Learning Representations for Multi-Dimensional Sentiment Analysis

Andrew Maas
Department of Computer Science
Stanford University
amaas@cs.stanford.edu

Peter Pham
Department of Computer Science
Stanford University
ptpham@cs.stanford.edu

Ying Wang
Department of Computer Science
Stanford University
ygw@stanford.edu

1 Introduction

Our work explores sentiment analysis using data from The Experience Project[1] (EP). On this website, users publicly share short stories about their experiences in life. Other users then read the stories and respond using a fixed set of reactions. There are five possible reactions: "You rock," "Teehee," "I understand," "Sorry, hugs," and "Wow, just wow." Here is an example document from the EP:

> I went and gave flowers to a retirement home! I got my income tax return yesterday and went and bought 5 dozen roses to give to each of the five floors of a retirement home in this city. It felt good to see some of the older ladies smile and one of the nurses gave me a hug. This is what I call a good day!

This document received 22 "You rock" reactions, 1 "Teehee" reaction, and 1 "Wow, just wow." reaction. We can treat these reactions as a normalized distribution over reactions, $[0.916, 0.041, 0, 0, 0.041]$. Treating the reaction data as a normalized distribution allows us to ignore the differences in reaction counts across documents and instead focus only on how reactions are distributed within documents.

Our project focuses on predicting the reaction distributions given only the text. This problem is substantially more challenging than the binary polarity classification tasks which have dominated the field of sentiment analysis to date [9, 4]. Although some previous work [11] considers continuous notions of sentiment for ranking and regression tasks, it is still restricted to a single axis of good vs. bad polarity. Work in the linguistics community paints a far more subtle picture of sentiment, one with multiple relevant dimensions and [14, 1, 2, 10]. Thus we believe sentiment analysis tasks which consider multi-dimensional notions of emotion present an exciting new set of challenges for the field. In this paper, we evaluate several baseline methods for the task and introduce models to learn feature representations which better capture the sentiment expressed in text.

In our previous work we developed a model which learns vector representations of words which capture both the semantic and sentimental information conveyed by that word [7]. This work applies related models to the more complex task of EP sentiment analysis. Because the concept is more complex we hypothesized that representing each word as a single vector would not be sufficient for this dataset. This lead us to develop a convolutional neural network to analyze the documents. This model learns convolution filters for n-grams which can potentially capture larger chunks of phrases as compared to the word vectors used in our previous work. In our control experiments we also

[1] http://www.experienceproject.com

develop a word vector model which is automatically tuned to better capture sentiment information in the dataset.

2 Softmax

To predict the reaction distribution y given some input features x we use a softmax model. This model parametrizes a categorical distribution over distinct classes. The softmax is also convenient in that its differentiability works well with the learning models we consider. Because the softmax is a parametric log-linear model of the distribution we need to define sufficiently expressive features x for the model to correctly capture the target distribution.

2.1 Model

We are given a set of examples $X \in \Re^{n \times m}$ where each column $x^{(i)}$ is an n-dimensional example. These examples have an associated label matrix $Y \in \Re^{k \times m}$ where a label y is a k-dimensional categorical distribution.

We want a model of the form $\hat{y} = f(x)$ to predict the categorical distribution associated with an example x. We use a softmax with parameters $W \in \Re^{k \times n}$ and $b \in \Re^k$. To obtain the predicted probability for the j-th dimension of the label vector for example i, we use

$$\hat{y}_j^{(i)} = \frac{\exp(w^{(j)} x^{(i)} + b_j)}{\sum_{j'} \exp(w^{(j')} x^{(i)} + b_{j'})}, \tag{1}$$

where $w^{(j)}$ is the j-th row of W. In practice we constrain $w^{(k)} = \vec{0}$ and $b_k = 0$ to keep the weights W from growing too large. This yields the more familiar formulation of softmax,

$$\hat{y}_j^{(i)} = \frac{1}{1 + \sum_{j=1}^{k-1} \exp(w^{(j)} x^{(i)} + b_j)}, \tag{2}$$

2.2 Learning

To learn the parameters W and b from labeled data, we minimize the Kullback Leibler (KL) divergence between the predicted and observed distributions for our training set. The KL-divergence between two distributions P and Q, defined as $KL(P||Q)$, is a measure of the difference between the distributions. This objective is defined over our dataset as

$$\ell(W, b) = \sum_{i=1}^{m} \sum_{j=1}^{k} y_j^{(i)} \log \left(\frac{y_j^{(i)}}{\hat{y}_j^{(i)}} \right), \tag{3}$$

where $y^{(i)}$ and $\hat{y}^{(i)}$ denote the actual and predicted labels of the i-th training example.

