), we let the learning rate vary in $\{0.05, 0.10, 0.25, 0.50\}$ and the number of epochs in $\{5, 10\}$. Some of our datasets (20Newsgroups, JRC-Acquis, Ohsumed, and Reuters-21578) are comparatively smaller than those tested in the original article; in those cases we let the number of epochs vary on $\{5, 50, 100, 200\}$ (we observed drastic improvements when using 100 or 200 epochs, but no further improvement when going beyond 200).

In our experiments we only consider the case in which `fastText` operates with unigram features. While `fastText` has been found to deliver better results

[30] `https://github.com/guoyinwang/LEAM`

[31] Note that by `fastText` we here mean its "supervised" mode, that is, `fastText` as a classifier. The set of embeddings that `fastText` produces when working in "unsupervised mode" are later used and discussed in Section 4.9, along with other sets of embeddings.

when using bigrams, the adoption of bigrams would likely lead to similar improvements for the rest of the methods being compared (Wang and Manning, 2012), and might only blur the purpose of this comparison.

For `fastText` we only report experiments involving sets of trainable vectors (namely, the combinations `Random`, `GloVe(trainable)`, `GloVe+Random`, and `GloVe+WCEs(trainable)`), since `fastText` is implicitly a method for learning embeddings, and thus does not cater for static vectors. In the case of `GloVe+WCEs(trainable)`, we do not apply supervised dropout, since we stick to the official implementation.[32]

4.3.5 Task-Oriented Convolutional Neural Networks

We include TCNN, a method proposed in (Lei et al., 2019) that uses TOR embeddings within a CNN. In particular, we report experiments using TCNN-WP (i.e., using the so-called *Word Probability* metric as the word-class correlation function) since it yielded best results in the experiments of (Lei et al., 2019).

TCNN-WP consists of a CNN operating with pre-trained embeddings concatenated with TOR embeddings. Since we also use CNN to test our WCEs (see Section 4.3.6), we use the same configuration in the interest of a fair comparison. Regarding the configuration tested in (Lei et al., 2019), the one we use differs only in the set of pre-trained embeddings (we use `GloVe` embeddings instead of `word2vec` embeddings), the window length of the filters (we used filters of sizes $\{3, 5, 7\}$ while they used sizes $\{2, 3, 4, 5\}$), and the optimizer (we use Adam instead of Adagrad). Any other hyperparameter is optimized as described in Section 4.3.6. TCNN-WP corresponds to CNN-`GloVe+WP(trainable)` in our nomenclature.

4.3.6 General-Purpose Deep-Learning Architectures

Since our goal is to quantify the relative improvement (if any) brought about by concatenating WCEs to pre-trained embeddings, we adopt the most general and simple formulation for each of our three deep-learning architectures, and leave the exploration of more sophisticated models for future research.

Let $(x_k, \mathbf{y}_k)$ be a training instance, with $x = [w_{1k}, \ldots, w_{lk}]$ a document consisting of a sequence of l words (padded where necessary) labelled with $\mathbf{y}_k \in \{0, 1\}^m$. Let $\mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times r}$ be the embedding matrix, containing $|\mathcal{V}|$ r-dimensional word embeddings $\mathbf{e_i} \in \mathbb{R}^r$ (where embeddings may either be initialized randomly, using pre-trained embeddings, or concatenations of pre-trained and supervised embeddings). All the tested architectures use an embedding layer as the first layer on the network, which transforms every input document x_k into $\mathbf{x}_k = [E(w_{1k}), \ldots, E(w_{lk})]$, where $E(w_{ik})$ is the word embedding in $\mathbf{E}$ for word w_{ik}. Different models implement different transformations

$$\mathbf{o}_k = N(\mathbf{x}_k; \Theta) \tag{12}$$

[32] `https://fasttext.cc/`

of the input (as defined below) parameterized by Θ, with $\mathbf{o}_k$ denoting the document embedding of x_k. Finally, $\mathbf{o}_k$ is mapped into the space of label outputs by

$$\hat{\mathbf{y}}_k = f(\mathbf{o}_k \cdot \mathbf{W} + \mathbf{b}) \tag{13}$$

where $\mathbf{W}$ and $\mathbf{b}$ are the parameters (weight and bias) of an affine transformation, f is a non-linear function, instantiated as the softmax function for single-label problems or as the sigmoid function for multilabel problems, and $\hat{\mathbf{y}}_k \in [0,1]^m$ is a vector of m predicted posterior probabilities, one for each class. The full set of model parameters to optimize is thus $\Theta' = [\Theta; \mathbf{W}, \mathbf{b}]$.

We initialize the model parameters using the *Xavier uniform* method described in (Glorot and Bengio, 2010). We then train the model by backprogating the errors, where error is computed as the *cross-entropy* loss (in the single-label case) or as the *binary cross-entropy* loss (in the multilabel case). We carry out optimization via stochastic gradient descent with the Adam update rule (Kingma and Ba, 2015). We set the learning rate to 1e-3 and the batch size to 100 documents, dynamically padding the sequences to $l = \min\{500, l_{max}\}$, where l_{max} is the length of the longest document within the batch.

We train the models for a maximum of 200 epochs, but we apply early stopping whenever 10 consecutive training epochs do not yield any improvement in the validation set in terms of F_1^M. An epoch consists of a full pass over all the training documents. Since WIPO-GAMMA is one order of magnitude larger than the other datasets, we consider an epoch to be over after 30,000 documents (300 batches) have been processed. We dump the model parameters whenever the value of F_1^M on the validation set improves. When the training epochs are over, we restore the best model parameters and perform one final training epoch on the validation set.

We consider the following network architectures as alternative implementations of the transformation N of Equation 12:

– *Convolutional Neural Networks.* Convolutional Neural Networks (CNN) are a special type of neural models particularly suited for computer vision, that apply convolved filters which are robust to position-invariant patterns. In text-related applications (Collobert et al., 2011; Kim, 2014) a convolution is the result of the application of a linear filter to a matrix consisting of the w word embeddings corresponding to the words that appear in a sliding window of length w, in order to produce a feature map.
A convolutional layer typically contains and applies many filters, each of which is followed by a non-linear activation function (typically: the *rectified linear unit* $ReLU(x) = \max\{0, x\}$, which is the one we use here) and a max-pooling operation that takes the maximum value for each filter. The result is a thus a vector with as many features as there are filters.
We consider one single convolutional layer (Le et al., 2018) with γ output channels for each window length $w \in \{3, 5, 7\}$ (i.e., 3γ output channels in total), where γ is a hyperparameter to be optimized on a validation set. We let γ vary in the range $\{64, 128, 256, 512\}$. The final representation is

a vector $\mathbf{o}_k \in \mathbb{R}^{3\gamma}$, which concatenates all convolved outputs, followed by the application of a dropout operator.

– *Long-Short Term Memory Networks.* Recurrent Neural Networks (RNNs) (Lai et al., 2015; Rumelhart et al., 1986) are a family of network architectures specially devised for processing sequential data. RNNs apply the same computation to each input in the sequence. The internal state $\mathbf{h}_{ik}$ at time i is defined recursively as $\mathbf{h}_{ik} = f(\mathbf{h}_{(i-1)k}, E(w_{ik}); \Theta)$, with $E(w_{ik})$ the embedding of word w_{ik} and Θ parameterizing the recurrent function. The model is trained by *Backprogragation Through Time* (BPTT) via unfolding the recursive computation and sharing the parameters Θ across all time steps. In this work we adopt the well-known *Long-Short Term Memory* (LSTM) (Hochreiter and Schmidhuber, 1997) as the recurrent cell. We apply gradient clipping at ± 0.1 in order to avoid exploding gradients. The size γ of the hidden state is a hyperparameter of the model, to be optimized on a validation set from the range $\{256, 512, 1024, 2048\}$. The output $\mathbf{o}_k \in \mathbb{R}^{\gamma}$ is the final state $\mathbf{h}_{lk}$ produced by the LSTM.

– *Attention Models.* Attention models (ATTNs) implement criteria that enable the model to weight differently (i.e., to pay different attention to) the contribution of intermediate factors in certain computations. Although attention mechanisms by their own (Vaswani et al., 2017) constitute nowadays an entire family of deep neural models, called *transformers* (Devlin et al., 2019; Peters et al., 2018; Yang et al., 2019b), we focus on a simpler formulation, called *soft-scaled dot-product attention mechanism* (Luong et al., 2015).
This attention mechanism takes all hidden states $\mathbf{H}_k = [\mathbf{h}_{1k}, \ldots, \mathbf{h}_{lk}]$ produced by a RNN (we use the LSTM here as well) and the document embedding $\mathbf{o}_k = \mathbf{h}_{lk}$, and computes a vector of attention weights over all intermediate states, i.e.,

$$\mathbf{a}_k = \text{softmax}(\mathbf{o}_k^{\top} \mathbf{H}_k) \tag{14}$$

and produces a new output $\mathbf{o}' \in \mathbb{R}^{\gamma}$ as a weighted sum

$$\mathbf{o}'_k = \sum_{a_{ik} \in \mathbf{a}_k} a_{ik} \mathbf{h}_{ik} \tag{15}$$

As for LSTM, for the hidden layer γ we choose the size from the range $\{256, 512, 1024, 2048\}$ that performs best on the validation set.

4.4 Results

Tables 2 and 3 report the F_1^M and F_1^{μ} results we have obtained. Since neural architectures use a random inizialization of the parameters, our results for them are averages across 10 runs. Most of the SVM-based configurations we use here (more precisely: all but `SVM-BERT-finetuned` and `SVM-BERT-finetuned+WCEs`) are deterministic, and thus excluded from the test of statistical significance, which requires the repetition of random trials. In all cases we also report

the results of paired sample, two-tailed t-tests at different confidence levels ($\alpha = 0.05$ and $\alpha = 0.005$) in order to assess the statistical significance of the differences in performance as measured by the averaged results.

