Bag of Tricks for Efficient Text Classification

Armand Joulin **Edouard Grave** **Piotr Bojanowski** **Tomas Mikolov**
Facebook AI Research
{ajoulin,egrave,bojanowski,tmikolov}@fb.com

Abstract

This paper explores a simple and efficient baseline for text classification. Our experiments show that our fast text classifier `fastText` is often on par with deep learning classifiers in terms of accuracy, and many orders of magnitude faster for training and evaluation. We can train `fastText` on more than one billion words in less than ten minutes using a standard multicore CPU, and classify half a million sentences among 312K classes in less than a minute.

1 Introduction

Text classification is an important task in Natural Language Processing with many applications, such as web search, information retrieval, ranking and document classification (Deerwester et al., 1990; Pang and Lee, 2008). Recently, models based on neural networks have become increasingly popular (Kim, 2014; Zhang and LeCun, 2015; Conneau et al., 2016). While these models achieve very good performance in practice, they tend to be relatively slow both at train and test time, limiting their use on very large datasets.

Meanwhile, linear classifiers are often considered as strong baselines for text classification problems (Joachims, 1998; McCallum and Nigam, 1998; Fan et al., 2008). Despite their simplicity, they often obtain state-of-the-art performances if the right features are used (Wang and Manning, 2012). They also have the potential to scale to very large corpus (Agarwal et al., 2014).

In this work, we explore ways to scale these baselines to very large corpus with a large output space, in the context of text classification. Inspired by the recent work in efficient word representation learning (Mikolov et al., 2013; Levy et al., 2015), we show that linear models with a rank constraint and a fast loss approximation can train on a billion words within ten minutes, while achieving performance on par with the state-of-the-art. We evaluate the quality of our approach `fastText`[1] on two different tasks, namely tag prediction and sentiment analysis.

2 Model architecture

A simple and efficient baseline for sentence classification is to represent sentences as bag of words (BoW) and train a linear classifier, e.g., a logistic regression or an SVM (Joachims, 1998; Fan et al., 2008). However, linear classifiers do not share parameters among features and classes. This possibly limits their generalization in the context of large output space where some classes have very few examples. Common solutions to this problem are to factorize the linear classifier into low rank matrices (Schutze, 1992; Mikolov et al., 2013) or to use multilayer neural networks (Collobert and Weston, 2008; Zhang et al., 2015).

Figure 1 shows a simple linear model with rank constraint. The first weight matrix A is a look-up table over the words. The word representations are then averaged into a text representation, which is in turn fed to a linear classifier. The text representa-

[1] `https://github.com/facebookresearch/fastText`

fasttext 的输入层 是 词 + n-grams词组(这是附加特征) 组成的特征向量(单词向量通过embeddings获得，n-grams中每个词组作为一个词来看待，通过另一个embeddings获得)，该特征向量经过线性变化(对所有单词向量做求和取平均)映射到隐藏层，得到text representation向量，隐藏层再经过层次softmax映射到标签。在预测的过程中，同时也学到了词的向量表示和n-grams的向量表示。在预测标签时使用非线性激活函数，而在中间的隐藏层不使用。

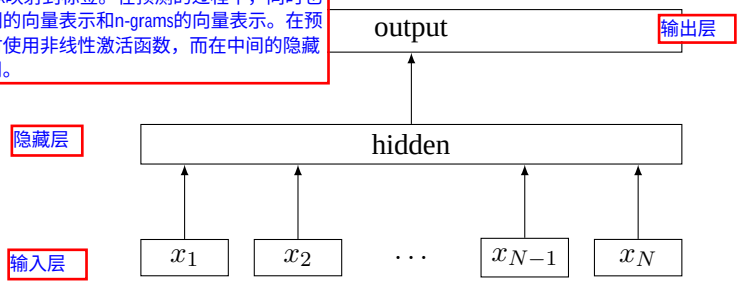

Figure 1: Model architecture of `fastText` for a sentence with N ngram features $x_1, \ldots, x_N$. The features are embedded and averaged to form the hidden variable.

tion is an hidden variable which can be potentially be reused. This architecture is similar to the cbow model of Mikolov et al. (2013), where the middle word is replaced by a label. We use the softmax function f to compute the probability distribution over the predefined classes. For a set of N documents, this leads to minimizing the negative log-likelihood over the classes:

$$-\frac{1}{N}\sum_{n=1}^{N} y_n \log(f(BAx_n)),$$

where x_n is the normalized bag of features of the n-th document, y_n the label, A and B the weight matrices. This model is trained asynchronously on multiple CPUs using stochastic gradient descent and a linearly decaying learning rate.

