Baseline Needs More Love: On Simple Word-Embedding-Based Models and Associated Pooling Mechanisms

Dinghan Shen[1], Guoyin Wang[1], Wenlin Wang[1], Martin Renqiang Min[2]
Qinliang Su[3], Yizhe Zhang[4], Chunyuan Li[1], Ricardo Henao[1], Lawrence Carin[1]
[1] Duke University [2] NEC Laboratories America [3] Sun Yat-sen University [4] Microsoft Research
dinghan.shen@duke.edu

Abstract

Many deep learning architectures have been proposed to model the *compositionality* in text sequences, requiring a substantial number of parameters and expensive computations. However, there has not been a rigorous evaluation regarding the added value of sophisticated compositional functions. In this paper, we conduct a point-by-point comparative study between Simple Word-Embedding-based Models (SWEMs), consisting of parameter-free pooling operations, relative to word-embedding-based RNN/CNN models. Surprisingly, SWEMs exhibit comparable or even superior performance in the majority of cases considered. Based upon this understanding, we propose two additional pooling strategies over learned word embeddings: (*i*) a max-pooling operation for improved interpretability; and (*ii*) a hierarchical pooling operation, which preserves spatial (n-gram) information within text sequences. We present experiments on 17 datasets encompassing three tasks: (*i*) (long) document classification; (*ii*) text sequence matching; and (*iii*) short text tasks, including classification and tagging. The source code and datasets can be obtained from https://github.com/dinghanshen/SWEM.

1 Introduction

Word embeddings, learned from massive unstructured text data, are widely-adopted building blocks for Natural Language Processing (NLP). By representing each word as a fixed-length vector, these embeddings can group semantically similar words, while implicitly encoding rich linguistic regularities and patterns (Bengio et al., 2003; Mikolov et al., 2013; Pennington et al., 2014). Leveraging the word-embedding construct, many deep architectures have been proposed to model the *compositionality* in variable-length text sequences. These methods range from simple operations like addition (Mitchell and Lapata, 2010; Iyyer et al., 2015), to more sophisticated compositional functions such as Recurrent Neural Networks (RNNs) (Tai et al., 2015; Sutskever et al., 2014), Convolutional Neural Networks (CNNs) (Kalchbrenner et al., 2014; Kim, 2014; Zhang et al., 2017a) and Recursive Neural Networks (Socher et al., 2011a).

Models with more expressive compositional functions, *e.g.*, RNNs or CNNs, have demonstrated impressive results; however, they are typically computationally expensive, due to the need to estimate hundreds of thousands, if not millions, of parameters (Parikh et al., 2016). In contrast, models with simple compositional functions often compute a sentence or document embedding by simply adding, or averaging, over the word embedding of each sequence element obtained via, *e.g.*, *word2vec* (Mikolov et al., 2013), or *GloVe* (Pennington et al., 2014). Generally, such a Simple Word-Embedding-based Model (SWEM) does not explicitly account for spatial, word-order information within a text sequence. However, they possess the desirable property of having significantly fewer parameters, enjoying much faster training, relative to RNN- or CNN-based models. Hence, there is a computation-*vs.*-expressiveness tradeoff regarding how to model the compositionality of a text sequence.

In this paper, we conduct an extensive experimental investigation to understand when, and why, simple pooling strategies, operated over word embeddings alone, already carry sufficient information for natural language understanding. To account for the distinct nature of various NLP tasks that may require different semantic features, we compare SWEM-based models with existing recurrent and convolutional networks in a point-by-point manner. Specifically, we consider 17 datasets, including three distinct NLP tasks: *document classification* (Yahoo news, Yelp reviews, *etc.*), *natural language sequence matching* (SNLI, WikiQA, *etc.*) and *(short) sentence classification/tagging* (Stanford sentiment treebank, TREC, *etc.*). Surprisingly, SWEMs exhibit comparable or even superior performance in the majority of cases considered.

In order to validate our experimental findings, we conduct additional investigations to understand to what extent *the word-order information* is utilized/required to make predictions on different tasks. We observe that in text representation tasks, many words (*e.g.*, stop words, or words that are not related to sentiment or topic) do not meaningfully contribute to the final predictions (*e.g.*, sentiment label). Based upon this understanding, we propose to leverage a *max-pooling* operation directly over the word embedding matrix of a given sequence, to select its most *salient* features. This strategy is demonstrated to extract complementary features relative to the standard averaging operation, while resulting in a more interpretable model. Inspired by a case study on sentiment analysis tasks, we further propose a *hierarchical pooling* strategy to abstract and preserve the spatial information in the final representations. This strategy is demonstrated to exhibit comparable empirical results to LSTM and CNN on tasks that are sensitive to word-order features, while maintaining the favorable properties of not having compositional parameters, thus fast training.

Our work presents a simple yet strong baseline for text representation learning that is widely ignored in benchmarks, and highlights the general computation-*vs.*-expressiveness tradeoff associated with appropriately selecting compositional functions for distinct NLP problems. Furthermore, we quantitatively show that the word-embedding-based text classification tasks can have the similar level of difficulty regardless of the employed models, using the subspace training (Li et al., 2018) to constrain the trainable parameters. Thus, according to Occam's razor, simple models are preferred.