model	variant	20Newsgroups	JRC-Acquis	Ohsumed	RCV1-v2	Reuters-21578	WIPO-gamma
SVM	tfidf	.670	**.397**	.673	.584	.636	—
	tfcw	.667	.391	.669	.595	.629	—
	GloVe(static)	.635	.277	.505	.495	.531	—
	GloVe+WCEs(static)	.678	.291	.629	.493	.556	—
	BERT-finetuned (static)	.665 ±.004	.389 ±.004	.694 ±.003	**.695** ±.003	**.652** ±.012	—
	BERT-finetuned+WCEs (static)	**.707** ±.004	.325 ±.010	.725† ±.002	.680 ±.002	**.652** ±.011	—
BERT	BERT-finetuned (trainable)	.684 ±.003	.141 ±.004	**.731** ±.004	.685 ±.018	.404 ±.022	.357 ±.007
TCNN	GloVe+WP(trainable)	.691 ±.002	.285 ±.006	.689 ±.004	.474 ±.007	.528 ±.013	.428 ±.016
LEAM	Random	.653 ±.009	.320 ±.003	.619 ±.009	.482 ±.009	.510 ±.024	.519 ±.010
	GloVe(trainable)	.686 ±.004	.362 ±.005	.690 ±.004	.575 ±.008	.580 ±.019	.518 ±.011
	GloVe+Random	.685 ±.003	.361 ±.005	.680 ±.004	.565 ±.007	.571 ±.023	.526 ±.018
	GloVe+WCEs(trainable)	.695 ±.003	.390 ±.004	.696 ±.003	.569 ±.007	.584 ±.015	.543 ±.009
fastText	Random	.614 ±.003	.312 ±.006	.609 ±.002	.523 ±.001	.511 ±.009	.513 ±.002
	GloVe(trainable)	.640 ±.003	.330 ±.005	.627 ±.002	.548 ±.001	.529 ±.008	.547 ±.003
	GloVe+Random	.640 ±.003	.331 ±.004	.624 ±.002	.548 ±.001	.528 ±.008	.547 ±.002
	GloVe+WCEs(trainable)	.693 ±.002	.348 ±.005	.655 ±.002	.491 ±.001	.569 ±.008	**.571** ±.001
CNN	Random	.636 ±.005	.318 ±.004	.666 ±.004	.446 ±.010	.518 ±.011	.453 ±.013
	GloVe(static)	.670 ±.003	.318 ±.003	.657 ±.006	.491 ±.010	.505 ±.016	.444 ±.013
	GloVe(trainable)	.691 ±.003	.323 ±.005	.698 ±.005	.515 ±.015	.537 ±.013	.479 ±.014
	GloVe+Random	.694 ±.003	.308 ±.023	.692 ±.008	.506 ±.008	.559 ±.010	.444 ±.007
	GloVe+WCEs(static)	.703† ±.003	.331 ±.007	.699 ±.004	.515 ±.007	.600 ±.016	.391 ±.007
	GloVe+WCEs(trainable)	.706†† ±.004	.346 ±.003	.706 ±.005	.523 ±.008	.611 ±.014	.424 ±.013
LSTM	Random	.426 ±.014	.191 ±.012	.567 ±.011	.349 ±.017	.371 ±.023	.473 ±.015
	GloVe(static)	.624 ±.007	.213 ±.007	.646 ±.014	.509 ±.013	.458 ±.034	.444 ±.116
	GloVe(trainable)	.629 ±.008	.185 ±.009	.655 ±.025	.398 ±.140	.469 ±.040	.512 ±.013
	GloVe+Random	.591 ±.087	.184 ±.006	.653 ±.006	.456 ±.017	.462 ±.026	.507 ±.017
	GloVe+WCEs(static)	.652 ±.047	.273 ±.008	.688 ±.013	.555 ±.016	.569 ±.018	.505 ±.009
	GloVe+WCEs(trainable)	.660 ±.010	.247 ±.007	.684 ±.013	.500 ±.014	.532 ±.057	.534 ±.008
ATTN	Random	.554 ±.004	.204 ±.071	.572 ±.004	.367 ±.011	.467 ±.016	.491 ±.011
	GloVe(static)	.624 ±.004	.249 ±.006	.649 ±.005	.546 ±.006	.524 ±.021	.498 ±.028
	GloVe(trainable)	.626 ±.005	.240 ±.006	.643 ±.012	.468 ±.018	.495 ±.052	.527 ±.020
	GloVe+Random	.623 ±.006	.242 ±.006	.636 ±.006	.455 ±.011	.521 ±.011	.530 ±.005
	GloVe+WCEs(static)	.682 ±.004	.310 ±.006	.681 ±.003	.508 ±.010	.594 ±.020	.512 ±.013
	GloVe+WCEs(trainable)	.659 ±.005	.283 ±.009	.676 ±.004	.527 ±.014	.566 ±.016	.533 ±.008

Table 2 Classification performance in terms of F_1^M. Boldface indicates the best absolute result for each dataset, while green cells indicate the best result locally to a specific neural architecture. Symbols †† and † indicate the methods, if any, whose performance is *not* statistically significantly different with respect to the best result obtained by any neural approach according to a two-tailed t-test at different confidence levels $\alpha = 0.05$ and $\alpha = 0.005$, respectively, i.e., given the p-value p, we use †† whenever $p > 0.05$, we use † when $0.005 < p \leq 0.05$, or no symbol if $p \leq 0.005$.

Various facts emerge from these results. First, beating well-optimized "traditional" baselines, such as `SVM-tfidf`, is not easy (especially in terms of F_1^M). This has already been noticed in past literature (Wang and Manning, 2012), and has recently stimulated debate (Lin, 2019; Yang et al., 2019a). For SVMs, our results for 20Newsgroups and Reuters-21578 are in line with those reported for the same datasets by Bekkerman et al. (2003). These authors found that the adoption of more sophisticated supervised representations helps SVMs to improve over simple tfidf features in 20Newsgroups but not on Reuters-21578, and argued that in this latter case a considerably high accuracy is achievable by simply using a handful of highly correlated words for each class (something that is already well represented in a bag-of-words model). Notwithstanding this, we should observe that SVMs, in their standard

model	variant	20Newsgroups	JRC-Acquis	Ohsumed	RCV1-v2	Reuters-21578	WIPO-gamma
SVM	tfidf	.679	.525	.702	.808	.881	—
	tfcw	.677	.521	.698	.803	.877	—
	GloVe(static)	.649	.404	.544	.702	.791	—
	GloVe+WCEs(static)	.688	.385	.658	.757	.834	—
	BERT-finetuned(static)	.672 ±.003	.533 ±.014	.722 ±.002	.847 ±.001	.877 ±.003	—
	BERT-finetuned+WCEs(static)	**.717** ±.004	.478 ±.018	**.749** ±.001	.857 ±.001	**.887** ±.005	—
BERT	BERT-finetuned(trainable)	.695 ±.002	.559 ±.005	.748††±.003	**.864** ±.001	.886 ±.001	.664 ±.004
TCNN	GloVe+WP(trainable)	.702 ±.002	.462 ±.008	.716 ±.004	.781 ±.004	.852 ±.003	.651 ±.006
LEAM	Random	.661 ±.009	.480 ±.002	.662 ±.009	.747 ±.007	.808 ±.010	.694 ±.003
	GloVe(trainable)	.697 ±.004	.538 ±.004	.718 ±.002	.804 ±.009	.854 ±.008	.695 ±.003
	GloVe+Random	.696 ±.003	.532 ±.004	.712 ±.005	.797 ±.007	.845 ±.011	.696 ±.005
	GloVe+WCEs(trainable)	.709 ±.002	**.568** ±.003	.721 ±.003	.817 ±.004	.864 ±.006	.699 ±.002
fastText	Random	.622 ±.003	.511 ±.002	.644 ±.002	.772 ±.001	.822 ±.004	.689 ±.001
	GloVe(trainable)	.649 ±.003	.530 ±.002	.662 ±.002	.781 ±.001	.839 ±.002	.696 ±.000
	GloVe+Random	.649 ±.003	.530 ±.002	.651 ±.001	.781 ±.001	.839 ±.003	.696 ±.000
	GloVe+WCEs(trainable)	.704 ±.002	.536 ±.004	.688 ±.001	.769 ±.001	.843 ±.003	.701† ±.000
CNN	Random	.644 ±.005	.509 ±.007	.691 ±.005	.746 ±.006	.835 ±.004	.673 ±.004
	GloVe(static)	.684 ±.002	.527 ±.004	.693 ±.006	.799 ±.004	.848 ±.005	.649 ±.005
	GloVe(trainable)	.703 ±.003	.528 ±.005	.720 ±.007	.803 ±.009	.857 ±.003	.675 ±.005
	GloVe+Random	.705 ±.002	.496 ±.020	.720 ±.006	.789 ±.005	.862 ±.002	.662 ±.003
	GloVe+WCEs(static)	.715††±.004	.509 ±.007	.723 ±.003	.789 ±.002	.858 ±.004	.623 ±.003
	GloVe+WCEs(trainable)	**.717** ±.003	.520 ±.004	.729 ±.004	.792 ±.002	.866 ±.005	.649 ±.005
LSTM	Random	.434 ±.013	.226 ±.094	.618 ±.009	.680 ±.015	.699 ±.024	.685 ±.005
	GloVe(static)	.633 ±.007	.360 ±.015	.687 ±.011	.793 ±.005	.823 ±.011	.662† ±.063
	GloVe(trainable)	.638 ±.008	.193 ±.063	.686 ±.020	.715 ±.100	.804 ±.018	.701† ±.004
	GloVe+Random	.604 ±.081	.208 ±.048	.683 ±.004	.743 ±.013	.794 ±.017	.699† ±.005
	GloVe+WCEs(static)	.661 ±.044	.449 ±.017	.714 ±.009	.807 ±.007	.860 ±.006	.686 ±.004
	GloVe+WCEs(trainable)	.668 ±.010	.403 ±.009	.709 ±.011	.765 ±.008	.849 ±.007	.703††±.003
ATTN	Random	.559 ±.004	.336 ±.112	.620 ±.006	.677 ±.008	.775 ±.009	.690 ±.005
	GloVe(static)	.634 ±.003	.421 ±.009	.680 ±.006	.797 ±.002	.818 ±.008	.688 ±.012
	GloVe(trainable)	.634 ±.007	.397 ±.013	.667 ±.009	.744 ±.008	.790 ±.014	**.707** ±.007
	GloVe+Random	.631 ±.006	.384 ±.014	.668 ±.010	.737 ±.007	.813 ±.005	**.707** ±.001
	GloVe+WCEs(static)	.691 ±.004	.489 ±.007	.709 ±.003	.783 ±.006	.858 ±.006	.687 ±.006
	GloVe+WCEs(trainable)	.668 ±.005	.459 ±.013	.698 ±.005	.773 ±.008	.841 ±.004	.703††±.002

Table 3 As Table 2, but with F_1^{μ} in place of F_1^{M}.

formulation, do not scale to very large training sets; as a result, we were unable to train them on WIPO-GAMMA. Additionally, `SVM-tfidf` seems to be comparable or even better than the supervised version `SVM-tfcw`. This might come as a surprise, since any supervised weighting factor should intuitively beat the "idf" heuristic, which is completely agnostic of the word-class distributions, especially given that the combination using ConfWeight and max-pooling was chosen as the best-performing variant from a pool of $6 \times 3 = 18$ methods. However, that STW criteria often work no better than tfidf has already been reported in past literature (Debole and Sebastiani, 2003). This is likely the reason why no STW function has managed to oust tfidf as a standard in text classification research (further discussions on this can be found in (Moreo et al., 2020)).

Concerning neural approaches, a method equipped with WCEs either turns out to be the best performer, or is comparable (in a statistically significant sense) to the best performer, both in terms of F_1^{M} and F_1^{μ}, and for all datasets but RCV1-v2, in which `BERT-finetuned` performed best. Concatenating `GloVe` embeddings and WCEs almost always yields superior performance with respect to only using `GloVe` embeddings, both for static and trainable pre-trained embeddings, across all models (this applies also to SVMs). Interestingly enough, concatenating WCEs with the document embeddings generated by BERT helps to improve the document representation for classi-

fication, as witnessed by the fact that a SVM trained on this concatenation (`SVM-BERT-finetuned+WCEs`) tends to work better than with the BERT embeddings alone (`SVM-BERT-finetuned`) in terms of F_1^μ (in 4 out of 5 cases). For the F_1^M measure there is no clear winner, since `SVM-BERT-finetuned` works better in 2 out of 5 cases, while `SVM-BERT-finetuned+WCEs` works better in other 2 out of 5 cases; the remaining case (REUTERS-21578) ends up in a tie, since both models score the best global result for that dataset. The BERT variants yield results that sometimes surpass, by a large margin, all other competitors both in terms of F_1^M and F_1^μ; datasets for which this happens include OHSUMED, RCV1-V2, and REUTERS-21578. Finally, of all the variants that we test on all 6 datasets, LEAM-`GloVe+WCEs(trainable)` obtains the best average results, both in terms of F_1^M and F_1^μ.

