CSC321 Introduction to Neural Networks and Machine Learning

Lecture 21
Using Boltzmann machines to initialize backpropagation

Geoffrey Hinton

Some problems with backpropagation

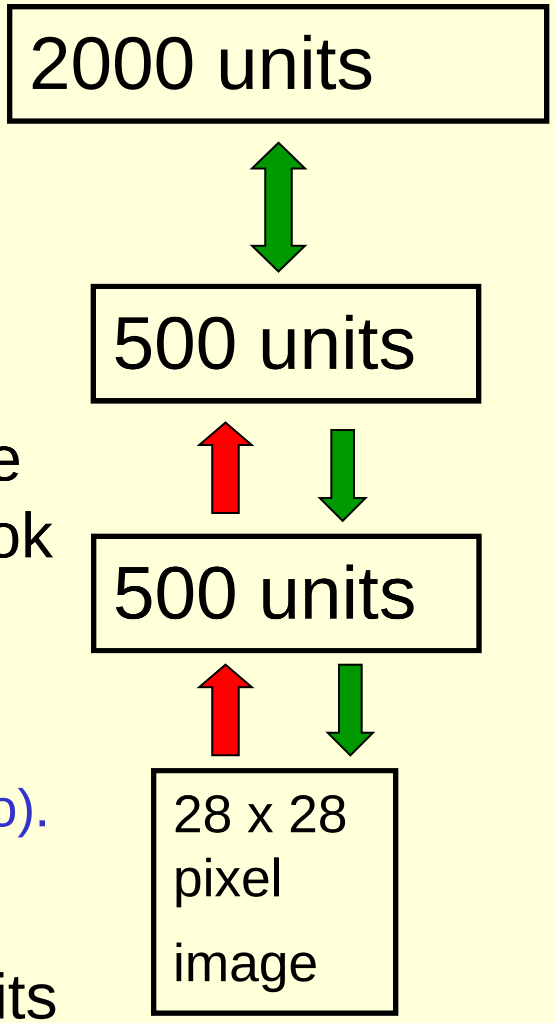
- The amount of information that each training case provides about the weights is at most the log of the number of possible output labels.
  - So to train a big net we need lots of labeled data.
- In nets with many layers of weights the backpropagated derivatives either grow or shrink multiplicatively at each layer.
  - Learning is tricky either way.
- Dumb gradient descent is not a good way to perform a global search for a good region of a very large, very non-linear space.
  - So deep nets trained by backpropagation are rare in practice.

A solution to all of these problems

- Use greedy unsupervised learning to find a sensible set of weights one layer at a time. Then fine-tune with backpropagation.
- Greedily learning one layer at a time scales well to really deep networks.
- Most of the information in the final weights comes from modeling the distribution of input vectors.
  - The precious information in the labels is only used for the final fine-tuning.
- We do not start backpropagation until we already have sensible weights that already do well at the task.
  - So the fine-tuning is well-behaved and quite fast.

Modelling the distribution of digit images

The top two layers form a restricted Boltzmann machine whose free energy landscape should model the low dimensional manifolds of the digits.



The network learns a density model for unlabeled digit images. When we generate from the model we often get things that look like real digits of all classes.

More hidden layers make the generated fantasies look better (YW Teh, Simon Osindero).

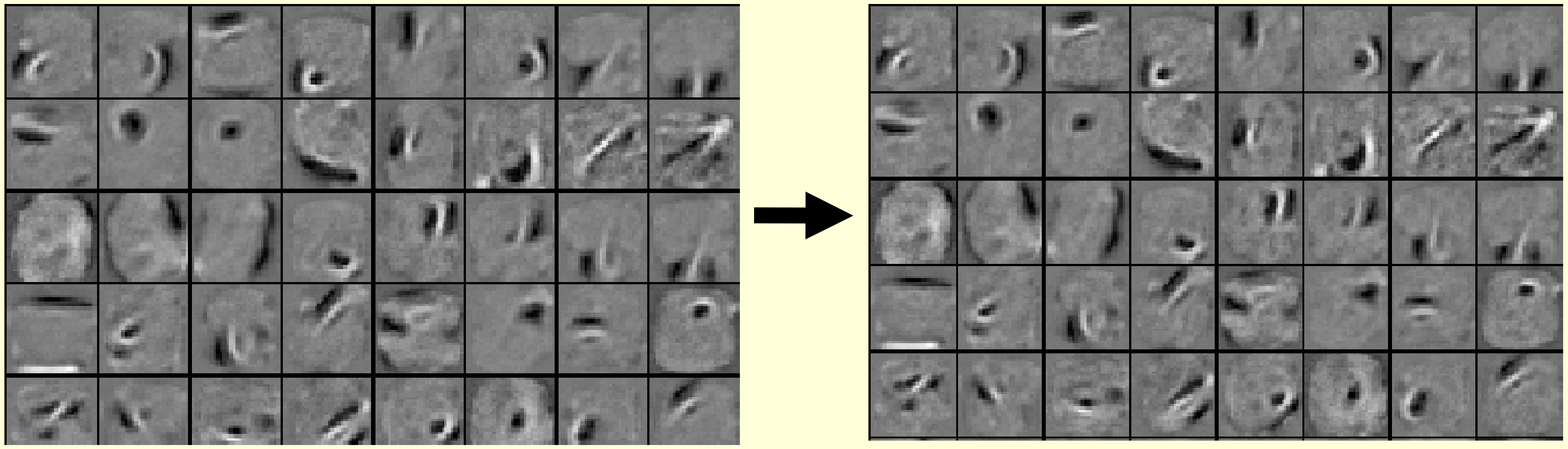
But do the hidden features really help with digit discrimination? Add 10 softmaxed units to the top and do backprop.

