Word Embedding Aware Convolutional Networks for Sentiment Analysis

Shrey Desai
University of Texas at Austin
shreydesai@utexas.edu

Abstract

Convolutional neural networks (CNNs) have become extremely popular models for text classification. Traditional architectures perform convolutions on dense vector representations of the input text, also known as *word embeddings*. While researchers have explored various architectures to advance the state-of-the-art, the importance of the initial word embedding layer has not been considered as closely. This paper seeks to bridge this gap by empirically analyzing how using different word embeddings impact performance.[1]

1 Introduction

Convolutional neural networks (CNNs) have achieved impressive results in computer vision and speech recognition [1]. Recently, CNNs have overtaken traditional linear models in NLP for tasks such as text classification [2]. These models leverage distributed representations of words—commonly referred to as *word embeddings*—as their inputs. CNNs utilize linear filters that convolve across input regions to extract salient features. For text classification specifically, the filters learn to emphasize certain n-gram structures through backpropagation.

CNN architectures begin with a word embedding layer that replaces tokens with word vector representations, forming a sentence matrix. These vectors are either sampled from standard distributions or pre-populated using popular unsupervised algorithms. Depending on the method used, the vectors differ in their embedded syntactic and semantic information. This paper seeks to explore the impact of such variation, as most architectures do not consider using different word embeddings in the optimization process.

Using a sentence-level CNN similar to Kim's [3], we explore the effects of various word embedding initializations on sentiment analysis tasks. We consider embeddings obtained through a standard Gaussian, Word2vec, GloVe, and Numberbatch. The models are supervised on multiple sentiment datasets that contain "positive" and "negative" labels for sentences. In addition to reporting the performance, we also analyze how the models use the embeddings to make predictions.

2 Model

Each corpus is a collection of sentences with variable lengths. Let V represent the vocabulary of the corpus. Sentences are encoded as a sequence $s = [s_1, s_2, \cdots, s_n]$ where $s_i \in V$. In order to create a uniform representation, we fix n to represent the maximum sequence length. If $length(s) > n$, the first n elements of the sequence are taken. If $length(s) < n$, the sequence is padded with a null symbol <PAD>. Consequently, <PAD> is not included in V.

[1] Code is available at `https://github.com/shreydesai/cnn-sentiment-analysis`

Table 1: Statistics of datasets after preprocessing. l: Average sentence length. $|V|$: Vocabulary size. $|V_1|$,$|V_2|$,$|V_3|$: Counts of words that appear in the pre-trained Word2vec, GloVe, and Numberbatch embeddings, respectively. *Train*: Number of training samples. *Test*: Number of testing samples; 10-fold cross-validation (CV) is used if a testing set is not available.

	l	$\|V\|$	$\|V_1\|$	$\|V_2\|$	$\|V_3\|$	*Train*	*Test*
MR	115	18766	16448	17614	16914	9595	CV
SST-2	872	16238	14829	15849	15268	6920	1821
MPQA	18	6249	6083	6189	6128	9544	CV

The embedding layer functions as a lookup table that replaces each encoded item s_i with a vector $\mathbf{w}_i \in \mathbb{R}^d$ that serves as the corresponding d-dimensional word embedding. The concatenation of n word embedding vectors forms a sentence matrix $\mathbf{S} \in \mathbb{R}^{n \times d}$—the input to the CNN.

Next, we define a convolutional filter $\mathbf{W} \in \mathbb{R}^{h \times d}$ where h represents the *height* of the filter. The filter is not strided, padded, or dilated. Let $\mathbf{S}[i:j]$ represent a sub-matrix of $\mathbf{S}$ from rows i through j inclusive. The feature map $\mathbf{c} \in \mathbb{R}^{n-h+1}$ is created by applying the filter to each possible window of h words, using a non-linear function f and bias $b \in \mathbb{R}$:

$$\mathbf{c} = \sum_{i=1}^{n-h+1} f(\mathbf{S}[i:i+h] \cdot \mathbf{W} + b) \tag{1}$$

1-max pooling [4], defined as $\hat{c} = \max\{\mathbf{c}\}$, picks the maximum scalar from the feature map. This allows us to (a) propagate the maximum signal further into the network, (b) reduce the dimension of the input, and (c) introduce additional non-linearities.

We define a single n-gram block as consisting of a convolutional and pooling layer, as described above. Our network uses multiple n-gram blocks—the height of each filter is varied across the blocks, so the CNN learns multiple n-gram representations from the input. The outputs of these n-gram blocks are concatenated, then fed into a fully connected layer. For regularization we use a dropout layer that stochastically disables neurons to force them to learn independently useful features.