2 Related Work

A fundamental goal in NLP is to develop expressive, yet computationally efficient compositional functions that can capture the linguistic structure of natural language sequences. Recently, several studies have suggested that on certain NLP applications, much simpler word-embedding-based architectures exhibit comparable or even superior performance, compared with more-sophisticated models using recurrence or convolutions (Parikh et al., 2016; Vaswani et al., 2017). Although complex compositional functions are avoided in these models, additional modules, such as attention layers, are employed on top of the word embedding layer. As a result, the specific role that the word embedding plays in these models is not emphasized (or explicit), which distracts from understanding how important the word embeddings alone are to the observed superior performance. Moreover, several recent studies have shown empirically that the advantages of distinct compositional functions are highly dependent on the specific task (Mitchell and Lapata, 2010; Iyyer et al., 2015; Zhang et al., 2015a; Wieting et al., 2015; Arora et al., 2016). Therefore, it is of interest to study the practical value of the additional expressiveness, on a wide variety of NLP problems.

SWEMs bear close resemblance to Deep Averaging Network (DAN) (Iyyer et al., 2015) or fastText (Joulin et al., 2016), where they show that average pooling achieves promising results on certain NLP tasks. However, there exist several key differences that make our work unique. First, we explore a series of pooling operations, rather than only average-pooling. Specifically, a *hierarchical* pooling operation is introduced to incorporate spatial information, which demonstrates superior results on sentiment analysis, relative to average pooling. Second, our work not only explores when simple pooling operations are enough, but also investigates the underlying reasons, *i.e.*, what semantic features are required for distinct NLP problems. Third, DAN and fastText only focused on one or two problems at a time, thus a comprehensive study regarding the effectiveness of various compositional functions on distinct NLP tasks, *e.g.*, categorizing short sentence/long documents, matching natural language sentences, has heretofore been absent. In response, our work seeks to perform a comprehensive comparison with respect to simple-*vs.*-complex compositional functions, across a wide range of NLP problems, and reveals some general rules for rationally selecting models to tackle different tasks.

3 Models & training

Consider a text sequence represented as X (either a sentence or a document), composed of a sequence of words: $\{w_1, w_2, ..., w_L\}$, where L is the number of tokens, *i.e.*, the sentence/document length. Let $\{v_1, v_2, ..., v_L\}$ denote the respective word embeddings for each token, where $v_l \in \mathbb{R}^K$. The compositional function, $X \rightarrow z$, aims to combine word embeddings into a fixed-length sentence/document representation z. These representations are then used to make predictions about sequence X. Below, we describe different types of functions considered in this work.

3.1 Recurrent Sequence Encoder

A widely adopted compositional function is defined in a recurrent manner: the model successively takes word vector v_t at position t, along with the hidden unit h_{t-1} from the last position $t-1$, to update the current hidden unit via $h_t = f(v_t, h_{t-1})$, where $f(\cdot)$ is the transition function.

To address the issue of learning long-term dependencies, $f(\cdot)$ is often defined as Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997), which employs *gates* to control the flow of information abstracted from a sequence. We omit the details of the LSTM and refer the interested readers to the work by Graves et al. (2013) for further explanation. Intuitively, the LSTM encodes a text sequence considering its word-order information, but yields additional compositional parameters that must be learned.

3.2 Convolutional Sequence Encoder

The Convolutional Neural Network (CNN) architecture (Kim, 2014; Collobert et al., 2011; Gan et al., 2017; Zhang et al., 2017b; Shen et al., 2018) is another strategy extensively employed as the compositional function to encode text sequences. The convolution operation considers windows of n consecutive words within the sequence, where a set of filters (to be learned) are applied to these word windows to generate corresponding *feature maps*. Subsequently, an aggregation operation (such as max-pooling) is used on top of the feature maps to abstract the most salient semantic features, resulting in the final representation. For most experiments, we consider a single-layer CNN text model. However, Deep CNN text models have also been developed (Conneau et al., 2016), and are considered in a few of our experiments.

3.3 Simple Word-Embedding Model (SWEM)

To investigate the raw modeling capacity of word embeddings, we consider a class of models with no additional compositional parameters to encode natural language sequences, termed SWEMs. Among them, the simplest strategy is to compute the element-wise average over word vectors for a given sequence (Wieting et al., 2015; Adi et al., 2016):

$$z = \frac{1}{L} \sum_{i=1}^{L} v_i \,. \qquad (1)$$

The model in (1) can be seen as an average pooling operation, which takes the mean over each of the K dimensions for all word embeddings, resulting in a representation z with the same dimension as the embedding itself, termed here SWEM-*aver*. Intuitively, z takes the information of every sequence element into account via the addition operation.

Max Pooling Motivated by the observation that, in general, only a small number of key words contribute to final predictions, we propose another SWEM variant, that extracts the most salient features from every word-embedding dimension, by taking the maximum value along each dimension of the word vectors. This strategy is similar to the max-over-time pooling operation in convolutional neural networks (Collobert et al., 2011):

$$z = \text{Max-pooling}(v_1, v_2, ..., v_L) \,. \qquad (2)$$

We denote this model variant as SWEM-*max*. Here the j-th component of z is the maximum element in the set $\{v_{1j}, \dots, v_{Lj}\}$, where v_{1j} is, for example, the j-th component of v_1. With this pooling operation, those words that are unimportant or unrelated to the corresponding tasks will be ignored in the encoding process (as the components of the embedding vectors will have small amplitude), unlike SWEM-*aver* where every word contributes equally to the representation.

Considering that SWEM-*aver* and SWEM-*max* are complementary, in the sense of accounting for different types of information from text sequences,

Model	Parameters	Complexity	Sequential Ops
CNN	$n \cdot K \cdot d$	$\mathcal{O}(n \cdot L \cdot K \cdot d)$	$\mathcal{O}(1)$
LSTM	$4 \cdot d \cdot (K + d)$	$\mathcal{O}(L \cdot d^2 + L \cdot K \cdot d)$	$\mathcal{O}(L)$
SWEM	0	$\mathcal{O}(L \cdot K)$	$\mathcal{O}(1)$

Table 1: Comparisons of CNN, LSTM and SWEM architectures. Columns correspond to the number of *compositional* parameters, computational complexity and sequential operations, respectively.

we also propose a third SWEM variant, where the two abstracted features are concatenated together to form the sentence embeddings, denoted here as SWEM-*concat*. For all SWEM variants, there are no additional compositional parameters to be learned. As a result, the models only exploit intrinsic word embedding information for predictions.

