Supporting Online Material for

Reducing the Dimensionality of Data with Neural Networks

G. E. Hinton* and R. R. Salakhutdinov

*To whom correspondence should be addressed. E-mail: hinton@cs.toronto.edu

Published 28 July 2006, *Science* **313**, 504 (2006)
DOI: 10.1126/science.1127647

This PDF file includes:

Supporting Online Material

Details of the pretraining: To speed up the pretraining of each RBM, we subdivided all datasets into mini-batches, each containing 100 data vectors and updated the weights after each mini-batch. For datasets that are not divisible by the size of a minibatch, the remaining data vectors were included in the last minibatch. For all datasets, each hidden layer was pretrained for 50 passes through the entire training set. The weights were updated after each mini-batch using the averages in Eq. 1 of the paper with a learning rate of 0.1. In addition, 0.9 times the previous update was added to each weight and 0.00002 times the value of the weight was subtracted to penalize large weights. Weights were initialized with small random values sampled from a normal distribution with zero mean and standard deviation of 0.1. The Matlab code we used is available at http://www.cs.toronto.edu/$\sim$hinton/MatlabForSciencePaper.html

Details of the fine-tuning: For the fine-tuning, we used the method of conjugate gradients on larger minibatches containing 1000 data vectors. We used Carl Rasmussen's "minimize" code (*1*). Three line searches were performed for each mini-batch in each epoch. To determine an adequate number of epochs and to check for overfitting, we fine-tuned each autoencoder on a fraction of the training data and tested its performance on the remainder. We then repeated the fine-tuning on the entire training set. For the synthetic curves and hand-written digits, we used 200 epochs of fine-tuning; for the faces we used 20 epochs and for the documents we used 50 epochs. Slight overfitting was observed for the faces, but there was no overfitting for the other datasets. Overfitting means that towards the end of training, the reconstructions were still improving on the training set but were getting worse on the validation set.

We experimented with various values of the learning rate, momentum, and weight-decay parameters and we also tried training the RBM's for more epochs. We did not observe any significant differences in the final results after the fine-tuning. This suggests that the precise weights found by the greedy pretraining do not matter as long as it finds a good region from which to start the fine-tuning.

How the curves were generated: To generate the synthetic curves we constrained the x coordinate of each point to be at least 2 greater than the x coordinate of the previous point. We also constrained all coordinates to lie in the range $[2, 26]$. The three points define a cubic spline which is "inked" to produce the 28×28 pixel images shown in Fig. 2 in the paper. The details of the inking procedure are described in (*2*) and the matlab code is at (*3*).

Fitting logistic PCA: To fit logistic PCA we used an autoencoder in which the linear code units were directly connected to both the inputs and the logistic output units, and we minimized the cross-entropy error using the method of conjugate gradients.

How pretraining affects fine-tuning in deep and shallow autoencoders: Figure S1 compares performance of pretrained and randomly initialized autoencoders on the curves dataset. Figure S2 compares the performance of deep and shallow autoencoders that have the same number of parameters. For all these comparisons, the weights were initialized with small random values sampled from a normal distribution with mean zero and standard deviation 0.1.

Details of finding codes for the MNIST digits: For the MNIST digits, the original pixel intensities were normalized to lie in the interval $[0, 1]$. They had a preponderance of extreme values and were therefore modeled much better by a logistic than by a Gaussian. The entire training procedure for the MNIST digits was identical to the training procedure for the curves, except that the training set had 60,000 images of which 10,000 were used for validation. Figure S3 and S4 are an alternative way of visualizing the two-dimensional codes produced by PCA and by an autencoder with only two code units. These alternative visualizations show many of the actual digit images. We obtained our results using an autoencoder with 1000 units in the first hidden layer. The fact that this is more than the number of pixels does not cause a problem for the RBM – it does not try to simply copy the pixels as a one-hidden-layer autoencoder would. Subsequent experiments show that if the 1000 is reduced to 500, there is very little change in the performance of the autoencoder.

