Semantic Hashing

Ruslan Salakhutdinov
Department of Computer Science
University of Toronto
Toronto, Ontario M5S 3G4
rsalakhu@cs.toronto.edu

Geoffrey Hinton
Department of Computer Science
University of Toronto
Toronto, Ontario M5S 3G4
hinton@cs.toronto.edu

ABSTRACT

We show how to learn a deep graphical model of the word-count vectors obtained from a large set of documents. The values of the latent variables in the deepest layer are easy to infer and give a much better representation of each document than Latent Semantic Analysis. When the deepest layer is forced to use a small number of binary variables (e.g. 32), the graphical model performs "semantic hashing": Documents are mapped to memory addresses in such a way that semantically similar documents are located at nearby addresses. Documents similar to a query document can then be found by simply accessing all the addresses that differ by only a few bits from the address of the query document. This way of extending the efficiency of hash-coding to approximate matching is much faster than locality sensitive hashing, which is the fastest current method. By using semantic hashing to filter the documents given to TF-IDF, we achieve higher accuracy than applying TF-IDF to the entire document set.

1. INTRODUCTION

One of the most popular and widely used algorithms for retrieving documents that are similar to a query document is TF-IDF[15, 14] which measures the similarity between documents by comparing their word-count vectors. The similarity metric weights each word by both its frequency in the query document (Term Frequency) and the logarithm of the reciprocal of its frequency in the whole set of documents (Inverse Document Frequency). TF-IDF has several major drawbacks:

- It computes document similarity directly in the word-count space, which can be slow for large vocabularies.
- It assumes that the counts of different words provide independent evidence of similarity.
- It makes no use of semantic similarities between words so it cannot see the similarity between "Wolfowitz resigns" and "Gonzales quits".

To remedy these drawbacks, numerous models for capturing low-dimensional, latent representations have been proposed and successfully applied in the domain of information retrieval. A simple and widely-used method is Latent Semantic Analysis (LSA) [5], which extracts low-dimensional semantic structure using SVD decomposition to get a low-rank approximation of the word-document co-occurrence matrix. This allows document retrieval to be based on "semantic" content rather than just on individually weighted words. LSA, however, is very restricted in the types of semantic content it can capture because it is a linear method so it can only capture pairwise correlations between words. A probabilistic version of LSA (pLSA) was introduced by [11], using the assumption that each word is modeled as a sample from a document-specific multinomial mixture of word distributions. A proper generative model at the level of documents, Latent Dirichlet Allocation, was introduced by [2], improving upon [11].

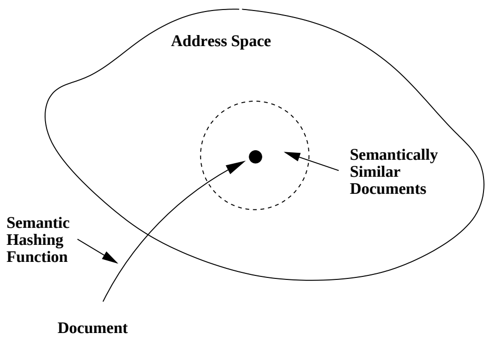


Figure 1: A schematic representation of semantic hashing.

These recently introduced probabilistic models can be viewed as graphical models in which hidden topic variables have directed connections to variables that represent word-counts. Their major drawback is that exact inference is intractable due to explaining away, so they have to resort to slow or inaccurate approximations to compute the posterior distribution over topics. This makes it difficult to fit the models to data. Also, as Welling *et. al.* [16] point out, fast inference is important for information retrieval. To achieve this [16] introduce a class of two-layer undirected graphical models that generalize Restricted Boltzmann Machines (RBM's)[7] to exponential family distributions. This allows them to model non-binary data and to use non-binary hidden (*i.e.* latent) variables. Maximum likelihood learning is intractable in these models, but learning can be performed efficiently by following an approximation to the gradient of a different objective function called "contrastive divergence" [7]. Several further developments of these undirected models [6, 17] show that they are competitive in terms of retrieval accuracy with their directed counterparts.

All of the above models, however, have important limitations.

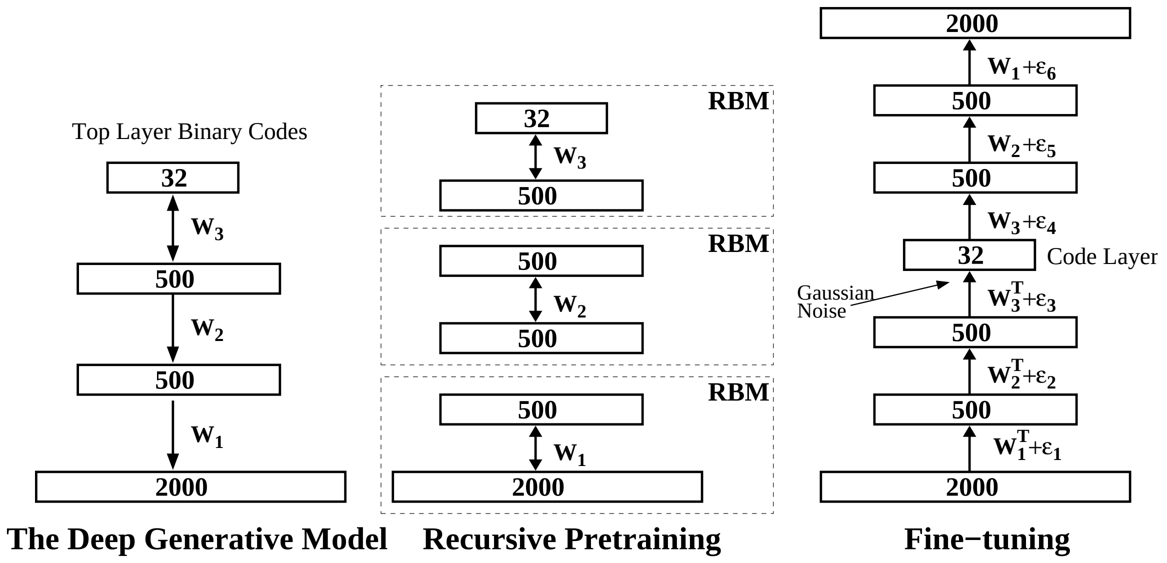


Figure 2: Left panel: The deep generative model. Middle panel: Pretraining consists of learning a stack of RBM's in which the feature activations of one RBM are treated as data by the next RBM. Right panel: After pretraining, the RBM's are "unrolled" to create a multi-layer autoencoder that is fine-tuned by backpropagation.