In practice it's often useful to add a regularization term or prior distribution on the model parameters. We use a Gaussian prior on W with mean 0 and variance λ. Minimizing W under this Gaussian prior is equivalent to minimizing $\lambda ||W||_F^2$, the squared Frobenious norm. Thus, our final objective seeks to minimize w.r.t. W and b,

$$\ell(W, b) = \lambda ||W||_F^2 + \sum_{i=1}^{m} \sum_{j=1}^{k} y_j^{(i)} \log \left(\frac{y_j^{(i)}}{\hat{y}_j^{(i)}} \right). \tag{4}$$

We minimize this loss using a second-order gradient method (L-BFGS) contained in the minFunc package. The partial derivatives are

$$\frac{\partial}{\partial W_{j,l}} \ell(W, b) = 2\lambda W_{j,l} + \sum_{i=1}^{m} \hat{y}_j^{(i)} x_l^{(i)} - y_j^{(i)} x_l^{(i)} \tag{5}$$

$$\frac{\partial}{\partial b_j} \ell(W, b) = \sum_{i=1}^{m} \hat{y}_j^{(i)} - y_j^{(i)} \tag{6}$$

3 Experiments

For each of the experiments, we trained a softmax model to minimize the regularized KL-divergence objective defined in (4). Performance was measured on a test set in terms of the KL-only component of the optimization objective, divided by the number of documents. This gives an average KL loss over the documents.

Because KL loss is not so easy to interpret, we also report classification accuracy as a more intuitive measurement. We define classification as the prediction of the most popular label for a particular document — which of the five labels received the most user votes. To measure classification, we compared the label with the highest probability in the predicted distribution with the most popular label in the actual distribution. The accuracy is the percentage of documents where the predicted and actual max labels match. If a document's actual vote distribution has more than one class tied for first place, we take the lowest-index class to be the most popular.

3.1 Bag of Words

In our bag-of-words experiments, we represent each document as a binary-valued vector whose i-th entry indicates whether the i-th dictionary word appears in the document.

We tested our model with 3 dictionaries:

- All unigrams appearing in our entire dataset (vocabulary size 45,392)
- All bigrams appearing in our entire dataset (vocabulary size 563,588)
- All unigrams and bigrams appear in our entire dataset (vocabulary size 608,980)

Traditional stop word removal was not used, because many frequently-appearing words, like "love" or "hate" are indicative of sentiment.

In each of our experiments, we discarded documents containing no dictionary words. This resulted in a smaller dataset for the bigram-only dictionary experiment, since there were some one-token documents that effectively contained no bigrams. For each experiment, we trained on 20,000 documents, which left 6659 test documents for the unigram and the combined unigram-bigram cases and 6644 for the bigram-only case. The regularization hyperparameter was set by cross-validation on the training set.

3.2 Latent Semantic Analysis

Vector space models (VSMs) model words using a real-valued multi-dimensional vector, where similarities among vectors can capture semantic or syntactic similarities among words [13]. Latent Semantic Analysis (LSA), perhaps the best known VSM, explicitly learns semantic word vectors by applying singular value decomposition (SVD) to factor a term–document co-occurrence matrix. It is typical to weight and normalize the matrix values prior to SVD. To obtain a k-dimensional representation for a given word, only the entries corresponding to the k largest singular values are taken from the word's basis in the factored matrix. Such matrix factorization-based approaches are often successful in practice, especially in document-level retrieval tasks, or when used as features for named entity recognition systems.

Using term frequency (tf) and inverse document frequency (idf) weighting to transform the values in a VSM often increases the performance of retrieval and categorization systems. Delta idf weighting [8] is a supervised variant of idf weighting in which the idf calculation is done for each document class and then one value is subtracted from the other. Martineau and Finin [8] present evidence that this weighting helps with binary sentiment classification. However this weighting scheme is not applicable to our present work because we have a multi-dimensional notion of sentiment as opposed to the previously studied binary tasks. We performed preliminary experiments using LSA on tf-idf weighted co-occurrence matrices and found lower performance when compared with a technique which did not use idf weighting. For this reason, our ultimate choice for LSA uses the SVD of a term count co-occurrence matrix.

