Ensembling Insights for Baseline Text Models

Henry Ehrenberg
Stanford University
henryre@cs.stanford.edu

Dan Iter
Stanford University
daniter@cs.stanford.edu

Abstract

Deep neural networks are in vogue for text classification. The lack of interpretability and computational cost associated with deep architectures has led to renewed interest in effective baseline models. In this paper, we review several popular baseline models which strike a balance between traditional and neural approaches, and propose improvements by combining their key contributions. In particular, we study gradient-tuned word embeddings, modeling n-grams, and generative sentence representation methods. We evaluate our methods by comparing end performance and training time on sentiment analysis and topic classification tasks. By combining techniques of popular baseline models into a single shallow architecture, we outperformed the individual models on all tasks, were competitive with traditional and deep approaches, and maintained fast training times.

1 Introduction

Baseline methods are an essential but often neglected component of the natural language processing field. As deep neural network-based methods for text classification continue to grow in popularity, it becomes increasingly more important to compare their performance to competitive, but simpler models. Though deep neural networks alleviate much of the burden of feature engineering and have more representational power than traditional models, baseline methods can be competitive on a wide range of tasks, such as text classification. Therefore, it is critical to include baseline comparisons to understand performance and efficiency trade-offs when introducing new models. High performing baseline methods are also frequently deployed in production themselves due to their low cost to train and serve, their interpretability, and the relative ease with which they can be refined.

Simple linear and maximum entropy models have always and will continue to hold an import place as baseline models. However, recent research interest in baseline models has centered around shallow neural architectures, which strike a balance between traditional and deep learning approaches to natural language processing. Shallow neural networks and embedding-based models have several advantages over traditional methods, since they are able to share parameters across features and classes. Our goal is identify the strengths and weaknesses of popular shallow baseline methods for text classification, and propose approaches to synthesize their key insights.

The remainder of this paper is organized as follows. In Section 2, we review several baseline approaches for text classification with a focus on shallow neural networks. We identify the key insights of these methods, and design a new shallow architecture that combines them in Section 3. Section 4 presents the results of experiments, in which we compare the performance of several baseline models to deep learning approaches on sentiment analysis and topic classification tasks. We conclude with a brief discussion and outlook for future work in Section 5.

2 Related work: popular models for text classification

Traditional sentence and document classification methods rely on a hand-tuned or statically generated feature set. Common feature sets include bag-of-n-gram (BoW) representations (Berger et al., 1996), decomposition-based embeddings (Levy et al., 2015), or representations of parse trees (Collins and Duffy, 2001). More recently, neural word embedding methods - such as skip-gram (Mikolov et al., 2013) and GloVe (Pennington et al., 2014) - and deep neural networks - such as long short-term memory networks (Hochreiter and Schmidhuber, 1997) and convolutional neural networks (Collobert and Weston, 2008) - became the fashionable approach for text classification. While these models can capture long-term and non-linear relationships between pieces of information without a laborious feature engineering process, they are generally not interpretable (Goldberg and Levy, 2014) and require either a great deal of time or a great deal of computing power to train.

In response to these drawbacks, there is renewed interest in developing effective baseline methods for text classification which strike a balance between traditional methods and deep neural network-based approaches. Wang and Manning (2012) show that using bigrams and a simple modification of BoW representations with a linear support vector machine yields performance competitive with deep learning approaches. Joulin et al. (2016) introduce a shallow neural architecture for sentence classification implemented in the fastText library[1] which achieves comparable results in seconds to models that took days to train. Finally, Arora et al. (2017) derive a method for combining fixed word embeddings to form sentence representations which beat long short-term memory (LSTM) models for some tasks. In the following section, we combine ideas from these papers to motivate new shallow architectures for text classification.