Details of finding codes for the Olivetti face patches: The Olivetti face dataset from which we obtained the face patches contains ten 64×64 images of each of forty different people. We constructed a dataset of 165,600 25×25 images by rotating (-90° to $+90^\circ$), scaling (1.4 to 1.8), cropping, and subsampling the original 400 images. The intensities in the cropped images were shifted so that every pixel had zero mean and the entire dataset was then scaled by a single number to make the average pixel variance be 1. The dataset was then subdivided into 124,200 training images, which contained the first thirty people, and 41,400 test images, which contained the remaining ten people. The training set was further split into 103,500 training and 20,700 validation images, containing disjoint sets of 25 and 5 people. When pretraining the first layer of 2000 binary features, each real-valued pixel intensity was modeled by a Gaussian distribution with unit variance. Pretraining this first layer of features required a much smaller learning rate to avoid oscillations. The learning rate was set to 0.001 and pretraining proceeded for 200 epochs. We used more feature detectors than pixels because a real-valued pixel intensity contains more information than a binary feature activation. Pretraining of the higher layers was carried out as for all other datasets. The ability of the autoencoder to reconstruct more of the perceptually significant, high-frequency details of faces is not fully reflected in the squared pixel error. This is an example of the well-known inadequacy of squared pixel error for assessing perceptual similarity.

Details of finding codes for the Reuters documents: The 804,414 newswire stories in the Reuters Corpus Volume II have been manually categorized into 103 topics. The corpus covers four major groups: corporate/industrial, economics, government/social, and markets. The labels were not used during either the pretraining or the fine-tuning. The data was randomly split into 402,207 training and 402,207 test stories, and the training set was further randomly split into 302,207 training and 100,000 validation documents. Common stopwords were removed from the documents and the remaining words were stemmed by removing common endings.

During the pretraining, the confabulated activities of the visible units were computed using a "softmax" which is the generalization of a logistic to more than 2 alternatives:

$$\hat{p}_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{1}$$

where $\hat{p}_i$ is the modeled probability of word i and x_i is the weighted input produced by the feature activations plus a bias term. The use of a softmax does not affect the learning rule (*4*), but to allow for the fact that the document contains N observations from the probability distribution over words, the weight w_{ik} from word i to feature k was set to be N times the weight w_{ki} from the feature to the word.

For our comparisons with Latent Semantic Analysis (LSA), we used the version of LSA in which each word count, c_i, is replaced by $\log(1+c_i)$. This standard preprocessing trick slightly improves the retrieval performance of LSA by down-weighting the influence of very frequent words.

A comparison with Local Linear Embedding: It is hard to compare autoencoders with Local Linear Embedding (*5*) on a reconstruction task because, like several other non-parametric dimensionality reduction methods, LLE does not give a simple way of reconstructing a test image. We therefore compared LLE with autoencoders on a document retrieval task. LLE is not a sensible method to use because it involves finding nearest neighbors in the high-dimensional space for each new query document, but document retrieval does, at least, give us a way of assessing how good the low-dimensional codes are. Fitting LLE takes a time that is quadratic in the number of training cases, so we used the "20 newsgroup" corpus that has only 11,314 training and 7,531 test documents. The "documents" are postings taken from the Usenet newsgroup collection. The corpus is partitioned fairly evenly into 20 different newsgroups, each corresponding to a separate topic. The data was preprocessed, organized by date, and is available at (*6*). The data was split by date, so the training and test sets were separated in time. Some newsgroups are very closely related to each other, e.g. soc.religion.christian and talk.religion.misc, while others are very different, e.g. rec.sport.hockey and comp.graphics. We removed common stopwords from the documents and also stemmed the remaining words by removing common endings. As for the Reuters corpus, we only considered the 2000 most frequent words in the training dataset.