Recent work in language modeling and other syntactic tasks introduces alternative techniques for VSM induction [5, 3]. We obtained word vectors from these models already trained by Turian et.

al. [12]. Despite a large set of word vectors, the coverage for our dataset was poor. In particular, we found that only about half of our vocabulary was represented by the Turian vocabulary. Upon examination we found that our dataset contains many misspellings and informal phrases which are not present in the news article corpus used by [12] to train word vectors. Here is a random sampling of the words in our corpus that do not appear in the Turian vocabulary: 'aaaahhhhhhh', 'nincompoops', 'onme', 'agggggggga', 'freckly', "i'am", 'new-graphics', 'overwith', 'missleading', 'opurtunity'. For this reason our preliminary experiments exhibited poor performance for these word vectors and we did not pursue them further. Training such models on a new dataset requires days or weeks and was thus not feasible for our project.

3.3 Fine-tuned LSA

Let $\phi^{(i)}$ denote the word vector corresponding to a word $w^{(i)}$, assuming there are V words in total. In LSA fine tuning, we construct pairs of $(\phi^{(i)}, \bar{y}^{(i)})$ where $\bar{y}^{(i)}$ represents the average of the labels distributions for all documents in which word $w^{(i)}$ appears. That is,

$$\bar{y}_j = \frac{1}{|\{d \in D | w^{(i)} \in d\}|} \sum_{\{d \in D | w^{(i)} \in d\}} y_j \tag{7}$$

Let our predicted distribution for the i-th word vector be

$$\hat{y}_j^{(i)} = \frac{\exp(\phi^{(i)} w^{(j)} + b_j)}{\sum_{j'} \exp(\phi^{(i)} w^{(j')} + b_{j'})} \tag{8}$$

Our objective is then to minimize the regularized KL divergence between $\bar{y}$ and $\hat{y}$ over all words:

$$\ell(W, b) = \lambda ||W||_F^2 + \sum_{i=1}^{V} \sum_{j=1}^{k} \bar{y}_j^{(i)} \log \left(\frac{\bar{y}_j^{(i)}}{\hat{y}_j^{(i)}} \right). \tag{9}$$

We perform alternate optimization steps on the partial derivatives:

$$\frac{\partial}{\partial W_{j,l}} \ell(W, b) = 2\lambda W_{j,l} + \sum_{i=1}^{V} \hat{y}_j^{(i)} \phi_l^{(i)} - y_j^{(i)} \phi_l^{(i)} \tag{10}$$

$$\frac{\partial}{\partial b_j} \ell(W, b) = \sum_{i=1}^{V} \hat{y}_j^{(i)} - y_j^{(i)} \tag{11}$$

$$\frac{\partial}{\partial \phi_j^{(i)}} \ell(W, b) = \sum_{i=1}^{V} \sum_{j=1}^{k} \bar{y}_j^{(i)} \left[-W_{j,l} + \frac{\sum_{j'} W_{j',l} \exp(\phi^{(i)} w^{(j')} + b_{j'})}{\sum_{j'} \exp(\phi^{(i)} w^{(j')} + b_{j'})} \right] \tag{12}$$

4 Results

4.1 Bag of Words

The results for bag-of-words are shown in table 4.1.

In these results, the unigram dictionary is the most successful in minimizing the test KL loss and also produces the highest test accuracy. A dictionary of unigrams plus bigrams performs second-best in both categories, while a bigram-only dictionary performs worst. On the test set, the performances of a unigram-only dictionary and a unigram plus bigram dictionary are very close, though the unigram plus bigram dictionary performs much better on the training set.

We suspect that using bigrams in the dictionary overfits the model to the training set. This can be seen in the training performances of the models: the bigram-only and bigram plus unigram dictionaries both show better performance on the training set than the unigram dictionary, but the unigram dictionary performs the best on the test set.

Table 1: Performance of bag-of-word models

	KL Divergence		Accuracy	
Dictionary	**Train**	**Test**	**Train**	**Test**
Baseline	3.7227	11.1809	42.34	42.92
Unigrams	0.8191	**1.0030**	58.37	**48.99**
Bigrams	0.6520	1.0363	69.07	45.85
Unigrams + Bigrams	0.5698	1.0162	73.94	48.96
LSA100	1.0306	1.0302	65.82	45.00
LSA100 Fine-Tuned	0.6136	1.5025	65.81	46.21

This overfitting is most likely due to the more specific nature of bigrams. For any training and testing split, it is likely that the two subsets will share more unigrams than bigrams. Therefore, we expect that many bigrams in the training set will not be in the test set, and vice versa. This leads to the model placing emphasis on bigram features that do not show up at test time, while ignoring bigrams that are in the test set but not in the training set.