3 Combining shallow baseline approaches

3.1 Embedding-based models for classification

The binary fastText classification model (Joulin et al., 2016) is a simple and effective approach to embedding-based text classification. Let $S^{(k)}$ be the unordered set of k-grams in the sentence S, and let $\mathbf{U}$ be an embedding table where the embedding for an entry v is $\mathbf{U}_v$. Then the class prediction of the model f is given by

$$f(S) = \sigma(\mathbf{w}^T \mathbf{\Phi}_S + b) \quad \text{where} \quad \mathbf{\Phi}_S = \frac{1}{\sum_{k=1}^{K} |S^{(k)}|} \sum_{k=1}^{K} \sum_{v \in S^{(k)}} \mathbf{U}_v,$$

K is the desired n-gram representation, and $\sigma(\cdot)$ is the sigmoid function. The key to the effectiveness of fastText is that the embedding table $\mathbf{U}$ (in addition to $\mathbf{w}$ and b) is tuned by gradient descent. Therefore, the model learns to represent each vocabulary entry by a vector which is discriminative for the task at hand. The table $\mathbf{U}$ can be initialized randomly or with a set of pretrained word vectors.

3.2 Improving sentence representations

The primary weakness of fastText is its overly simple sentence representation. By taking an unweighted mean of word vectors, the representation of unmeaningful words (such as "the" or "in") contribute equally to the representations of words which are important to the classification. Although the model may train the representations of unmeaningful words to be close to the origin, there is no guarantee of this.

Therefore, we propose replacing the sentence representation in the fastText model with the representation for fixed word embeddings introduced by Arora et al. (2017). The sentence embedding method is derived via maximum likelihood of a generative language model, and combines word vector reweighting with denoising via matrix factorization. The authors demonstrate that it performs competitively with LSTM models on several text classification tasks.

[1] fastText on GitHub: https://github.com/facebookresearch/fastText

Given a d-dimensional embedding table $\mathbf{U}$, a table of unigram frequency estimates $\mathbf{p}$, a smoothing parameter λ, and a $d \times d$ matrix $\mathbf{T}$, we define

$$\mathbf{\Phi}_S(\lambda, \mathbf{T}) = (\mathbf{I}_d - \mathbf{T}) \frac{1}{|S^{(1)}|} \sum_{v \in S^{(1)}} \frac{\lambda}{\lambda + \mathbf{p}_v} \mathbf{U}_v$$

For a model using static embeddings, the final sentence representation is given by $\mathbf{\Phi}_S(\lambda, \mathbf{z}\mathbf{z}^T)$ where $\mathbf{z}$ is the first right singular vector of the matrix formed by $\mathbf{\Phi}_S(\lambda, \mathbf{0})$ for each S in the training set, and λ remains a parameter to be chosen.

As described by the authors, the direction $\mathbf{z}$ (the "common component") represents a noise direction and correlates with words like "the" and "in". By removing each sentence embedding's project along this direction, the representation retains only meaningful context. However, the fastText training paradigm updates the embedding table via backpropagation every training batch, and the original common component may not be a noise direction in the new embedding space. In the following sections, we address this issue and discuss other nuances.

Updating the common component vector

We explore the following three methods for updating the vector $\mathbf{z}$ in section 4.3.

- **Exact update**: At the beginning of each training batch, the vector $\mathbf{z}$ is computed exactly using singular value decomposition over the full training set. This is computationally expensive, but guarantees the vector is updated correctly.
- **Lazy update**: At the beginning of every Tth training *epoch*, the vector $\mathbf{z}$ is computed exactly using singular value decomposition over the full training set. This is less expensive, but does not account for word vector updates within each epoch.
- **Gradient update**: The vector $\mathbf{z}$ is initialized using singular value decomposition over the full training set and is updated by gradient descent at each training batch. This is the cheapest option, but does not guarantee that $\mathbf{z}$ represents the originally intended direction.

Including *n*-grams

Motivated by large performance gains reported from using n-grams (Wang and Manning, 2012), we extend the above representation to include n-grams. However, pretrained word embeddings and word frequency estimates for large domain-specific corpora (used to populate $\mathbf{p}_v$) are typically only available for unigrams. Therefore, we define

$$\hat{\mathbf{p}}_v = \min_{v' \in v} \mathbf{p}_{v'}$$

where $v' \in v$ are the sub-grams of v, and randomly initialize the entries of $\mathbf{U}_v$ for n-grams in the training set which do not have a pretrained embedding. Then we have

$$\mathbf{\Phi}_S(\lambda, \mathbf{T}) = (\mathbf{I}_d - \mathbf{T}) \frac{1}{\sum_{k=1}^{K} |S^{(k)}|} \sum_{k=1}^{K} \sum_{v \in S^{(k)}} \frac{\lambda}{\lambda + \hat{\mathbf{p}}_v} \mathbf{U}_v$$