Hierarchical Pooling Both SWEM-*aver* and SWEM-*max* do not take word-order or spatial information into consideration, which could be useful for certain NLP applications. So motivated, we further propose a *hierarchical* pooling layer. Let $v_{i:i+n-1}$ refer to the *local* window consisting of n consecutive words words, $v_i, v_{i+1}, ..., v_{i+n-1}$. First, an average-pooling is performed on each local window, $v_{i:i+n-1}$. The extracted features from all windows are further down-sampled with a *global* max-pooling operation on top of the representations for every window. We call this approach SWEM-*hier* due to its layered pooling.

This strategy preserves the local spatial information of a text sequence in the sense that it keeps track of how the sentence/document is constructed from individual word windows, *i.e.*, n-grams. This formulation is related to bag-of-n-grams method (Zhang et al., 2015b). However, SWEM-*hier* learns fixed-length representations for the n-grams that appear in the corpus, rather than just capturing their occurrences via count features, which may potentially advantageous for prediction purposes.

3.4 Parameters & Computation Comparison

We compare CNN, LSTM and SWEM wrt their parameters and computational speed. K denotes the dimension of word embeddings, as above. For the CNN, we use n to denote the filter width (assumed constant for all filters, for simplicity of analysis, but in practice variable n is commonly used). We define d as the dimension of the final sequence representation. Specifically, d represents the dimension of hidden units or the number of filters in LSTM or CNN, respectively.

We first examine the number of *compositional parameters* for each model. As shown in Table 1, both the CNN and LSTM have a large number of parameters, to model the semantic compositionality of text sequences, whereas SWEM has no such parameters. Similar to Vaswani et al. (2017), we then consider the computational complexity and the minimum number of sequential operations required for each model. SWEM tends to be more efficient than CNN and LSTM in terms of computation complexity. For example, considering the case where $K = d$, SWEM is faster than CNN or LSTM by a factor of nd or d, respectively. Further, the computations in SWEM are highly parallelizable, unlike LSTM that requires $\mathcal{O}(L)$ sequential steps.

4 Experiments

We evaluate different compositional functions on a wide variety of supervised tasks, including document categorization, text sequence matching (given a sentence pair, X_1, X_2, predict their relationship, y) as well as (short) sentence classification. We experiment on 17 datasets concerning natural language understanding, with corresponding data statistics summarized in the Supplementary Material.

We use GloVe word embeddings with $K = 300$ (Pennington et al., 2014) as initialization for all our models. Out-Of-Vocabulary (OOV) words are initialized from a uniform distribution with range $[-0.01, 0.01]$. The GloVe embeddings are employed in two ways to learn refined word embeddings: (i) directly updating each word embedding during training; and (ii) training a 300-dimensional Multilayer Perceptron (MLP) layer with ReLU activation, with GloVe embeddings as input to the MLP and with output defining the refined word embeddings. The latter approach corresponds to learning an MLP model that adapts GloVe embeddings to the dataset and task of interest. The advantages of these two methods differ from dataset to dataset. We choose the better strategy based on their corresponding performances on the validation set. The final classifier is implemented as an MLP layer with dimension selected from the set $[100, 300, 500, 1000]$, followed by a sigmoid or softmax function, depending on the specific task.

Adam (Kingma and Ba, 2014) is used to optimize all models, with learning rate selected from

Model	Yahoo! Ans.	AG News	Yelp P.	Yelp F.	DBpedia
Bag-of-means*	60.55	83.09	87.33	53.54	90.45
Small word CNN*	69.98	89.13	94.46	58.59	98.15
Large word CNN*	70.94	91.45	95.11	59.48	98.28
LSTM*	70.84	86.06	94.74	58.17	98.55
Deep CNN (29 layer)†	73.43	91.27	**95.72**	**64.26**	**98.71**
fastText ‡	72.0	91.5	93.8	60.4	98.1
fastText (bigram)‡	72.3	92.5	95.7	63.9	98.6
SWEM-*aver*	73.14	91.71	93.59	60.66	98.42
SWEM-*max*	72.66	91.79	93.25	59.63	98.24
SWEM-*concat*	**73.53**	**92.66**	93.76	61.11	**98.57**
SWEM-*hier*	73.48	92.48	**95.81**	**63.79**	98.54

Table 2: Test accuracy on (long) document classification tasks, in percentage. Results marked with $*$ are reported in Zhang et al. (2015b), with † are reported in Conneau et al. (2016), and with ‡ are reported in Joulin et al. (2016).

Politics	Science	Computer	Sports	Chemistry	Finance	Geoscience
philipdru	coulomb	system32	billups	sio2 (SiO_2)	proprietorship	fossil
justices	differentiable	cobol	midfield	nonmetal	ameritrade	zoos
impeached	paranormal	agp	sportblogs	pka	retailing	farming
impeachment	converge	dhcp	mickelson	chemistry	mlm	volcanic
neocons	antimatter	win98	juventus	quarks	budgeting	ecosystem

Table 3: Top five words with the largest values in a given word-embedding dimension (each column corresponds to a dimension). The first row shows the (manually assigned) topic for words in each column.

the set $[1 \times 10^{-3}, 3 \times 10^{-4}, 2 \times 10^{-4}, 1 \times 10^{-5}]$ (with cross-validation used to select the appropriate parameter for a given dataset and task). Dropout regularization (Srivastava et al., 2014) is employed on the word embedding layer and final MLP layer, with dropout rate selected from the set $[0.2, 0.5, 0.7]$. The batch size is selected from $[2, 8, 32, 128, 512]$.