Results on permutation-invariant MNIST task

• Very carefully trained backprop net with one or two hidden layers (Platt; Hinton)	1.53%
• SVM (Decoste & Schoelkopf)	1.4%
• Generative model of joint density of images and labels (with unsupervised fine-tuning)	1.25%
• Generative model of unlabelled digits followed by gentle backpropagtion	1.2%
• Generative model of joint density followed by gentle backpropagation	1.1%

Learning Dynamics of Deep Nets

the next 4 slides describe work by Yoshua Bengio's group

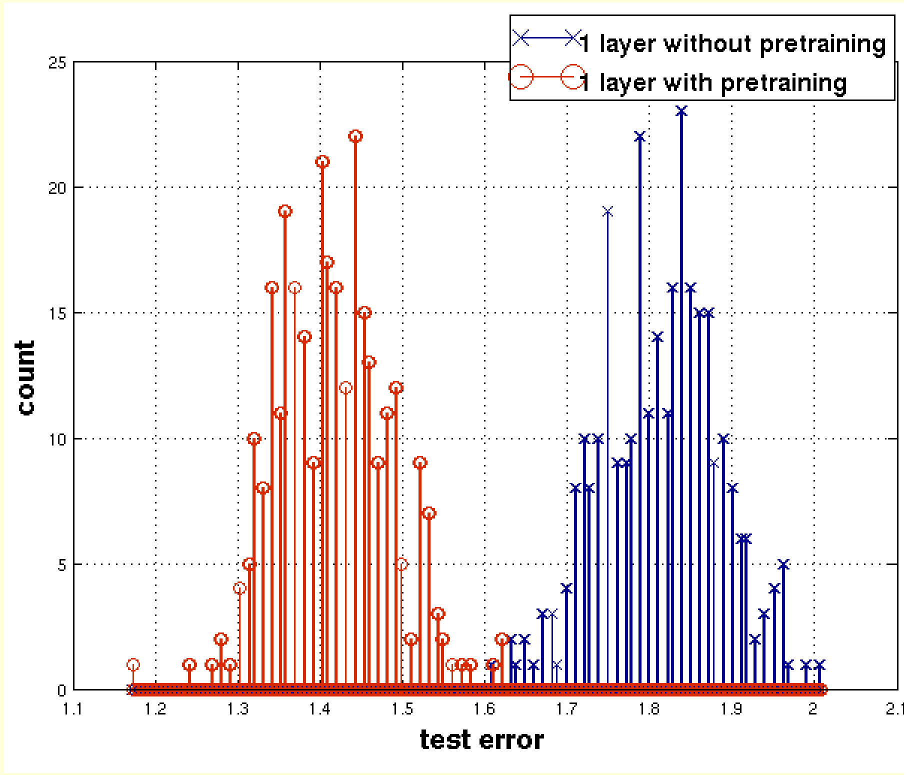


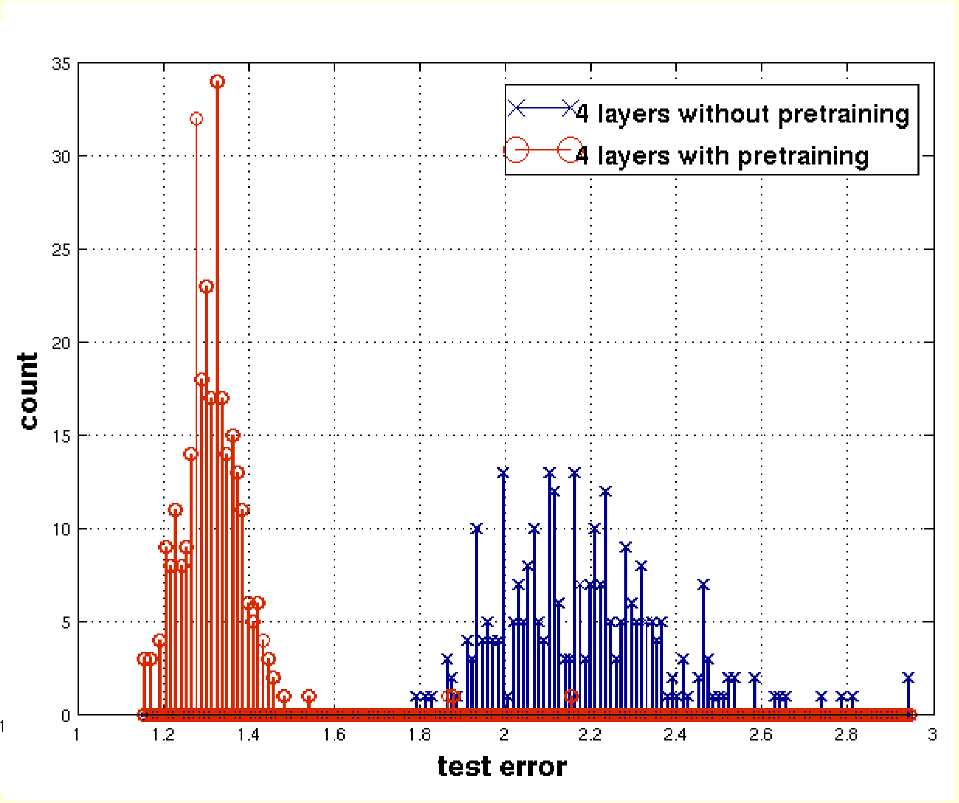
Before fine-tuning

After fine-tuning

Effect of Unsupervised Pre-training

