Convolutional Neural Networks for Sentence Classification

Yoon Kim
New York University
yhk255@nyu.edu

Abstract
We report on a series of experiments with convolutional neural networks (CNN) trained on top of pre-trained word vectors for sentence-level classification tasks. We show that a simple CNN with little hyperparameter tuning and static vectors achieves excellent results on multiple benchmarks. Learning task-specific vectors through fine-tuning offers further gains in performance. We additionally propose a simple modification to the architecture to allow for the use of both task-specific and static vectors. The CNN models discussed herein improve upon the state of the art on 4 out of 7 tasks, which include sentiment analysis and question classification.

1 Introduction
Deep learning models have achieved remarkable results in computer vision (Krizhevsky et al., 2012) and speech recognition (Graves et al., 2013) in recent years. Within natural language processing, much of the work with deep learning methods has involved learning word vector representations through neural language models (Bengio et al., 2003; Yih et al., 2011; Mikolov et al., 2013) and performing composition over the learned word vectors for classification (Collobert et al., 2011). Word vectors, wherein words are projected from a sparse, 1-of-\(V\) encoding (here \(V\) is the vocabulary size) onto a lower dimensional vector space via a hidden layer, are essentially feature extractors that encode semantic features of words in their dimensions. In such dense representations, semantically close words are likewise close—in euclidean or cosine distance—in the lower dimensional vector space.

Convolutional neural networks (CNN) utilize layers with convolving filters that are applied to local features (LeCun et al., 1998). Originally invented for computer vision, CNN models have subsequently been shown to be effective for NLP and have achieved excellent results in semantic parsing (Yih et al., 2014), search query retrieval (Shen et al., 2014), sentence modeling (Kalchbrenner et al., 2014), and other traditional NLP tasks (Collobert et al., 2011).

In the present work, we train a simple CNN with one layer of convolution on top of word vectors obtained from an unsupervised neural language model. These vectors were trained by Mikolov et al. (2013) on 100 billion words of Google News, and are publicly available. We initially keep the word vectors static and learn only the other parameters of the model. Despite little tuning of hyperparameters, this simple model achieves excellent results on multiple benchmarks, suggesting that the pre-trained vectors are ‘universal’ feature extractors that can be utilized for various classification tasks. Learning task-specific vectors through fine-tuning results in further improvements. We finally describe a simple modification to the architecture to allow for the use of both pre-trained and task-specific vectors by having multiple channels.

Our work is philosophically similar to Razavian et al. (2014) which showed that for image classification, feature extractors obtained from a pre-trained deep learning model perform well on a variety of tasks—including tasks that are very different from the original task for which the feature extractors were trained.

2 Model
The model architecture, shown in figure 1, is a slight variant of the CNN architecture of Collobert et al. (2011). Let \(x_i \in \mathbb{R}^k\) be the \(k\)-dimensional word vector corresponding to the \(i\)-th word in the sentence. A sentence of length \(n\) (padded where

1https://code.google.com/p/word2vec/
We employ language from computer vision where a color image has red, green, and blue channels. Each feature map is important feature—one with the highest value—for activity distribution over labels. The penultimate layer and are passed to a fully connected layer whose output is the probability distribution over labels.

In one of the model variants, we experiment with having two ‘channels’ of word vectors—one that is kept static throughout training and one that is fine-tuned via backpropagation (section 3.2). In the multichannel architecture, illustrated in figure 1, each filter is applied to both channels and the results are added to calculate $c_i$ in equation (2). The model is otherwise equivalent to the single channel architecture.

2.1 Regularization

For regularization we employ dropout on the penultimate layer with a constraint on $l_2$-norms of the weight vectors (Hinton et al., 2012). Dropout prevents co-adaptation of hidden units by randomly dropping out—i.e., setting to zero—a proportion $p$ of the hidden units during forward-backpropagation. That is, given the penultimate layer $z = [c_1, \ldots, c_m]$ (note that here we have $m$ filters), instead of using

$$y = w \cdot z + b$$

for output unit $y$ in forward propagation, dropout uses

$$y = w \cdot (z \circ r) + b,$$

where $\circ$ is the element-wise multiplication operator and $r \in \mathbb{R}^m$ is a ‘masking’ vector of Bernoulli random variables with probability $p$ of being 1. Gradients are backpropagated only through the unmasked units. At test time, the learned weight vectors are scaled by $p$ such that $\hat{w} = pw$, and $\hat{w}$ is used (without dropout) to score unseen sentences. We additionally constrain $l_2$-norms of the weight vectors by rescaling $w$ to have $||w||_2 = s$ whenever $||\hat{w}||_2 > s$ after a gradient descent step.

