Semi-supervised Learning of Compact Document Representations with Deep Networks

Marc'Aurelio Ranzato RANZATO@COURANT.NYU.EDU
Courant Institute, New York University, 719 Broadway 12th fl., New York NY 10003, USA

Martin Szummer SZUMMER@MICROSOFT.COM
Microsoft Research Cambridge, 7 J J Thomson Avenue, Cambridge CB3 0FB, UK

Abstract

Finding good representations of text documents is crucial in information retrieval and classification systems. Today the most popular document representation is based on a vector of word counts in the document. This representation neither captures dependencies between related words, nor handles synonyms or polysemous words. In this paper, we propose an algorithm to learn text document representations based on semi-supervised autoencoders that are stacked to form a deep network. The model can be trained efficiently on partially labeled corpora, producing very compact representations of documents, while retaining as much class information and joint word statistics as possible. We show that it is advantageous to exploit even a few labeled samples during training.

1. Introduction

Document representations are a key ingredient in all information retrieval and processing systems. The goal of the representation is to make certain aspects of the document readily accessible, e.g. the document topic. To identify a document topic, we cannot rely on specific words in the document, as it may use other synonymous words or misspellings. Likewise, the presence of a word does not warrant that the document is related to it, as it may be taken out of context, or polysemous, or unimportant to the document topic.

The most widespread representations for document classification and retrieval today are based on a vector of counts. These include various term-weighting retrieval schemes, such as tf-idf and BM25 (Robertson and Walker, 1994), and bag-of-words generative models such as naive Bayes text classifiers. The pertinent feature of these representations is that they represent individual words. A serious drawback of the basic tf-idf and BM25 representations is that all dimensions are treated as independent, whereas in reality word occurrences are highly correlated.

There have been many attempts at modeling word correlations by rotating the vector space and projecting documents onto principal axes that expose related words. Methods include LSI (Deerwester et al., 1990) and pLSI (Hofmann, 1999). These methods constitute a linear re-mapping of the original vector space, and while an improvement, still can only capture very limited relations between words. As a result they need a large number of projections in order to give an appropriate representation.

Other models, such as LDA (Blei et al., 2003), have shown superior performance over pLSI and LSI. However, inferring the representation is computationally expensive because of the "explaining away" effect that plagues all directed graphical models.

More recently, a number of authors have proposed undirected graphical models that can make inference efficient at the cost of more complex learning due to a global (rather than local) partition function whose exact gradient is intractable. These models build on Restricted Boltzmann Machines (RBMs) by adapting the conditional distribution of the input visible units to model discrete counts of words (Hinton and Salakhutdinov, 2006; Gehler et al., 2006; Salakhutdinov and Hinton, 2007a,b). These models have shown state-of-the-art performance in retrieval and clustering, and can be easily used as a building block for deep multi-layer networks (Hinton et al., 2006). This might allow the

Appearing in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008. Copyright 2008 by the author(s)/owner(s).

top-level representation to capture high-order correlations that would be difficult to efficiently represent with similar but shallow models (Bengio and LeCun, 2007). Many authors have pointed out that RBMs are robust to uncorrelated noise in the input since they model the distribution of the input data, and they implicitly perform automatic model selection by not using unnecessary hidden units. But they are also somewhat cumbersome to train, relying on two disparate steps: unsupervised pre-training using an approximate sampling technique such as contrastive divergence (Hinton, 2000), followed by supervised back-propagation. It is rather difficult to predict when training can be stopped and how long the Markov Chain has to run. An alternative is to replace RBMs with autoencoders (Bengio et al., 2006), or special autoencoders that produce sparse representations (Ranzato et al., 2007b). According to these authors, the performance of RBMs and standard autoencoders is quite similar as long as the dimensionality of the latent space is smaller than the input. Seeking an algorithm that can be trained efficiently, and that can produce a representation with just a few matrix multiplications, we propose a deep network whose building blocks are autoencoders, with a specially designed first layer for modeling discrete counts of words.

Previously, deep networks have been trained either from fully labeled data, or purely unlabeled data. Neither method is ideal, as it is expensive to label large collections, whereas purely unsupervised learning may not capture the relevant class information in the data. Inspired by the experiments by Bengio, Lamblin et al. (2006), we learn the parameters of the model by using *both a supervised and an unsupervised objective*. In other words, we require the representation to produce good reconstructions of the input documents and, at the same time, to give good predictions of the document class labels. Besides demonstrating better accuracy in retrieval, we also extend the deep network framework to a *semi-supervised* setting where we deal with partially labeled collections of documents. This allows us to use relatively few labeled documents yet leverage language structure learned from large corpora, see Sec. 3.1.