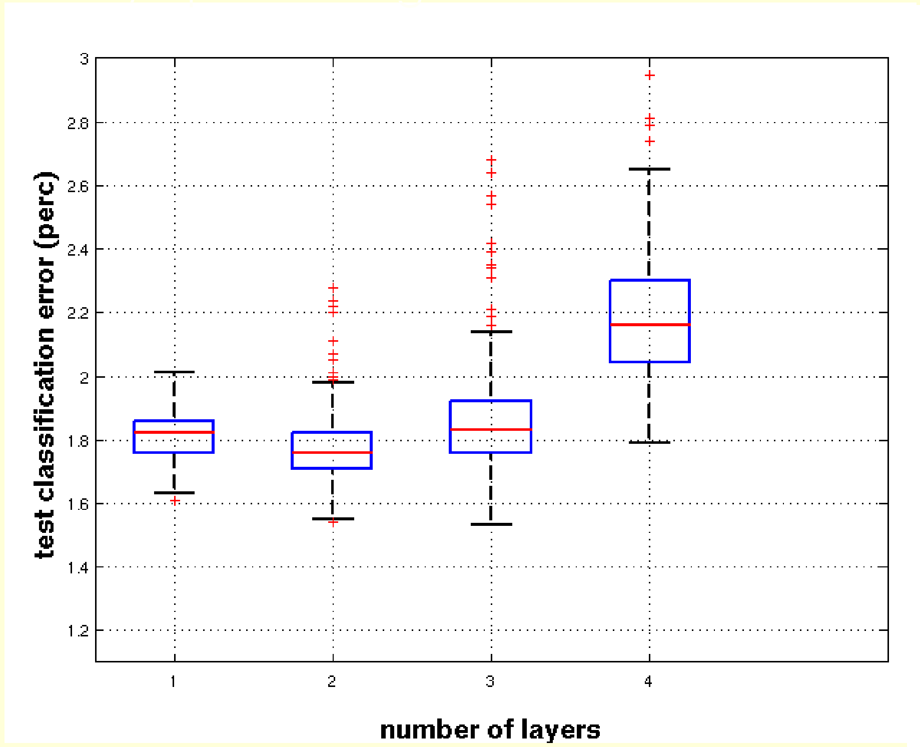
Erhan et. al.   AISTATS'2009



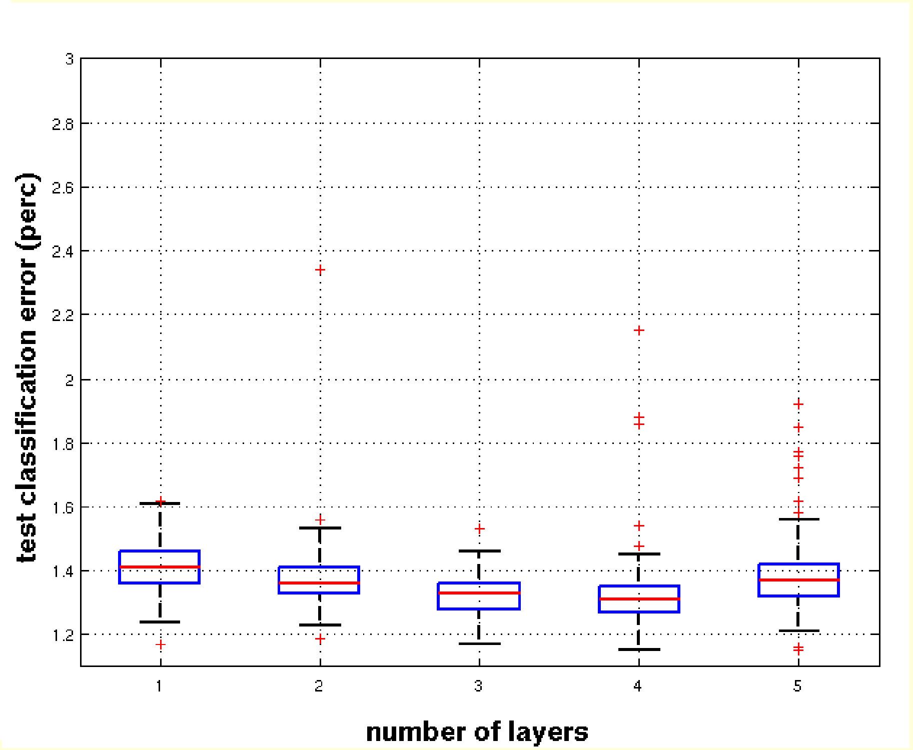


Effect of Depth

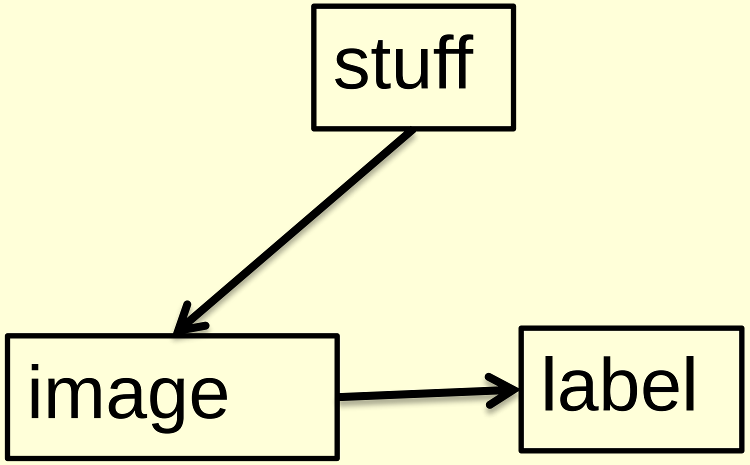
without pre-training

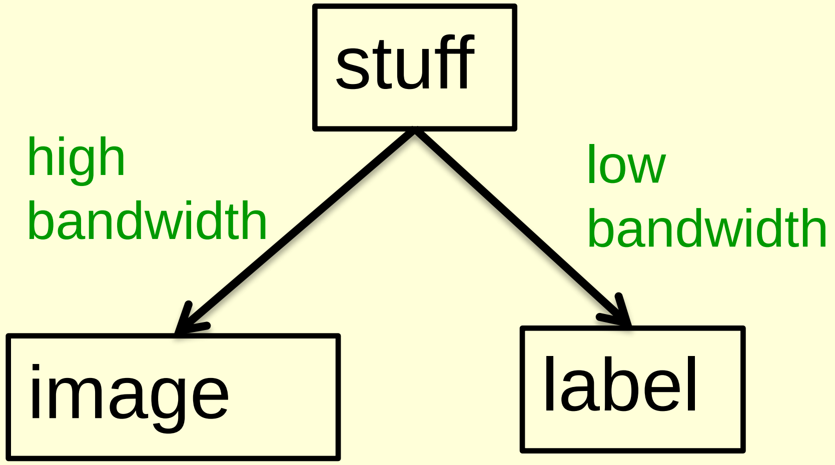


with pre-training



Why unsupervised pre-training makes sense





If image-label pairs were generated this way, it would make sense to try to go straight from images to labels.
For example, do the pixels have even parity?

If image-label pairs are generated this way, it makes sense to first learn to recover the stuff that caused the image by inverting the high bandwidth pathway.

An early use of neural nets (~1989)