Table 4 compares, in terms of the relative improvement of F_1^M and F_1^μ, variants equipped with WCEs against the baselines, i.e., against variants that either use only `GloVe` or `GloVe+Random` or the alternative supervised embeddings `GloVe+WP(trainable)` proposed by Lei et al. (2019). The comparison is carried out on pairs of variants across all six datasets, and across all the neural models on which both configurations have been tested. Regarding the static case, `GloVe+WCEs(static)` shows a relative improvement of +7.69% in F_1^M and +3.36% in F_1^μ over `GloVe(static)` on average. Concerning the case in which the embedding layer is trainable, `GloVe+WCEs(trainable)` shows considerable relative improvements over all other variants, both in terms of F_1^M and F_1^μ. The fact that `GloVe+WCEs(trainable)` also substantially outperforms `GloVe+Random` rules out the possibility that the boost in performance that WCEs add is a mere consequence of adding parameters or regularizing part of the embedding. A two-tailed paired t-test reveals that the differences in performance, as averaged across datasets and networks, are statistically significant at high confidence level ($\alpha = 0.005$) in all cases explored in Table 4, for both F_1^M and F_1^μ.

	RI(F_1^M)	RI(F_1^μ)
`GloVe+WCEs(static)` vs. `GloVe(static)`	+7.69%	+3.36%
`GloVe+WCEs(trainable)` vs. `Random`	+15.91%	+10.02%
`GloVe+WCEs(trainable)` vs. `GloVe(trainable)`	+5.06%	+3.45%
`GloVe+WCEs(trainable)` vs. `GloVe+Random`	+5.44%	+3.95%
`GloVe+WCEs(trainable)` vs. `GloVe+WP(trainable)`	+7.18%	+2.58%

Table 4 Relative improvement, expressed as a percentage, of F_1^M and F_1^μ of different variants across all our datasets and networks.

The superiority of the models also equipped with WCEs cannot be merely explained by the higher number of parameters in their embedding layer. By inspecting the execution logs, we find out that approximately 50% of the times the best hyperparameters chosen by the variants with WCEs were consistent with those chosen by the same variant without them. In such cases, the `GloVe+WCEs(trainable)` setting contains exactly the same number of train-

able parameters as the corresponding `GloVe+Random` variants, yet it performs better (with relative average improvements of +5.44% and +3.95% in F_1^M and F_1^μ, respectively, and with statistical significance at $\alpha = 0.005$). For other approximately 40% of the times, the model selected when using WCEs happens to be comparatively smaller than when not using them, and only in the remaining 10% of the cases the variants using WCEs turn out to be larger.

Concerning supervised embeddings, WCEs prove superior to TOR embeddings (`GloVe+WP(trainable)`). Although both types of embeddings build on top of a similar rationale, WCEs incorporate many mechanisms devised to counter problematic issues that arise in multiclass text classification. In other words, differently from TOR embeddings (Lei et al., 2019), our correlation measure is unbiased towards the word and class prevalences (more on this on Section 4.8), WCEs are regularized via supervised dropout (more on this on Section 4.6), and allow for a compact representation in large codeframes. In this respect, it is worth noting that TOR embeddings for JRC-Acquis have 2,706 dimensions and thus use up a lot of GPU memory. Despite the fact that in this dataset WCEs have just 300 dimensions, they still perform better.

4.5 Learning Curves

In this section we look at the learning curves for the three deep learning architectures when equipped with different types of embeddings. For the sake of clarity, we plot only three representative variants: `Random`, `GloVe(static)` and `GloVe+WCEs(static)` (that, for simplicity, we here simply denote by `GloVe` and `GloVe+WCEs`). Unless specified differently, these plots and all the subsequent ones are generated with the same, fixed hyperparameters for all variants, i.e., 256 channels for CNN and 512 hidden nodes for LSTM and ATTN, a supervised dropout probability of 0.5 for `GloVe+WCEs`, and 200 dimensions for random embeddings; we run 100 training epochs and deactivate early stop. For these experiments we choose RCV1-v2, since it is arguably the most widely adopted benchmark in the literature of text classification by topic. (In similar experiments that we have run on other datasets we have verified similar trends.)

Figure 1 shows a grid of plots that visualize learning curves as a function of the number of epochs for RCV1-v2. The first and second rows display (the logarithm of) the training and validation loss, respectively, while the third and fourth rows display the values of F_1^M and F_1^μ, respectively, on the validation set. Columns correspond, from left to right, to the CNN, LSTM, and ATTN architectures.

There are a few observations that we can make from these plots. Networks with random embeddings have more parameters to tune than networks which rely on static pre-trained embeddings (since the latter are fixed, and thus are not trainable parameters), and thus lower the training loss faster than the rest. Notwithstanding this, the validation loss is always higher for them than that of `GloVe` and `GloVe+WCEs`, which shows that the presence of random embeddings

brings about a tendency to overfit the training data. This tendency to generate overfitting is well countered by the variants that use static pre-trained embeddings. The knowledge incorporated in `GloVe` embeddings is generic and thus consistent for validation documents as well.

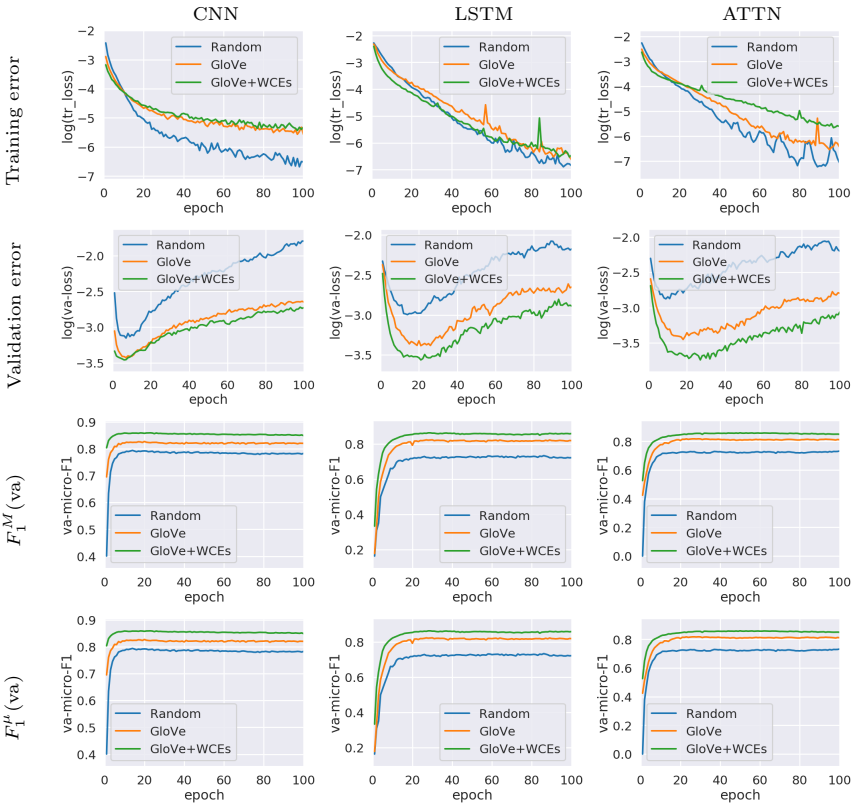

Fig. 1 Learning curves on dataset RCV1-v2

We can also observe that the use of WCEs eases parameter optimization across the three architectures tested. That is, models equipped with WCEs reach promising regions of the parameter space faster (i.e., in fewer epochs) than those not using them.

However, since models relying exclusively on pre-trained embeddings are anyway connected to class labels through the loss function, it is legitimate to wonder what is the real contribution of WCEs to the supervised learning task. Clearly, WCEs are not bringing any new information to the model, since they are computed using the very same amount of information the learner has when training a classifier (this is in contrast to pre-trained embeddings, which are learnt from external data). We conjecture that this has to do with the way WCEs inject supervised information from the bottom (word level) and the top

(document level), instead of only from the top, which may favour the gradient flow. In other words, WCEs may not really be adding any new information, but are handling the available class label information in a more efficient way.

It is worth noting that WCEs model the correlations between words and labels globally (at the dataset level) and not locally (at the batch level). Global word-class dependencies remain reachable to batched optimization only in the long term. The same principle, according to which the word-class distributions are mined globally beforehand, and later serve the purpose of a model prior, was already used in (Moreo et al., 2020) (more details are given in Section 5.1).

4.6 The Importance of Regularization

WCEs directly inject into the model information from the label distribution as available in the training set. This might somehow compromise the generalization capability of the classifier when dealing with future unseen data, for the reasons discussed at the beginning of Section 3.3. During preliminary experiments we observed that this is indeed the case, and that there is thus a need for properly regularizing the model. Note though that what we say in this section exclusively applies to the CNN, LSTM, and ATTN models. In fact, supervised dropout is not applied in the other models, for the simple reason that, for the latter, we use the original implementations made available by their authors, and these implementations do not make use of supervised dropout.

Figure 2 shows the effect of supervised dropout (which, as explained in Section 3.3, is the device we use for performing regularization) at varying drop probability rates for the `GloVe+WCEs` configuration. In the last two rows of this figure we plot F_1^M and F_1^μ as directly computed on the test set of RCV1-v2 every 10 epochs (last two columns). The rationale behind showing the values of effectiveness on the test set (instead of on the validation set, as in Figure 1) is to better illustrate the effect of regularization when dealing with unseen data: as observed in Section 4.1, news stories in the RCV1-v2 test set are from a disjoint time window with respect to those in the training set, which is not true for validation documents, which are randomly drawn from the original training set.

We can observe that applying no regularization at all ($p = 0.0$) yields a faster minimization of the training loss, but also results in poorer generalization on unseen data. The model generalizes better for $p > 0.0$; setting $p = 0.5$ actually yields the best results. Of course, in general the optimal value for p has to be explored on a validation set in a case-by-case fashion, but we have observed the setting $p = 0.5$ to generally entail, across all methods and datasets, a good tradeoff between loss minimization in training and performance on validation/test data (this is indeed in agreement with general considerations regarding dropout as reported in the literature).

However, our use of supervised dropout in the WCE-based variants might, in theory, make our experimental comparisons unfair, since supervised dropout

is not used in the baseline methods based on CNN, LSTM, or ATTN. In other words, it might happen that these latter methods overfit the embeddings (for the `trainable` variants) or the layer on top of them, while in the WCE-based variants overfitting is mitigated by the use of supervised dropout. We have carried out additional experiments (that we here omit to report in detail for the sake of brevity) in order to confirm / disconfirm this hypothesis. In these experiments we compare the effect of dropout on the baselines. Our results indicate that, in general, applying dropout after the embedding layer, helps the baselines improve their results, but that this improvement is not consistent (i.e., sometimes a deterioration occurs instead of an improvement, and sometimes this deterioration is even large), is not statistically significant, and is not marked enough as to surpass the performance of the WCE-based variants. In cases in which applying dropout helps to improve the baselines by a substantial margin, we have verified that similar improvements would be observed if dropout were applied to the entire embedding layer (and not only to the supervised embedding part) in the WCE-based variants as well. Aside from the fact that applying dropout is often beneficial (but sometimes harmful) we note that `GloVe+Random` establishes a perfectly fair baseline in terms of regularization and number of parameters, that is nonetheless beaten by WCE-based variants in a systematic way.