First, there are limitations on the types of structure that can be represented efficiently by a single layer of hidden variables. We will show that a network with multiple hidden layers and with millions of parameters can discover latent representations that work much better for information retrieval. Second, all of these text retrieval algorithms are based on computing a similarity measure between a query document and other documents in the collection. The similarity is computed either directly in the word space or in a low-dimensional latent space. If this is done naively, the retrieval time complexity of these models is $\mathcal{O}(NV)$, where N is the size of the document corpus and V is the size of vocabulary or dimensionality of the latent space. By using an inverted index, the time complexity for TF-IDF can be improved to $\mathcal{O}(BV)$, where B is the average, over all terms in the query document, of the number of other documents in which the term appears. For LSA, the time complexity can be improved to $\mathcal{O}(V \log N)$ by using special data structures such as KD-trees, provided the intrinsic dimensionality of the representations is low enough for KD-trees to be effective. For all of these models, however, the larger the size of document collection, the longer it will take to search for relevant documents.

In this paper we describe a new retrieval method called "semantic hashing" that produces a shortlist of similar documents in a time that is *independent* of the size of the document collection and linear in the size of the shortlist. Moreover, only a few machine instructions are required per document in the shortlist. Our method must store additional information about every document in the collection, but this additional information is only about one word of memory per document. Our method depends on a new way of training deep graphical models one layer at a time, so we start by describing the type of graphical model we use and how we train it.

The lowest layer in our graphical model represents the word-count vector of a document and the highest (*i.e.* deepest) layer represents a learned binary code for that document. The top two layers of the generative model form an undirected bipartite graph and the remaining layers form a belief net with directed, top-down connections (see fig. 2). The model can be trained efficiently by using a Restriced Boltzmann Machine (RBM) to learn one layer of hidden variables at a time [8]. After learning is complete, the mapping from a word-count vector to the states of the top-level variables is fast, requiring only a matrix multiplication followed by a componentwise non-linearity for each hidden layer.

After the greedy, layer-by-layer training, the generative model is not significantly better than a model with only one hidden layer. To take full advantage of the multiple hidden layers, the layer-by-layer learning must be treated as a "pretraining" stage that finds a good region of the parameter space. Starting in this region, a gradient search can then fine-tune the model parameters to produce a much better model [10].

In the next section we introduce the "Constrained Poisson Model" that is used for modeling word-count vectors. This model can be viewed as a variant of the Rate Adaptive Poisson model [6] that is easier to train and has a better way of dealing with documents of different lengths. In section 3, we describe both the layer-by-layer pretraining and the fine-tuning of the deep multi-layer model. We also show how "deterministic noise" can be used to force the fine-tuning to discover binary codes in the top layer. In section 4, we describe two different ways of using binary codes for retrieval. For relatively small codes we use semantic hashing and for larger binary codes we simply compare the code of the query document to the codes of all candidate documents. This is still very fast because it can use bit operations. We present experimental results showing that both methods work very well on a collection of about a million documents as well as on a smaller collection.

2. THE CONSTRAINED POISSON MODEL

We use a conditional "constrained" Poisson distribution for modeling observed "visible" word count data $\mathbf{v}$ and a conditional Bernoulli distribution for modeling "hidden" topic features $\mathbf{h}$:

$$p(v_i = n|\mathbf{h}) = \text{Ps}\left(n, \frac{\exp\left(\lambda_i + \sum_j h_j w_{ij}\right)}{\sum_k \exp\left(\lambda_k + \sum_j h_j w_{kj}\right)} N\right) \quad (1)$$

$$p(h_j = 1|\mathbf{v}) = \sigma(b_j + \sum_i w_{ij} v_i) \quad (2)$$

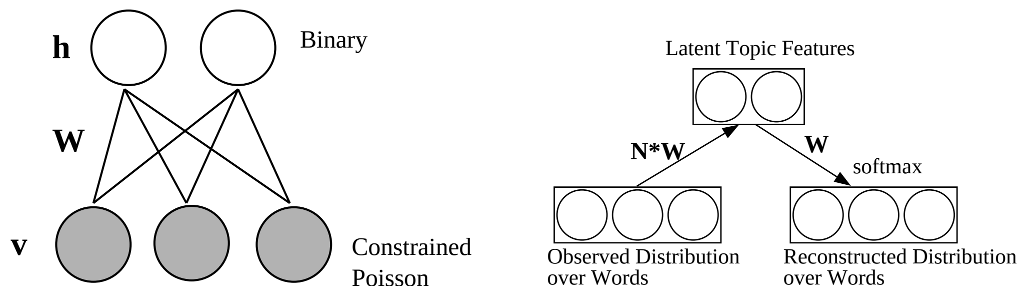

h
Binary
W
v
Constrained
Poisson
Latent Topic Features
N*W
W
softmax
Observed Distribution over Words
Reconstructed Distribution over Words

Figure 3: The left panel shows the Markov random field of the constrained Poisson model. The top layer represents a vector, $\mathbf{h}$, of stochastic, binary, latent, topic features and and the bottom layer represents a Poisson visible vector $\mathbf{v}$. The right panel shows a different interpretation of the constrained Poisson model in which the visible activities have all been divided by the number of words in the document so that they represent a probability distribution. The factor of N that multiplies the upgoing weights is a result of having N *i.i.d.* observations from the observed distribution.

where $\text{Ps}(n, \lambda) = e^{-\lambda}\lambda^n/n!$, $\sigma(x) = 1/(1+e^{-x})$, w_{ij} is a symmetric interaction term between word i and feature j, $N = \sum_i v_i$ is the total length of the document, λ_i is the bias of the conditional Poisson model for word i, and b_j is the bias of feature j. The Poisson rate, whose log is shifted by the weighted combination of the feature activations, is normalized and scaled up by N. We call this the "Constrained Poisson Model" (see fig. 3) since it ensures that the mean Poisson rates across all words sum up to the length of the document. This normalization is significant because it makes learning stable and it deals appropriately with documents of different lengths.