We see the worst test performance on the bigram-only dictionary because it potentially creates the smallest overlap between train and test features. However, when we include both unigrams and bigrams in the dictionary, the performance improves to a level comparable to the unigram-only dictionary.

5 Convolutional Neural Network

Because the EP contains complex, multi-dimensional notions of sentiment we explored a class of models which can learn representations for multi-word phrases rather than just single tokens. To this end, we develop convolutional neural networks for EP documents. Related models have recent success as a strong general approach to word sense disambiguation, named entity recognition, and part of speech tagging [5]. These tasks are of a more syntactic nature where the relevant scope is a single sentence or smaller. In our present work, we develop such models to handle entire EP documents, which are substantially longer.

Our convolutional model consists of several layers of representation. As a preprocessing step, we transform each word into it a dense vector representation. The word vectors used can come from any vector space model. The first layer of the model is the convolutional layer. A set of learned filters look at each n-gram in the sentence and compute a transformed representation for each n-gram. The transformed representations at each position are then averaged, resolving the issue of documents containing an unknown number of words. This averaged representation forms the input to a softmax layer and the KL divergence previously discussed is optimized. During learning, the KL divergence is back-propagated through the network to train the convolution filters as well as the softmax parameters.

5.1 Model

Given a document represented as one-on vectors $(w_1, w_2, ...w_N)$ we transform the words into the corresponding sequence of word vectors $(\phi_1, \phi_2, ...\phi_N)$ using some fixed mapping of words to vectors derived from a VSM. The convolution layer acts upon this sequence of word vectors. For simplicity of exposition we discuss convolution filters which compute representations for bigrams, however the method extends to arbitrary n-grams, and heterogeneous mixtures of variable order n-grams. We represent the document in matrix form as a collection of bigrams,

$$V = \begin{bmatrix} \phi_1 \oplus \phi_2 \\ \phi_2 \oplus \phi_3 \\ \vdots \\ \phi_{N-1} \oplus \phi_N \end{bmatrix}, \tag{13}$$

where V is a matrix of dimension 2β x P. P is the number of bigrams present in the document. The symbol $\oplus$ denotes concatenating two vectors, and here we treat the word vectors ϕ as row vectors.

Another free parameter of the architecture is the *step size* of the convolutional filters, in the above and in our experiments we use a step size of 1. However, we could just as easily have the second row in v be $[\phi_3 \oplus \phi_4]$ corresponding to a step size of 2 and causing each word to enter into only a single row of v when using bigram filters.

The convolutional filters are represented by a matrix W with dimensions $K \times 2\beta$. The number of convolutional filters K is a free parameter of the model. To obtain the hidden representations H from W and V we use,

$$H = \sigma(WV + B). \tag{14}$$

The function $\sigma(x)$ is the point-wise sigmoid function, $\sigma(x) = \frac{1}{1+e^{-x}}$. The matrix of hidden unit representations for the document has dimension $K \times P$ and the entry $H_{i,j}$ is the i^{th} hidden unit's response to the j^{th} bigram in the document. We aso have a vector of K biases for the hidden units which is replicated to form the rows of B.

Because the number of rows P in H is dependent upon the document length, it is difficult to define the next layer of the model directly in terms of H. To avoid this variable length problem, we introduce an averaging layer where we obtain a K-dimensional vector h by averaging across the rows of H.

The convolution and averaging layers yield a fixed length vector h for the entire document. We use h as input to a softmax layer as described in 2. The final loss function is the same KL divergence as (4) but with the regularization weight λ set to 0.

5.2 Learning

Training the convolutional model amounts to finding the optimal convolution filters W and classifier parameters ψ, as well as their corresponding biases. This is a high-dimensional non-convex problem so in general we can't find the globally optimal solutions. Instead we ignore the non-convexity and find a local optimum, which can work well in practice for neural networks.

We again use L-BFGS optimization as implemented by the minFunc package in matlab. The gradients of our model are continuous and fairly easy to derive, which is one of the reason we used an average of the convolution layer representations as opposed to a max or other non-differentiable function. Using several implementation tricks, we can forward-propagate the entire EP dataset through the model with 100 trigram convolution filters in less than 30 seconds.

5.3 Results

We evaluated the convolutional model using the same KL and accuracy metrics introduced previously. Due to minor differences in preprocessing, the convolutional model was trained on 17,588 and evaluated using 4,277 examples.