Finally, we study the relative advantages of different deep models. For instance, we investigate when deep models are better than shallow ones. Our experiments in Sec. 3.2 show that for learning compact representations of documents, deep architectures greatly outperform shallow models. Compact representations are beneficial because they require less storage (an important consideration for large search engines), and they are more computationally efficient when used in indexing. We also explored the possibility to use deep networks to

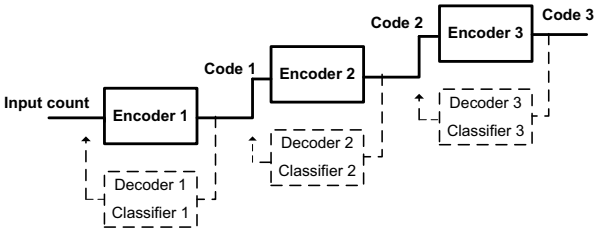

Figure 1. Architecture of a model with three stages. The system is trained layer by layer. During the training of the n-th layer, the n-th encoder is coupled with the n-th decoder and classifier (shown in dashed line). The n-th encoder will provide the codes to train the layer above. After training, the feedback decoding modules are discarded and the system is used to produce very compact codes by a feed-forward pass through the chain of encoders.

learn binary *high-dimensional* representations instead of *compact* representations. These high-dimensional representations were trained using the Symmetric Encoding Sparse Machine (SESM) (Ranzato et al., 2007b). However, the compact representations proved to be far more efficient in terms of memory usage and CPU time, as described in Sec. 3.3. Also, training is more computationally efficient than for related models such as RBMs.

2. The model

The input to the system is a bag of words representation of each text document in the form of a count vector. The length of the vector equals the number of unique words in the collection, and its i-th entry stores the number of times the corresponding word occurs in the document. The goal of the system is to extract a *compact* representation from this very high-dimensional but sparse input vector. A compact representation is good because it requires less storage, and allows fast index lookup. Since the representation is produced by a deep multi-layer model, it can efficiently discover latent topics by grouping similar words and by activating features whenever some "interesting" combination of words is detected (see visualization in Sec. 3.4).

We propose a system that is composed of multiple layers. Each layer computes a weighted sum of its input followed by a logistic nonlinearity. Each layer can be seen as an *encoder* producing a representation, or *code*, from its input. This code will be propagated and used as the input to the next layer of the model. This architecture is quite similar to a neural network model, but is trained differently and has a special first layer able to encode discrete count data. The goal of training is to find the parameters in each layer.

In order to successfully learn the parameters we follow the strategy advocated by recent work (Hinton et al., 2006; Hinton and Salakhutdinov, 2006; Bengio et al., 2006) on deep multi-layer models. Learning proceeds greedily layer by layer. When the parameters of one layer have been found, the data is fed through that layer and the output becomes the input for the next layer, which is trained subsequently.

Let us consider a generic layer in the model, and let $\boldsymbol{x}$ be its input and let $\boldsymbol{z}$ be the representation produced by the layer. In order to warrant the fidelity of the code $\boldsymbol{z}$, we attach a feedback module which aims to reconstruct the input $\boldsymbol{x}$ from the code $\boldsymbol{z}$. The reason is that if the model achieves a good reconstruction from the code $\boldsymbol{z}$, then we can be sure that the representation has preserved most of the information from $\boldsymbol{x}$. The original layer can then be interpreted as an *encoder* that computes a code from the input, while the feedback module can be seen as a *decoder* that reconstructs the input $\boldsymbol{x}$ from the code $\boldsymbol{z}$. Learning consists of minimizing a reconstruction error E_R with respect to the parameters in the encoder and decoder when the input $\boldsymbol{x}$ is drawn from the training dataset. Since we learn by stochastic gradient descent, any type of encoder and decoder is allowed as long as it is differentiable.