LLE must be given the K nearest neighbors of each datapoint and we used the cosines of the angles between count vectors to find the nearest neighbors. The performance of LLE depends on the value chosen for K so we tried $K = 5, 10, 15, 20, 25, 30$ and we always report

the results using the best value of K. We also checked that K was large enough for the whole dataset to form one connected component (for K=5, there was a disconnected component of 21 documents). During the test phase, for each query document q, we identify the K nearest count vectors from the training set and compute the best weights $\mathbf{w}$ for reconstructing the count vector from its neighbors. We then use the same weights $\mathbf{w}$ to generate the low-dimensional code for q from the low-dimensional codes of its high-dimensional K nearest neighbors (*7*). LLE code is available at (*8*).

For 2-dimensional codes, LLE ($K = 10$) performs better than LSA but worse than our autoencoder. For higher dimensional codes, the performance of LLE ($K = 25$) is very similar to the performance of LSA (see Fig. S5) and much worse than the autoencoder. We also tried normalizing the squared lengths of the document count vectors before applying LLE but this did not help.

Using the pretraining and fine-tuning for digit classification: To show that the same pretraining procedure can improve generalization on a classification task, we used the "permutation invariant" version of the MNIST digit recognition task. Before being given to the learning program, all images undergo the same random permutation of the pixels. This prevents the learning program from using prior information about geometry such as affine transformations of the images or local receptive fields with shared weights (*9*). On the permutation invariant task, Support Vector Machines achieve 1.4% (*10*). The best published result for a randomly initialized neural net trained with backpropagation is 1.6% for a 784-800-10 network (*11*). Pretraining reduces overfitting and makes learning faster, so it makes it possible to use a much larger 784-500-500-2000-10 neural network that achieves 1.2%.

We pretrained a 784-500-500-2000 net for 100 epochs on all 60,000 training cases in the just same way as the autoencoders were pretrained, but with 2000 logistic units in the top layer. The pretraining did not use any information about the class labels. We then connected ten "softmaxed" output units to the top layer and fine-tuned the whole network using simple gradient descent in the cross-entropy error with a very gentle learning rate to avoid unduly perturbing the weights found by the pretraining. For all but the last layer, the learning rate was 0.03 for the weights and 0.1 for the biases. To speed learning, 0.8 times the previous weight increment was added to each weight update. For the biases and weights of the 10 output units, there was no danger of destroying information from the pretraining, so their learning rates were five times larger and they also had a penalty which was 5×10^{-5} times their squared magnitude. After 77 epochs of fine-tuning, the average cross-entropy error on the *training* data fell below a pre-specified threshold value and the fine-tuning was stopped. The test error at that point was 1.17%. The threshold value was determined by performing the pretraining and fine-tuning on only 50,000 training cases and using the remaining 10,000 training cases as a validation set to determine the training cross-entropy error that gave the fewest classification errors on the validation set.

We have also fine-tuned the whole network using using the method of conjugate gradients (*1*) on minibatches containing 1000 data vectors. Three line searches were performed for each

mini-batch in each epoch. After 48 epochs of fine-tuning, the test error was 1.14%. The stopping criterion for fine-tuning was determined in the same way as described above. The Matlab code for training such a classifier is available at
http://www.cs.toronto.edu/~hinton/MatlabForSciencePaper.html

Supporting text

How the energies of images determine their probabilities: The probability that the model assigns to a visible vector, $\mathbf{v}$ is

$$p(\mathbf{v}) = \sum_{\mathbf{h} \in \mathcal{H}} p(\mathbf{v}, \mathbf{h}) = \frac{\sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}))}{\sum_{\mathbf{u},\mathbf{g}} \exp(-E(\mathbf{u}, \mathbf{g}))} \tag{2}$$

where $\mathcal{H}$ is the set of all possible binary hidden vectors. To follow the gradient of $\log p(\mathbf{v})$ *w.r.t.* w_{ij} it is necessary to alternate between updating the states of the features and the states of the pixels until the stationary distribution is reached. The expected value of $v_i h_j$ in this stationary distribution is then used instead of the expected value for the one-step confabulations. In addition to being much slower, this maximum likelihood learning procedure suffers from much more sampling noise.

To further reduce noise in the learning signal, the binary states of already learned feature detectors (or pixels) in the "data" layer are replaced by their real-valued probabilities of activation when learning the next layer of feature detectors, but the new feature detectors always have stochastic binary states to limit the amount of information they can convey.