Choosing the smoothing parameter

For fixed word embeddings, λ is the only tunable parameter for the sentence embedding. The authors report $\lambda = 10^{-3}$ to be a generally effective value. Therefore, in experiments where we tune the embeddings with backpropagation, we initialize $\lambda = 10^{-3}$ by convention and update the value of the exponent via gradient descent during training.

3.3 The SHALO library

The SHALO library[2] implements all of the above models in TensorFlow[3]. The library also contains sparse BoW and LSTM models for comparison. Table 1 summarizes the models in SHALO.

[2]SHALO library on GitHub: https://github.com/henryre/shalo

[3]https://www.tensorflow.org/

Model	Word representation	Sentence representation	Details
Sparse BoW	One-hot encoding	Word vector sum	Different model graph
LSTM	Trained embeddings	Output of last LSTM cell	
Linear embedding	Fixed embeddings	Word vector mean	
fastText	Trained embeddings	Word vector mean	
TTBB	Fixed embeddings	Arora et al. (2017)	
Tuned TTBB	Trained embeddings	Arora et al. (2017)	Gradient-tuned common component
Exact TTBB	Trained embeddings	Arora et al. (2017)	Common component computed exactly

Table 1: Summary of the models in the SHALO library

Most of the embedding-based models allow the user to choose whether they want the word embeddings to remain fixed from a pretrained set, be tuned from a pretrained set, or be trained from a random initialization. SHALO allows users to choose between training with ℓ_2-regularized logistic loss, which induces a logistic classifier, or with ℓ_2-regularized hinge loss, which induces a maximum margin classifier. The library also contains grid search utilities to tune standard hyperparameters such as learning rate, regularization, and embedding dimension.

The models in SHALO[4] differ only in their text preprocessing and sentence representation methods, and therefore the library uses a minimal class hierarchy to construct each model in a few lines of code. For example, fastText is defined in only seven lines.

4 Experiments

4.1 Datasets

We evaluate the models on three datasets for two binary text classification tasks: sentiment analysis and topic classification. *AG News* (Gulli, 2005) is a dataset of news articles that are organized into four categories. We used only two categories: "Sports" and "Science/Technology". AG News articles have a title and a body, and we concatenate them to create the document for prediction. *Amazon Reviews* (McAuley and Leskovec, 2013) is a dataset of reviews for Amazon products categorized into positive and negative. We only use the review and not the title. *IMDB Reviews* (Potts, 2011) is a dataset of movie reviews, also categorized into positive and negative. These reviews tend to be much longer than the other datasets, averaging 240 words per review.

Due to limitation on compute availability, we randomly sampled a set of 5,000 examples from each dataset for the training set and 1,000 for the development set. We use the full test set for each task to evaluate out methods. Note that if we used the entire dataset, training a single model could take over a day on some of the datasets.

4.2 Training procedure and evaluation

The raw text for each data set was only minimally processed: words were lowercased and tokens with only one character were discarded. The embedding tables for all models were initialized with a set of 174,015 300-dimensional embeddings trained using word2vec with dependency context windows on English Wikipedia (Levy and Goldberg, 2014). The embedding for any n-gram found in the training set but not the pretrained set were randomly initialized. Marginal unigram frequency estimates were derived from the Google Web Trillion Word Corpus (Norvig, 2008).

We fit each model listed in Table 1 to the three datasets described above. To optimally train each model, we must tune a number of hyperparameters. For each model we searched over the loss functions (log or hinge), the learning rate (between 10^{-3} and 10^{-1}), and the ℓ_2 regularization parameter (between 10^{-5} and 10). Because of the large space of hyperparameters, we use a random exploration of the full grid search to attempt to find the optimal parameters. We evaluate the sampled hyperparameter sets on the held out development set, and record the development accuracy at every five epochs. We then choose the snapshot of the model with the best performance on the development set for our final evaluation on the test set.