Softmax converts the fully connected layer output $\mathbf{x} \in \mathbb{R}^K$ into a distribution $\mathbf{y}$ over the K classes:

$$P(\mathbf{y}|\mathbf{x}) = \frac{\exp(\mathbf{x})}{\sum_{j=1}^{K} \mathbf{x}_j} \tag{2}$$

Finally, we use binary cross-entropy loss as the objective function since we have two classes to categorize sentiment—positive and negative.

3 Experiments

3.1 Datasets

We test our models on several datasets. The datasets are summarized as follows: (1) **MR:** Sentence polarity dataset (Pang and Lee) [5]. (2) **SST-2:** Stanford Sentiment Treebank. Adapted from SST-1, which contained fine-grained labels (very positive, positive, neutral, negative, very negative), but this version has binary labels only with neutral sentences removed (Socher et al.) [6]. (3) **MPQA:** Opinion polarity dataset from the Multi-Perspective Question Answering corpus (Wiebe et al.) [7]. Further dataset information is detailed in Table 1.

3.2 Hyperparameters

For all datasets, we define the maximum sequence length as the average sentence length, use word embedding dimensions of 300, and do not modify the original vocabulary. The CNN is constructed with three n-gram blocks of filter sizes 3, 4, and 5, respectively, with 128 channels each. ReLU [8], defined as $y = \max(0, x)$, is used as the non-linear activation function. The model is trained for 30

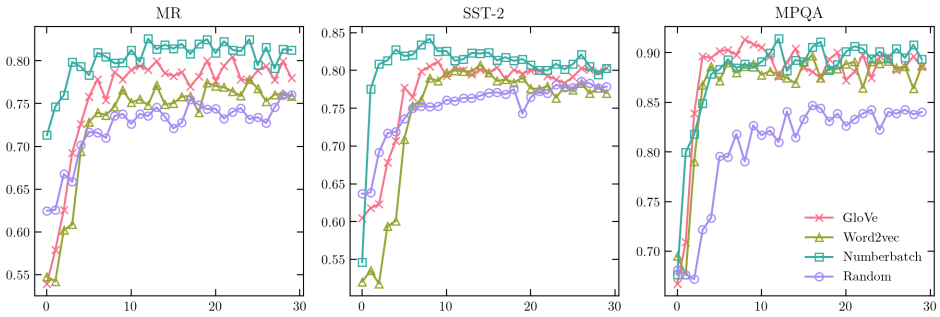

Figure 1: Validation accuracies of the models across training epochs.

epochs using the Adam [9] optimizer, learning rate of $1e{-}3$, no weight decay, dropout rate of 0.5, and a mini-batch size of 32. Early stopping is used to select checkpointed models with the highest accuracies on the validation sets. These hyperparameters were chosen through a minimal grid search on the MR dataset.

3.3 Model Variations

Using a standard text CNN as a base model, we define several variations by experimenting with different types of embedding layers:

- **CNN-random:** Embeddings are randomly initialized through a standard Gaussian.
- **CNN-word2vec:** Embeddings are initialized with Word2vec [10]. Word2vec describes a family of unsupervised neural language models that predict the linguistic context of words in a corpus. We use Mikolov et al.'s embeddings trained on 300 million words from Google News with a continuous bag-of-words model.
- **CNN-glove:** Embeddings are initialized with GloVe [11]. GloVe is an unsupervised algorithm that finds co-occurrence statistics from a global word-word co-occurrence matrix. We use Pennington et al.'s embeddings trained on the Wikipedia 2014 dump and Gigaword 5.
- **CNN-numberbatch:** Embeddings are initialized with Numberbatch [12]. Numberbatch is an ensemble method that combines data from ConceptNet 5.5, Word2vec, GloVe, and Open Subtitles 2016 [13]. We use Speer et al.'s English-only embeddings (version 16.04).

The embeddings used in each model have a dimensionality of 300. Further, for words that do not have entries in the pre-trained embeddings, their corresponding word vectors are randomly initialized from a standard Gaussian.

4 Discussion

4.1 Results

The validation accuracies are visualized in Figure 1. Models with random embeddings perform consistently worse than models with pre-trained embeddings. The MPQA dataset especially shows a large drop-off in performance when using random embeddings. The model accuracies are closer together for the MR and SST-2 datasets, but the pre-trained embedding models still perform better than the random embedding models. These conclusions are also consistent with the test accuracies detailed in Table 2.

The results cohere with the idea that models with random embeddings require more epochs to train, converge to suboptimal minima, and suffer from low accuracies. In contrast, models with pre-trained embeddings collectively perform well. In particular, they converge rapidly to a more optimal solution. For example, the Numberbatch models converge in about 10 epochs as opposed to the random models which converge in 20-30 epochs while maintaining strong accuracies.

Table 2: Test accuracies of the models.