The energy function for real-valued data: When using linear visible units and binary hidden units, the energy function and update rules are particularly simple if the linear units have Gaussian noise with unit variance. If the variances are not 1, the energy function becomes:

$$E(\mathbf{v}, \mathbf{h}) = \sum_{i \in \text{pixels}} \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{j \in \text{features}} b_j h_j - \sum_{i,j} \frac{v_i}{\sigma_i} h_j w_{ij} \tag{3}$$

where σ_i is the standard deviation of the Gaussian noise for visible unit i. The stochastic update rule for the hidden units remains the same except that each v_i is divided by σ_i. The update rule for visible units i is to sample from a Gaussian with mean $b_i + \sigma_i \sum_j h_j w_{ij}$ and variance σ_i^2.

The differing goals of Restricted Boltzmann machines and autoencoders: Unlike an autoencoder, the aim of the pretraining algorithm is not to accurately reconstruct each training image but to make the *distribution* of the confabulations be the same as the distribution of the images. This goal can be satisfied even if an image A that occurs with probability $p(A)$ in the training set is confabulated as image B with probability $p(B|A)$ provided that, for all A, $p(A) = \sum_B p(B)p(A|B)$.

Supporting figures

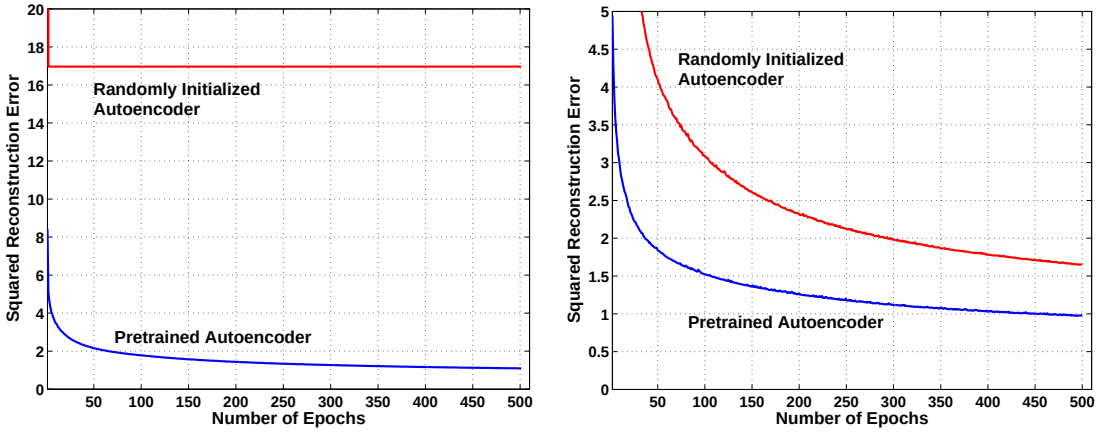

Fig. S1: The average squared reconstruction error per test image during fine-tuning on the curves training data. Left panel: The deep 784-400-200-100-50-25-6 autoencoder makes rapid progress after pretraining but no progress without pretraining. Right panel: A shallow 784-532-6 autoencoder can learn without pretraining but pretraining makes the fine-tuning much faster, and the pretraining takes less time than 10 iterations of fine-tuning.

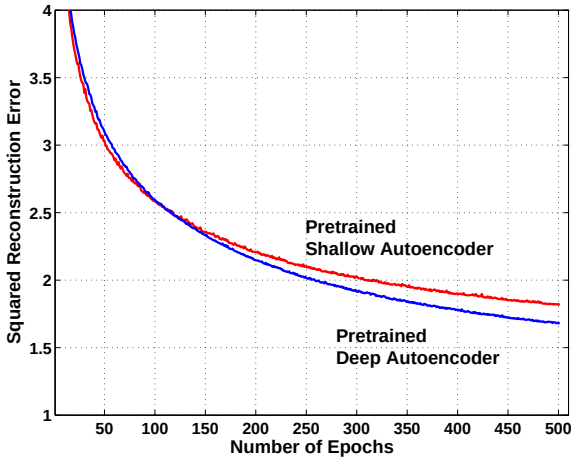

Fig. S2: The average squared reconstruction error per image on the test dataset is shown during the fine-tuning on the curves dataset. A 784-100-50-25-6 autoencoder performs slightly better than a shallower 784-108-6 autoencoder that has about the same number of parameters. Both autoencoders were pretrained.