Figure 1: Model architecture with two channels for an example sentence.
### 3 Datasets and Experimental Setup

We test our model on various benchmarks. Summary statistics of the datasets are in table 1.

| Data   | c | l | N     | |V| | |Vpre| | Test |
|--------|---|---|-------|---|---|---|---|---|-----|
| MR     | 2 | 20| 10662 | 18765 | 16448 | CV |
| SST-1  | 5 | 18| 11855 | 17836 | 16262 | 2210 |
| SST-2  | 2 | 19| 9613  | 16185 | 14838 | 1821 |
| Subj   | 2 | 23| 10000 | 21323 | 17913 | CV |
| TREC   | 6 | 10| 5952  | 9592  | 9125  | 500 |
| CR     | 2 | 19| 3775  | 5340  | 5046  | CV |
| MPQA   | 2 | 3 | 10606 | 6246  | 6083  | CV |

Table 1: Summary statistics for the datasets after tokenization. c: Number of target classes. l: Average sentence length. N: Dataset size. |V|: Vocabulary size. |Vpre|: Number of words present in the set of pre-trained word vectors. Test: Test set size (CV means there was no standard train/test split and thus 10-fold CV was used).

#### 3.1 Hyperparameters and Training

For all datasets we use: rectified linear units, filter windows \((h)\) of 3, 4, 5 with 100 feature maps each, dropout rate \((p)\) of 0.5, \(l_2\) constraint \((s)\) of 3, and mini-batch size of 50. These values were chosen via a grid search on the SST-2 dev set.

We do not otherwise perform any dataset-specific tuning other than early stopping on dev sets. For datasets without a standard dev set we randomly select 10% of the training data as the dev set. Training is done through stochastic gradient descent over shuffled mini-batches with the Adadelta update rule (Zeiler, 2012).

#### 3.2 Pre-trained Word Vectors

Initializing word vectors with those obtained from an unsupervised neural language model is a popular method to improve performance in the absence of a large supervised training set (Collobert et al., 2011; Socher et al., 2011; Iyyer et al., 2014). We use the publicly available word2vec vectors that were trained on 100 billion words from Google News. The vectors have dimensionality of 300 and were trained using the continuous bag-of-words architecture (Mikolov et al., 2013). Words not present in the set of pre-trained words are initialized randomly.

#### 3.3 Model Variations

We experiment with several variants of the model.

- **CNN-rand**: Our baseline model where all words are randomly initialized and then modified during training.
- **CNN-static**: A model with pre-trained vectors from word2vec. All words—including the unknown ones that are randomly initialized—are kept static and only the other parameters of the model are learned.
- **CNN-non-static**: Same as above but the pre-trained vectors are fine-tuned for each task.
- **CNN-multichannel**: A model with two sets of word vectors. Each set of vectors is treated as a ‘channel’ and each filter is applied
- **MPQA**: Opinion polarity detection subtask of the MPQA dataset (Wiebe et al., 2005).7

---

1. https://www.cs.cornell.edu/people/pabo/movie-review-data/  
2. http://nlp.stanford.edu/sentiment/ Data is actually provided at the phrase-level and hence we train the model on both phrases and sentences but only score on sentences at test time, as in Socher et al. (2013), Kalchbrenner et al. (2014), and Le and Mikolov (2014). Thus the training set is an order of magnitude larger than listed in table 1.  
3. http://cogcomp.cs.illinois.edu/Data/QA/QC/  
4. http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html  
5. http://www.cs.pitt.edu/mpqa/
Table 2: Results of our CNN models against other methods. RAE: Recursive Autoencoders with pre-trained word vectors from Wikipedia (Socher et al., 2011). MV-RNN: Matrix-Vector Recursive Neural Network with parse trees (Socher et al., 2012). RNTN: Recursive Neural Tensor Network with tensor-based feature function and parse trees (Socher et al., 2013). DCNN: Dynamic Convolutional Neural Network with k-max pooling (Kalchbrenner et al., 2014). Paragraph-Vect: Recursive Autoencoders with pre-trained category grammar operators (Hermann and Blunsom, 2013). CCAE: Combinatoric Category Autoencoders with combinatorial category grammar operators (Hermann and Blunsom, 2013). Sent-Parser: Sentiment analysis-specific parser (Dong et al., 2014). NBSVM: Naive Bayes SVM and Multinomial Naive Bayes with uni-bigrams from Wang and Manning (2012). SVMs: SVM with uni-bi-trigrams, wh word, head word, POS, parser, hypernyms, and 60 hand-coded rules as features from Silva et al. (2011).