4.1 Document Categorization

We begin with the task of categorizing documents (with approximately 100 words in average per document). We follow the data split in Zhang et al. (2015b) for comparability. These datasets can be generally categorized into three types: *topic categorization* (represented by Yahoo! Answer and AG news), *sentiment analysis* (represented by Yelp Polarity and Yelp Full) and *ontology classification* (represented by DBpedia). Results are shown in Table 2. Surprisingly, on topic prediction tasks, our SWEM model exhibits stronger performances, relative to both LSTM and CNN compositional architectures, this by leveraging both the average and max-pooling features from word embeddings. Specifically, our SWEM-*concat* model even outperforms a 29-layer deep CNN model (Conneau et al., 2016), when predicting topics. On the ontology classification problem (DBpedia dataset), we observe the same trend, that SWEM exhibits comparable or even superior results, relative to CNN or LSTM models.

Since there are no compositional parameters in SWEM, our models have an order of magnitude fewer parameters (excluding embeddings) than LSTM or CNN, and are considerably more computationally efficient. As illustrated in Table 4, SWEM-*concat* achieves better results on Yahoo! Answer than CNN/LSTM, with only 61K parameters (one-tenth the number of LSTM parameters, or one-third the number of CNN parameters), while taking a fraction of the training time relative to the CNN or LSTM.

Model	Parameters	Speed
CNN	541K	171s
LSTM	1.8M	598s
SWEM	**61K**	**63s**

Table 4: Speed & Parameters on Yahoo! Answer dataset.

Interestingly, for the sentiment analysis tasks, both CNN and LSTM compositional functions perform better than SWEM, suggesting that word-order information may be required for analyzing sentiment orientations. This finding is consistent with Pang et al. (2002), where they hypothesize that the positional information of a word in text sequences may be beneficial to predict sentiment. This is intuitively reasonable since, for

Model	SNLI	MultiNLI Matched	MultiNLI Mismatched	WikiQA		Quora	MSRP	
	Acc.	*Acc.*	*Acc.*	*MAP*	*MRR*	*Acc.*	*Acc.*	*F1*
CNN	82.1	65.0	65.3	0.6752	0.6890	79.60	69.9	80.9
LSTM	80.6	66.9*	66.9*	**0.6820**	**0.6988**	82.58	70.6	80.5
SWEM-*aver*	82.3	66.5	66.2	**0.6808**	**0.6922**	82.68	71.0	81.1
SWEM-*max*	**83.8**	**68.2**	**67.7**	0.6613	0.6717	82.20	70.6	80.8
SWEM-*concat*	83.3	67.9	67.6	0.6788	0.6908	**83.03**	**71.5**	**81.3**

Table 5: Performance of different models on matching natural language sentences. Results with * are for Bidirectional LSTM, reported in Williams et al. (2017). Our reported results on MultiNLI are only trained MultiNLI training set (without training data from SNLI). For MSRP dataset, we follow the setup in Hu et al. (2014) and do not use any additional features.

instance, the phrase "not really good" and "really not good" convey different levels of negative sentiment, while being different only by their word orderings. Contrary to SWEM, CNN and LSTM models can both capture this type of information via convolutional filters or recurrent transition functions. However, as suggested above, such word-order patterns may be much less useful for predicting the topic of a document. This may be attributed to the fact that word embeddings alone already provide sufficient topic information of a document, at least when the text sequences considered are relatively long.

4.1.1 Interpreting model predictions

Although the proposed SWEM-*max* variant generally performs a slightly worse than SWEM-*aver*, it extracts complementary features from SWEM-*aver*, and hence in most cases SWEM-*concat* exhibits the best performance among all SWEM variants. More importantly, we found that the word embeddings learned from SWEM-*max* tend to be sparse. We trained our SWEM-*max* model on the Yahoo datasets (randomly initialized). With the learned embeddings, we plot the values for each of the word embedding dimensions, for the entire vocabulary. As shown in Figure 1, most of the values are highly concentrated around zero, indicating that the word embeddings learned are very sparse. On the contrary, the GloVe word embeddings, for the same vocabulary, are considerably denser than the embeddings learned from SWEM-*max*. This suggests that the model may only depend on a few key words, among the entire vocabulary, for predictions (since most words do not contribute to the max-pooling operation in SWEM-*max*). Through the embedding, the model learns the important words for a given task (those words with non-zero embedding components).

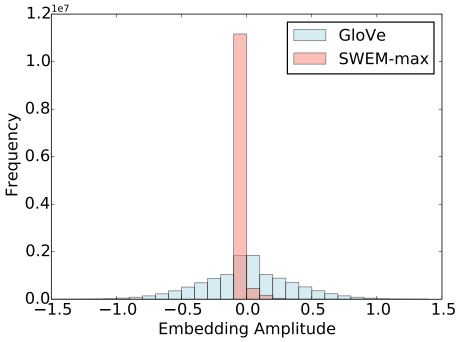


Figure 1: Histograms for learned word embeddings (randomly initialized) of SWEM-*max* and GloVe embeddings for the same vocabulary, trained on the Yahoo! Answer dataset.