Some inputs may have labels specifying the class to which they belong. In order to incorporate this information, we add another module to the decoder. The feedback module now not only has a decoder reconstructing the input $\boldsymbol{x}$, but also a *classifier* predicting the label y from the code $\boldsymbol{z}$, see Fig. 1. During training the parameters of the encoder, classifier and decoder are learned by minimizing the loss

$$L = E_R + \alpha_c E_C, \tag{1}$$

where E_R and E_C are terms measuring the reconstruction and classification error respectively, and α_c is a coefficient balancing them. The first term is common to many unsupervised learning algorithms and makes the system model the structure and the dependencies among the input components of $\boldsymbol{x}$. The second term represents the supervised goal ensuring that codes are also going to be good for discriminating between classes.

For the classifier module we used a linear classifier trained by cross-entropy error E_C. Denoting with $(W_C)_i$ the i-th row of the classifier weight matrix, with b_{Ci} the i-th bias of the classifier, and with h_j the j-th output unit of the classifier passed through a soft-max:

$$h_j = \frac{\exp((W_C)_j \cdot \boldsymbol{z} + b_{Cj})}{\sum_i \exp((W_C)_i \cdot \boldsymbol{z} + b_{Ci})},$$

we define $E_C = -\sum_i y_i \log h_i$, where $\boldsymbol{y}$ is a 1-of-N encoding of the target class label.

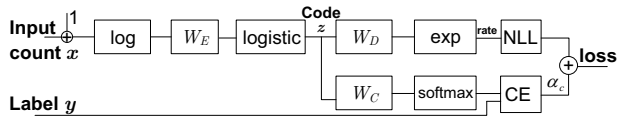

Figure 2. The architecture of the first stage has three components: (1) an encoder, (2) a decoder (Poisson regressor), and (3) a classifier. The loss is the weighted sum of cross-entropy (CE) and negative log-likelihood (NLL) under the Poisson model.

2.1. Training the First Stage

The first stage is special because the input $\boldsymbol{x}$ is a discrete vector of word counts, with x_i counting the number of occurrences of the i-th word in the document. The decoder is a *Poisson regression model* aiming to predict $\boldsymbol{x}$ from the code $\boldsymbol{z}$. A Poisson regressor is a log-linear model which assigns the following probability to an observed $\boldsymbol{x}$:

$$P(\boldsymbol{x}) = \prod_i P(x_i) = \prod_i \mathrm{e}^{-\lambda_i} \frac{\lambda_i^{x_i}}{x_i!}, \tag{2}$$

where the set of rates is given by $\lambda = \beta \mathrm{e}^{W_D \boldsymbol{z} + \boldsymbol{b}_D}$, with a decoder weight matrix W_D, decoder biases $\boldsymbol{b}_D$, and a constant β proportional to the document length. This normalization handles documents of different lengths and makes learning stable. The reconstruction error E_R minimized at the first stage (eq. 1) is the negative log-likelihood of the data

$$E_R = \sum_i (\beta \mathrm{e}^{((W_D)_i \cdot \boldsymbol{z} + b_{Di})} - x_i (W_D)_i \cdot \boldsymbol{z} - x_i b_{Di} + \log x_i!), \tag{3}$$

averaged over the samples $\boldsymbol{x}$ in the training dataset.

We design the encoder by "reverse-engineering" the decoder to make the machine symmetric. Since the decoder computes an exponential of a weighted sum, the encoder performs a weighted sum of the log-transformed input $\boldsymbol{x}$. In addition to this, the encoder applies a logistic nonlinearity. Hence, the code $\boldsymbol{z}$ is given by $\boldsymbol{z} = \boldsymbol{\sigma}(W_E \log(\boldsymbol{x}) + \boldsymbol{b}_E)$, where W_E and $\boldsymbol{b}_E$ are the weight matrix and the biases in the encoder, and $\boldsymbol{\sigma}$ is the logistic. Since many components in $\boldsymbol{x}$ are zero (because only a few dictionary words are actually present in a given document), and since the rate at which a word might appear is generally fairly low, this architecture is prone to numerical problems in the evaluation of the logarithm, and would possibly require large negative weights in W_D (in order to make the rate λ small). Thus, we shift the Poisson regression by adding one to the input. As a result, if a word does not occur in the document, the input to the encoder weight matrix W_E will be zero (and not minus infinity), and

if a word is rare its rate will be one forcing the corresponding weights in W_D to be close to zero (and not to minus infinity). Fig. 2 shows the final architecture of the first stage.