4.7 Computational Cost

In this section we turn to analyzing the additional computational cost that the use of WCEs entails. Directly comparing execution times woud lead to confusion, since there are many experimental variables that impact on them. For example, some methods run on CPU (`fastText` and SVMs) while others run on GPU (all others); some methods are implemented using PyTorch (CNN, LSTM, ATTN, BERT) while others use Tensorflow (LEAM) and yet others are written in plain C++ (`fastText`) or jointly in C++ and Python (SVMs are implemented in C/C++ in the `liblinear` package, which is wrapped in `scikit-learn`, which is written in Python).

We thus restrict our attention to methods that adopt early stopping (i.e., CNN, LSTM, ATTN, and LEAM[33]) and compare, for each method, the number of epochs it took them to converge. For each architecture we focus on the variants `GloVe+Random` and `GloVe+WCEs(trainable)`, since they are equipped with embeddings of the same number of trainable parameters, i.e., of the same capacity (we have separately verified that our conclusions hold also for the other variants). Table 5 reports, for each method and dataset, the number of training epochs before invoking early stopping, averaged across 10 runs. In most cases, the average numbers of epochs required for `GloVe+Random` and `GloVe+WCEs(trainable)` to converge are not different in a statistically significant sense. This indicates that using WCEs does not come at a cost.

[33] We modified the official implementation of `https://github.com/guoyinwang/LEAM` to use early-stop.

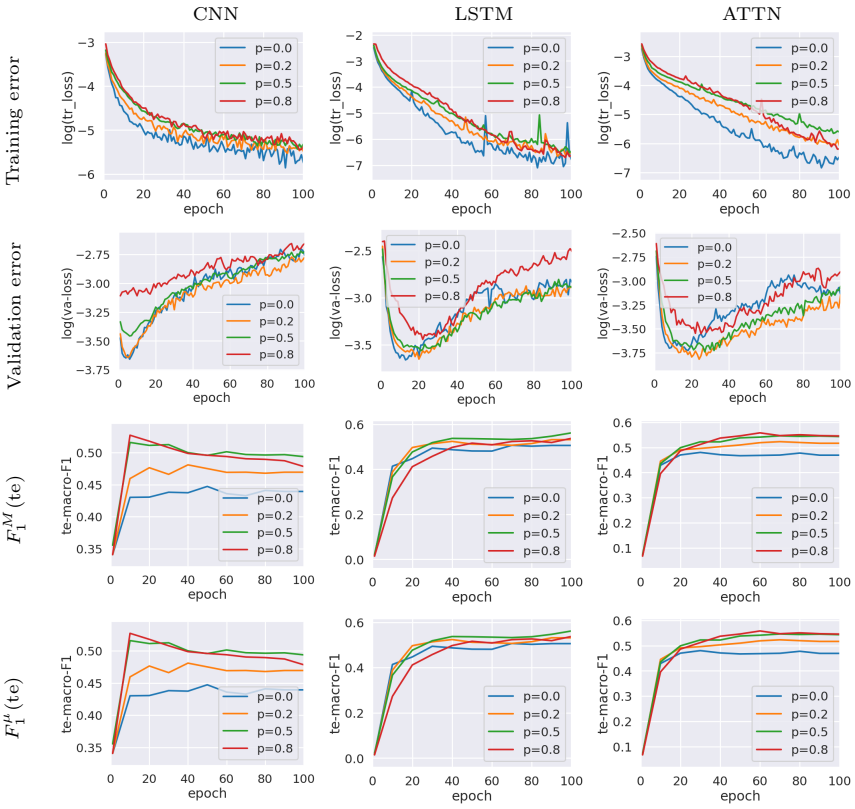

Fig. 2 Effect of supervised dropout regularization in RCV1-v2

Model	Variant	20Newsgroups	JRC-Acquis	Ohsumed	RCV1-v2	Reuters21578	WIPO-gamma
LEAM	GloVe+Random	29.7 ± 5.2	166.2† ± 21.6	**23.7** ± 1.6	56.0 ± 10.8	64.8 ± 11.0	**122.7** ± 46.7
	GloVe+WCEs(trainable)	**18.8** ± 2.0	**148.6** ± 13.9	24.6†† ± 2.1	**33.4** ± 3.1	**45.5** ± 6.0	151.9†† ± 35.1
CNN	GloVe+Random	23.3†† ± 4.0	**26.3** ± 20.8	**13.1** ± 6.5	**15.3** ± 5.8	**18.8** ± 5.1	**51.2** ± 10.4
	GloVe+WCEs(trainable)	**20.3** ± 3.6	44.4† ± 6.8	16.2†† ± 2.3	22.5† ± 5.1	28.3 ± 6.3	58.2†† ± 15.1
LSTM	GloVe+Random	**22.8** ± 10.0	**107.4** ± 22.4	**17.2** ± 4.7	**41.5** ± 10.1	61.0†† ± 7.3	**61.0** ± 12.9
	GloVe+WCEs(trainable)	24.1†† ± 3.8	137.1† ± 20.7	18.9†† ± 5.5	46.2†† ± 12.3	**60.3** ± 14.1	83.6† ± 18.6
ATTN	GloVe+Random	**20.9** ± 6.1	**109.3** ± 7.2	**12.5** ± 5.6	**34.5** ± 9.5	56.9†† ± 16.9	72.0†† ± 10.1
	GloVe+WCEs(trainable)	21.8†† ± 7.1	133.3 ± 7.9	18.8 ± 1.1	37.8†† ± 4.5	**53.9** ± 11.9	**67.4** ± 10.4

Table 5 Average number of epochs required for convergence.

Table 6 reports the time required to create the WCEs, which can be broken down into two components, i.e., (i) the time needed to generate matrix $\mathbf{X}$ (which encodes the bag-of-words model with tfidf weighting), and (ii) the time needed to subsequently generate matrix $\mathbf{S}$ (which contains the WCEs). Most of the total time is accounted for by the generation of matrix $\mathbf{X}$ (1st row of Table 6); the computational cost of generating the WCEs from matrix $\mathbf{X}$ is almost negligible (2nd row of Table 6), and typically represents less than

Computation	20Newsgroups	Ohsumed	RCV1-v2	JRC-Acquis	Reuters21578	WIPO-gamma
X	2.10s (99.16%)	3.85s (98.95%)	4.74s (97.34%)	14.00s (56.00%)	1.11s (96.78%)	4m 39.00s (91.56%)
S	0.02s (0.84%)	0.04s (1.05%)	0.13s (2.66%)	11.00s (44.00%)	0.04s (3.22%)	25.79s (8.44%)
Total	2.12s	3.89s	4.87s	25.00s	1.15s	5m 5.00s

Table 6 Total time required by the computation of WCEs (**X** indicates the "weighted bag-of-words" matrix mentioned at the beginning of Section 3, while **S** indicates the WCE matrix of Equation 3).

10% of the total time. The higher times clocked for JRC-Acquis and WIPO-gamma are due to the application of PCA, which is not necessary for the other datasets (see Section 3.1). The total times (last row of Table 6) are, in all cases, much smaller than the times needed for optimizing the models.

4.8 Other Measures of Correlation

In this section we explore other correlation measures as alternative ways for computing the WCEs. In other words, we explore alternatives to the use of the dot product for instantiating the η function of Equation 6. For this, we use well-known functions from information theory or statistics that have been routinely used for feature selection purposes in text classification, including Positive Pointwise Mutual Information (PPMI – see Footnote 3), Information Gain (IG), Chi-square (χ^2), and ConfWeight (CW) (Soucy and Mineau, 2005). In preliminary experiments we had carried out using these functions we had indeed found that standardizing the resulting matrix **A** (Equation 2) improves accuracy. We thus report the results of using as the η function one that also performs standardizing, e.g., $\eta_{\chi^2}(w_i, c_j) = z_j(\chi^2(w_i, c_j))$ (similarly for PPMI and IG).

Figure 3 compares the classification performance, in terms of F_1^M, of the dot product (as originally used in Equation 2, and here abbreviated as "Dot") against PPMI, IG, χ^2, and CW. As can be observed from Figure 3, "Dot" is almost always superior to all other functions, or at least comparable to the best-performing function, across all datasets and network architectures. Other functions behave irregularly across experiments; for example, PPMI seems to be the most competitive method for WIPO-gamma, but is a weak one in Reuters-21578 (this applies to all architectures) and JRC-Acquis (in CNN and LSTM). Interestingly, the weighting function CW, which proved more effective than PPMI, IG, and χ^2 when combined with SVMs (in experiments omitted but mentioned in Section 4.3.1), does not show a clear superiority, if at all, with respect to these weighting functions.

We conjecture that the superiority of the dot product partly depends on its ability to take non-binary (in our case: tfidf) weights into account, while

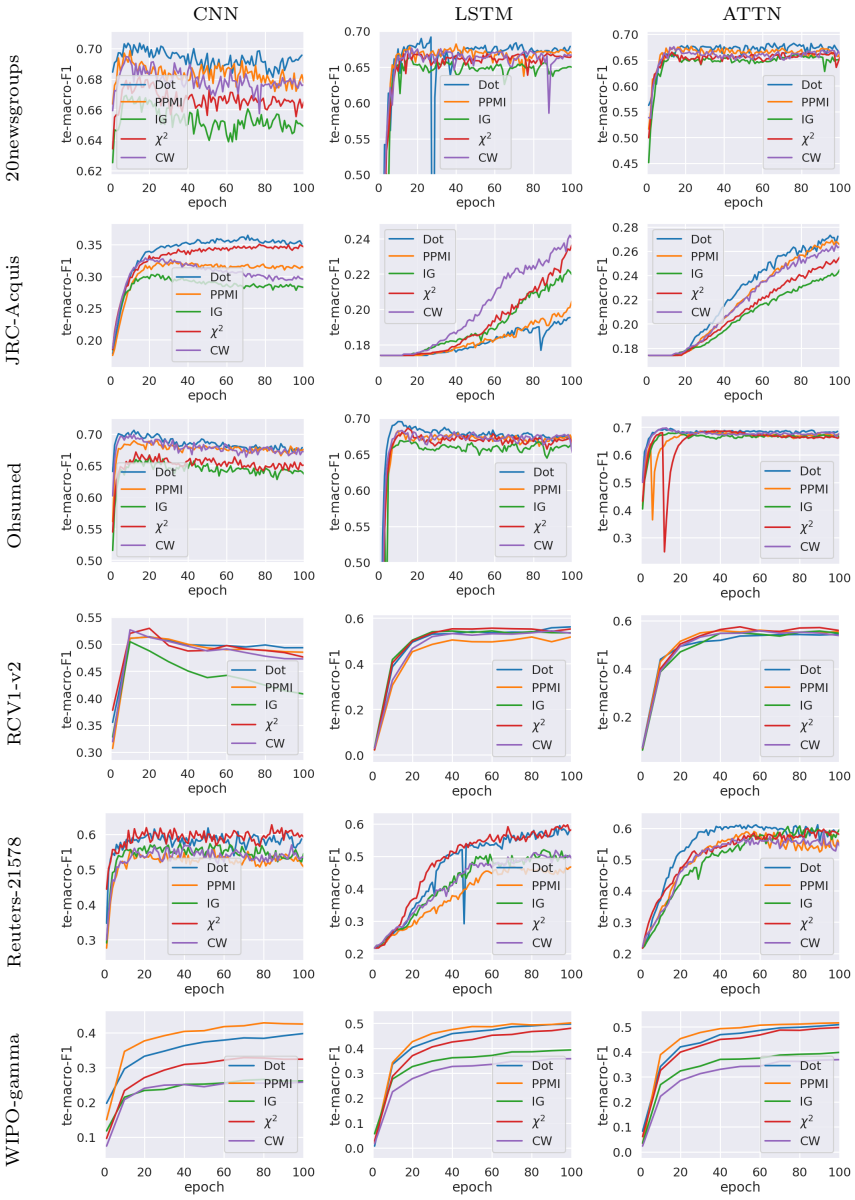

Fig. 3 Classification accuracy (in terms of F_1^M) resulting from the use of measures of correlation alternative to the one ("Dot") that we use for generating WCEs.