The marginal distribution over visible count vectors v is:

$$p(\mathbf{v}) = \sum_{\mathbf{h}} \frac{\exp\left(-E(\mathbf{v},\mathbf{h})\right)}{\sum_{\mathbf{u},\mathbf{g}} \exp\left(-E(\mathbf{u},\mathbf{g})\right)} \tag{3}$$

with an "energy" term (i.e. the negative log probability + unknown constant offset) given by:

$$\begin{aligned} E(\mathbf{v},\mathbf{h}) = &-\sum_i \lambda_i v_i + \sum_i \log\left(v_i!\right) \\ &-\sum_j b_j h_j - \sum_{i,j} v_i h_j w_{ij} \end{aligned} \tag{4}$$

The parameter updates required to perform gradient ascent in the log-likelihood can be obtained from Eq. 3:

$$\Delta w_{ij} = \epsilon \frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \epsilon(<v_i h_j>_{data} - <v_i h_j>_{model})$$

where ϵ is the learning rate, $<\cdot>_{data}$ denotes an expectation with respect to the data distribution and $<\cdot>_{model}$ is an expectation with respect to the distribution defined by the model. To avoid computing $<\cdot>_{model}$, we use 1-step Contrastive Divergence [7]:

$$\Delta w_{ij} = \epsilon(<v_i h_j>_{data} - <v_i h_j>_{recon}) \tag{5}$$

The expectation $<v_i h_j>_{data}$ defines the frequency with which word i and feature j are on together when the features are being driven by the observed count data from the training set using Eq. 2. After stochastically activating the features, Eq. 1 is used to "reconstruct" the Poisson rates for each word. Then Eq. 2 is used again to activate the features and $<v_i h_j>_{recon}$ is the corresponding expectation when the features are being driven by the reconstructed counts. The learning rule for the biases is just a simplified version of Eq. 5.

3. PRETRAINING AND FINE-TUNING A DEEP GENERATIVE MODEL

A single layer of binary features is not the best way to capture the structure in the count data. We now describe an efficient way to learn additional layers of binary features.

After learning the first layer of hidden features we have an undirected model that defines $p(\mathbf{v},\mathbf{h})$ via the energy function in Eq. 4. We can also think of the model as defining $p(\mathbf{v},\mathbf{h})$ by defining a consistent pair of conditional probabilities, $p(\mathbf{h}|\mathbf{v})$ and $p(\mathbf{v}|\mathbf{h})$ which can be used to sample from the model distribution. A different way to express what has been learned is $p(\mathbf{v}|\mathbf{h})$ and $p(\mathbf{h})$. Unlike a standard directed model, this $p(\mathbf{h})$ does not have its own separate parameters. It is a complicated, non-factorial prior on $\mathbf{h}$ that is defined implicitly by the weights. This peculiar decomposition into $p(\mathbf{h})$ and $p(\mathbf{v}|\mathbf{h})$ suggests a recursive algorithm: keep the learned $p(\mathbf{v}|\mathbf{h})$ but replace $p(\mathbf{h})$ by a better prior over $\mathbf{h}$, *i.e.* a prior that is closer to the average, over all the data vectors, of the conditional posterior over $\mathbf{h}$.

We can sample from this average conditional posterior by simply applying $p(\mathbf{h}|\mathbf{v})$ to the training data. The sampled $\mathbf{h}$ vectors are then the "data" that is used for training a higher-level RBM that learns the next layer of features. We could initialize the higher-level RBM model by using the same parameters as the lower-level RBM but with the roles of the hidden and visible units reversed. This ensures that $p(\mathbf{v})$ for the higher-level RBM starts out being exactly the same as $p(\mathbf{h})$ for the lower-level one. Provided the number of features per layer does not decrease, [8] show that each extra layer increases a variational lower bound on the log probability of the data. This bound is described in more detail in the appendix.

The directed connections from the first hidden layer to the visible units in the final, composite graphical model (see figure 1) are a consequence of the the fact that we keep the $p(\mathbf{v}|\mathbf{h})$ but throw away the $p(\mathbf{h})$ defined by the first level RBM. In the final composite model, the only undirected connections are between the top two layers, because we do not throw away the $p(\mathbf{h})$ for the highest-level RBM.

The first layer of hidden features is learned using a constrained Poisson RBM in which the visible units represent word-counts and the hidden units are binary. All the higher-level RBM's use binary units for both their hidden and their visible layers. The update rules for each layer are then:

$$p(h_j = 1|\mathbf{v}) = \sigma(b_j + \sum_i w_{ij} v_i) \tag{6}$$

$$p(v_i = 1|\mathbf{h}) = \sigma(b_i + \sum_j w_{ij} h_j) \tag{7}$$

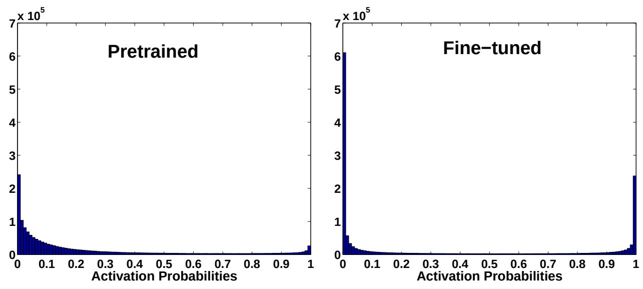


Figure 4: **The distribution of the activities of the 128 code units on the 20 Newsgroups training data before and after fine-tuning with backpropagation and deterministic noise.**

The learning rule provided in the previous section remains the same [7]. This greedy, layer-by-layer training can be repeated several times to learn a deep, hierarchical model in which each layer of features captures strong high-order correlations between the activities of features in the layer below.