In this regard, the nature of max-pooling process gives rise to a more interpretable model. For a document, only the word with largest value in each embedding dimension is employed for the final representation. Thus, we suspect that semantically similar words may have large values in some shared dimensions. So motivated, after training the SWEM-*max* model on the Yahoo dataset, we selected five words with the largest values, among the entire vocabulary, for each word embedding dimension (these words are selected preferentially in the corresponding dimension, by the max operation). As shown in Table 3, the words chosen wrt each embedding dimension are indeed highly relevant and correspond to a common topic (the topics are inferred from words). For example, the words in the first column of Table 3 are all political terms, which could be assigned to the *Politics & Government* topic. Note that our model can even learn locally interpretable structure that is not explicitly indicated by the label information. For instance, all words in the fifth column are *Chemistry*-related. However, we do not have a chemistry label in the dataset, and regardless they should belong to the *Science* topic.

4.2 Text Sequence Matching

To gain a deeper understanding regarding the modeling capacity of word embeddings, we further investigate the problem of sentence matching, including natural language inference, answer sentence selection and paraphrase identification. The corresponding performance metrics are shown in Table 5. Surprisingly, on most of the datasets considered (except WikiQA), SWEM demonstrates the best results compared with those with CNN or the LSTM encoder. Notably, on SNLI dataset, we observe that SWEM-*max* performs the best among all SWEM variants, consistent with the findings in Nie and Bansal (2017); Conneau et al. (2017), that *max-pooling* over BiLSTM hidden units outperforms average pooling operation on SNLI dataset. As a result, with only 120K parameters, our SWEM-*max* achieves a test accuracy of 83.8%, which is very competitive among state-of-the-art sentence encoding-based models (in terms of both performance and number of parameters)[1].

The strong results of the SWEM approach on these tasks may stem from the fact that when matching natural language sentences, it is sufficient in most cases to simply model the word-level alignments between two sequences (Parikh et al., 2016). From this perspective, word-order information becomes much less useful for predicting relationship between sentences. Moreover, considering the simpler model architecture of SWEM, they could be much easier to be optimized than LSTM or CNN-based models, and thus give rise to better empirical results.

4.2.1 Importance of word-order information

One possible disadvantage of SWEM is that it ignores the word-order information within a text sequence, which could be potentially captured by CNN- or LSTM-based models. However, we empirically found that except for sentiment analysis, SWEM exhibits similar or even superior performance as the CNN or LSTM on a variety of tasks. In this regard, one natural question would be: how important are word-order features for these tasks? To this end, we randomly shuffle the words for every sentence in the training set, while keeping the original word order for samples in the test set. The motivation here is to remove the word-order features from the training set and examine how sensitive the performance on different tasks are to word-order information. We use LSTM as the model for this purpose since it can captures word-order information from the original training set.

Datasets	**Yahoo**	**Yelp P.**	**SNLI**
Original	72.78	95.11	78.02
Shuffled	72.89	93.49	77.68

Table 6: Test accuracy for LSTM model trained on original/shuffled training set.

The results on three distinct tasks are shown in Table 6. Somewhat surprisingly, for Yahoo and SNLI datasets, the LSTM model trained on shuffled training set shows comparable accuracies to those trained on the original dataset, indicating that word-order information does not contribute significantly on these two problems, *i.e.*, topic categorization and textual entailment. However, on the Yelp polarity dataset, the results drop noticeably, further suggesting that word-order does matter for sentiment analysis (as indicated above from a different perspective).

Notably, the performance of LSTM on the Yelp dataset with a shuffled training set is very close to our results with SWEM, indicating that the main difference between LSTM and SWEM may be due to the ability of the former to capture word-order features. Both observations are in consistent with our experimental results in the previous section.

Case Study To understand what type of sentences are sensitive to word-order information, we further show those samples that are wrongly predicted because of the shuffling of training data in Table 7. Taking the first sentence as an example, several words in the review are generally positive, *i.e. friendly*, *nice*, *okay*, *great* and *likes*. However, the most vital features for predicting the sentiment of this sentence could be the phrase/sentence *'is just okay'*, *'not great'* or *'makes me wonder why everyone likes'*, which cannot be captured without considering word-order features. It is worth noting the hints for predictions in this case are actually n-gram phrases from the input document.

4.3 SWEM-*hier* for sentiment analysis

As demonstrated in Section 4.2.1, word-order information plays a vital role for sentiment analysis tasks. However, according to the case study above, the most important features for sentiment prediction may be some key n-gram phrase/words from

[1] See leaderboard at https://nlp.stanford.edu/projects/snli/ for details.

Negative:	Friendly staff and nice selection of vegetarian options. Food **is just okay**, **not great**. **Makes me wonder why everyone likes** food fight so much.
Positive:	The store is small, but it carries specialties that are difficult to find in Pittsburgh. I **was particularly excited** to find middle eastern chili sauce and chocolate covered turkish delights.

Table 7: Test samples from Yelp Polarity dataset for which LSTM gives wrong predictions with shuffled training data, but predicts correctly with the original training set.

the input document. We hypothesize that incorporating information about the local word-order, *i.e.*, n-gram features, is likely to largely mitigate the limitations of the above three SWEM variants. Inspired by this observation, we propose using another simple pooling operation termed as hierarchical (SWEM-*hier*), as detailed in Section 3.3. We evaluate this method on the two document-level sentiment analysis tasks and the results are shown in the last row of Table 2.

SWEM-*hier* greatly outperforms the other three SWEM variants, and the corresponding accuracies are comparable to the results of CNN or LSTM (Table 2). This indicates that the proposed hierarchical pooling operation manages to abstract spatial (word-order) information from the input sequence, which is beneficial for performance in sentiment analysis tasks.