| Model              | MR   | SST-1 | SST-2 | Subj | TREC | CR   | MPQA |
|--------------------|------|-------|-------|------|------|------|------|
| CNN-rand           | 76.1 | 45.0  | 82.7  | 89.6 | 91.2 | 79.8 | 83.4 |
| CNN-static         | 81.0 | 45.5  | 86.8  | 93.0 | 92.8 | 84.7 | 89.6 |
| CNN-non-static     | 81.5 | 48.0  | 87.2  | 93.4 | 93.6 | 84.3 | 89.5 |
| CNN-multichannel   | 81.1 | 47.4  | 88.1  | 93.2 | 92.2 | 85.0 | 89.4 |
| RAEC (Socher et al., 2011) | 77.7 | 43.2  | 82.4  | -    | -    | -    | 86.4 |
| MV-RNN (Socher et al., 2012) | 79.0 | 44.4  | 82.9  | -    | -    | -    | -    |
| RNTN (Socher et al., 2013) | -    | 45.7  | 85.4  | -    | -    | -    | -    |
| DCNN (Kalchbrenner et al., 2014) | -    | 48.5  | 86.8  | -    | 93.0 | -    | -    |
| Paragraph-Vect (Le and Mikolov, 2014) | -    | 48.7  | 87.8  | -    | -    | -    | -    |
| CCAE (Hermann and Blunsom, 2013) | 77.8 | -     | -     | -    | 81.8 | 89.6 | 86.3 |
| Sent-Parser (Dong et al., 2014) | 79.5 | -     | -     | -    | 93.6 | -    | -    |
| NBSVM (Wang and Manning, 2012) | 79.4 | -     | 93.2  | -    | 80.0 | 86.3 | 86.3 |
| MNB (Wang and Manning, 2012) | 79.0 | -     | -     | 93.4 | -    | 82.1 | 86.1 |
| G-Dropout (Wang and Manning, 2013) | 79.1 | -     | -     | 93.6 | -    | 81.9 | 86.3 |
| F-Dropout (Wang and Manning, 2013) | 79.4 | -     | -     | 93.6 | -    | 81.4 | 86.1 |
| Tree-CRF (Nakagawa et al., 2010) | 77.3 | -     | -     | -    | 82.7 | -    | -    |
| CRF-PR (Yang and Cardie, 2014) | -    | -     | -     | -    | 95.0 | -    | -    |

Table 2: Results of our CNN models against other methods. RAEC: Recursive Autoencoders with pre-trained word vectors from Wikipedia (Socher et al., 2011). MV-RNN: Matrix-Vector Recursive Neural Network with parse trees (Socher et al., 2012). RNTN: Recursive Neural Tensor Network with tensor-based feature function and parse trees (Socher et al., 2013). DCNN: Dynamic Convolutional Neural Network with k-max pooling (Kalchbrenner et al., 2014). Paragraph-Vect: Recursive Autoencoders with pre-trained category grammar operators (Hermann and Blunsom, 2013). CCAE: Combinatoric Category Autoencoders with combinatorial category grammar operators (Hermann and Blunsom, 2013). Sent-Parser: Sentiment analysis-specific parser (Dong et al., 2014). NBSVM, MNB: Naive Bayes SVM and Multinomial Naive Bayes with uni-bigrams from Wang and Manning (2012). G-Dropout, F-Dropout: Gaussian Dropout and Fast Dropout from Wang and Manning (2013). Tree-CRF: Dependency tree with Conditional Random Fields (Nakagawa et al., 2010). CRF-PR: Conditional Random Fields with Posterior Regularization (Yang and Cardie, 2014). SVMs: SVM with uni-bi-trigrams, wh word, head word, POS, parser, hypernyms, and 60 hand-coded rules as features from Silva et al. (2011).

to both channels, but gradients are backpropagated only through one of the channels. Hence the model is able to fine-tune one set of vectors while keeping the other static. Both channels are initialized with word2vec.