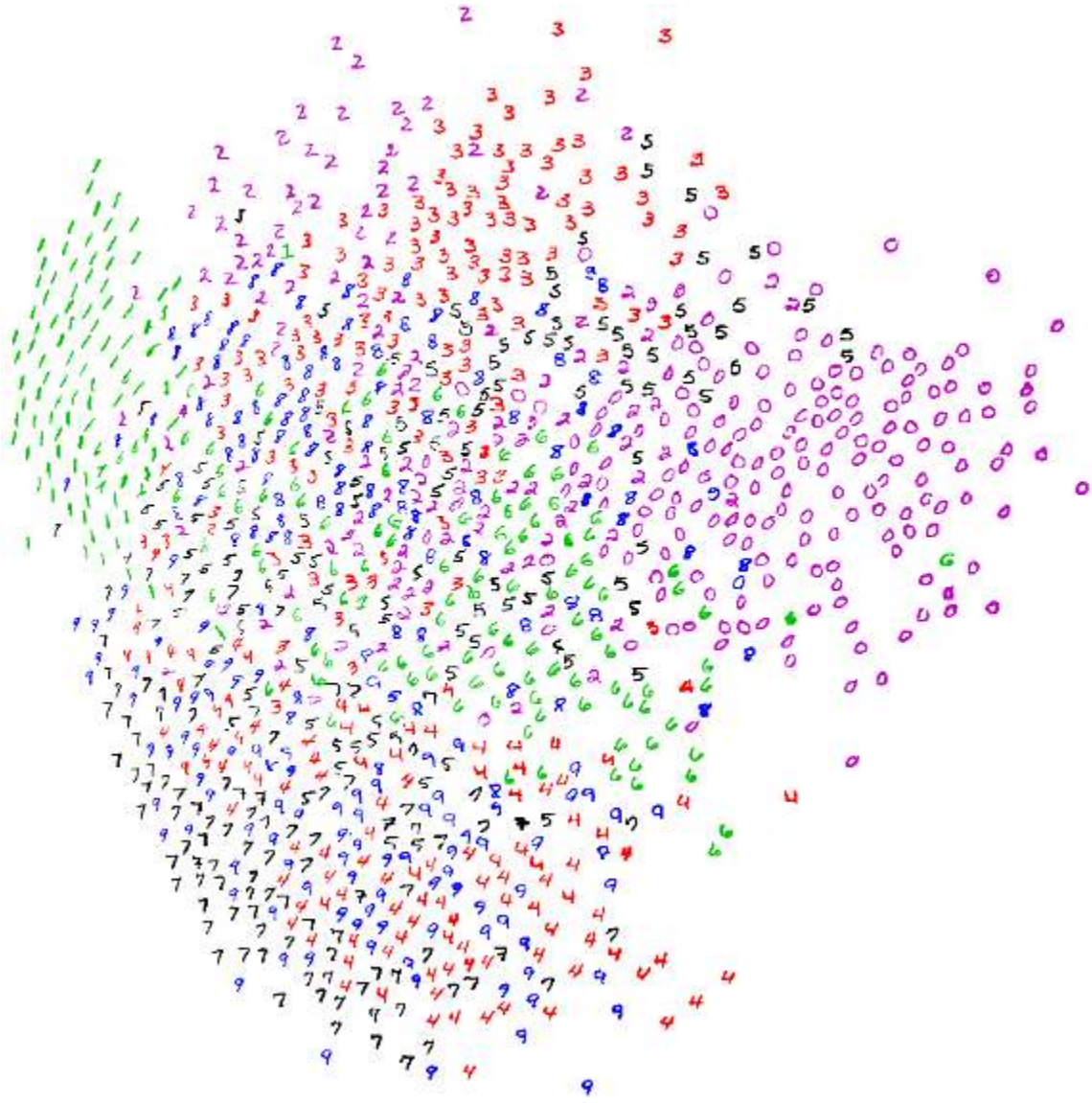

Fig. S3: An alternative visualization of the 2-D codes produced by taking the first two principal components of all 60,000 training images. 5,000 images of digits (500 per class) are sampled in random order. Each image is displayed if it does not overlap any of the images that have already been displayed.