PPMI, IG, and χ^2 only consider binary presence/absence indicators.[34] Additionally, the dot product only leverages positive correlations (this is also true for PPMI), and is thus much faster than other methods that compute negative correlations at intermediate steps, as is the case of IG and χ^2. The reason is that our tfidf matrix $\mathbf{X}$ and the label matrix $\mathbf{Y}$ (see Equation 2) are highly sparse; many machine learning software packages (including the ones we use) do cater for the presence of sparse matrices, and thus do not explicitly represent zero values, causing any operation involving non-zero values to substantially increase computation times. For example, it takes 20s to compute matrix $\mathbf{S}$ for REUTERS-21578 when using χ^2, while the same only takes 0.04s when using the dot product (see Table 6).

4.9 Different Pre-trained Embeddings

Up to now, we have tested the performance of WCEs as an extension of `GloVe` vectors. It might be interesting to check how well WCEs could perform when concatenated to embedded representations other than those generated by `GloVe`.

Figure 4 shows the results of experiments in which WCEs are concatenated to different types of embeddings, including pre-trained `word2vec`[35] (Mikolov et al., 2013b), pre-trained `fastText`[36] (Mikolov et al., 2018), and the contextualized embeddings produced by BERT[37] (Devlin et al., 2019). Note that `fastText` is here used in its unsupervised modality[38], i.e., one that learns word embeddings from the observation of large quantities of unlabelled text. For BERT, we took the output of the last layer as the contextualized embeddings. All the pre-trained embeddings used in this experiment are used "as they are", i.e., we do not fine-tuned any of the models and we keep the embeddings static during the training.[39] For the sake of clarity we report the relative improvement (in terms of F_1^M) that concatenating WCEs brings to bear with respect to each type of embedding.

[34] Though most traditional functions used for feature selection can only use presence/absence, other metrics exist that work with weighted scores, e.g., the *Fisher score*. In initial experiments not described in this paper we have indeed tried to use the Fisher score, but we have eventually given up, due to the fact that (a) its computation is very slow, and (b) the classification accuracy that we have observed is not much different from what can be obtained with the other functions mentioned above, and is often intermediate between the best and the worst recorded values.

[35] Available at `https://code.google.com/archive/p/word2vec/`

[36] Available at `https://fasttext.cc/docs/en/english-vectors.html`

[37] We used the Huggingface's implementation available at `https://github.com/huggingface/transformers`.

[38] See `https://fasttext.cc/docs/en/unsupervised-tutorial.html`

[39] More often than not, BERT is used by fine-tuning the entire model to the task at hand. In this set of experiments we prefer to reproduce a simpler scenario, in which the practitioner simply uses pre-trained models as made available by the developers of BERT. Fine-tuning models such as BERT requires a considerable amount of computational power, which might not be at everyone's reach. Experiments showcasing how a properly fine-tuned BERT works (with and without WCEs) on our datasets are illustrated in Section 4.4.

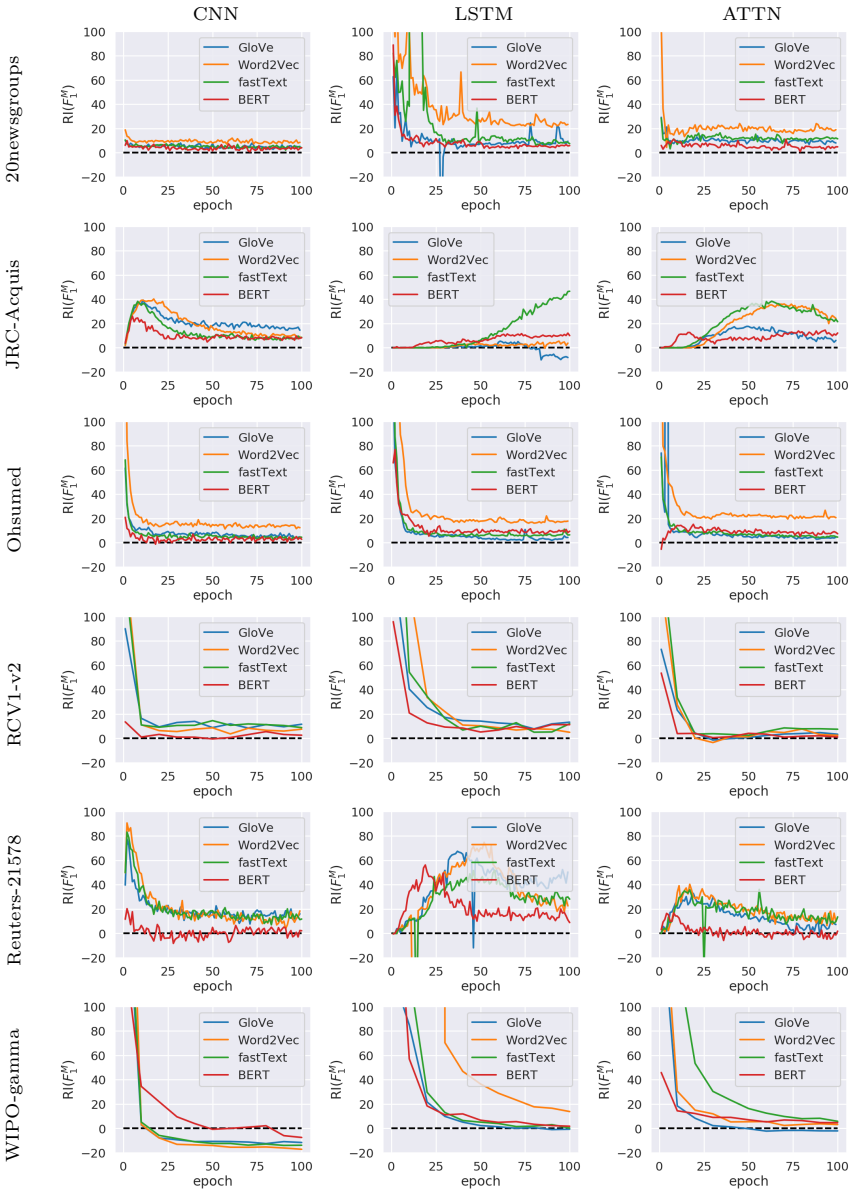

Fig. 4 Percentage of relative improvement brought about by concatenating WCEs to different types of word embeddings.

The results show that the improvements brought about by WCEs are not restricted to the case in which `GloVe` vectors are used. In the vast majority of cases the relative improvement has a positive sign. The few exceptions concern BERT embeddings: they benefit from also using WCEs in most cases, but not always. Interestingly enough, the largest improvements almost always occur during the first epochs of training. This supports our belief that WCEs represent a useful initialization for classification purposes. It also speaks about the fact that WCEs help bringing to bear corpus-level correlations between words and labels yet from the very beginning. It is only in the long term that such correlations become meaningful for methods relying on unsupervised representations, as witnessed by the fact that the margins of improvement tend to become thinner as the training progresses.

In the future we plan to validate our WCEs along with other contextualized vector representations beyond BERT, e.g., (McCann et al., 2017; Peters et al., 2018; Yang et al., 2019b), sub-word representations (Bojanowski et al., 2017), character-based embeddings (Kim et al., 2016; Zhang et al., 2015), or high-dimensional sparse representations (Sahlgren, 2005).

4.10 Visualizing WCEs

In this section we try to gain an understanding on how words represented by WCEs are topologically distributed in the embedding space, and how WCEs alter the distribution of pre-trained embeddings once they are concatenated to them. To do so we use `Embedding Projector`, a publicly available tool for data visualization based on the *t-distributed Stochastic Neighbor Embedding* (t-SNE) technique (van der Maaten and Hinton, 2008)[40], and which allows to map word embeddings onto a 2-dimensional space. We use its default parameters (perplexity=18 and learning rate=10) and perform 1,000 iterations. We only represent 5,000 words, in order to get a clearer visualization; the words we select are the most predictive ones (as quantified via information gain) for each class, following a "round robin" policy which selects the same number of highly predictive words for each class (Forman, 2004). We assign different colours to the classes, and colour each embedding according to the class for which it was selected. For this experiment we choose 20Newsgroups (the dataset with fewest classes), with the aim of keeping the colour coding simple enough, thus maximizing visual clarity.

Figure 5 shows the distribution of `GloVe` vectors. The top part shows that, to some extent, some word clusters seem to correlate with some classes. This was to be expected, since words relevant to a given class tend to be semantically related to each other. The enlarged region (bottom) shows how `GloVe` succeeds at producing meaningful local structures containing smaller clusters of semantically interrelated words, e.g., {"diet", "dietary", "vitamin"}, or {"doctor", "medical", "hospital"}, within class `sci.med`.

[40] `https://projector.tensorflow.org/`

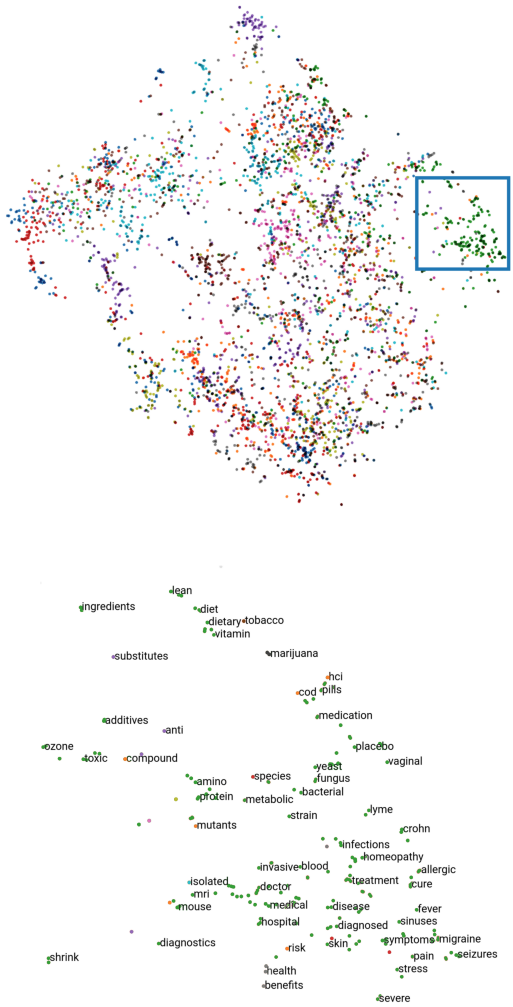

Fig. 5 Visualization (best viewed in color) of the space of `GloVe` embeddings for 20NEWSGROUPS (top), and enlargement of the part in the blue square (bottom). Each point represents an embedding, and (in the top part) each colour represents one of the 20 classes in the codeframe, and the one the word represented by the embedding is highly predictive of.