2.2. Training the Upper Stages

The outputs of earlier layers are fed as inputs of subsequent layers. The architecture of the subsequent layers differs from the first one in that the decoder uses a Gaussian regressor instead of a Poisson regressor. Accordingly, the encoder computes a weighted sum of its input and applies a logistic nonlinearity. This architecture is similar to an autoencoder neural network, but here the feedback layer also includes a supervised classifier. If $\boldsymbol{z}^{(n-1)}$ is the input to the n-th layer, the code $\boldsymbol{z}^{(n)}$ produced at this stage is $\boldsymbol{z}^{(n)} = \boldsymbol{\sigma}(W_E \boldsymbol{z}^{(n-1)} + \boldsymbol{b}_E)$. The reconstruction error E_R in the loss of eq. 1 can be written as $E_R = \|\boldsymbol{z}^{(n-1)} - W_D \boldsymbol{z}^{(n)} - \boldsymbol{b}_D\|_2^2$.

2.3. Training the Whole Model

Learning consists of determining the parameters at each layer of the deep model. The algorithm proceeds as follows:
(1) attach a Poisson regressor and a linear classifier to the first layer, and minimize the loss in eq. 1 with respect to the parameters $(W_E, b_E, W_D, b_D, W_C, b_C)$ by stochastic gradient descent;
(2) transform the training samples $\boldsymbol{x}$ into codes $\boldsymbol{z}^{(1)}$ using the trained encoder of the first layer;
(3) train the second layer by attaching a Gaussian regressor and a linear classifier to the encoder, using the codes $\boldsymbol{z}^{(1)}$ as input;
(4) use the trained encoder of the second layer to transform the codes $\boldsymbol{z}^{(1)}$ into the higher-level codes $\boldsymbol{z}^{(2)}$;
(5) repeat the previous two steps for as many layers as desired.

When the input sample is not accompanied by a label, the classifier is not updated and the loss function simply reduces to $L = E_R$. In order to minimize the loss with respect to the parameters we use stochastic gradient descent and we back-propagate the derivatives through the decoder, classifier and encoder (LeCun et al., 1998).

The learning algorithm is particularly efficient. The computational cost of learning is linear in the number of training samples (sublinear for redundant datasets, which are frequent). For each training document at any given layer, the cost is given by a forward and backward pass through encoder, decoder and classifier. Each pass is dominated by a matrix-vector multiplication whose complexity depends on the size of the matrix. Since at each layer we reduce the dimensionality of the input, the first layer dominates the computational cost. However, the sparsity of the input count vector can be exploited to speed-up the computation by taking into account only those rows in W_E that are involved in the computation. In general, the computational cost at a given layer scales as $4MN + 2NK$, where M is the dimensionality of the input, N is the dimensionality of the code, and K is the number of classes.

If we are interested in classification we can also use the trained classifier to predict the labels from the features (at any layer), without training a separate supervised system (see Sec. 3.1 for an example). Also, our experiments show that there is not much advantage in "fine-tuning" the parameters by doing global non-greedy supervised training of the machine as performed by Hinton et al. (2006). The label injection during the greedy training of each layer renders this final supervised training stage unnecessary. This saves a lot of time because it is expensive to do forward and backward propagation through a large and deep network.

Inference is also very efficient. Once the model is trained the encoders are stacked and the decoder and classifier modules are removed. A feature vector is computed by a forward propagation of the input sparse count vector through the sequence of encoders. This computation requires a few matrix vector multiplications, where the most expensive one is at the first layer, which can benefit further from a sparse computation.

3. Experiments

In our experiments we considered three standard datasets: 20 Newsgroups, Reuters-21578, and Ohsumed[1]. The 20 Newsgroups dataset contains 18845 postings taken from the Usenet newsgroup collection. Documents are partitioned into 20 topics. The dataset is split into 11314 training documents and 7531 test documents. Training and test articles are separated in time. Reuters has a predefined ModApte split of the data into 11413 training documents and 4024 test documents. Documents belong to one of 91 topics. The Ohsumed dataset has 34389 documents with 30689 words and each document might be assigned to more than one topic, for a total of 23 topics. The dataset is split into training and test by randomly selecting the 67% and the 33% of the data. Rainbow[2] was used to pre-process these datasets by stemming the documents,

[1] These corpora were downloaded from `http://people.csail.mit.edu/jrennie/20Newsgroups`, and `http://www.kyb.mpg.de/bs/people/pgehler/rap`