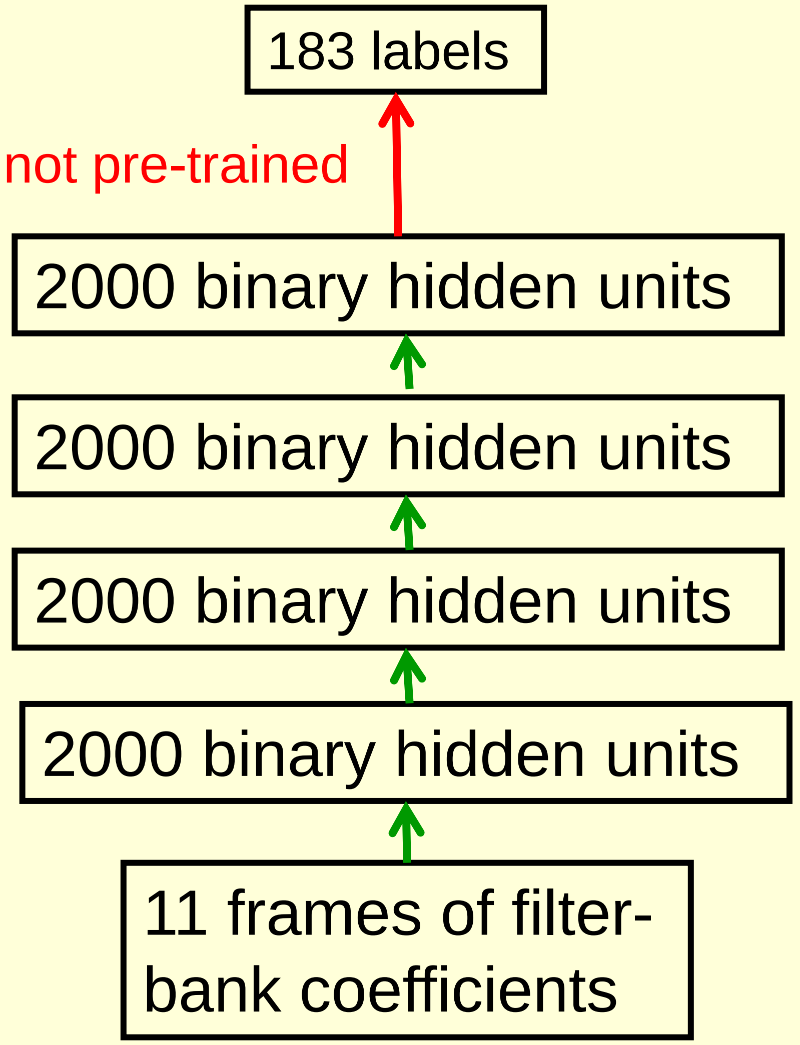
- Use a feedforward neural net to convert a window of speech coefficients into a posterior probability distribution over short pieces of phonemes (61 phones each with 3 pieces)
  – To train this net we need to know the “correct” label for each window, so we need to bootstrap from an existing speech recognition system.
- The trained neural net produces a posterior distribution over phone pieces at each time.
  – We feed these distributions to a decoder which finds the most likely sequence of phonemes.

How to make the phone recognizer work much better

- Train lots of big layers, one at a time, without using the labels.
- Add 183-way softmax over labels as the final layer.
- Fine-tune with bckpropagation on a big GPU board for several days.

A very deep belief net for phone recognition

Mohamed, Dahl & Hinton (2011)



Many of the major speech recognition groups (Google, Microsoft, IBM) are now trying this approach.

Deep Autoencoders

- They always looked like a really nice way to do non-linear dimensionality reduction:
  - They provide mappings both ways
  - The learning time is linear (or better) in the number of training cases.
  - The final model is compact and fast.
- But it turned out to be very very difficult to optimize deep autoencoders using backprop.
  - We now have a much better way to optimize them.

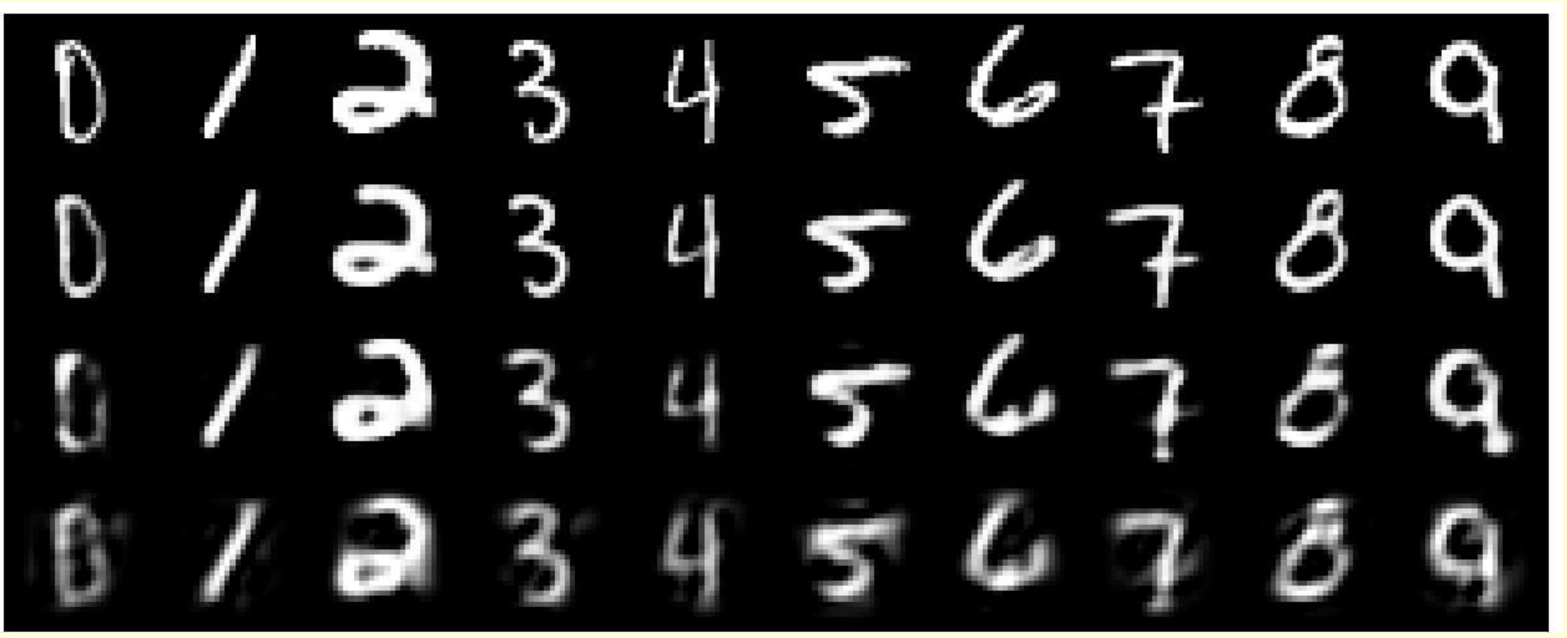
The deep autoencoder

$$\begin{tikzcd}
784 \arrow[r, "W_1"] & 1000 \arrow[r, "W_2"] & 500 \arrow[r, "W_3"] & 250 \arrow[dr, "W_4"] & \\
 & & & & 30 \text{ linear units} \arrow[dl, "W_4^T"] \\
784 & 1000 \arrow[l, "W_1^T"'] & 500 \arrow[l, "W_2^T"'] & 250 \arrow[l, "W_3^T"'] &
\end{tikzcd}$$

If you start with small random weights it will not learn. If you break symmetry randomly by using bigger weights, it will not find a good solution.

So we train a stack of 4 RBM's and then "unroll" them. Then we fine-tune with gentle backprop.

A comparison of methods for compressing digit images to 30 real numbers.