Figure 6 shows the distribution of WCEs for the very same words represented in Figure 5. The visualization shows almost perfect word clusters for classes (top). This should come at no surprise, since the WCEs explicitly encode class structure. As a counterpart, local semantics within clusters vanishes (bottom), given that WCEs disregard word-word interactions. For example, for certain pairs of words (e.g., {"attacking", "attacks"}, {"terror","terrorism"}),

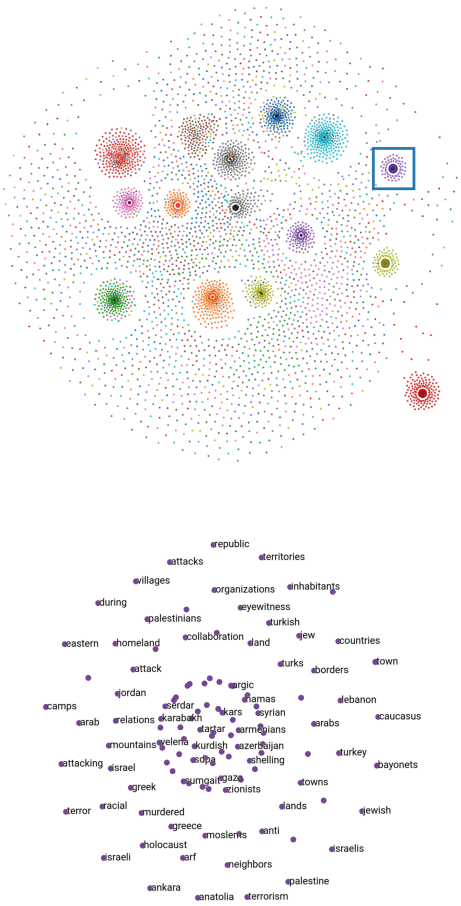

Fig. 6 Same as Figure 5, but with WCEs instead of `GloVe` embeddings.

the two words happen to lie far from each other within the cluster for class `talk.politics.mideast`.

Figure 7 shows the distribution of the concatenation of `GloVe` vectors and WCEs for the very same words of Figures 5 and 6. This representation brings together the best of the two worlds. Globally, a neat class structure emerges, as imposed by the WCEs (top). Locally, clusters exhibit a meaningful inner structure, thanks to `GloVe` vectors (bottom). As an example, relevant words for class `sci.space` organize in smaller clusters such as {"moon", "earth", "lunar"} and {"allen", "grifin", "dani"}.

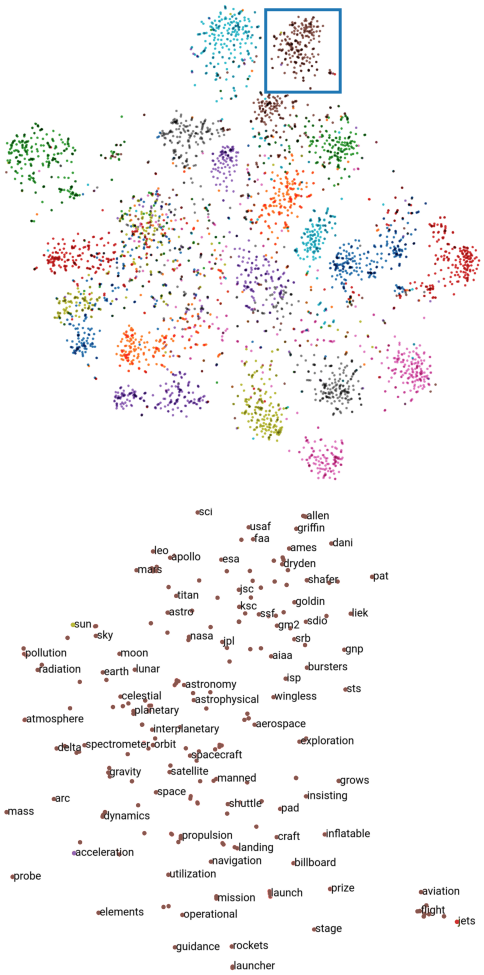

Fig. 7 Same as Figure 5, but with `GloVe`+WCE embeddings instead of `GloVe` embeddings.

4.11 Can we Learn WCEs for Out-of-Vocabulary Terms?

In contrast to unsupervised word embeddings, WCEs are inherently task-dependent, and thus heavily rely on the distribution of the words in the training set. This means that, for any word encountered at testing time that was not observed during training, the corresponding WCE will be a vector of zeroes (or a a randomly initialized vector), and this could harm performance. This is yet another manifestation of the well-known problem of *out-of-vocabulary* (OOV) *words*, which represents an active area of research in the field of word embeddings (Garneau et al., 2019).

Classifiers that make use of pre-trained embeddings do have a chance to make sense of OOV words (i.e., words not encountered at classifier training time). When (as is usually the case) pre-trained embeddings are generated from huge quantities of text, an OOV word is fairly likely to have a corresponding pre-trained embedding. If this is the case, what is learnt during classifier training is thus related to this word too, inasmuch as its pre-trained embedding is at least partly aligned with the embeddings of some words encountered during classifier training.

In this section we cope with the problem of WCEs for OOV words. Our idea is to predict the WCE for an OOV word from its pre-trained embedding (when available), on the grounds that semantically similar words (as observed in general language use) can be expected to exhibit similar class-conditional distributions.

We frame the problem of generating the WCE for an OOV word as a multivariate regression task.[41] As from Section 3, let $\mathbf{E} = [\mathbf{U} \oplus \mathbf{S}]$ be the embedding matrix, where rows $\mathbf{e}_i = [\mathbf{u}_i \oplus \mathbf{s}_i]$ are the embeddings, consisting of a concatenation of a pre-trained embedding $\mathbf{u}_i$ and a WCE $\mathbf{s}_i$. We train a two-layered feed-forward network (in the experiments we use 64 units in the hidden layer, ReLU activation, and 0.5 dropout) to predict the WCE $\mathbf{s}_i \in \mathbf{S}$ from the pre-trained embedding $\mathbf{u}_i \in \mathbf{U}$, using the words in the vocabulary as the training examples. We adopt Mean Square Error (MSE) as the loss criterion. Once trained, and once an OOV word is encountered, the regressor is asked to generate the WCE for it, provided this word has a pre-trained embedding.

We have run experiments, using RCV1-v2 (the dataset that contains the highest number of OOV words) as the dataset and all the learners of Section 4.3, in which we compare the accuracy of two different configurations: (i) a configuration in which the WCEs of an OOV word is a vector of zeros (which is the setting we have used so far), and (ii) a configuration in which the WCEs of an OOV word has been predicted from the pre-trained embedding of the word, using all the non-OOV words as training examples.

Unfortunately, the differences in classification accuracy between the two configurations turned out to be barely discernible; in the interest of brevity, we thus omit to plot them out explicitly. The likely reason of this result is that, while RCV1-v2 contains no less than 384,327 OOV words, occurring 2,073,278 times in the test set, these represent only 2.16% of the total number of word occurrences, which means that their impact on classification accuracy is minimal. We have separately verified similar results for the rest of our datasets.

However, in a qualitative (although somehow "anecdotal") evaluation we have verified that the predicted WCEs for OOV words look meaningful. In or-

[41] Another technique for solving this problem is *Latent Semantic Imputation* (Yao et al., 2019). This methods allows filling the missing representation in a vector space (in our case: in the space of WCEs) by analyzing the neighborhood of the word representation in another vector space (in our case: the space of unsupervised embeddings) via techniques inspired by manifold learning.

der to do this, we looked for OOV words whose predicted WCE displays a large correlation with some class, i.e., whose WCE is such that the value for some of its features is high. Some interesting examples of such OOV words include "astronauts", "invincible", "battlefields", "indiscriminate" for class `GDEF` (which is about armed forces, defence policy, and defence budget); "windfarm" for class `GENV` (about environment, pollution, conservation, green issues); "pneumatic" and "prostatectomy" for class `GHEA` (dealing with health and diseases).

We also found many misspelled words whose predicted WCE displays a large correlation with some class, e.g., "sellling" and "exchang" for classes `C311` (domestic markets, sales and imports) and `C312` (external markets and exports); "emploment" for class `C41` (all management issues); "manufatures" for class `E12` (monetary/economic policy and intervention, interest rates), among others. This is interesting, because the ability to make sense of misspelled words is important for many text management applications. Incidentally, these findings also speak about the ability of `GloVe` to model rare words.

Surprisingly, we have also found new correlations for non-English words, like Spanish words "aseguradora" (insurer) for class `E121` (money supply), and "mantenimiento" (maintenance) for class `E313` (inventories and stocks of manufacturing raw materials).

Concerning the qualitative analysis we have presented, it is fair to mention that this experiment is preliminary. More sophisticated methods for predicting supervised OOV representations could well lead to significant improvements. This is something we plan to delve deeper into in our future research.

5 Discussion

5.1 Term Semantics: From Unsupervised to Supervised

In Section 1 we have touched upon the connections between WCEs and `fastText`, and argued that the two methods are the supervised counterparts of SPPMI and `word2vec`, respectively. These are just the most recent examples of a trend, in the field of extracting word semantics from data for text classification purposes, that over the years has seen a move from unsupervised to supervised techniques.

The first interesting example of this trend is that of *word clustering*. In the '90s, word clustering was advocated, among others, as a means to implement dimensionality reduction for text classification purposes, according to the idea that clusters of semantically related words, instead of individual words, would serve as features. While standard unsupervised techniques were initially used (Lewis, 1992), the field slowly moved to using supervised ones, such as *distributional clustering by class distribution* (DCCD) (Baker and McCallum, 1998; Bekkerman et al., 2003). While unsupervised word clustering has the simple goal of grouping together semantically related words, DCCD has the goal of grouping together words that are discriminative for the same classes; as such, it constitutes a technique for building special-purpose word

clusters, i.e., ones that are to be used for text classification only, and only for the specific codeframe on which they have been trained.

The second interesting example is that of *term weighting*. In text classification, and also in other tasks such as text search and text clustering, term weighting serves the purpose of emphasizing the importance of words that are deemed to be more important in describing the semantics of the documents they occur in. While standard unsupervised techniques were initially used (Yang and Chute, 1994), *supervised term weighting* (STW) techniques later started to gain prominence (Debole and Sebastiani, 2003), based on the notion that the words that should weigh more in representing a document are not the ones that are rarest in the collection, but the ones which are most correlated with the labels of interest. As in the case of DCCD, STW techniques leverage the class labels of the training documents, and generate document representations that have a special-purpose nature, i.e., should be used only for the specific text classification task on which they have been trained. The notion of STW is brought one step further in *learning to weight* (Moreo et al., 2020), where the STW function is not given but is learnt from data.