2.1 Hierarchical softmax

When the number of classes is large, computing the linear classifier is computationally expensive. More precisely, the computational complexity is $O(kh)$ where k is the number of classes and h the dimension of the text representation. In order to improve our running time, we use a hierarchical softmax (Goodman, 2001) based on the Huffman coding tree (Mikolov et al., 2013). During training, the computational complexity drops to $O(h \log_2(k))$.

The hierarchical softmax is also advantageous at test time when searching for the most likely class. Each node is associated with a probability that is the probability of the path from the root to that node. If the node is at depth $l+1$ with parents $n_1, \ldots, n_l$, its probability is

$$P(n_{l+1}) = \prod_{i=1}^{l} P(n_i).$$

This means that the probability of a node is always lower than the one of its parent. Exploring the tree with a depth first search and tracking the maximum probability among the leaves allows us to discard any branch associated with a small probability. In practice, we observe a reduction of the complexity to $O(h \log_2(k))$ at test time. This approach is further extended to compute the T-top targets at the cost of $O(\log(T))$, using a binary heap.

2.2 N-gram features

Bag of words is invariant to word order but taking explicitly this order into account is often computationally very expensive. Instead, we use a bag of n-grams as additional features to capture some partial information about the local word order. This is very efficient in practice while achieving comparable results to methods that explicitly use the order (Wang and Manning, 2012).

We maintain a fast and memory efficient mapping of the n-grams by using the *hashing trick* (Weinberger et al., 2009) with the same hashing function as in Mikolov et al. (2011) and 10M bins if we only used bigrams, and 100M otherwise.

3 Experiments

We evaluate `fastText` on two different tasks. First, we compare it to existing text classifers on the problem of sentiment analysis. Then, we evaluate its capacity to scale to large output space on a tag prediction dataset. Note that our model could be implemented with the Vowpal Wabbit library,[2] but we observe in practice, that our tailored implementation is at least 2-5× faster.

3.1 Sentiment analysis

Datasets and baselines. We employ the same 8 datasets and evaluation protocol of Zhang et al. (2015). We report the n-grams and TFIDF baselines from Zhang et al. (2015), as well as the character level convolutional model (char-CNN) of Zhang and LeCun (2015), the character based convolution recurrent network (char-CRNN) of (Xiao and Cho, 2016) and the very deep convolutional network (VDCNN) of Conneau et al. (2016). We also compare

[2] Using the options `--nn`, `--ngrams` and `--log_multi`

Model	AG	Sogou	DBP	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
BoW (Zhang et al., 2015)	88.8	92.9	96.6	92.2	58.0	68.9	54.6	90.4
ngrams (Zhang et al., 2015)	92.0	97.1	98.6	95.6	56.3	68.5	54.3	92.0
ngrams TFIDF (Zhang et al., 2015)	92.4	97.2	98.7	95.4	54.8	68.5	52.4	91.5
char-CNN (Zhang and LeCun, 2015)	87.2	95.1	98.3	94.7	62.0	71.2	59.5	94.5
char-CRNN (Xiao and Cho, 2016)	91.4	95.2	98.6	94.5	61.8	71.7	59.2	94.1
VDCNN (Conneau et al., 2016)	91.3	96.8	98.7	95.7	64.7	73.4	63.0	95.7
`fastText`, $h = 10$	91.5	93.9	98.1	93.8	60.4	72.0	55.8	91.2
`fastText`, $h = 10$, bigram	92.5	96.8	98.6	95.7	63.9	72.3	60.2	94.6

Table 1: Test accuracy [%] on sentiment datasets. `FastText` has been run with the same parameters for all the datasets. It has 10 hidden units and we evaluate it with and without bigrams. For char-CNN, we show the best reported numbers without data augmentation.