[4]The sparse BoW model uses a different model graph for efficient sparse matrix-vector multiply.

***n*-gram**	**cos sim.**
art imitating	0.270
pbs program	0.244
cautionary tale	0.219
shorty resonate	0.215
composition of	0.210
so densely	0.206
and cunning	0.205
exhilarating funny	0.201
possibility of	0.201
and joe	0.202

(a) Initial value from exact computation, $\lambda = 10^{-3}$

***n*-gram**	**cos sim.**
it's	0.525
he	0.501
are	0.496
of modern	0.489
three	0.477
kind	0.467
pace	0.452
fare	0.448
hits	0.442
enough	0.441

(b) Epoch 1, $\lambda = 10^{-2.73}$

***n*-gram**	**cos sim.**
in	0.796
the	0.735
he	0.700
that	0.695
his	0.694
its	0.679
but	0.677
pace	0.666
would	0.659
seems	0.659

(c) Epoch 2, $\lambda = 10^{-2.47}$

Table 2: Closest vocabulary entries (by cosine distance) to common component vector across training epochs for the gradient update model. The vocabulary entries most similar to the initial common component vector do not appear to be meaningless. However, after only two epochs, the vector is trained to correlate highly with semantically unimportant words.

4.3 Common component update strategies

We experimented with the three common component update strategies listed in Section 3.2. Even when updating every epoch, the lazy update approach resulted in increasing loss as training progressed since the word embedding space evolved fast enough to render the stale common component vector deleterious to learning. Training loss did decrease with the exact update strategy, but final accuracy results were surprisingly poor, and further examination is needed to determine why. Therefore, we do not include these approaches in the following results section.

On the other hand, the gradient update strategy yielded good results, and demonstrated that the common component vector can be improved beyond the exact singular vector computation. Table 2 shows the evolution of the common component vector being trained by gradient descent. The initial low quality of the common component vector was due to the quantity of randomly initialized embeddings when using both unigrams and bigrams.

4.4 Text classification results

Table 3 shows the classification accuracy for each of the models on the three tasks, and Figure 1 shows their training time. Due to dataset size and the binary problem formulation, the selected tasks were particularly well suited for BoW classification, which had the best scores on both sentiment analysis tasks. In particular, it had at least a 10 point improvement over all other models on the IMDB Reviews (although its training time was drastically greater than all methods besides the LSTM). This finding joins a long list of others that show BoW models are a critical baseline for binary text classification. Our results here motivate further experiments in regimes where models like fastText outperformed BoW models significantly, such as multiclass problems and very large datasets.

The tuned TTBB model outperformed both vanilla TTBB and fastText on all tasks. This validates our hypothesis that combining gradient-tuned word embeddings, modeling n-grams, and generative sentence representation methods produces a more powerful model than these techniques produce individually. Surprisingly, standard TTBB was the worst performing method (including the linear embedding which is a strictly simpler model) on all tasks. The high classification performance reported by Arora et al. (2017) used a multilayer neural network on top of the sentence embedding, rather than just a single linear layer as used here. Therefore, the static sentence embedding method they presented may require more representational power downstream to perform well.

The LSTM model had the best score on AG News, where almost all models had very high accuracy. It had middling performance on the sentiment analysis tasks. On one hand, the fact that the tuned TTBB model outperformed the LSTM on Amazon and IMDB Reviews is an encouraging sign for shallow modeling approaches. However, the more important lesson is that text classification tasks are not demonstrative targets for deep learning methods. Instead, proposed deep neural networks

should prove their efficacy on structured prediction tasks which require modeling non-linear and long-term relationships.

The linear embedding model and standard TTBB had the fastest training times across all tasks since neither uses backpropagation to update word embeddings. The tuned TTBB model and fastText were similar in speed, and offer a worthwhile trade-off to the faster models given the large accuracy improvements they provide. As expected, the LSTM was by far the slowest model to train. However, the sparse BoW model was much slower than fastText or the tuned TTBB model for the IMDB task due to the length of the documents. It is important to note that hand-rolled learning code, such as the official fastText library or LIBLINEAR (Fan et al., 2008), will be orders of magnitude faster than the TensorFlow implementations in SHALO. However, we trained all models in Tensorflow in order to fairly study relative training speeds.