To suppress noise in the learning signal, we use the real-valued activation *probabilities* for the visible units of all the higher-level RBM's, but to prevent hidden units from transmitting more than one bit of information from the data to its reconstruction, the pretraining always uses stochastic binary values for the hidden units.

The variational bound does not apply if the layers get smaller, as they do in an autoencoder, but, as we shall see, the pretraining algorithm still works very well as a way to initialize a subsequent stage of fine-tuning. The pretraining finds a point that lies in a good region of parameter space and the myopic fine-tuning then performs a local gradient search that finds a nearby point that is considerably better.

Recursive Greedy Learning of the Deep Generative Model:

1. Learn the parameters $\theta^1 = (W^1, \lambda^1, b^1)$ of the Constrained Poisson Model.
2. Freeze the parameters of the Constrained Poisson model and use the activation probabilities of the binary features, when they are being driven by training data, as the data for training the next layer of binary features.
3. Freeze the parameters θ^2 that define the 2^{nd} layer of features and use the activation probabilities of those features as data for training the 3^{rd} layer of binary features.
4. Proceed recursively for as many layers as desired.

3.1 Fine-tuning the weights

After pretraining, the individual RBM's at each level are "unrolled" as shown in figure 2 to create a deep autoencoder. If the stochastic activities of the binary features are replaced by deterministic, real-valued probabilities, we can then backpropagate through the entire network to fine-tune the weights for optimal reconstruction of the count data. For the fine tuning, we divide the count vector by the number of words so that it represents a probability distribution across words. Then we use the cross-entropy error function with a "softmax" at the output layer. The fine-tuning makes the codes in the central layer of the autoencoder work much better for information retrieval.

3.2 Making the codes binary

During the fine-tuning, we want backpropagation to find codes that are good at reconstructing the count data but are as close to binary as possible. To make the codes binary, we add Gaussian noise to the bottom-up input received by each code unit[1]. Assuming that the decoder network is insensitive to very small differences in the output of a code unit, the best way to communicate information in the presence of added noise is to make the bottom-up input received by a code unit be large and negative for some training cases and large and positive for others. Figure 4 shows that this is what the fine-tuning does.

To prevent the added Gaussian noise from messing up the conjugate gradient fine-tuning, we used "deterministic noise" with mean zero and variance 16. For each training case, the sampled noise values are fixed in advance and do not change during training. With a limited number of training cases, the optimization could tailor the parameters to the fixed noise values, but this is not possible when the total number of sampled noise values is much larger than the number of parameters.

3.3 Details of the training

To speed-up the pretraining, we subdivided both datasets into small mini-batches, each containing 100 cases[2], and updated the weights after each mini-batch. For both datasets each layer was greedily pretrained for 50 passes (epochs) through the entire training dataset. The weights were updated using a learning rate of 0.1, momentum of 0.9, and a weight decay of $0.0002 \times$weight$\times$learning rate. The weights were initialized with small random values sampled from a zero-mean normal distribution with variance 0.01.

For fine-tuning we used the method of conjugate gradients[3] on larger mini-batches of 1000 data vectors, with three line searches performed for each mini-batch in each epoch. To determine an adequate number of epochs and to avoid overfitting, we fine-tuned on a fraction of the training data and tested performance on the remaining validation data. We then repeated fine-tuning on the entire training dataset for 50 epochs. Slight overfitting was observed on the 20 Newsgroups corpus but not on the Reuters corpus. After fine-tuning, the codes were thresholded to produce binary code vectors. The asymmetry between 0 and 1 in the energy function of an RBM causes the unthresholded codes to have many more values near 0 than near 1, so we used a threshold of 0.1.

We experimented with various values for the noise variance and the threshold, as well as the learning rate, momentum, and weight-decay parameters used in the pretraining. Our results are fairly robust to variations in these parameters and also to variations in the number of layers and the number of units in each layer. The precise weights found by the pretraining do not matter as long as it finds a good region of the parameter space from which to start the fine-tuning.

4. EXPERIMENTAL RESULTS

To evaluate performance of our model on an information retrieval task we use Precision-Recall curves where we define:

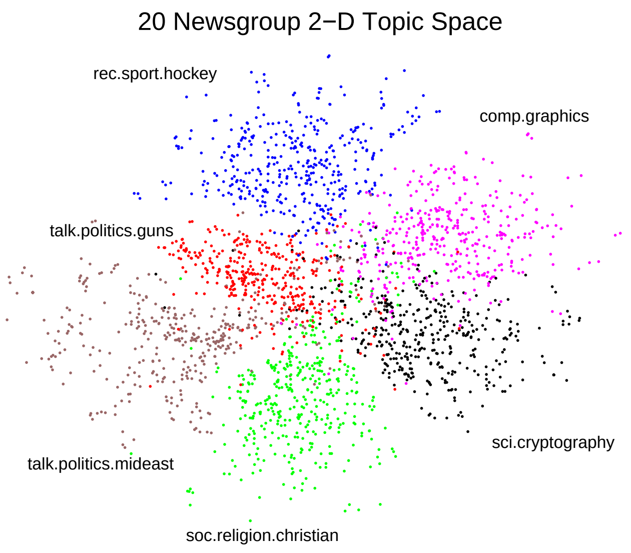
$$\begin{aligned} \text{Recall} &= \frac{\text{Number of retrieved relevant documents}}{\text{Total number of all relevant documents}} \\ \text{Precision} &= \frac{\text{Number of relevant retrieved documents}}{\text{Total number of retrieved documents}} \end{aligned}$$

To decide whether a retrieved document is relevant to the query document, we simply look to see if they have the same class label.