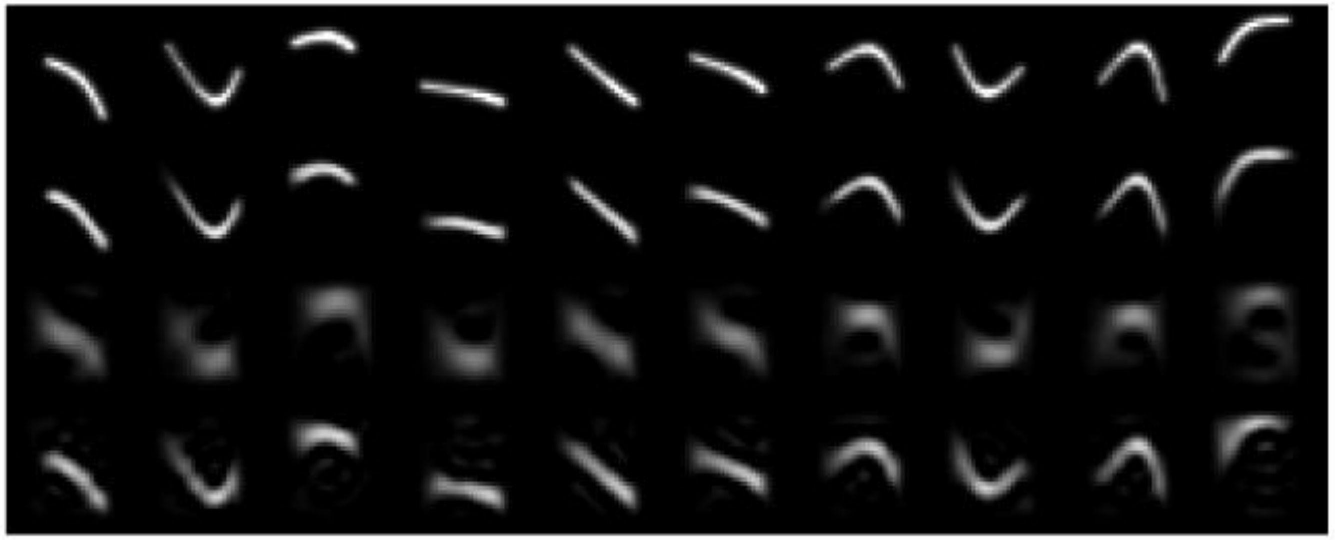
real data

30-D deep auto

30-D logistic PCA

30-D PCA

A very deep autoencoder for synthetic curves that only have 6 degrees of freedom



	squared error
Data	0.0
Auto:6	1.5
PCA:6	10.3
PCA:30	3.9

An autoencoder for patches of real faces

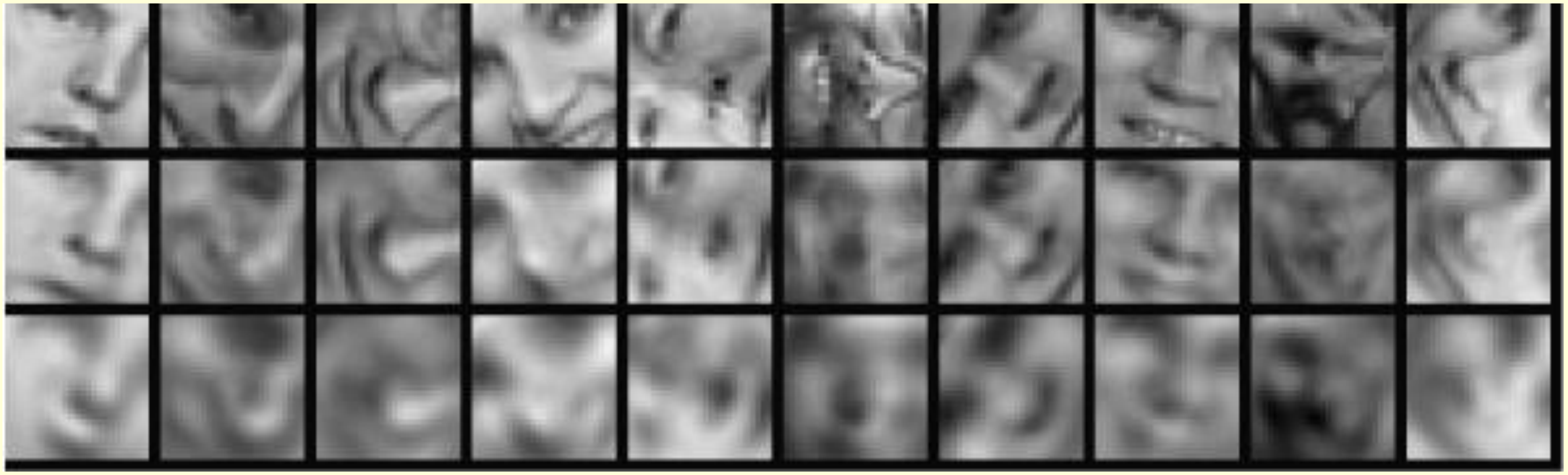
- 625 → 2000 → 1000 → 641 → 30   and back out again

linear   logistic units   linear

Train on 100,000 denormalized face patches from 300 images of 30 people. Use 100 epochs of CD at each layer followed by backprop through the unfolded autoencoder.

Test on face patches from 100 images of 10 new people.