4.4 Short Sentence Processing

We now consider sentence-classification tasks (with approximately 20 words on average). We experiment on three sentiment classification datasets, *i.e.*, MR, SST-1, SST-2, as well as subjectivity classification (Subj) and question classification (TREC). The corresponding results are shown in Table 8. Compared with CNN/LSTM compositional functions, SWEM yields inferior accuracies on sentiment analysis datasets, consistent with our observation in the case of document categorization. However, SWEM exhibits comparable performance on the other two tasks, again with much less parameters and faster training. Further, we investigate two sequence tagging tasks: the standard CoNLL2000 chunking and CoNLL2003 NER datasets. Results are shown in the Supplementary Material, where LSTM and CNN again perform better than SWEMs. Generally, SWEM is less effective at extracting representations from *short* sentences than from *long* documents. This may be due to the fact that for a shorter text sequence, word-order features tend to be more important since the semantic information provided by word embeddings alone is relatively limited.

Moreover, we note that the results on these relatively small datasets are highly sensitive to model regularization techniques due to the overfitting issues. In this regard, one interesting future direction may be to develop specific regularization strategies for the SWEM framework, and thus make them work better on small sentence classification datasets.

5 Discussion

5.1 Comparison via subspace training

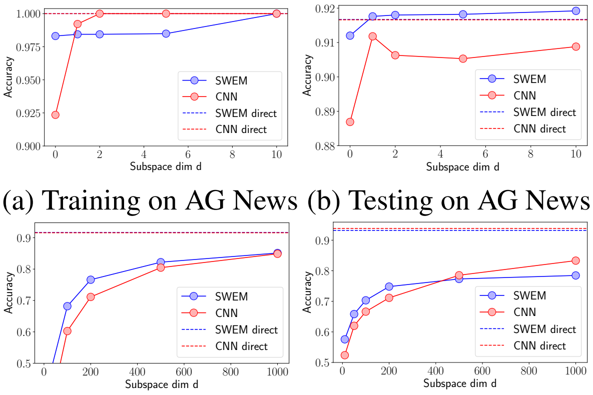
We use *subspace training* (Li et al., 2018) to measure the model complexity in text classification problems. It constrains the optimization of the trainable parameters in a subspace of low dimension d, the intrinsic dimension d_{int} defines the minimum d that yield a good solution. Two models are studied: the SWEM-*max* variant, and the CNN model including a convolutional layer followed by a FC layer. We consider two settings:

(1) The word embeddings are randomly intialized, and optimized jointly with the model parameters. We show the performance of direct and subspace training on AG News dataset in Figure 2 (a)(b). The two models trained via direct method share almost identical perfomrnace on training and testing. The subspace training yields similar accuracy with direct training for very small d, even when model parameters are not trained at all ($d = 0$). This is because the word embeddings have the full degrees of freedom to adjust to achieve good solutions, regardless of the employed models. SWEM seems to have an easier loss landspace than CNN for word embeddings to find the best solutions. According to Occam's razor, simple models are preferred, if all else are the same.

(2) The pre-trained GloVe are frozen for the word embeddings, and only the model parameters are optimized. The results on testing datasets of AG News and Yelp P. are shown in Figure 2 (c)(d), respectively. SWEM shows significantly higher accuracy than CNN for a large range of low subspace dimension, indicating that SWEM is more parameter-efficient to get a decent solution. In Figure 2(c), if we set the performance threshold

Model	MR	SST-1	SST-2	Subj	TREC
RAE (Socher et al., 2011b)	77.7	43.2	82.4	–	–
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–
LSTM (Tai et al., 2015)	–	46.4	84.9	–	–
RNN (Zhao et al., 2015)	77.2	–	–	**93.7**	90.2
Constituency Tree-LSTM (Tai et al., 2015)	-	**51.0**	88.0	-	-
Dynamic CNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0
CNN (Kim, 2014)	**81.5**	48.0	**88.1**	93.4	**93.6**
DAN-ROOT (Iyyer et al., 2015)	-	46.9	85.7	-	-
SWEM-*aver*	77.6	45.2	83.9	92.5	**92.2**
SWEM-*max*	76.9	44.1	83.6	91.2	89.0
SWEM-*concat*	**78.2**	**46.1**	**84.3**	**93.0**	91.8

Table 8: Test accuracies with different compositional functions on (short) sentence classifications.



(a) Training on AG News (b) Testing on AG News

(c) Testing on AG News (d)Testing on Yelp P.

Figure 2: Performance of subspace training. Word embeddings are optimized in (a)(b), and frozen in (c)(d).

as 80% testing accuracy, SWEM exhibits a lower d_{int} than CNN on AG News dataset. However, in Figure 2(d), CNN can leverage more trainable parameters to achieve higher accuracy when d is large.

5.2 Linear classifiers

To further investigate the quality of representations learned from SWEMs, we employ a linear classifier on top of the representations for prediction, instead of a non-linear MLP layer as in the previous section. It turned out that utilizing a linear classifier only leads to a very small performance drop for both Yahoo! Ans. (from 73.53% to 73.18%) and Yelp P. datasets (from 93.76% to 93.66%) . This observation highlights that SWEMs are able to extract robust and informative sentence representations despite their simplicity.

5.3 Extension to other languages

We have also tried our SWEM-concat and SWEM-hier models on Sogou news corpus (with the same experimental setup as (Zhang et al., 2015b)), which is a *Chinese* dataset represented by Pinyin (a phonetic romanization of Chinese). SWEM-concat yields an accuracy of 91.3%, while SWEM-hier (with a local window size of 5) obtains an accuracy of 96.2% on the test set. Notably, the performance of SWEM-hier is comparable to the best accuracies of CNN (95.6%) and LSTM (95.2%), as reported in (Zhang et al., 2015b). This indicates that hierarchical pooling is more suitable than average/max pooling for Chinese text classification, by taking spatial information into account. It also implies that Chinese is more sensitive to local word-order features than English.