5.2 WCEs with few training data

How many training examples are needed in order to obtain good-quality WCEs? Can WCEs be used with small training sets?

A specific study to estimate the gain in performance that WCEs bring about as a function of the number of training examples is something we have not carried out (and that we plan to do in future work). However, our experiments suggest that WCEs can bring about benefits even when there are few training examples. One such indirect piece of evidence is in our experiments on the Reuters-21578 dataset as evaluated according to F_1^M, as reported in Table 2. Reuters-21578 is known to contain a very large number of classes with very few training examples (down to 1 example per class), and F_1^M is known to reward those classifiers that perform well also on the rare classes (unlike F_1^μ, which rewards those classifiers that perform well especially on the frequent classes). The fact that, on Reuters-21578, WCEs bring about benefits in terms of F_1^M (e.g., from .804 to .849 for LSTM, from .790 to .841 for ATTN) is an indication that they are beneficial for rare classes too.

5.3 Known Limitations

In this article we have focused our attention on multiclass classification by topic. Other classification scenarios remain unexplored. Two important such scenarios include (i) simple binary classification, and (ii) classification by dimensions other than topic, such as, e.g., sentiment classification (an active area of research where deep learning is already showing interesting results (Zhang et al., 2018)).

In their current form, WCEs are not suitable for binary classification. The reason is that, since the dimensionality of WCEs is the number m of classes in the codeframe of interest, in binary classification the dimensionality of WCEs would be 1, which indicates that WCEs would convey very little information to the classification process.[42] One possible strategy to counter this problem might be based on increasing the number of classes artificially. A possible approach to do so might gain inspiration from the structural learning framework (Ando and Zhang, 2005). This strategy would consist of adding new classes that account for the presence or absence of certain highly predictive words for the task (i.e., adding to $\mathbf{Y}$ new columns corresponding to binary versions of columns in $\mathbf{X}$ for highly predictive words), and then computing the WCEs across them (a similar intuition has been explored in (Moreo et al., 2016)). However, preliminary experiments we have conducted along this vein are still inconclusive.

In addition to this, WCEs depend on the labelled data from the training set, and thus might depend on the prevalences of the classes in the codeframe. This might compromise their contribution to tasks characterized by the presence of prior probability shift, i.e., by the fact that the prevalence of a class in the training data is substantially different from the prevalence of the same class in the unseen data. This might make the use of WCEs problematic for tasks such as text quantification (González et al., 2017), which indeed targets scenarios characterized by prior probability shift.

6 Conclusions

In this article we have presented word-class embeddings (WCEs), i.e., distributed representations of words specifically designed for multiclass text classification. The hypothesis underlying the present study is that the class-conditional distributions of a word defines a fingerprint that might help to refine its pre-trained representation for applications of multiclass text classification. The extensive empirical evaluation we have conducted indeed confirms this hypothesis.[43]

[42] The fact that WCEs are not suitable for codeframes containing just a few classes is the reason why all the datasets we have chosen for our experiments are for classification *by topic* (CBT). While WCEs are not *inherently* about CBT, it is a matter of fact that large enough codeframes are mostly to be found in CBT (e.g., when classifying text according to domain-specific taxonomies / thesauri). Other classification tasks of a non-topical nature are often characterized by codeframes consisting of two or three classes; examples of this are classification by sensitivity (Sensitive vs. NonSensitive) (Berardi et al., 2015), sentiment classification (Positive vs. Neutral vs. Negative) (Pang and Lee, 2008), or classification by subjectivity (Subjective vs. Objective) (Riloff et al., 2005).

[43] While in this paper we have focused on classification, we should note that WCEs are straightforwardly applicable to regression tasks too. One reason why we exclusively concentrate on classification is that, in the realm of text, classification is a way more popular task than regression. In other words, there are many more applications of text classification than of text regression, which also means that there are fewer publicly available datasets for experimenting on text regression. A second reason why we have focused on classification

WCEs are a straightforward (or: "frustratingly easy" (Daumé, 2007)) implementation of a conceptually simple idea. They are meant to expand, and not modify, pre-trained word representations that model general language usage. Although this implies adding some parameters to the model, we have seen that this additional cost is negligible in practice; they contribute to the classification task far more than they complicate the model.

We have investigated, also with the aid of a visualization tool, how these new embeddings alter the topology of the embedding space when WCEs are concatenated to embeddings pre-trained on generic, unlabelled text corpora. Our findings suggest that the new representation adds global class structure while preserving local word semantics, and is thus better suited for the classification task.

Unsupervised word embeddings are known to encode a mixture of the senses of polysemous words (Camacho-Collados and Pilehvar, 2018). We think WCEs indirectly help to disambiguate relevant domain-dependent words. The word-class distribution might uncover the meaning of a word prevalent in the domain of interest, thus leaving a domain-specific mark on the resulting representation. This information might be thought of as a form of task-dependent word bias, which reframes the general-purpose word meaning through the lens of the codeframe of interest.

In future work we will try to provide solutions for the limitations discussed in Section 5.3. For instance, the fact that WCEs might not perform optimally in scenarios characterized by distribution shift might be addressed by rescaling each word-class correlation by the updated class prevalence estimates as computed by a quantification method robust to distribution shift, such as the one discussed by Saerens et al. (2002). Other directions worth exploring would be that of modelling, in sentiment classification contexts, the sentiment prior of words across different domains (e.g., reviews of books, music, films, kitchen appliances, etc.), thus producing word-sentiment embeddings with as many dimensions as there are source domains available. This would hopefully help in cross-domain sentiment classification tasks (Blitzer et al., 2006; Moreo et al., 2016). It should similarly be interesting to investigate the implications of this idea on multi-task learning (Caruana, 1993), where each task might contribute with a dedicated task-specific embedding to the representation for other tasks. We would also like to validate WCEs in scenarios having to do with shorter texts, e.g., tweet classification, or question type classification. Finally, we have started investigating the use of WCEs in multi-lingual TC (Moreo et al., 2021), since WCEs are naturally aligned across languages inasmuch as the codeframes are common across all language-specific training sets.

---

is that most text regression tasks are not multiclass, i.e., there is a single class (or "concept") of interest and the regressor must label a document with a real-valued score for that concept. "Single-class regression" is the regression equivalent of binary classification, and in Section 5.3 we have argued that WCEs are not suitable for binary classification; for the very same reasons they are not suitable to "single-class regression". For all these reasons, in this paper we restrict our interest to (multiclass) classification.

Acknowledgements The present work has been supported by the ARIADNEplus project, funded by the European Commission (Grant 823914) under the H2020 Programme INFRAIA-2018-1, by the SoBigdata++ project, funded by the European Commission (Grant 871042) under the H2020 Programme INFRAIA-2019-1, and by the AI4Media project, funded by the European Commission (Grant 951911) under the H2020 Programme ICT-48-2020 . The authors' opinions do not necessarily reflect those of the European Commission. Thanks to NVidia for donating the two Titan GPUs on which many of the experiments discussed in this paper were run. We thank the anonymous reviewers for valuable feedback that allowed us to improve the quality of this paper.

References

Ando RK, Zhang T (2005) A framework for learning predictive structures from multiple tasks and unlabeled data. Journal of Machine Learning Research 6:1817–1853

Baker D, McCallum AK (1998) Distributional clustering of words for text classification. In: Proceedings of the 21st ACM International Conference on Research and Development in Information Retrieval (SIGIR 1998), Melbourne, AU, pp 96–103, DOI 10.1145/290941.290970

Baldi P (2011) Autoencoders, unsupervised learning, and deep architectures. In: Proceedings of the ICML 2011 Workshop on Unsupervised and Transfer Learning, Bellevue, US, pp 37–49

Baroni M, Dinu G, Kruszewski G (2014) Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014), Baltimore, US, pp 238–247, DOI 10.3115/v1/p14-1023

Bekkerman R, El-Yaniv R, Tishby N, Winter Y (2003) Distributional word clusters vs. words for text categorization. Journal of Machine Learning Research 3:1183–1208

Bengio Y, Ducharme R, Vincent P, Jauvin C (2003) A neural probabilistic language model. Journal of Machine Learning Research 3:1137–1155

Berardi G, Esuli A, Macdonald C, Ounis I, Sebastiani F (2015) Semi-automated text classification for sensitivity identification. In: Proceedings of the 24th ACM International Conference on Information and Knowledge Management (CIKM 2015), Melbourne, AU, pp 1711–1714, DOI 10.1145/2806416.2806597

Bhatia K, Jain H, Kar P, Varma M, Jain P (2015) Sparse local embeddings for extreme multi-label classification. In: Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NIPS 2015), Montreal, CA, pp 730–738

Blei DM, Ng AY, Jordan MI (2003) Latent Dirichlet allocation. Journal of Machine Learning Research 3:993–1022

Blitzer J, McDonald R, Pereira F (2006) Domain adaptation with structural correspondence learning. In: Proceedings of the 4th Conference on Empirical Methods in Natural Language Processing (EMNLP 2006), Sydney, AU, pp 120–128, DOI 10.3115/1610075.1610094

Bojanowski P, Grave E, Joulin A, Mikolov T (2017) Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics 5:135–146, DOI 10.1162/tacl_a_00051

Bullinaria JA, Levy JP (2007) Extracting semantic representations from word co-occurrence statistics: A computational study. Behavior Research Methods 39(3):510–526, DOI 10.3758/bf03193020

Camacho-Collados J, Pilehvar MT (2018) From word to sense embeddings: A survey on vector representations of meaning. Journal of Artificial Intelligence Research 63:743–788, DOI 10.1613/jair.1.11259

Caruana R (1993) Multitask learning: A knowledge-based source of inductive bias. In: Proceedings of the 10th International Conference on Machine Learning (ICML 1993), Amherst, US, pp 41–48, DOI 10.1016/b978-1-55860-307-3.50012-5

Collobert R, Weston J, Bottou L, Karlen M, Kavukcuoglu K, Kuksa P (2011) Natural language processing (almost) from scratch. Journal of Machine Learning Research 12:2493–2537

Cortes C, Vapnik V (1995) Support vector networks. Machine Learning 20(3):273–297

Daumé H (2007) Frustratingly easy domain adaptation. In: Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL 2007), Prague, CZ, pp 256–263

Debole F, Sebastiani F (2003) Supervised term weighting for automated text categorization. In: Proceedings of the 18th ACM Symposium on Applied Computing (SAC 2003), Melbourne, US, pp 784–788, DOI doi.org/10.1145/952532.952688

Deerwester S, Dumais ST, Furnas GW, Landauer TK, Harshman R (1990) Indexing by latent semantic analysis. Journal of the American Society for Information Science 41(6):391–407

Devlin J, Chang M, Lee K, Toutanova K (2019) BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2019), Minneapolis, US, pp 4171–4186

Dong Y, Liu P, Zhu Z, Wang Q, Zhang Q (2020) A fusion model-based label embedding and self-interaction attention for text classification. IEEE Access 8:30548–30559, DOI 10.1109/access.2019.2954985

Dumais ST, Platt J, Heckerman D, Sahami M (1998) Inductive learning algorithms and representations for text categorization. In: Proceedings of the 7th ACM International Conference on Information and Knowledge Management (CIKM 1998), Bethesda, US, pp 148–155, DOI 10.1145/288627.288651

Erhan D, Bengio Y, Courville A, Manzagol PA, Vincent P, Bengio S (2010) Why does unsupervised pre-training help deep learning? Journal of Machine Learning Research 11:625–660