[1]We tried other ways of encouraging the code units to be binary, such as penalizing the entropy of $-p \log p - (1-p) \log{(1-p)}$ for each code unit, but Gaussian noise worked better.

[2]The last mini-batch contained more than 100 cases.

[3]Code is available at
http://www.kyb.tuebingen.mpg.de/bs/people/carl/code/minimize/



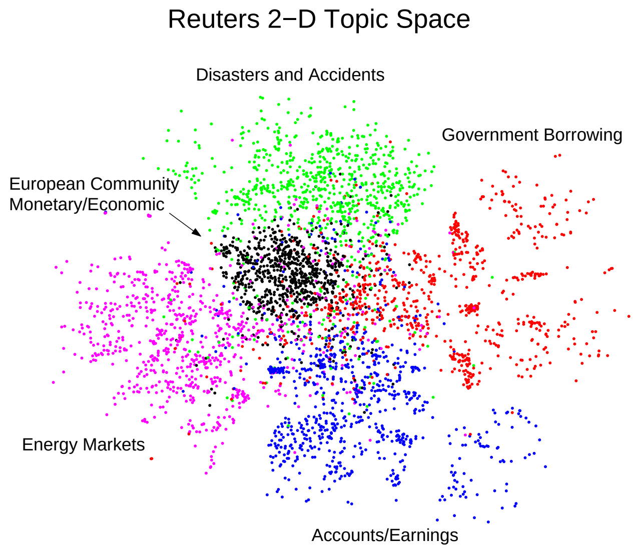


Figure 5: A 2-dimensional embedding of the 128-bit codes using stochastic neighbor embedding for the 20 Newsgroups data (left panel) and the Reuters RCV2 corpus (right panel). See in color for better visualization.

This is the only time that the class labels are used. It is not a particularly good measure of relevance, but it is the same for all the methods we compare.

Results of [6] show that pLSA and LDA models do not generally outperform LSA and TF-IDF. Therefore for comparison we only used LSA and TF-IDF as benchmark methods. For LSA each word count, c_i, was replaced by $\log(1 + c_i)$ before the SVD decomposition, which slightly improved performance. For both these methods we used the cosine of the angle between two vectors as a measure of their similarity.

4.1 Description of the Text Corpora

In this section we present experimental results for document retrieval on two text datasets: 20-Newsgroups and Reuters Corpus Volume II.

The 20 newsgroups corpus contains 18,845 postings taken from the Usenet newsgroup collection. The corpus is partitioned fairly evenly into 20 different newsgroups, each corresponding to a separate topic.[4] The data was split by date into 11,314 training and 7,531 test articles, so the training and test sets were separated in time. The training set was further randomly split into 8,314 training and 3,000 validation documents. Newsgroups such as soc.religion.christian and talk.religion.misc are very closely related to each other, while newsgroups such as comp.graphics and rec.sport.hockey are very different (see fig. 5)

We further preprocessed the data by removing common stopwords, stemming, and then only considering the 2000 most frequent words in the training dataset. As a result, each posting was represented as a vector containing 2000 word counts. No other preprocessing was done.

The Reuters Corpus Volume II is an archive of 804,414 newswire stories[5] that have been manually categorized into 103 topics. The corpus covers four major groups: corporate/industrial, economics, government/social, and markets. Sample topics are displayed in figure 5. The topic classes form a tree which is typically of depth 3. For this dataset, we define the relevance of one document to another to be the fraction of the topic labels that agree on the two paths from the root to the two documents.

The data was randomly split into 402,207 training and 402,207 test articles. The training set was further randomly split into 302,207 training and 100,000 validation documents. The available data was already in the preprocessed format, where common stopwords were removed and all documents were stemmed. We again only considered the 2000 most frequent words in the training dataset.

4.2 Results using 128-bit codes

For both datasets we used a 2000-500-500-128 architecture which is like the architecture shown in figure 2, but with 128 code units. To see whether the learned 128-bit codes preserve class information, we used stochastic neighbor embedding [9] to visualize the 128-bit codes of all the documents from 5 or 6 separate classes. Figure 5 shows that for both datasets the 128-bit codes preserve the class structure of the documents.

In addition to requiring very little memory, binary codes allow very fast search because fast bit counting routines[6] can be used to compute the Hamming distance between two binary codes. On a 3GHz Intel Xeon running C, for example, it only takes 3.6 milliseconds to search through 1 million documents using 128-bit codes. The same search takes 72 milliseconds for 128-dimensional LSA.

Figures 6 and 7 (left panels) show that our 128-bit codes are better at document retrieval than the 128 real-values produced by LSA. We tried thresholding the 128 real-values produced by LSA to get binary codes. The thresholds were set so that each of the 128 components was a 0 for half of the training set and a 1 for the other half. The results of figure 6 reveal that binarizing LSA significantly reduces its performance. This is hardly surprising since LSA has not been optimized to make the binary codes perform well.

TF-IDF is slightly more accurate than our 128-bit codes when retrieving the top few documents in either dataset. If, however, we

[4]Available at http://people.csail.mit.edu/jrennie/20Newsgroups (20news-bydate.tar.gz). It has been preprocessed and organized by date.