Table 2 shows the results on both the training and test sets. We found a model with 100 bigram filters to perform best in our experiments, though a model using 50 trigram filters performed almost as well. Overall the performance of the convolutional class of models was lower than we hoped given it's a more expressive class of models. We hypothesized this relatively poor performance was due to the model being stuck in a local minimum of its objective function.

We also evaluated built convolutional models using word vectors already tuned via our LSA fine tuning procedure. Surprisingly these models performed quite poorly even compared to the baseline bag of words models. We are unsure whether this performance is due to a bug in our code or actually reflects that learning convolutional filters for already tuned vectors being a poor model.

6 Discussion

Our project explored methods to learn features for a complex multi-dimensional sentiment analysis task. We showed that adaptive representations hold some promise in this area, especially those which can capture phrasal information beyond the unigram level.

Table 2: Performance of convolutional models. We evaluate several instantiations of the model which used different filter sizes(n-gram lengths), and number of filters (K). The models were evaluated using KL divergence and accuracy metrics.

			KL Divergence		Accuracy	
Word Vectors	**Filter Size**	**# Filters**	**Train**	**Test**	**Train**	**Test**
LSA50	2	50	0.964	**1.103**	46.51	49.75
LSA50	2	100	0.982	1.104	45.22	50.11
LSA50	3	50	0.961	1.109	46.38	**50.20**

We plan to continue exploring the convolutional class of models for this task as we see several avenues for improving performance. First, we can improve the locally optimal solution by pre-training the convolutional filters to capture the natural statistics of n-gram occurrence. This regime is known as self-taught learning as has been shown to boost performance in other domains such as image and audio classification [6]. We also want to further explore the possible network architectures for convolutional models. Primarily we plan to focus on replacing the averaging layer with something to better preserve document structure. For example we could use multiple layers of convolution before eventually averaging to obtain a fixed length vector.

The EP dataset provides a rich, challenging task for sentiment analysis. Document polarity classification is already approaching 90% accuracy on many datasets, but we found classification and reaction distribution prediction extremely challenging tasks in our work. Linguists and computer scientists have a challenging task in developing sentiment analysis techniques capable of capturing such multi-dimensional expressions of emotion commonly found in text.

Acknowledgments

We thank Andrew Ng and Chris Potts for helpful conversations. We also thank Chris for providing the EP dataset and a nice string tokenizer.

References

[1] C. O. Alm, D. Roth, and R. Sproat. Emotions from text: machine learning for text-based emotion prediction. In *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, 2005.

[2] A. Andreevskaia and S. Bergler. Mining WordNet for a fuzzy sentiment: Sentiment tag extraction from WordNet glosses. In *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL)*, 2006.

[3] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3(6):1137–1155, August 2003.

[4] J. Blitzer, M. Dredze, and F. Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Proceedings of the ACL*, 2007.

[5] R. Collobert and J. Weston. A unified architecture for natural language processing. *Proceedings of the 25th ICML*, 2008.

[6] Honglak Lee, R. Raina, A. Teichman, and A.Y. Ng. Exponential family sparse coding with applications to self-taught learning. In *Proceedings of the 21st international jont conference on Artifical intelligence*, pages 1113–1119. Morgan Kaufmann Publishers Inc., 2009.

[7] A. L. Maas, R. E. Daly, P.T. Pham, D. Huang, A.Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of the ACL (To appear)*, 2011.

[8] J. Martineau and T. Finin. Delta tfidf: An improved feature space for sentiment analysis. In *Proceedings of the third AAAI internatonal conference on weblogs and social media*, 2009.

[9] B. Pang and L. Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the ACL*, volume 2004, 2004.

[10] B. Pang and L. Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 115–124, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[11] Benjamin Snyder and Regina Barzilay. Multiple aspect ranking using the good grief algorithm. In *Proceedings of NAACL HLT*, pages 300–307, 2007.

[12] J. Turian, L. Ratinov, and Y. Bengio. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the ACL*, 2010.

[13] P. D. Turney and P. Pantel. From Frequency to Meaning : Vector Space Models of Semantics. *Journal of Artificial Intelligence Research*, 37:141–188, 2010.

[14] T. Wilson, J. Wiebe, and R. Hwa. Just how mad are you? Finding strong and weak opinion clauses. In *Proceedings of AAAI*, pages 761–769, 2004. Extended version in *Computational Intelligence* 22(2, Special Issue on Sentiment Analysis):73–99, 2006.