Forman G (2004) A pitfall and solution in multi-class feature selection for text classification. In: Proceedings of the 21st International Conference on Machine Learning (ICML 2004), Banff, CA, pp 38–45, DOI 10.1145/1015330.1015356

Garneau N, Leboeuf J, Lamontagne L (2019) Contextual generation of word embeddings for out-of-vocabulary words in downstream tasks. In: Proceedings of the 32nd Canadian Conference on Artificial Intelligence (Canadian AI), Kingston, CA, pp 563–569, DOI 10.1007/978-3-030-18305-9_60

Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS 2010), Chia Laguna, Italy, pp 249–256

González P, Castaño A, Chawla NV, del Coz JJ (2017) A review on quantification learning. ACM Computing Surveys 50(5):74:1–74:40, DOI 10.1145/3117807

Grave E, Mikolov T, Joulin A, Bojanowski P (2017) Bag of tricks for efficient text classification. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2017), Valencia, ES, pp 427–431, DOI 10.18653/v1/e17-2068

Gupta S, Kanchinadam T, Conathan D, Fung G (2019) Task-optimized word embeddings for text classification representations. Frontiers in Applied Mathematics and Statistics 5:67

Harris ZS (1954) Distributional structure. Word 10(2-3):146–162, DOI 10.1007/978-94-017-6059-1_36

Hersh W, Buckley C, Leone T, Hickman D (1994) OHSUMED: An interactive retrieval evaluation and new large text collection for research. In: Proceedings of the 17th ACM International Conference on Research and Development in Information Retrieval (SIGIR 1994), Dublin, IE, pp 192–201, DOI 10.1007/978-1-4471-2099-5_20

Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Computation 9(8):1735–1780

Hsu DJ, Kakade SM, Langford J, Zhang T (2009) Multi-label prediction via compressed sensing. In: Proceedings of the 23rd Annual Conference on Neural Information Processing Systems (NIPS 2009), Vancouver, CA, pp 772–780

Jiang M, Liang Y, Feng X, Fan X, Pei Z, Xue Y, Guan R (2018) Text classification based on deep belief network and softmax regression. Neural Computing and Applications 29(1):61–70, DOI 10.1007/s00521-016-2401-x

Jin P, Zhang Y, Chen X, Xia Y (2016) Bag-of-embeddings for text classification. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2016), New York, US, pp 2824–2830

Joachims T (1998) Text categorization with support vector machines: Learning with many relevant features. In: Proceedings of the 10th European Conference on Machine Learning (ECML 1998), Chemnitz, DE, pp 137–142, DOI 10.1007/bfb0026683

Joachims T (2001) A statistical learning model of text classification for support vector machines. In: Proceedings of the 24th ACM Conference on Research and Development in Information Retrieval (SIGIR 2001), New Orleans, US, pp 128–136, DOI 10.1145/383952.383974

Kim Y (2014) Convolutional neural networks for sentence classification. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), Doha, QA, pp 1746–1751

Kim Y, Jernite Y, Sontag D, Rush AM (2016) Character-aware neural language models. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI 2016), Phoenix, US, pp 2741–2749

Kingma DP, Ba J (2015) Adam: A method for stochastic optimization. In: Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015), San Diego, US

Lai S, Xu L, Liu K, Zhao J (2015) Recurrent convolutional neural networks for text classification. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI 2015), Austin, US, pp 2267–2273

Lan M, Tan CL, Su J, Lu Y (2009) Supervised and traditional term weighting methods for automatic text categorization. IEEE Transactions on Pattern Analysis and Machine Intelligence 31(4):721–735

Le HT, Cerisara C, Denis A (2018) Do convolutional networks need to be deep for text classification? In: Proceedings of the AAAI 2018 Workshop on Affective Content Analysis, New Orleans, US, pp 29–36

LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444

Lei X, Cai Y, Xu J, Ren D, Li Q, Leung HF (2019) Incorporating task-oriented representation in text classification. In: Proceedings of the 24th International Conference on Database Systems for Advanced Applications (DASFAA 2019), Chiang Mai, TH, pp 401–415

Levy O, Goldberg Y (2014) Neural word embedding as implicit matrix factorization. In: Proceedings of the 28th Annual Conference on Neural Information Processing Systems (NIPS 2014), Montreal, CA, pp 2177–2185

Levy O, Goldberg Y, Dagan I (2015) Improving distributional similarity with lessons learned from word embeddings. Transactions of the Association for Computational Linguistics 3:211–225

Lewis DD (1992) An evaluation of phrasal and clustered representations on a text categorization task. In: Proceedings of the 15th ACM International Conference on Research and Development in Information Retrieval (SIGIR 1992), Kobenhavn, DK, pp 37–50

Lin J (2019) The neural hype and comparisons against weak baselines. SIGIR Forum 52(1):40–51

Luong T, Pham H, Manning CD (2015) Effective approaches to attention-based neural machine translation. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP 2015), Lisbon, PT, pp 1412–1421

van der Maaten L, Hinton G (2008) Visualizing data using t-SNE. Journal of Machine Learning Research 9:2579–2605

McCann B, Bradbury J, Xiong C, Socher R (2017) Learned in translation: Contextualized word vectors. In: Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, US, pp 6294–6305

Mikolov T, Chen K, Corrado G, Dean J (2013a) Efficient estimation of word representations in vector space. In: Workshop Track Proceedings of the 1st International Conference on Learning Representations (ICLR 2013), Scottsdale, US

Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013b) Distributed representations of words and phrases and their compositionality. In: Proceedings of the 27th Annual Conference on Neural Information Processing Systems (NIPS 2013), Lake Tahoe, US, pp 3111–3119

Mikolov T, Grave E, Bojanowski P, Puhrsch C, Joulin A (2018) Advances in pre-training distributed word representations. In: Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC 2018), Miyazaki, JP

Mnih A, Kavukcuoglu K (2013) Learning word embeddings efficiently with noise-contrastive estimation. In: Proceedings of the 27th Annual Conference on Neural Information Processing Systems (NIPS 2013), Lake Tahoe, US, pp 2265–2273

Moreo A, Esuli A, Sebastiani F (2016) Distributional correspondence indexing for cross-lingual and cross-domain sentiment classification. Journal of Artificial Intelligence Research 55:131–163

Moreo A, Esuli A, Sebastiani F (2020) Learning to weight for text classification. IEEE Transactions on Knowledge and Data Engineering 32(2):302–316, DOI 10.1109/TKDE.2018.2883446

Moreo A, Pedrotti A, Sebastiani F (2021) Heterogeneous document embeddings for cross-lingual text classification. In: Proceedings of the 36th ACM Symposium on Applied Computing (SAC 2021), Gwangju, KR, DOI 10.1145/3412841.3442093, forthcoming

Morik K, Brockhausen P, Joachims T (1999) Combining statistical learning with a knowledge-based approach. A case study in intensive care monitoring. In: Proceedings of the 16th International Conference on Machine Learning (ICML 1999), Bled, SL, pp 268–277

Pang B, Lee L (2008) Opinion mining and sentiment analysis. Foundations and Trends in Information Retrieval 2(1/2):1–135

Pappas N, Henderson J (2019) Gile: A generalized input-label embedding for text classification. Transactions of the Association for Computational Linguistics 7:139–155

Pennington J, Socher R, Manning C (2014) Glove: Global vectors for word representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), Doha, QA, pp 1532–1543

Peters ME, Neumann M, Iyyer M, Gardner M, Clark C, Lee K, Zettlemoyer L (2018) Deep contextualized word representations. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2018), New Orleans, US, pp 2227–2237

Ren H, Zeng Z, Cai Y, Du Q, Li Q, Xie H (2019) A weighted word embedding model for text classification. In: Proceedings of the 24th International Conference on Database Systems for Advanced Applications (DASFAA 2019), Chiang Mai, TH, pp 419–434

Riloff E, Wiebe J, Phillips W (2005) Exploiting subjectivity classification to improve information extraction. In: Proceedings of the 12th Conference of the American Association for Artificial Intelligence (AAAI 2005), Pittsburgh, US, pp 1106–1111

Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323(6088):533–536, DOI 10.1038/323533a0

Saerens M, Latinne P, Decaestecker C (2002) Adjusting the outputs of a classifier to new a priori probabilities: A simple procedure. Neural Computation 14(1):21–41, DOI 10.1162/089976602753284446

Sahlgren M (2005) An introduction to random indexing. In: Proceedings of the TKE Workshop on Methods and Applications of Semantic Indexing, Copenhagen, DK

Socher R, Perelygin A, Wu J, Chuang J, Manning CD, Ng A, Potts C (2013) Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP 2013), Seattle, US, pp 1631–1642

Soucy P, Mineau GW (2005) Beyond TFIDF weighting for text categorization in the vector space model. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005), Edinburgh, UK, pp 1130–1135

Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research 15:1929–1958

Steinberger R, Pouliquen B, Widiger A, Ignat C, Erjavec T, Tufis D, Varga D (2006) The JRC-Acquis: A multilingual aligned parallel corpus with 20+ languages. In: Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006), Genova, IT, pp 2142–2147

Tang J, Qu M, Mei Q (2015) PTE: Predictive text embedding through large-scale heterogeneous text networks. In: Proceedings of the 21st ACM International Conference on Knowledge Discovery and Data Mining (KDD 2015), Sydney, AU, pp 1165–1174

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. In: Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, US, pp 5998–6008

Wang G, Li C, Wang W, Zhang Y, Shen D, Zhang X, Henao R, Carin L (2018) Joint embedding of words and labels for text classification. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL 2018), Melbourne, AU, pp 2321–2331

Wang S, Manning CD (2012) Baselines and bigrams: Simple, good sentiment and topic classification. In: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL 2012), Jeju Island, KR, pp 90–94

Yang W, Lu K, Yang P, Lin J (2019a) Critically examining the "neural hype": Weak baselines and the additivity of effectiveness gains from neural ranking models. In: Proceedings of the 42nd ACM Conference on Research and De-

velopment in Information Retrieval (SIGIR 2019), Paris, FR, pp 1129–1132, DOI 10.1145/3331184.3331340

Yang Y, Chute CG (1994) An example-based mapping method for text categorization and retrieval. ACM Transactions on Information Systems 12(3):252–277

Yang Z, Dai Z, Yang Y, Carbonell JG, Salakhutdinov R, Le QV (2019b) XLNet: Generalized autoregressive pretraining for language understanding. In: Proceedings of the 33rd Annual Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, CA, pp 5754–5764

Yao S, Yu D, Xiao K (2019) Enhancing domain word embedding via latent semantic imputation. In: Proceedings of the 25th ACM Conference on Knowledge Discovery and Data Mining (KDD 2019), Anchorage, US, pp 557–565, DOI 10.1145/3292500.3330926

Yu HF, Jain P, Kar P, Dhillon I (2014) Large-scale multi-label learning with missing labels. In: Proceedings of the 31st International Conference on Machine Learning (ICML 2014), Beijing, CN, pp 593–601

Zhang L, Wang S, Liu B (2018) Deep learning for sentiment analysis: A survey. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 8(4):e1253, DOI 10.1002/widm.1253

Zhang X, Zhao J, LeCun Y (2015) Character-level convolutional networks for text classification. In: Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NIPS 2015), Montreal, CA, pp 649–657