6 Conclusions

We have performed a comparative study between SWEM (with parameter-free pooling operations) and CNN or LSTM-based models, to represent text sequences on 17 NLP datasets. We further validated our experimental findings through additional exploration, and revealed some general rules for rationally selecting compositional functions for distinct problems. Our findings regarding when (and why) simple pooling operations are enough for text sequence representations are summarized as follows:

- Simple pooling operations are surprisingly effective at representing longer documents (with hundreds of words), while recurrent/convolutional compositional functions are most effective when constructing representations for short sentences.

- Sentiment analysis tasks are more sensitive to word-order features than topic categorization tasks. However, a simple *hierarchical pooling layer* proposed here achieves comparable results to LSTM/CNN on sentiment analysis tasks.

- To match natural language sentences, *e.g.*, textual entailment, answer sentence selection, *etc.*, simple pooling operations already exhibit similar or even superior results, compared to CNN and LSTM.

• In SWEM with max-pooling operation, each *individual dimension* of the word embeddings contains interpretable semantic patterns, and groups together words with a common theme or *topic*.

References

Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. 2016. Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. *ICLR*.

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2016. A simple but tough-to-beat baseline for sentence embeddings. In *ICLR*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *JMLR*, 3(Feb):1137–1155.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *JMLR*, 12(Aug):2493–2537.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. *EMNLP*.

Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. 2016. Very deep convolutional networks for natural language processing. *arXiv preprint arXiv:1606.01781*.

Zhe Gan, Yunchen Pu, Ricardo Henao, Chunyuan Li, Xiaodong He, and Lawrence Carin. 2017. Learning generic sentence representations using convolutional neural networks. In *EMNLP*, pages 2380–2390.

Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. 2013. Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278. IEEE.

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

Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *NIPS*, pages 2042–2050.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015. Deep unordered composition rivals syntactic methods for text classification. In *ACL*, volume 1, pages 16 81–1691.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *EMNLP*.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. 2018. Measuring the intrinsic dimension of objective landscapes. In *International Conference on Learning Representations*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119.

Jeff Mitchell and Mirella Lapata. 2010. Composition in distributional models of semantics. *Cognitive science*, 34(8):1388–1429.

Yixin Nie and Mohit Bansal. 2017. Shortcut-stacked sentence encoders for multi-domain inference. *arXiv preprint arXiv:1708.02312*.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up?: sentiment classification using machine learning techniques. In *EMNLP*, pages 79–86. ACL.

Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. *EMNLP*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543.

Dinghan Shen, Martin Renqiang Min, Yitong Li, and Lawrence Carin. 2017. Adaptive convolutional filter generation for natural language understanding. *arXiv preprint arXiv:1709.08294*.

Dinghan Shen, Yizhe Zhang, Ricardo Henao, Qinliang Su, and Lawrence Carin. 2018. Deconvolutional latent-variable model for text sequence matching. *AAAI*.

Richard Socher, Brody Huval, Christopher D Manning, and Andrew Y Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *EMNLP*, pages 1201–1211. Association for Computational Linguistics.

Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. 2011a. Parsing natural scenes and natural language with recursive neural networks. In *ICML*, pages 129–136.

Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. 2011b. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *EMNLP*, pages 151–161. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *NIPS*.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2015. Towards universal paraphrastic sentence embeddings. *ICLR*.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Shiliang Zhang, Hui Jiang, Mingbin Xu, Junfeng Hou, and Lirong Dai. 2015a. The fixed-size ordinally-forgetting encoding method for neural network language models. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 495–500.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015b. Character-level convolutional networks for text classification. In *NIPS*, pages 649–657.

Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, and Lawrence Carin. 2017a. Adversarial feature matching for text generation. In *ICML*.

Yizhe Zhang, Dinghan Shen, Guoyin Wang, Zhe Gan, Ricardo Henao, and Lawrence Carin. 2017b. Deconvolutional paragraph representation learning. *NIPS*.

Han Zhao, Zhengdong Lu, and Pascal Poupart. 2015. Self-adaptive hierarchical sentence model. In *IJCAI*, pages 4069–4076.

Appendix I: Experimental Setup

6.1 Data statistics

We consider a wide range of text-representation-based tasks in this paper, including *document categorization*, *text sequence matching* and *(short) sentence classification*. For document classification tasks, we use the same data splits in (Zhang et al., 2015b) (downloaded from https://goo.gl/QaRpr7); for short sentence classification, we employ the same training/testing data and preprocessing procedure with (Kim, 2014). The statistics and corresponding types of these datasets are summarized in Table 9

Datasets	#w	#c	Train	Types
Yahoo	104	10	1,400K	Topic categorization
AG News	43	4	120K	Topic categorization
Yelp P.	138	2	560K	Sentiment analysis
Yelp F.	152	5	650K	Sentiment analysis
DBpedia	57	14	560K	Ontology classification
SNLI	11 / 6	3	549K	Textual Entailment
MultiNLI	21/11	3	393K	Textual Entailment
WikiQA	7 / 26	2	20K	Question answering
Quora	13 / 13	2	384K	Paraphrase identification
MSRP	23 / 23	2	4K	Paraphrase identification
MR	20	2	11K	Sentiment analysis
SST-1	18	5	12K	Sentiment analysis
SST-2	19	2	10K	Sentiment analysis
Subj	23	2	10K	Subjectivity classification
TREC	10	6	6K	Question classification

Table 9: Data Statistics. Where **#w**, **#c** and Train denote the average number of words, the number of classes and the size of training set, respectively. For sentence matching datasets, **#w** stands for the average length for the two corresponding sentences.