Reconstructions of face patches from new people

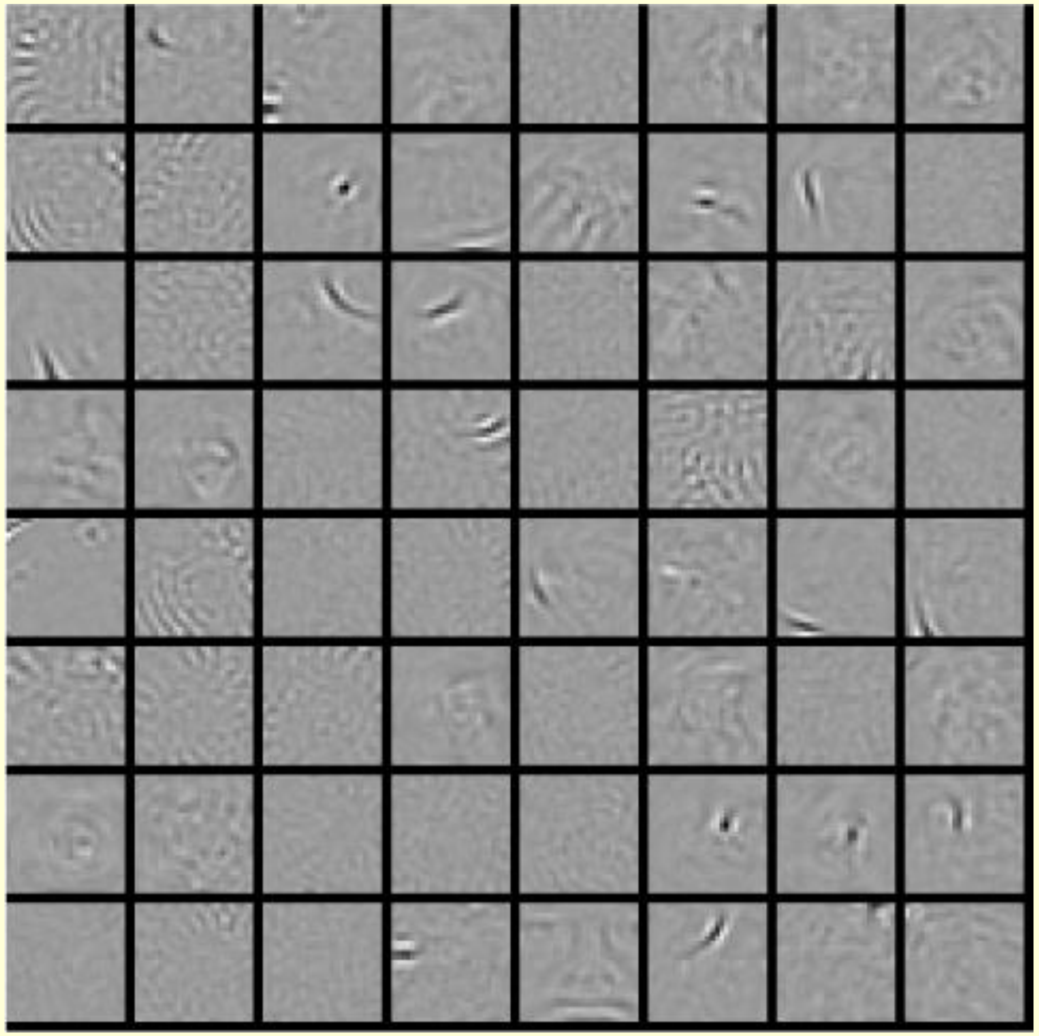


Data

Auto:30
126

PCA:30
135

64 of the hidden units in the first hidden layer

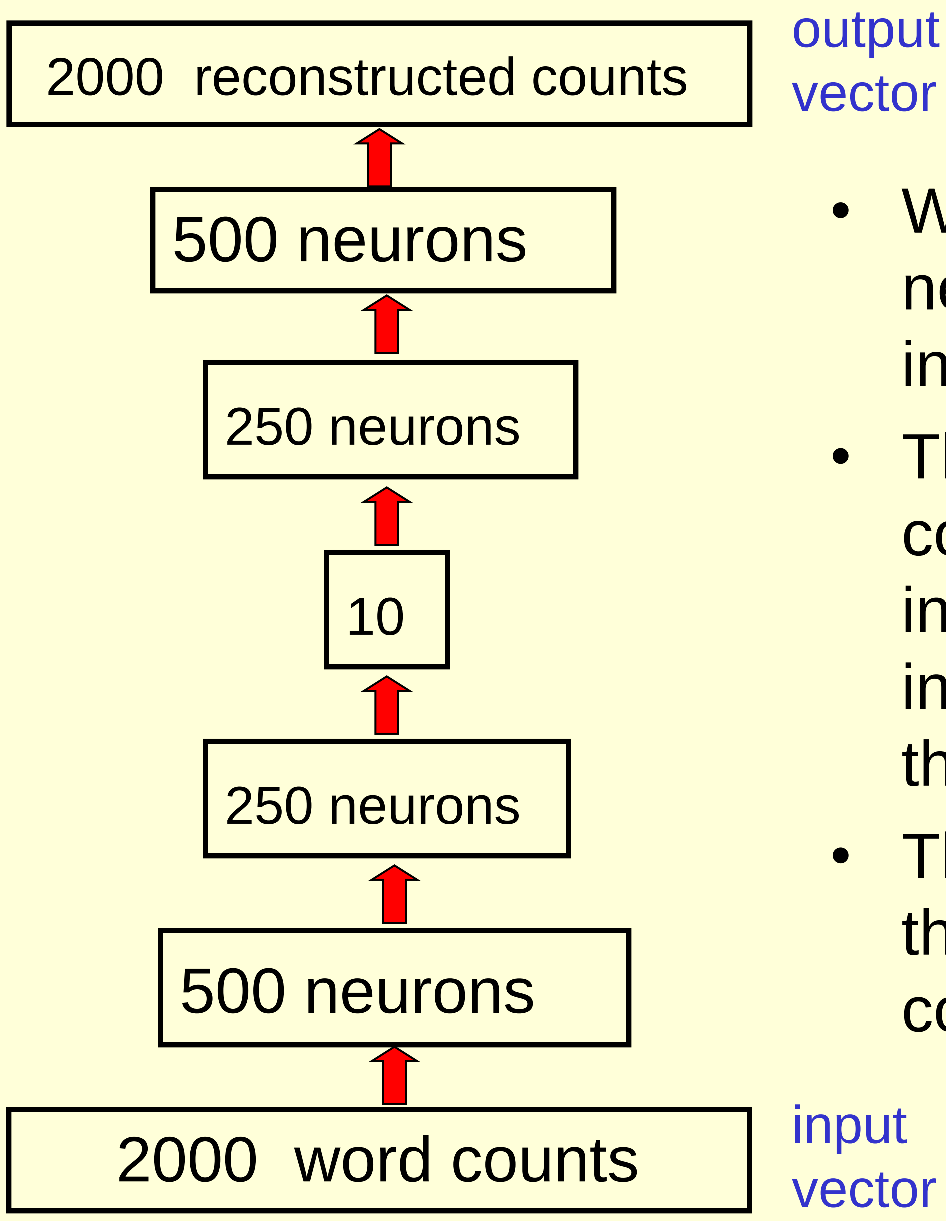


How to find documents that are similar to a query document

- Convert each document into a “bag of words”.
  - This is a vector of word counts ignoring the order.
  - Ignore stop words (like “the” or “over”)
- We could compare the word counts of the query document and millions of other documents but this is too slow.
  - So we reduce each query vector to a much smaller vector that still contains most of the information about the content of the document.