[5]The Reuter Corpus Volume 2 dataset is available at http://trec.nist.gov/data/reuters/reuters.html

[6]Code is available at http://www-db.stanford.edu/∼manku/bitcount/bitcount.html

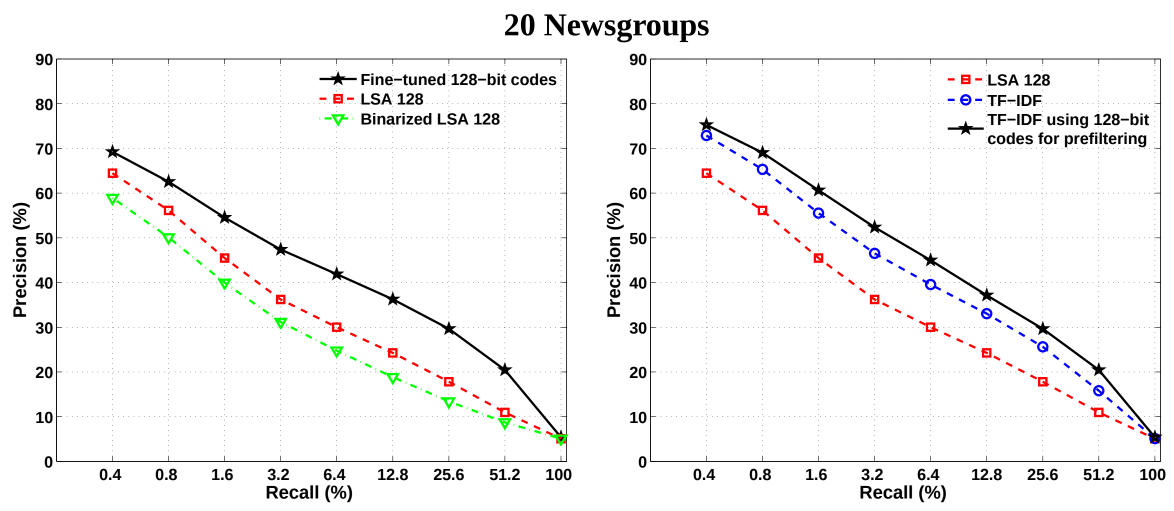


Figure 6: **Precision-Recall curves for the 20 Newsgroups dataset, when a query document from the test set is used to retrieve other test set documents, averaged over all 7,531 possible queries.**

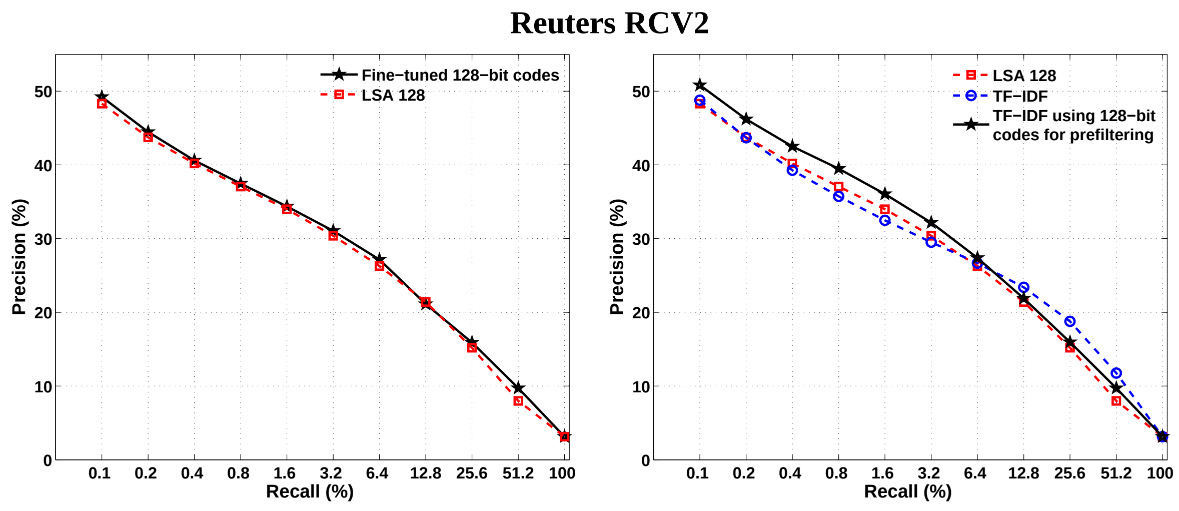


Figure 7: **Precision-Recall curves for the Reuters RCV2 dataset, when a query document from the test set is used to retrieve other test set documents, averaged over all 402,207 possible queries.**

use our 128-bit codes to preselect the top 100 documents for the 20 Newsgroups data or the top 1000 for the Reuters data, and then re-rank these preselected documents using TF-IDF, we get better accuracy than running TF-IDF alone on the whole document set (see figures 6 and 7). This means that some documents which TF-IDF would have considered a very good match to the query document have been correctly eliminated by using the 128-bit codes as a filter.

4.3 Results using 20-bit codes

Using 20-bit codes, we also checked whether our learning procedure could discover a way to model similarity of count-vectors by similarity of 20-bit addresses that was good enough to allow high precision and retrieval for our set of 402,207 Reuters RCV2 test documents. After learning to assign 20-bit addresses to documents using the training data, we compute the 20-bit address of each test document and place a pointer to the document at its address.[7]

For the 402,207 test documents, a 20-bit address space gives a density of about 0.4 documents per address. For a given a query document, we compute its 20-bit address and then retrieve all of the documents stored in a hamming ball of radius 4 (about $6196 \times 0.402207 \approxeq 2500$ documents) without performing any search at all. Figure 8 shows that neither precision nor recall is lost by restricting TF-IDF to this fixed, preselected set.

Using a simple implementation of Semantic Hashing in C, it takes about 0.5 milliseconds to create the short-list of about 2500 semantically similar documents and about 10 milleseconds to retrieve the top few matches from that short-list using TF-IDF. Locality-Sensitive Hashing (LSH) [4, 1] takes about 500 milleseconds to perform the same search using E^{2}LSH 0.1 software, provided by Alexandr Andoni and Piotr Indyk. Also, locality sensitive hashing is an approximation to nearest-neighbor matching in the word-count space, so it cannot be expected to perform better than TF-IDF and it generally performs slightly worse. Figure 8 shows that using semantic hashing as a filter, in addition to being much faster, achieves higher accuracy than either LSH or TF-IDF applied to the whole document set.