In order to disentangle the effect of the above variations versus other random factors, we eliminate other sources of randomness—CV-fold assignment, initialization of unknown word vectors, initialization of CNN parameters—by keeping them uniform within each dataset.

4 Results and Discussion

Results of our models against other methods are listed in table 2. Our baseline model with all randomly initialized words (CNN-rand) does not perform well on its own. While we had expected performance gains through the use of pre-trained vectors, we were surprised at the magnitude of the gains. Even a simple model with static vectors (CNN-static) performs remarkably well, giving competitive results against the more sophisticated deep learning models that utilize complex pooling schemes (Kalchbrenner et al., 2014) or require parse trees to be computed beforehand (Socher et al., 2013). These results suggest that the pre-trained vectors are good, ‘universal’ feature extractors and can be utilized across datasets. Fine-tuning the pre-trained vectors for each task gives still further improvements (CNN-non-static).

4.1 Multichannel vs. Single Channel Models

We had initially hoped that the multichannel architecture would prevent overfitting (by ensuring that the learned vectors do not deviate too far from the original values) and thus work better than the single channel model, especially on smaller datasets. The results, however, are mixed, and further work on regularizing the fine-tuning process is warranted. For instance, instead of using an additional channel for the non-static portion, one could maintain a single channel but employ extra dimensions that are allowed to be modified during training.
As is the case with the single channel non-static model, the multichannel model is able to fine-tune the non-static channel to make it more specific to the task-at-hand. For example, good is most similar to bad in word2vec, presumably because they are (almost) syntactically equivalent. But for vectors in the non-static channel that were fine-tuned on the SST-2 dataset, this is no longer the case (table 3). Similarly, good is arguably closer to nice than it is to great for expressing sentiment, and this is indeed reflected in the learned vectors.

For (randomly initialized) tokens not in the set of pre-trained vectors, fine-tuning allows them to learn more meaningful representations: the network learns that exclamation marks are associated with effusive expressions and that commas are conjunctive (table 3).

4.3 Further Observations

We report on some further experiments and observations:

- Kalchbrenner et al. (2014) report much worse results with a CNN that has essentially the same architecture as our single channel model. For example, their Max-TDNN (Time Delay Neural Network) with randomly initialized words obtains 37.4% on the SST-1 dataset, compared to 45.0% for our model. We attribute such discrepancy to our CNN having much more capacity (multiple filter widths and feature maps).

  - Dropout proved to be such a good regularizer that it was fine to use a larger than necessary network and simply let dropout regularize it. Dropout consistently added 2%–4% relative performance.

  - When randomly initializing words not in word2vec, we obtained slight improvements by sampling each dimension from $U[-a, a]$ where $a$ was chosen such that the randomly initialized vectors have the same variance as the pre-trained ones. It would be interesting to see if employing more sophisticated methods to mirror the distribution of pre-trained vectors in the initialization process gives further improvements.

  - We briefly experimented with another set of publicly available word vectors trained by Collobert et al. (2011) on Wikipedia and found that word2vec gave far superior performance. It is not clear whether this is due to Mikolov et al. (2013)’s architecture or the 100 billion word Google News dataset.

  - Adadelta (Zeiler, 2012) gave similar results to Adagrad (Duchi et al., 2011) but required fewer epochs.

5 Conclusion

In the present work we have described a series of experiments with convolutional neural networks built on top of word2vec. Despite little tuning of hyperparameters, a simple CNN with one layer of convolution performs remarkably well. Our results add to the well-established evidence that unsupervised pre-training of word vectors is an important ingredient in deep learning for NLP.

Acknowledgments

We would like to thank Yann LeCun and the anonymous reviewers for their helpful feedback and suggestions.

8http://ronan.collobert.com/senna/