6.2 Sequence Tagging Results

Datasets	CoNLL2000	CoNLL2003
CNN-CRF	94.32	89.59
BI-LSTM-CRF	94.46	90.10
SWEM-CRF	90.34	86.28

Table 10: The results (F1 score) on sequence tagging tasks.

SWEM-CRF indicates that CRF is directly operated on top of the word embedding layer and make predictions for each word (there is no contextual/word-order information before CRF layer, compared to CNN-CRF or BI-LSTM-CRF). As shown above, CNN-CRF and BI-LSTM-CRF consistently outperform SWEM-CRF on both sequence tagging tasks, although the training takes around 4 to 5 times longer (for BI-LSTM-CRF) than SWEM-CRF. This suggests that for chunking and NER, compositional functions such as LSTM or CNN are very necessary, because of the sequential (order-sensitive) nature of sequence tagging tasks.

6.3 What are the key words used for predictions?

Given the sparsity of word embeddings, one natural question would be: What are those *key words* that are leveraged by the model to make predictions? To this end, after training SWEM-*max* on Yahoo! Answer dataset, we selected the top-10 words (with the maximum values in that dimension) for every word embedding dimension. The results are visualized in Figure 3. These words are indeed very predictive since they are likely to occur in documents with a specific topic, as discussed above. Another interesting observation is that the frequencies of these words are actually quite low in the training set (*e.g. colston*: 320, *repubs*: 255 *win32*: 276), considering the large size of the training set (1,400K). This suggests that the model is utilizing those relatively rare, yet representative words of each topic for the final predictions.

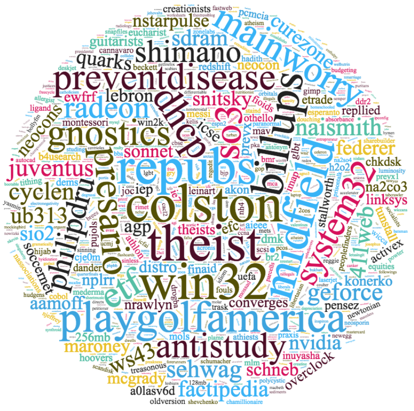


Figure 3: The top 10 words for each word embeddings' dimension.

6.4 How many word embedding dimensions are needed?

Since there are no compositional parameters in SWEM, the component that contains the semantic

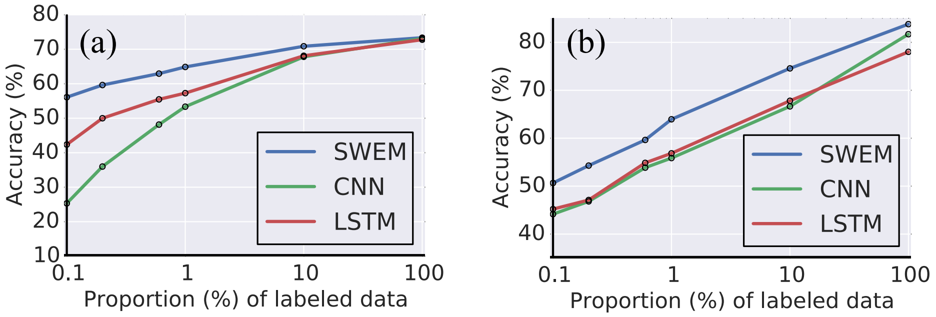


Figure 4: The test accuracy comparisons between SWEM and CNN/LSTM models on (a) Yahoo! Answers dataset and (b) SNLI dataset, with different proportions of training data (ranging from 0.1% to 100%).

information of a text sequence is the word embedding. Thus, it is of interest to see how many word embedding dimensions are needed for a SWEM architecture to perform well. To this end, we vary the dimension from 3 to 1000 and train a SWEM-*concat* model on the Yahoo dataset. For fair comparison, the word embeddings are randomly initialized in this experiment, since there are no pretrained word vectors, such as GloVe (Pennington et al., 2014), for some dimensions we consider. As shown in Table 11, the model exhibits higher accuracy with larger word embedding dimensions. This is not surprising since with more embedding dimensions, more semantic features could be potentially encapsulated. However, we also observe that even with only 10 dimensions, SWEM demonstrates comparable results relative to the case with 1000 dimensions, suggesting that word embeddings are very efficient at abstracting semantic information into fixed-length vectors. This property indicates that we may further reduce the number of model parameters with lower-dimensional word embeddings, while still achieving competitive results.

6.5 Sensitivity of compositional functions to sample size

To explore the robustness of different compositional functions, we consider another application scenario, where we only have a limited number of training data, *e.g.*, when labeled data are expensive to obtain. To investigate this, we re-run the experiments on Yahoo and SNLI datasets, while employing increasing proportions of the original training set. Specifically, we use 0.1%, 0.2%, 0.6%, 1.0%, 10%, 100% for comparison; the corresponding results are shown in Figure 4.

# Dim.	3	10	30	100	300	1000
Yahoo	64.05	**72.62**	73.13	73.12	73.24	73.31

Table 11: Test accuracy of SWEM on Yahoo dataset with a wide range of word embedding dimensions.

Surprisingly, SWEM consistently outperforms CNN and LSTM models by a large margin, on a wide range of training data proportions. For instance, with 0.1% of the training samples from Yahoo dataset (around 1.4K labeled data), SWEM achieves an accuracy of 56.10%, which is much better than that of models with CNN (25.32%) or LSTM (42.37%). On the SNLI dataset, we also noticed the same trend that the SWEM architecture result in much better accuracies, with a fraction of training data. This observation indicates that overfitting issues in CNN or LSTM-based models on text data mainly stems from over-complicated compositional functions, rather than the word embedding layer. More importantly, SWEM tends to be a far more robust model when only limited data are available for training.