[2] Rainbow is available at `http://www.cs.cmu.edu/~mccallum/bow/rainbow`

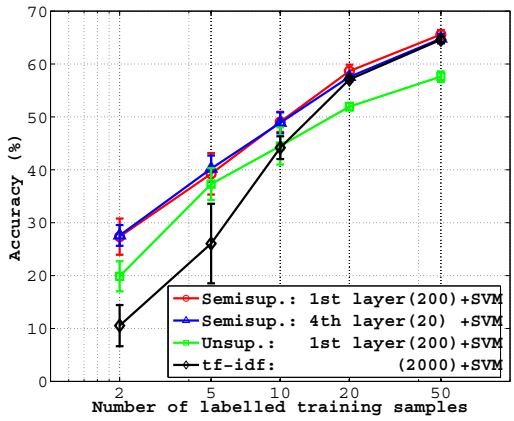

Figure 3. SVM classification of documents from the 20 Newsgroups dataset (2000 word vocabulary) trained with between 2 and 50 labeled samples per class. The SVM was applied to representations from the deep model trained in a semi-supervised or unsupervised way, and to the tf-idf representation. The numbers in parentheses denote the number of code units. Error bars indicate one standard deviation. The fourth layer representation has only 20 units, and is much more compact and computationally efficient than all the other representations.

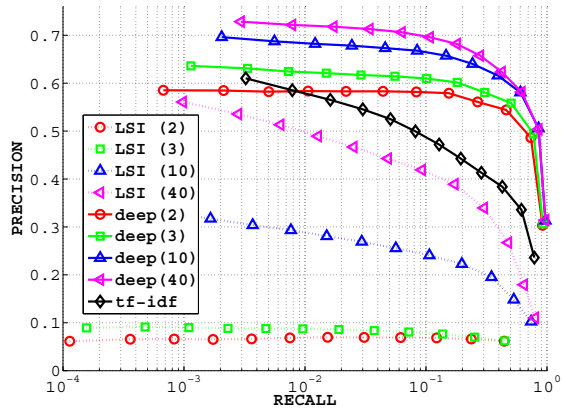

Figure 4. Precision-recall curves for the Reuters dataset comparing a linear model (LSI) to the nonlinear deep model with the same number of code units (in parentheses). Retrieval is done using the k most similar documents according to cosine similarity, with $k \in [1 \ldots 4095]$.

removing stop words and words appearing less than three times or in only a single document, and retaining between 1000 and 30,000 words with the highest mutual information.

Unless stated otherwise, we trained each layer of the network for only 4 epochs over the whole training dataset. Convergence took only a couple of epochs, and was robust to the choice of the learning rate. This was set to about 10^{-4} when training the first layer, and to 10^{-3} when training the layers above. The learning rate was exponentially decreased by multiplying it by 0.97 every 1000 samples. A small L1 regularizer on the parameters was added to the loss. Each weight was randomly initialized, and was updated by taking a gradient step with a regularizer given by the value of the learning rate times $5 \cdot 10^{-4}$ the sign of the weight. The value of α_c in eq. 1 was set to the ratio between the number of input units in the layer and the number of classes in order to make the two error terms E_R and E_C comparable. Its exact value did not affect the performance as long as it had the right order of magnitude.

3.1. The Value of Labels

In order to assess whether semi-supervised training was better than purely unsupervised training, we trained the deep model on the 20 Newsgroup dataset using only 2, 5, 10, 20 and 50 samples per class. During training we showed the system 10 labeled samples every 100 examples by sweeping more often over the labeled data. This procedure was repeated at each layer during training. We trained 4 layers for 10 epochs with an architecture of 2000-200-100-50-20, denoting 2000 inputs, 200 hidden units at the first layer, 100 at the second, 50 at the third, and 20 at the fourth. Then, we trained a Support Vector Machine[3] (SVM) with a Gaussian kernel on (1) the codes that corresponded to the labeled documents, and we compared the accuracy of the semi-supervised model to the one achieved by a Gaussian SVM trained on the features produced by (2) the same model but trained in an unsupervised way, and by (3) the tf-idf representation of the same labeled documents. The SVM was generally tuned by five-fold cross validation on the available labeled samples (but two-fold cross validation when using only two samples per class). Fig. 3 demonstrates that the learned features gave much better accuracy than the tf-idf representation overall when labeled data was scarce. The model was able to exploit the very few labeled samples producing features that were easier to discriminate. The performance actually improved when the dimensionality of the code was reduced and only 2 or 5 labeled samples per class were available, probably because a more compact code implicitly enforces a stronger regularization. Semi-supervised training outperformed unsupervised training, and the gap widened as we increased the number of labeled samples, indicat-