0	fish
0	cheese
2	vector
2	count
0	school
2	query
1	reduce
1	bag
0	pulpit
0	iraq
2	word

How to compress the count vector



- We train the neural network to reproduce its input vector as its output
- This forces it to compress as much information as possible into the 10 numbers in the central bottleneck.
- These 10 numbers are then a good way to compare documents.

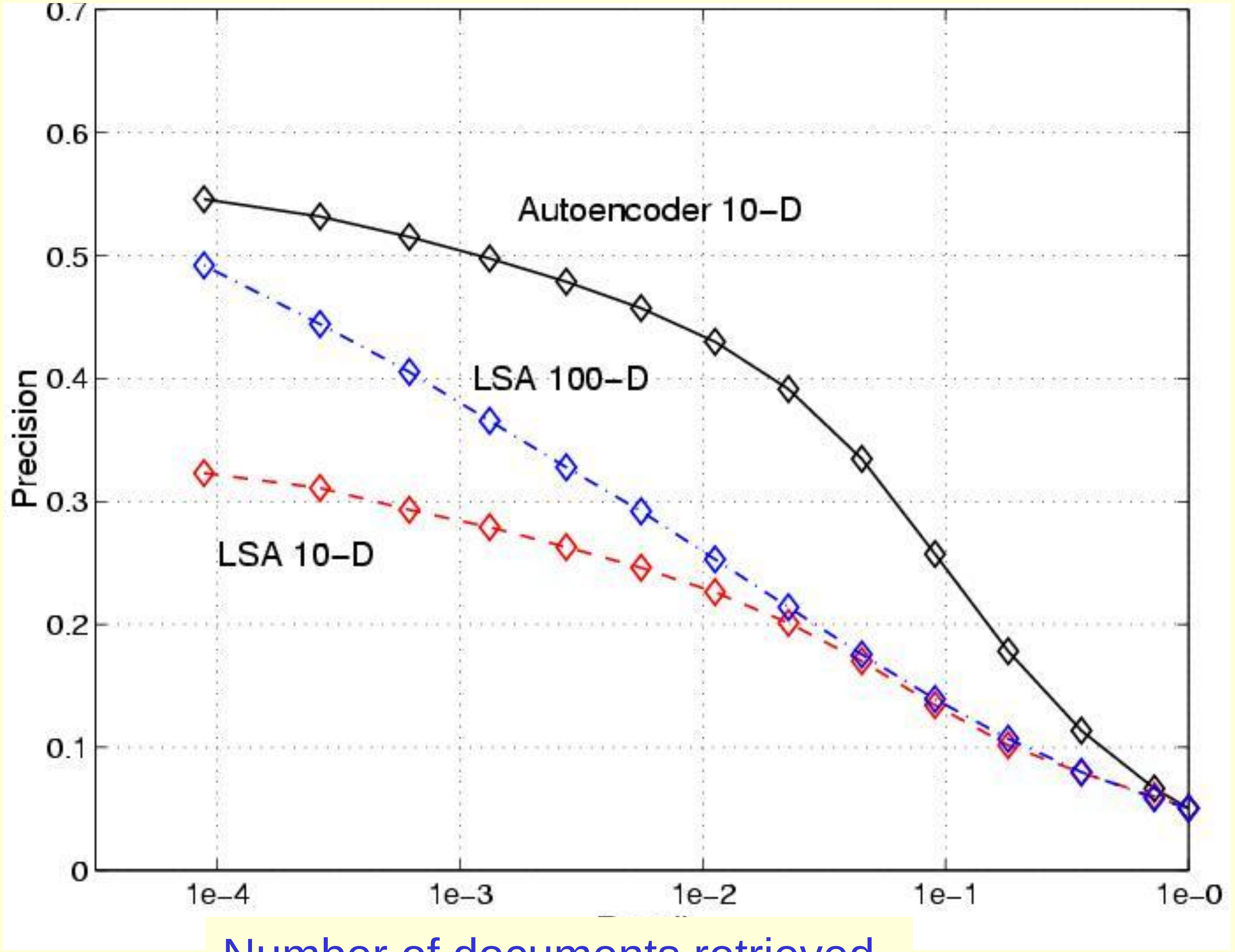
The non-linearity used for reconstructing bags of words

- Divide the counts in a bag of words vector by N, where N is the total number of non-stop words in the document.
  - The resulting probability vector gives the probability of getting a particular word if we pick a non-stop word at random from the document.
- At the output of the autoencoder, we use a softmax.
  - The probability vector defines the desired outputs of the softmax.
- When we train the first RBM in the stack we use the same trick.
  - We treat the word counts as probabilities, but we make the visible to hidden weights N times bigger than the hidden to visible because we have N observations from the probability distribution.