Model	AG News	Amazon Reviews	IMDB Reviews
Sparse BoW	97.50	**81.99**	**87.76**
LSTM	**98.96**	80.91	75.84
Linear embedding	97.74	77.11	75.07
fastText	97.13	80.85	76.43
TTBB	65.84	75.97	75.06
Tuned TTBB	**98.32**	**81.26**	**77.33**

Table 3: Accuracy (%) for evaluation tasks. The two best results for each task are shown in **bold**.

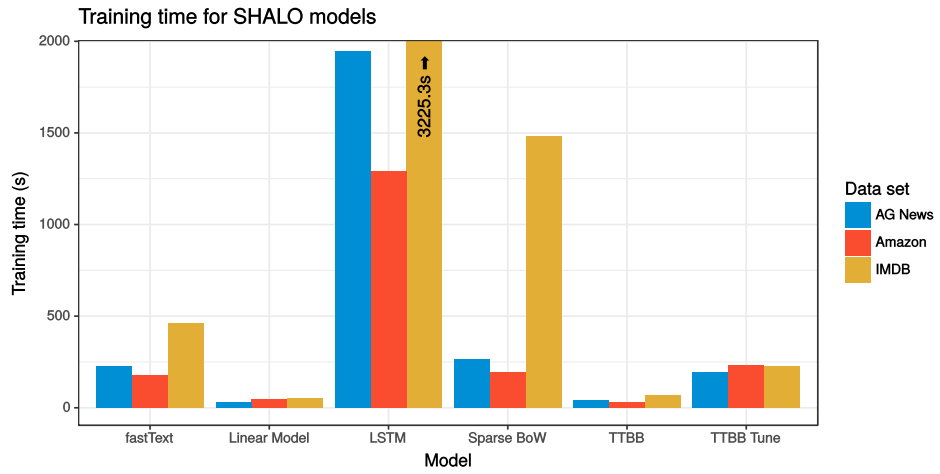

Figure 1: Training time (seconds) for each of the methods. Training was terminated after exceeding a maximum number of epochs (due to computing constraints), or after training loss convergence.

5 Conclusions and future work

In this paper, we reviewed several popular baseline models which strike a balance between traditional approaches and deep learning, and proposed improvements by combining their key contributions. We found that the new shallow model outperformed the previous embedding-based baseline models on all tasks we performed. We also provide more evidence that traditional bag-of-n-gram models are powerful despite their simplicity, and our results suggest that text classification may not be an interesting target for deep learning methods.

There are several directions for future work. Given the promising results of the combined baseline model, we want to investigate its performance further in regimes where fastText and deep methods significantly outperformed BoW models. These include larger datasets with more classes. We are also interested in applying these methods to structured prediction, such as sequence tagging. We hope that research interest in baseline models continue, as they are a critical component of natural language processing in the age of deep learning.

Acknowledgments

We would like to thank our mentor Danqi Chen, the rest of the CS224n course staff, and our advisor Chris Ré for unknowingly providing us with computing resources.

References

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. *ICLR*, 2017.

Adam L Berger, Vincent J Della Pietra, and Stephen A Della Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 1996.

Michael Collins and Nigel Duffy. Convolution kernels for natural language. *NIPS*, 2001.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. *ICML*, 2008.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 2008.

Yoav Goldberg and Omer Levy. word2vec explained: Deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

A. Gulli. The anatomy of a news search engine. *International World Wide Web Conference*, 2005.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

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

Omer Levy and Yoav Goldberg. Dependency-based word embeddings. *ACL*, 2014.

Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *TACL*, 3, 2015.

J. McAuley and J. Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. *RecSys*, 2013.

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

Peter Norvig. Natural language corpus data. In *Beautiful Data*. O'Reilly, 2008. URL http://norvig.com/ngrams/.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global vectors for word representation. *EMNLP*, 2014.

Christopher Potts. On the negativity of negation. *Proceedings of Semantics and Linguistic Theory 20*, 2011.

Sida Wang and Christopher D Manning. Baselines and bigrams: Simple, good sentiment and topic classification. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 90–94, 2012.