[3]We used libsvm package available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

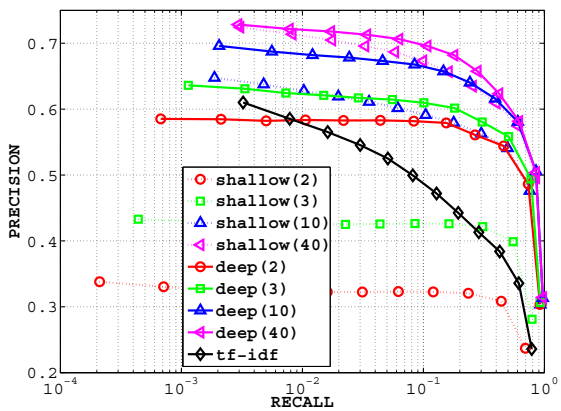

Figure 5. Precision-recall curves for the Reuters dataset comparing one-layer models (shallow) to deep models with the same numbers of code units. The deep models are more accurate overall when the codes are extremely compact. This also suggests that the number of hidden units has to be *gradually* decreased from layer to layer.

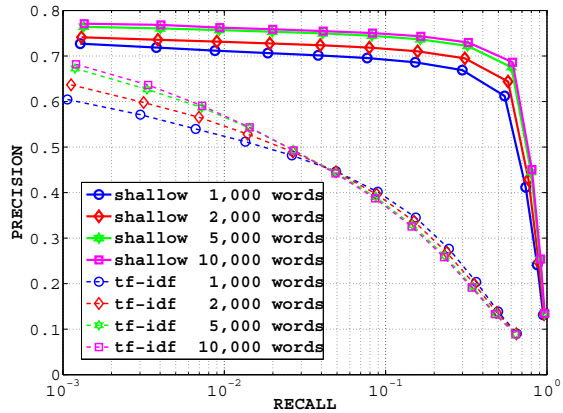

Figure 6. Precision-recall curves for the 20 Newsgroups dataset comparing the performance of tf-idf versus a one-layer shallow model for varying sizes of the word dictionary (from 1000 to 10000 words).

ing that the unsupervised method had failed to model information relevant for classification when compressing to a low-dimensional space.

Interestingly, if we classify the data using the classifier of the feedback module we obtain a performance similar to the one achieved by the Gaussian SVM. For example, when *all* training samples are labeled the classifier at the first stage achieves accuracy of 76.3% (as opposed to 75.5% of the SVM trained either on the learned representation or on tf-idf), while the one on the fourth layer achieves accuracy of 74.8%. Hence, the training algorithm provides an accurate classifier as a side product of the training, reducing the overall learning time.

3.2. Deep or Shallow?

In all the experiments discussed in this section the model was trained using fully labeled data (still, training also includes an unsupervised objective as discussed earlier). In order to retrieve documents after training the model, all documents are mapped into the latent low-dimensional space, the cosine similarity between each document in the test dataset and each document in the training dataset is measured, and the k most similar documents are retrieved. k is chosen to be equal to 1, 3, 7, ..., 4095. Based on the topic label of the documents, we assess the performance by computing the *recall* and the *precision* averaged over the whole test dataset.

In the first experiment, we compared the linear mapping produced by LSI to the nonlinear mapping produced by our model. We considered the Reuters dataset with a 12317 word vocabulary and trained a network with 3 layers. The first layer had 100 code units, the second layer had 40 units in one experiment and 10 in another, the third layer was trained with either 3 or 2 code units. As shown in Fig. 4, the nonlinear representation is more powerful than the linear one, when the representation is very compact.

Another interesting question is whether adding layers is useful. Fig. 5 shows that for a given dimensionality of the output latent space the deep architecture outperforms the shallow one. The deep architecture is capable of capturing more complex dependencies among the input variables than the shallow one, while the representation remains compact. The compactness allows us to efficiently handle very large vocabularies (more than 30,000 words for the Ohsumed, see Sec. 3.4). Fig. 6 shows that increasing the number of words (i.e. the dimensionality of the input) does give better retrieval performance.