[7]We actually start with a pointer to *null* at all addresses and then replace it by a one-dimensional array that contains pointers to all the documents that have that address.

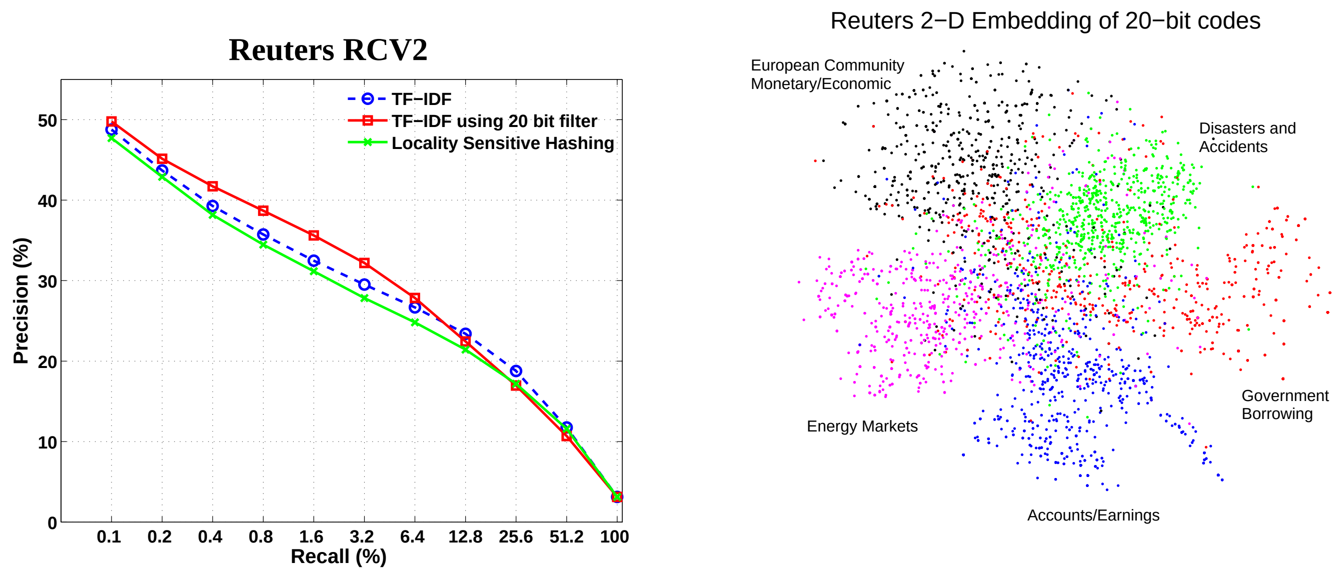


Figure 8: **Left panel: Precision-Recall curves for the Reuters RCV2 dataset, when a query document from the test set is used to retrieve other test set documents, averaged over all 402,207 possible queries. Right panel: 2-dimensional embedding of the 20-bit codes using stochastic neighbor embedding for the Reuters RCV2 corpus. See in color for better visualization.**

5. SEMANTIC HASHING FOR VERY LARGE DOCUMENT COLLECTIONS

For a billion documents, a 30-bit address space gives a density of about 1 document per address and semantic hashing only requires a few Gigabytes of memory. Using a hamming-ball of radius 5 around the address of a query document, we can create a long "shortlist" of about 175,000 similar documents with no search. There is no point creating a new data-structure for each shortlist. When items from the shortlist are required we can simply produce them by enumerating the addresses in the hamming ball. So, assuming we already have the 30-bit code for the query document, the time required to create this long shortlist is zero, which compares favorably with other methods. The items in the long shortlist could then be further filtered using, say, 128-bit binary codes produced by a deep belief net. This second level of filtering would only take a fraction of a millesecond. It requires additional storage of two 64-bit words of memory for every document in the collection, but this is only about twice the space already required for semantic hashing.

Scaling up the learning to a billion training cases would not be particularly difficult. Using mini-batches, the learning time is slightly sublinear in the size of the dataset if there is redundancy in the data, and different cores can be used to compute the gradients for different examples within a large mini-batch. So training on a billion documents would take at most a few weeks on 100 cores and a large organization could train on many billions of documents.

Unlike almost all other machine learning applications, overfitting need not be an issue because there is no need to generalize to new data. If the learning is ongoing, the deep belief net can be trained on all of the documents in the collection which should significantly improve the results we obtained when training the deep belief net on one half of a document collection and then testing on the other half.

Many elaborations are possible. We could learn several different semantic hashing functions on disjoint training sets and then preselect documents that are close to the query document in any of the semantic address spaces. This would ameliorate one of the weaknesses of semantic hashing: Documents with similar addresses have similar content but the converse is not necessarily true. Given the way we learn the deep belief net, it is quite possible for the semantic address space to contain widely separated regions that are semantically very similar. The seriousness of this problem for very large document collections remains to be determined, but it did not prevent semantic hashing from having good recall in our experiments.

There is a simple way to discourage the deep belief net from assigning very different codes to similar documents. We simply add an extra penalty term during the optimization that pulls the codes for similar documents towards each other. This attractive force can easily be backpropagated through the deep belief net. Any available information about the relevance of two documents can be used for this penalty term, including class labels if they are available. This resembles non-linear Neighborhood Components Analysis (NCA) [13, 3], but scales much better to very large datasets because the derivatives produced by the autoencoder prevent the codes from all becoming identical, so there is no need for the quadratically expensive normalization term used in NCA.