	Zhang and LeCun (2015)		Conneau et al. (2016)			`fastText`
	small char-CNN	big char-CNN	depth=9	depth=17	depth=29	$h = 10$, bigram
AG	1h	3h	24m	37m	51m	1s
Sogou	-	-	25m	41m	56m	7s
DBpedia	2h	5h	27m	44m	1h	2s
Yelp P.	-	-	28m	43m	1h09	3s
Yelp F.	-	-	29m	45m	1h12	4s
Yah. A.	8h	1d	1h	1h33	2h	5s
Amz. F.	2d	5d	2h45	4h20	7h	9s
Amz. P.	2d	5d	2h45	4h25	7h	10s

Table 2: Training time for a single epoch on sentiment analysis datasets compared to char-CNN and VDCNN.

to Tang et al. (2015) following their evaluation protocol. We report their main baselines as well as their two approaches based on recurrent networks (Conv-GRNN and LSTM-GRNN).

Results. We present the results in Figure 1. We use 10 hidden units and run `fastText` for 5 epochs with a learning rate selected on a validation set from $\{0.05, 0.1, 0.25, 0.5\}$. On this task, adding bigram information improves the performance by 1-4%. Overall our accuracy is slightly better than char-CNN and char-CRNN and, a bit worse than VDCNN. Note that we can increase the accuracy slightly by using more n-grams, for example with trigrams, the performance on Sogou goes up to 97.1%. Finally, Figure 3 shows that our method is competitive with the methods presented in Tang et al. (2015). We tune the hyperparameters on the validation set and observe that using n-grams up to 5 leads to the best performance. Unlike Tang et al. (2015), `fastText` does not use pre-trained word embeddings, which can be explained the 1% difference in accuracy.

Model	Yelp'13	Yelp'14	Yelp'15	IMDB
SVM+TF	59.8	61.8	62.4	40.5
CNN	59.7	61.0	61.5	37.5
Conv-GRNN	63.7	65.5	66.0	42.5
LSTM-GRNN	65.1	67.1	67.6	45.3
`fastText`	64.2	66.2	66.6	45.2

Table 3: Comparision with Tang et al. (2015). The hyperparameters are chosen on the validation set. We report the test accuracy.

Training time. Both char-CNN and VDCNN are trained on a NVIDIA Tesla K40 GPU, while our models are trained on a CPU using 20 threads. Table 2 shows that methods using convolutions are several orders of magnitude slower than `fastText`. While it is possible to have a $10\times$ speed up for char-CNN by using more recent CUDA implementations of convolutions, `fastText` takes less than a minute to train on these datasets. The GRNNs method of Tang et al. (2015) takes around 12 hours per epoch on CPU with a single thread. Our speed-

Input	Prediction	Tags
taiyoucon 2011 digitals: individuals digital photos from the anime convention taiyoucon 2011 in mesa, arizona. if you know the model and/or the character, please comment.	#cosplay	#24mm #anime #animeconvention #arizona #canon #con #convention #cos **#cosplay** #costume #mesa #play #taiyou #taiyoucon
2012 twin cities pride 2012 twin cities pride parade	#minneapolis	#2012twincitiesprideparade **#minneapolis** #mn #usa
beagle enjoys the snowfall	#snow	#2007 #beagle #hillsboro #january #maddison #maddy #oregon **#snow**
christmas	#christmas	#cameraphone #mobile
euclid avenue	#newyorkcity	#cleveland #euclidavenue

Table 4: Examples from the validation set of YFCC100M dataset obtained with `fastText` with 200 hidden units and bigrams. We show a few correct and incorrect tag predictions.

up compared to neural network based methods increases with the size of the dataset, going up to at least a 15,000× speed-up.

3.2 Tag prediction

Dataset and baselines. To test scalability of our approach, further evaluation is carried on the YFCC100M dataset (Thomee et al., 2016) which consists of almost 100M images with captions, titles and tags. We focus on predicting the tags according to the title and caption (we do not use the images). We remove the words and tags occurring less than 100 times and split the data into a train, validation and test set. The train set contains 91,188,648 examples (1.5B tokens). The validation has 930,497 examples and the test set 543,424. The vocabulary size is 297,141 and there are 312,116 unique tags. We will release a script that recreates this dataset so that our numbers could be reproduced. We report precision at 1.

We consider a frequency-based baseline which predicts the most frequent tag. We also compare with Tagspace (Weston et al., 2014), which is a tag prediction model similar to ours, but based on the Wsabie model of Weston et al. (2011). While the Tagspace model is described using convolutions, we consider the linear version, which achieves comparable performance but is much faster.