	MR	SST-2	MPQA
CNN-random	0.739	0.777	0.817
CNN-word2vec	0.757	0.823	0.859
CNN-glove	0.778	0.824	**0.885**
CNN-numberbatch	**0.794**	**0.846**	0.876

Table 3: Top 4 neighboring words across SST-2 models.

	Random	Word2vec	GloVe	Numberbatch
good	*<PAD>*	*nice*	*great*	*better*
	dumbest	*excellent*	*excellent*	*well*
	fidgeted	*decent*	*decent*	*always*
	moral	*great*	*nice*	*excellent*
n't	*<PAD>*	*no*	*lisa*	*do*
	atrociously	*not*	*ca*	*did*
	introspective	*nothing*	*os*	*not*
	discovering	*neither*	*ritchie*	*know*
,	*<PAD>*	*of*	*disconnects*	*.*
	limply	*and*	*overtly*	*and*
	boisterous	*recent*	*abrasive*	*but*
	unfulfilled	*our*	*apex*	*though*

When comparing the pre-trained embedding models, the Numberbatch models seem to perform the best. Numberbatch models achieve the highest accuracies on the MR and SST-2 datasets, but get slightly edged out by the GloVe model on the MPQA dataset. These results are intuitive—the Numberbatch embeddings are created through an ensemble of other pre-trained embeddings, so they maximize the efficacy of each individual embedding.

4.2 Semantic Analysis

To further understand why certain embeddings perform better than others, we test the embeddings' abilities to predict context. We search for the top neighboring words for "good", "n't", and "," using Euclidean distance in the word embedding layers of SST-2 models; the results are shown in Table 3. It is immediately apparent that the random embedding model is unable to predict the context of words. The neighboring words in the random embedding model carry no relevance to the original words.

In contrast, the pre-trained embedding models generally perform better. For "good", the pre-trained embedding models produce similar neighboring words. However, the results are mixed for "n't" and ",". The GloVe model is unable to recognize the context in which "n't" and "," appear, while the Word2vec model gives slightly more convincing results. The Numberbatch model is unequivocally the best as it recognizes the context of contractions, determiners, and connectives.

The syntactic and semantic information stored in word embeddings is therefore strongly linked to performance. Intuitively speaking, words that cluster together in a vector space receive similar weights from a convolutional filter. This is because there is lower variability among the vectors of a cluster in comparison to the vectors of another cluster. This principle can be exploited in sentiment analysis tasks since positive and negative words often cluster together—these words can easily be used to predict subjectivity in a text.

Figure 2 empirically verifies this claim. Most notably, the filter learns to emphasize positive trigrams such as "visually stunning rumination". Further, it does not weight trigrams with function words—"the", "and", "on", "between"—as much since these words are not representative of a sentence's sentiment. Convolutions are location invariant operations by nature, so the filter is able to weight the trigrams correctly irrespective of where they appear in the sentence.

Figure 2: Heatmap of the weights a trigram convolutional filter assigns to the tokens of a sentence. Darker colors imply higher weights while lighter colors imply lower weights. The visualization is generated from a Numberbatch embedding model trained on SST-2.

5 Conclusion

Word embeddings have a significant impact on CNN performance. We show that models with random embeddings consistently perform worse than models with pre-trained embeddings. Pre-trained embeddings capture syntactic and semantic information about a corpus that deep learning models use to make classification decisions. CNNs exploit the fact that semantically similar words cluster together in low-dimensional spaces.

In particular, we show that the Numberbatch embedding models have faster training times and convergence than Word2vec and Glove embedding models. This is primarily attributed to the Numberbatch embeddings' ability to best predict the linguistic context of words. Further, the model fine tunes the embeddings during the training process, leading to better performance over time.

For future work, there are a couple of things we can do. First, we could try using different datasets—the ones used in our experiments were quite small. Second, we primarily focus on sentence-level sentiment analysis tasks, but we can also expand to other forms of text classification. Third, it would be interesting to see if our results differ for deeper convolutional architectures.

References

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015.

[2] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. 2015.

[3] Yoon Kim. Convolutional neural networks for sentence classification. 2014.

[4] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. *ICML*, 2010.

[5] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. *EMNLP*, 10, 2002.

[6] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. *EMNLP*, 2013.

[7] Janyce Wiebe, Theresa Wilson, and Claire Cardie. Annotating expressions of opinions and emotions in language. *Language re- sources and evaluation*, 2005.

[8] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. *ICML*, 30, 2013.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2014.

[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *NIPS*, 2013.

[11] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. *EMNLP*, 2014.

[12] Robert Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. *AAAI*, 2017.

[13] Pierre Lison and Jorg Tiedemann. Opensubtitles2016: Extracting large parallel corpora from movie and tv subtitles. *LREC*, 2016.