6. AN ALTERNATIVE VIEW OF SEMANTIC HASHING

Fast retrieval methods often rely on intersecting sets of documents that contain a particular word or feature. Semantic hashing is no exception. Each of the binary values in the code assigned to a document represents a set containing about half the entire document collection. Intersecting such sets would be slow if they were represented by explicit lists, but all computers come with a special piece of hardware – the address bus – that can intersect sets in a single machine instruction. Semantic hashing is simply a way of mapping the set intersections required for document retrieval directly onto the available hardware.

7. CONCLUSION

In this paper we described a two-stage learning procedure for finding binary codes that can be used for fast document retrieval. During the *pretraining* stage, we greedily learn a deep generative model in which the lowest layer represents the word-count vector of a document and the highest layer represents a learned binary code for that document. During the *fine-tuning* stage, the model is "unfolded" to produce a deep autoencoder network and backpropagation is used to fine-tune the weights for optimal reconstruction. By adding noise at the code layer, we force backpropagation to use codes that are close to binary.

By treating the learned binary codes as memory addresses, we can find semantically similar documents in a time that is independent of the size of the document collection – something that no other retrieval method can achieve. Using the Reuters RCV2 dataset, we showed that by using semantic hashing as a filter for TF-IDF, we achieve higher precision and recall than TF-IDF or Locality Sensitive Hashing applied to the whole document collection in a very small fraction of the time taken by Locality-Sensitive Hashing, which is the fastest current method.

Acknowledgments

We thank Yoram Singer, John Lafferty, Sam Roweis, Vinod Nair, and two anonymous reviewers for helpful comments. This research was supported by NSERC, CFI and OTI. GEH is a fellow of CIAR and holds a CRC chair.

8. REFERENCES

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, pages 459–468. IEEE Computer Society, 2006.

[2] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[3] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *IEEE Computer Vision and Pattern Recognition or CVPR*, pages I: 539–546, 2005.

[4] Datar, Immorlica, Indyk, and Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *COMPGEOM: Annual ACM Symposium on Computational Geometry*, 2004.

[5] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[6] P. Gehler, A. Holub, and M. Welling. The rate adapting poisson (RAP) model for information retrieval and object recognition. In *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

[7] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1711–1800, 2002.

[8] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[9] G. E. Hinton and S. T. Roweis. Stochastic neighbor embedding. In *Advances in Neural Information Processing Systems*, pages 833–840. MIT Press, 2002.

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

[11] T. Hofmann. Probabilistic latent semantic analysis. In *Proceedings of the 15th Conference on Uncertainty in AI*, pages 289–296, San Fransisco, California, 1999. Morgan Kaufmann.

[12] R. M. Neal and G. E. Hinton. A view of the EM algorithm that justifies incremental, sparse and other variants. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 355–368. Kluwer Academic Press, 1998.

[13] R. Salakhutdinov and G. E. Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *AI and Statistics*, 2007.

[14] Salton. Developments in automatic text retrieval. *Science*, 253, 1991.

[15] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[16] M. Welling, M. Rosen-Zvi, and G. Hinton. Exponential family harmoniums with an application to information retrieval. In *Advances in Neural Information Processing Systems 4*, pages 1481–1488, Cambridge, MA, 2005. MIT Press.

[17] E. Xing, R. Yan, and A. G. Hauptmann. Mining associated text and images with dual-wing harmoniums. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI-2005)*, 2005.

9. APPENDIX: THE VARIATIONAL BOUND FOR THE GREEDY LEARNING

Consider a restricted Boltzmann machine with parameters W that determine $p(\mathbf{v}|\mathbf{h})$ and $p(\mathbf{h}|\mathbf{v})$, where $\mathbf{v}$ is a visible vector and $\mathbf{h}$ is a hidden vector.

It is shown in [12] that for any approximating distribution $Q(\mathbf{h}|\mathbf{v})$ we can write:

$$\begin{aligned}\log p(\mathbf{v}) \quad \geq \quad & \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{v})[\log p(\mathbf{h}) + \log p(\mathbf{v}|\mathbf{h})] \\ & - \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{v}) \log Q(\mathbf{h}|\mathbf{v}) \end{aligned} \tag{8}$$

If we set $Q(\mathbf{h}|\mathbf{v})$ to be the true posterior distribution, $p(\mathbf{h}|\mathbf{v})$ given by Eq. 1, the bound becomes tight. By freezing the parameter vector W at the value W_{frozen}, we freeze $p(\mathbf{v}|\mathbf{h})$, and $Q(\mathbf{h}|\mathbf{v}, W^T_{\text{frozen}})$. When $p(\mathbf{h})$ is implicitly defined by W_{frozen}, $Q(\mathbf{h}|\mathbf{v}, W^T_{\text{frozen}})$ is the true posterior, but when this $p(\mathbf{h})$ is replaced by a better distribution that is learned by a higher-level RBM, $Q(\mathbf{h}|\mathbf{v}, W^T_{\text{frozen}})$ is only an approximation to the true posterior. Nevertheless, the loss caused by using an approximate posterior is less than the gain caused by using a better model for $p(\mathbf{h})$, provided this better model is learned by optimizing the variational bound in Eq. 8 and provided the better model is initialized so that $p(\mathbf{v})$ for the better model is equal to $p(\mathbf{h})$ for the first model. Maximizing the bound with W frozen is equivalent to maximizing:

$$\sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{v}, W^T_{\text{frozen}}) \log p(\mathbf{h})$$

or replacing $p(\mathbf{h})$ by a prior that is closer to the average, over all the data vectors, of the conditional posterior $Q(\mathbf{h}|\mathbf{v}, W^T_{\text{frozen}})$. This is exactly what is being learned when samples from $p(\mathbf{h}|\mathbf{v})$ are used as the training data for a higher-level RBM.

In practice, we typically do not bother to initialize each RBM so that its $p(\mathbf{v})$ equals the $p(\mathbf{h})$ for the previous model. Initializing the weights in this way would force the hidden units of the new RBM to be of the same type as the visible units of the lower-level RBM.