3.3. Compact or Binary High-Dimensional?

The most popular representation of documents is tf-idf, a very high-dimensional and sparse representation. One might wonder whether we should learn a high-dimensional representation instead of a compact representation. Unfortunately, the autoencoder based learning algorithm forces us to map data into a lower-dimensional space at each layer, as without additional constraints (Ranzato et al., 2007a) the trivial identity function would be learned. We used the sparse encod-

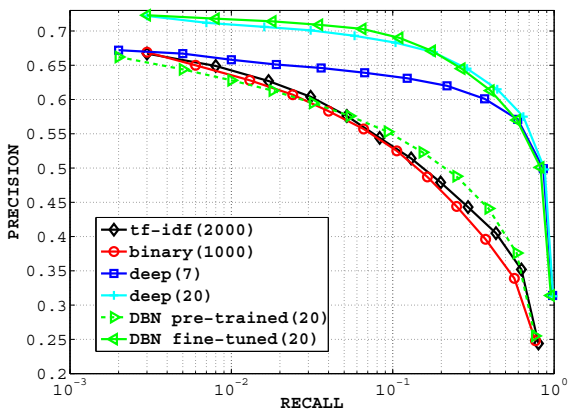

Figure 7. Precision-recall curves comparing compact representations vs. high-dimensional binary representations. Compact representations can achieve better performance using less memory and CPU time.

ing symmetric machine (SESM) (Ranzato et al., 2007b) as a building block for training a deep network producing sparse features. SESM is a symmetric autoencoder with a sparsity constraint on the representation, and it is trained unsupervised. In order to make the sparse representation at the final layer computationally appealing we thresholded it to make it binary. We trained a 2000-1000-1000 SESM network on the Reuters dataset. In order to make a fair comparison with our compact representation, we fixed the information content of the code in terms of precision[4] at $k = 1$. We measured the precision and recall of the binary representation of a test document by computing its Hamming distance from the representation of the training documents. We then trained our model with the following number of units 2000-200-100-7. The last number of units was set to match the precision of the binary representation at $k = 1$. Fig. 7 shows that our compact representation outperforms the high-dimensional and binary representation at higher values of k. Just 7 continuous units are able to achieve better retrieval than 1000 binary units! Storing the Reuters dataset with the compact representation takes less than half the memory space than using the binary representation, and comparing a test document against the whole training dataset is five times faster with the compact representation. The best accuracy for our model is given with a 20-unit representation. Fig. 7 shows the performance of a representation with the same number of units learned by a deep belief network (DBN) following Salakhutdinov and Hinton's constrained Poisson model (2007). Their model was greedily pre-trained for one epoch in an unsupervised way (200 pre-training epochs gave similar fine-tuned accuracy), and then fine-tuned with supervision for 100 epochs. While fine-tuning does not help our model, it significantly improves the DBN which eventually achieves the same accuracy as our model. Despite the similar accuracy, the computational cost of training a DBN (with our implementation using conjugate gradient on mini-batches) is several times higher due to this supervised training through a large and deep network. By looking at how words are mapped to the top-level feature space, we can get an intuition about the learned mapping. For instance, the code closest to the representation of the word "jakarta" corresponds to the word "indonesia", similarly,"meat" is closest to "beef" (table 1). As expected, the model implicitly clusters synonymous and related words.

[4]The entropy of the representation would be more natural, but its value depends on the quantization level.

Table 1. Neighboring word stems for the model trained on Reuters. The number of units is 2000-200-100-7.

Word stem	Neighboring word stems
livestock	beef, meat, pork, cattle
lend	rate, debt, bond, downgrad
acquisit	merger, stake, takeov
port	ship, port, vessel, freight
branch	stake, merger, takeov, acquisit
plantat	coffe, cocoa, rubber, palm
barrel	oil, crude, opec, refineri
subcommitte	bill, trade, bond, committe
coconut	soybean, wheat, corn, grain
meat	beef, pork, cattl, hog
ghana	cocoa, buffer, coffe, icco
varieti	wheat, grain, agricultur, crop
warship	ship, freight, vessel, tanker
edibl	beef, pork, meat, poultri

3.4. Visualization

The deep model can also be used to visualize documents. When the top layer is two-dimensional we can visualize high-dimensional nonlinear manifolds in the space of bags of words. Fig. 8 shows how documents in the Ohsumed test set are mapped to the plane. The model exposes clusters documents according to the topic class, and places similar topics next to each other. The dimensionality reduction is extreme in this case, from more than 30000 to 2.