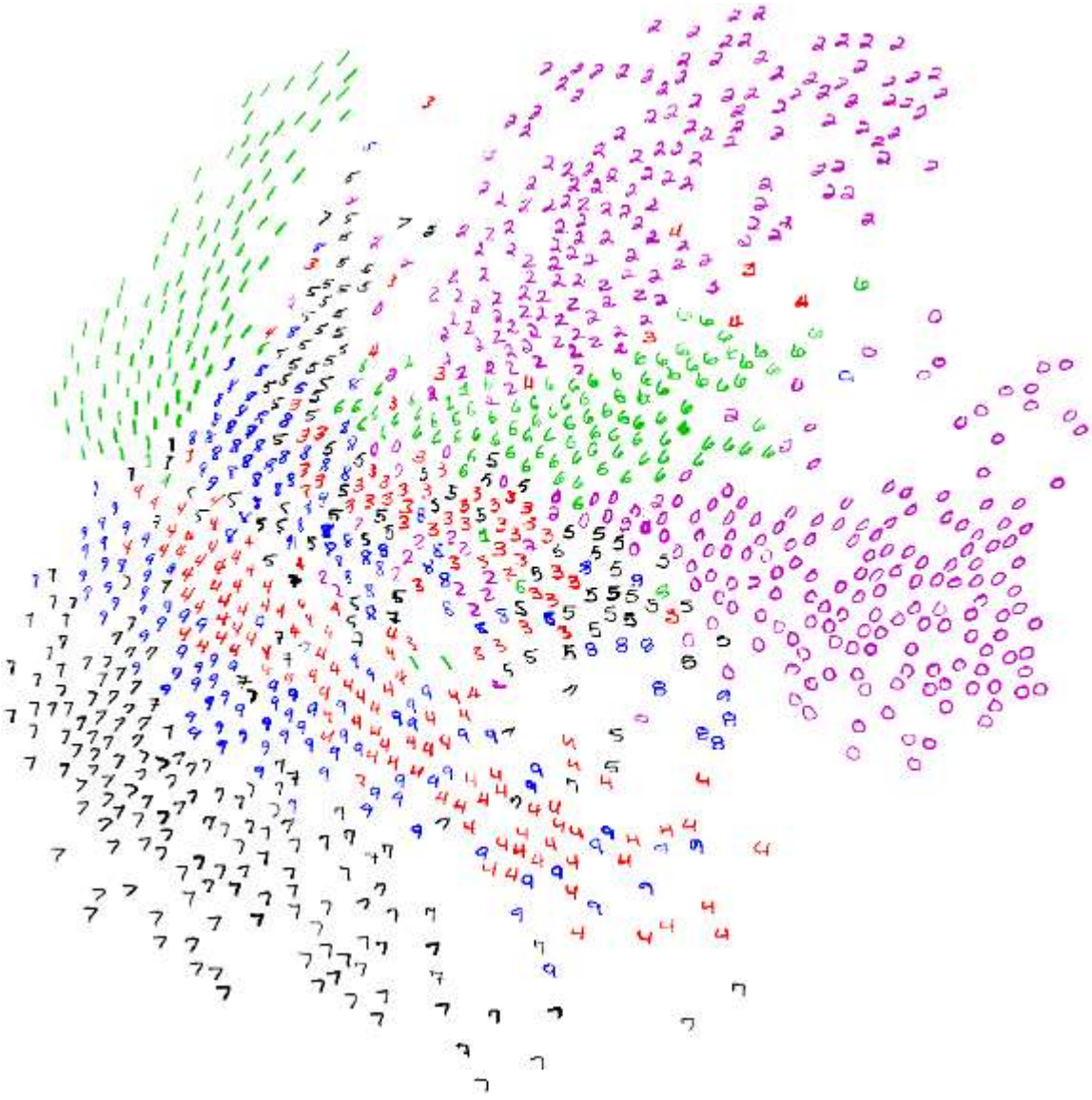

Fig. S4: An alternative visualization of the 2-D codes produced by a 784-1000-500-250-2 autoencoder trained on all 60,000 training images. 5,000 images of digits (500 per class) are sampled in random order. Each image is displayed if it does not overlap any of the images that have already been displayed.

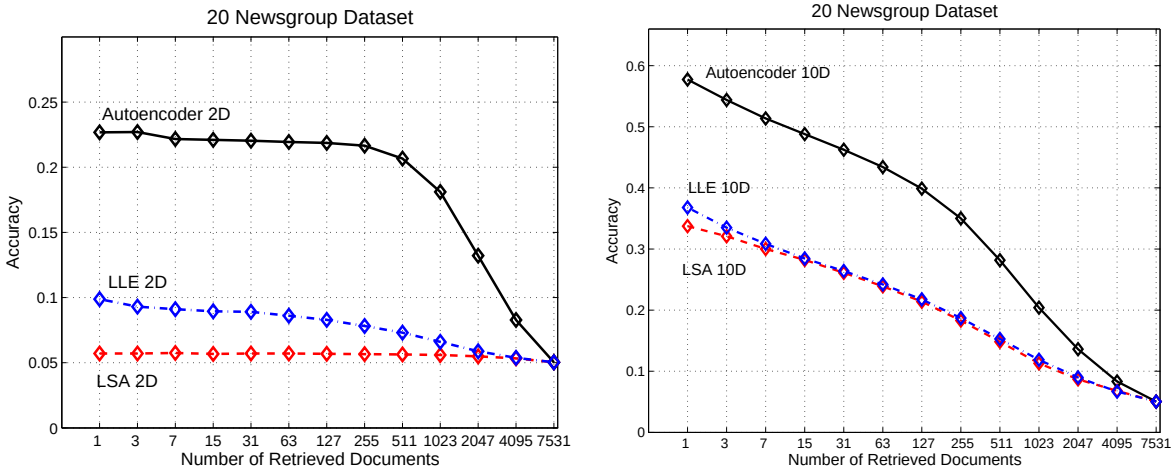

Fig. S5: Accuracy curves when a query document from the test set is used to retrieve other test set documents, averaged over all 7,531 possible queries.

References and Notes

1. For the conjugate gradient fine-tuning, we used Carl Rasmussen's "minimize" code available at http://www.kyb.tuebingen.mpg.de/bs/people/carl/code/minimize/.

2. G. Hinton, V. Nair, *Advances in Neural Information Processing Systems* (MIT Press, Cambridge, MA, 2006).

3. Matlab code for generating the images of curves is available at http://www.cs.toronto.edu/∼hinton.

4. G. E. Hinton, *Neural Computation* **14**, 1711 (2002).

5. S. T. Roweis, L. K. Saul, *Science* **290**, 2323 (2000).

6. The 20 newsgroups dataset (called 20news-bydate.tar.gz) is available at http://people.csail.mit.edu/jrennie/20Newsgroups.

7. L. K. Saul, S. T. Roweis, *Journal of Machine Learning Research* **4**, 119 (2003).

8. Matlab code for LLE is available at http://www.cs.toronto.edu/∼roweis/lle/index.html.

9. Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, *Proceedings of the IEEE* **86**, 2278 (1998).

10. D. V. Decoste, B. V. Schoelkopf, *Machine Learning* **46**, 161 (2002).

11. P. Y. Simard, D. Steinkraus, J. C. Platt, *Proceedings of Seventh International Conference on Document Analysis and Recognition* (2003), pp. 958–963.