Performance of the autoencoder at document retrieval

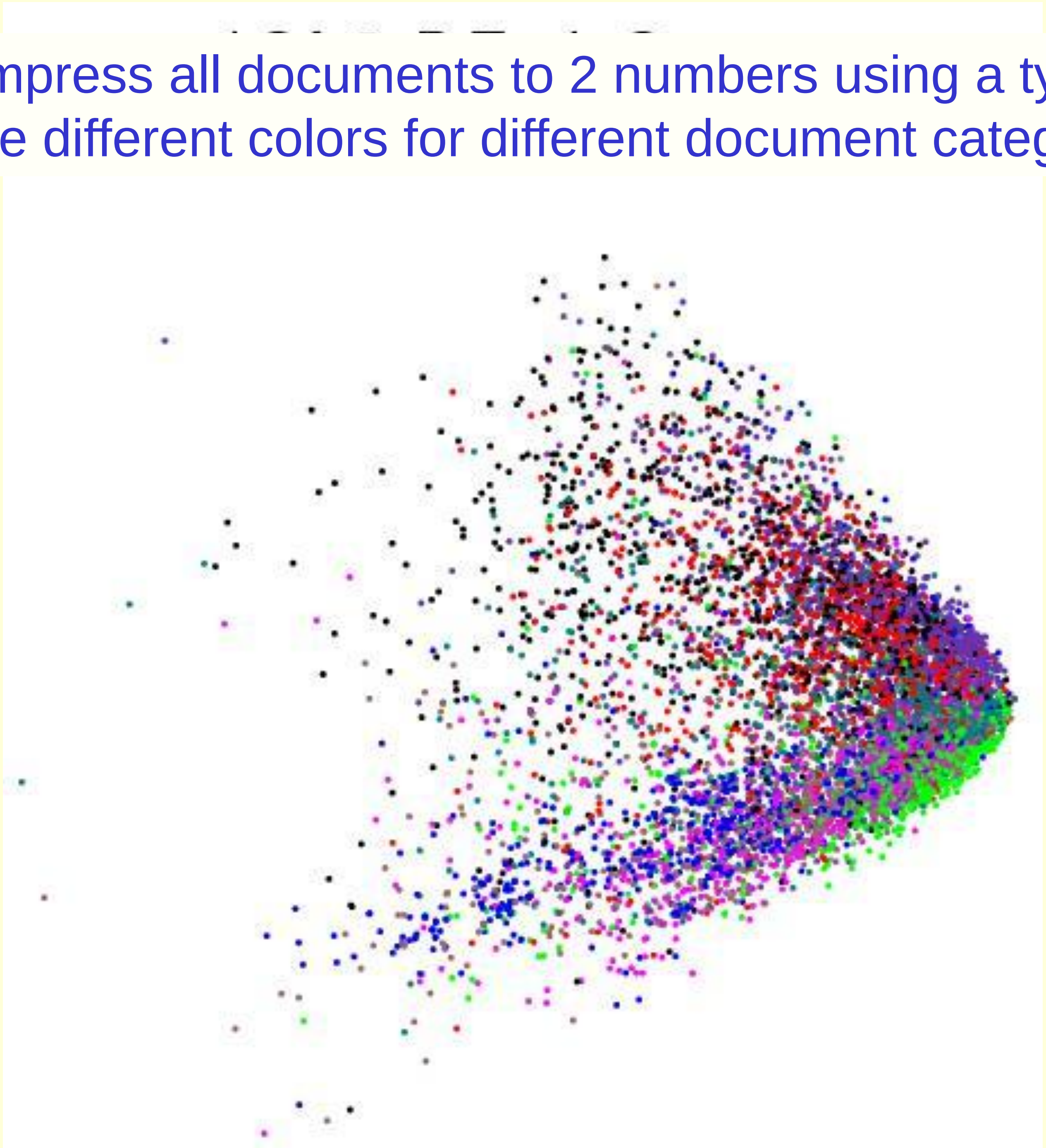
- Train on bags of 2000 words for 400,000 training cases of business documents.
  - First train a stack of RBM's. Then fine-tune with backprop.
- Test on a separate 400,000 documents.
  - Pick one test document as a query. Rank order all the other test documents by using the cosine of the angle between codes.
  - Repeat this using each of the 400,000 test documents as the query (requires 0.16 trillion comparisons).
- Plot the number of retrieved documents against the proportion that are in the same hand-labeled class as the query document. Compare with LSA (a version of PCA).

Proportion of retrieved documents in same class as query

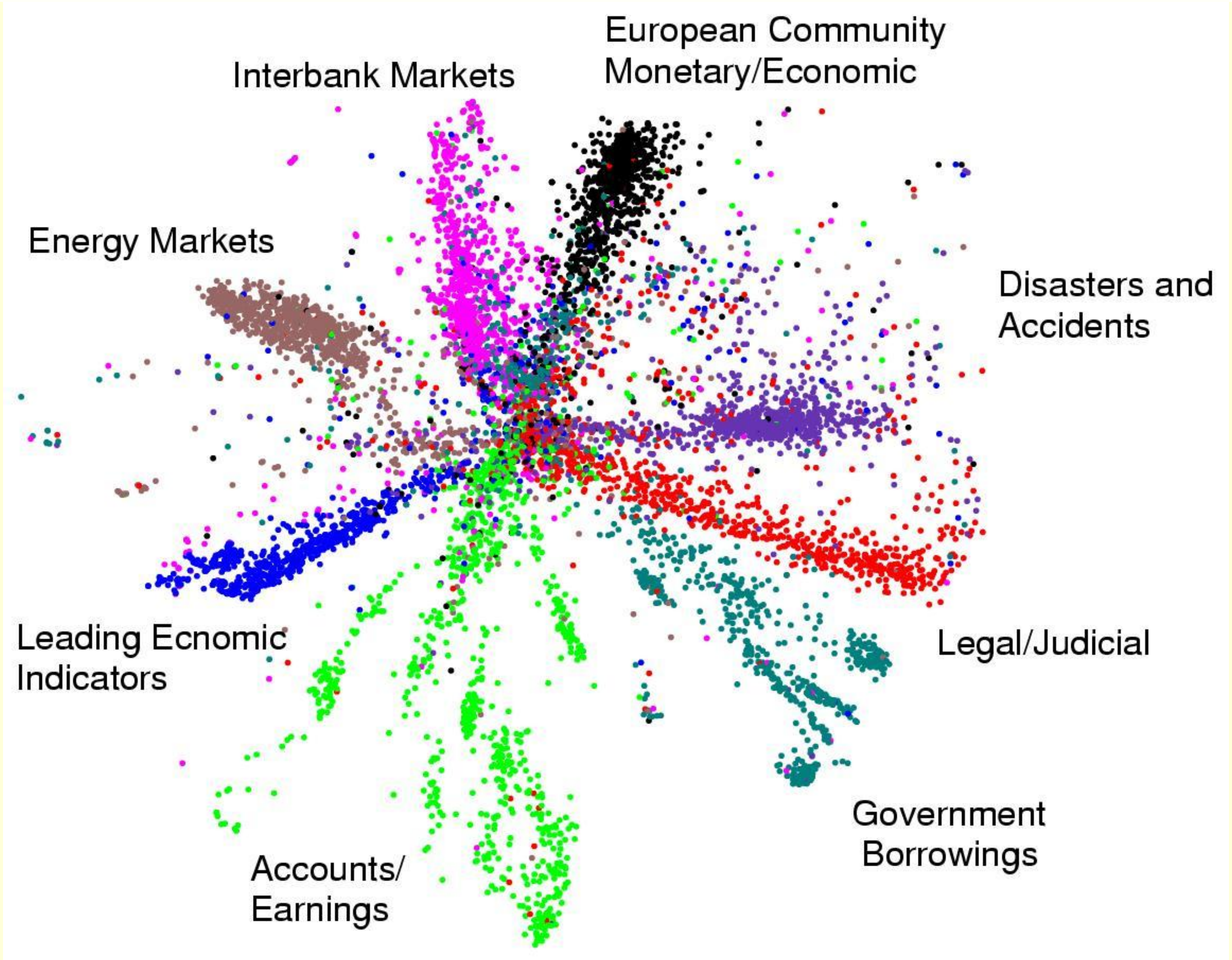


Number of documents retrieved

First compress all documents to 2 numbers using a type of PCA
Then use different colors for different document categories



First compress all documents to 2 numbers. Then use different colors for different document categories



THE END