4. Conclusions

We have proposed and demonstrated a simple and efficient algorithm to learn document representations from

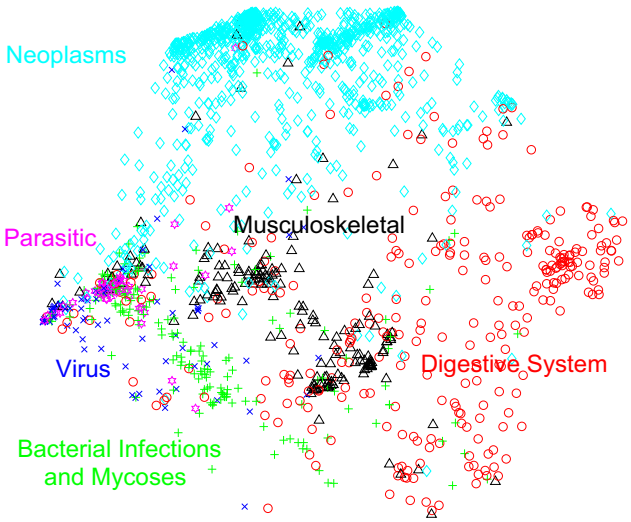

Figure 8. Two-dimensional codes produced by the deep model 30689-100-10-5-2 trained on the Ohsumed dataset (only the 6 most numerous classes are shown). The codes results from propagating documents in the test set through the four-layer network.

partially labeled datasets. The representation is rich in that it can model complex dependencies between words, which allows us to capture higher-level semantic aspects of documents than is possible with linear models. Capturing such complex structure would not be possible based on labeled data alone; by leveraging unlabeled documents we get access to a much larger amount data.

This algorithm trains faster than a similar model based on RBMs, and it finds more efficient representations than a network trained with SESMs that produce high-dimensional binary features. We have shown that these deep models greatly outperform similar but shallow models when the learning task is very hard, such as learning very compact representations. Compact representations are very important for search engines because they are cheap to store, and fast to compute and to compare. Also, we have shown that even a few labels can be exploited to make the features more discriminative.

For future work, we are interested in applying the representation for clustering and ranking. It would also be interesting to go beyond the bag of words model to capture word proximity.

Acknowledgments

The authors would like to thank Y. LeCun for his insights and suggestions.

References

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2006). Greedy layer-wise training of deep networks. In *NIPS*.

Bengio, Y. and LeCun, Y. (2007). *Scaling learning algorithms towards AI*. MIT press.

Blei, D., Ng, A. Y., and Jordan, M. I. (2003). Latent Dirichlet allocation. *JMLR*.

Deerwester, S., Dumais, S., Landauer, T., Furnas, G., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journ. of American Society of Information Science*, 41:391–407.

Gehler, P. V., Holub, A. D., and Welling, M. (2006). The rate adapting Poisson model for information retrieval and object recognition. In *ICML*.

Hinton, G. (2000). Training products of experts by minimizing contrastive divergence. Technical report, U. Toronto.

Hinton, G., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554.

Hinton, G. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Hofmann, T. (1999). Probabilisitc latent semantic analysis. In *Proc. of Uncertainty in Artificial Intelligence*.

LeCun, Y., Bottou, L., Orr, G., and Muller, K. (1998). Efficient backprop. In Orr, G. and K., M., editors, *Neural Networks: Tricks of the trade*. Springer.

Ranzato, M., Boureau, Y., Chopra, S., and LeCun, Y. (2007a). A unified energy-based framework for unsupervised learning. In *AI-STATS*.

Ranzato, M., Boureau, Y., and LeCun, Y. (2007b). Sparse feature learning for deep belief networks. In *NIPS*. MIT Press.

Robertson, S. and Walker, S. (1994). Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In *Proc. ACM SIGIR*, pages 232–241.

Salakhutdinov, R. and Hinton, G. (2007a). Semantic hashing. In *ACM SIGIR workshop on Information Retrieval and Applications of Graphical Models*.

Salakhutdinov, R. and Hinton, G. (2007b). Using deep belief nets to learn covariance kernels for gaussian processes. In *NIPS 20*. MIT Press.