Results and training time. Table 5 presents a comparison of `fastText` and the baselines. We run `fastText` for 5 epochs and compare it to Tagspace for two sizes of the hidden layer, i.e., 50 and 200. Both models achieve a similar performance with a small hidden layer, but adding bigrams gives us a significant boost in accuracy. At test time, Tagspace needs to compute the scores for all the classes which makes it relatively slow, while our fast inference gives a significant speed-up when the number of classes is large (more than 300K here). Overall, we are more than an order of magnitude faster to obtain model with a better quality. The speedup of the test phase is even more significant (a 600× speedup). Table 4 shows some qualitative examples.

Model	prec@1	Running time	
		Train	Test
Freq. baseline	2.2	-	-
Tagspace, $h = 50$	30.1	3h8	6h
Tagspace, $h = 200$	35.6	5h32	15h
`fastText`, $h = 50$	31.2	6m40	48s
`fastText`, $h = 50$, bigram	36.7	7m47	50s
`fastText`, $h = 200$	41.1	10m34	1m29
`fastText`, $h = 200$, bigram	46.1	13m38	1m37

Table 5: Prec@1 on the test set for tag prediction on YFCC100M. We also report the training time and test time. Test time is reported for a single thread, while training uses 20 threads for both models.

4 Discussion and conclusion

In this work, we propose a simple baseline method for text classification. Unlike unsupervisedly trained word vectors from word2vec, our word features can

be averaged together to form good sentence representations. In several tasks, `fastText` obtains performance on par with recently proposed methods inspired by deep learning, while being much faster. Although deep neural networks have in theory much higher representational power than shallow models, it is not clear if simple text classification problems such as sentiment analysis are the right ones to evaluate them. We will publish our code so that the research community can easily build on top of our work.

Acknowledgement. We thank Gabriel Synnaeve, Hervé Gégou, Jason Weston and Léon Bottou for their help and comments. We also thank Alexis Conneau, Duyu Tang and Zichao Zhang for providing us with information about their methods.

References

[Agarwal et al.2014] Alekh Agarwal, Olivier Chapelle, Miroslav Dudík, and John Langford. 2014. A reliable effective terascale linear learning system. *JMLR*.

[Collobert and Weston2008] Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*.

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

[Deerwester et al.1990] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science*.

[Fan et al.2008] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. Liblinear: A library for large linear classification. *JMLR*.

[Goodman2001] Joshua Goodman. 2001. Classes for fast maximum entropy training. In *ICASSP*.

[Joachims1998] Thorsten Joachims. 1998. *Text categorization with support vector machines: Learning with many relevant features*. Springer.

[Kim2014] Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *EMNLP*.

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

[McCallum and Nigam1998] Andrew McCallum and Kamal Nigam. 1998. A comparison of event models for naive bayes text classification. In *AAAI workshop on learning for text categorization*.

[Mikolov et al.2011] Tomáš Mikolov, Anoop Deoras, Daniel Povey, Lukáš Burget, and Jan Černockỳ. 2011. Strategies for training large scale neural network language models. In *Workshop on Automatic Speech Recognition and Understanding*. IEEE.

[Mikolov et al.2013] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[Pang and Lee2008] Bo Pang and Lillian Lee. 2008. Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*.

[Schutze1992] Hinrich Schutze. 1992. Dimensions of meaning. In *Supercomputing*.

[Tang et al.2015] Duyu Tang, Bing Qin, and Ting Liu. 2015. Document modeling with gated recurrent neural network for sentiment classification. In *EMNLP*.

[Thomee et al.2016] Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. 2016. Yfcc100m: The new data in multimedia research. volume 59, pages 64–73. ACM.

[Wang and Manning2012] Sida Wang and Christopher D Manning. 2012. Baselines and bigrams: Simple, good sentiment and topic classification. In *ACL*.

[Weinberger et al.2009] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In *ICML*.

[Weston et al.2011] Jason Weston, Samy Bengio, and Nicolas Usunier. 2011. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*.

[Weston et al.2014] Jason Weston, Sumit Chopra, and Keith Adams. 2014. #tagspace: Semantic embeddings from hashtags. In *EMNLP*.

[Xiao and Cho2016] Yijun Xiao and Kyunghyun Cho. 2016. Efficient character-level document classification by combining convolution and recurrent layers. *arXiv preprint arXiv:1602.00367*.

[Zhang and LeCun2015] Xiang Zhang and Yann LeCun. 2015. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*.

[Zhang et al.2015] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *NIPS*.