Reducing the Dimensionality of Data with Neural Networks

参考：

https://www.cnblogs.com/shuzirank/p/5818320.html

Abstract

High-dimensional data can be converted to low-dimensional codes by training a multilayer neural network with a small central layer to reconstruct high-dimensional input vectors.

Gradient descent can be used for fine-tuning the weights in such ‘‘autoencoder’’ networks, but this works well only if the initial weights are close to a good solution.

We describe an effective way of initializing the weights that allows deep autoencoder networks to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data.

用带有很小的中间层的MNN对高维数据降维，用GD fine-tuning weights需要初始的权重足够好，本文提出一种初始化权重的方法，效果比PCA好

Dimensionality reduction facilitates the classification, visualization, communication, and storage of high-dimensional data.(降维好）

A simple and widely used method is principal components analysis (PCA), which finds the directions of greatest variance in the data set and represents each data point by its coordinates along each of these directions.

We describe a nonlinear generalization of PCA that uses an adaptive, multilayer “encoder” network to transform the high-dimensional data into a low-dimensional code and a similar “encoder” network to recover the data from the code.

Starting with random weights in the two networks, they can be trained together by minimizing the discrepancy(差异) between the original data and its reconstruction. The required gradients are easily obtained by using the chain rule to backpropagate error derivatives first through the decoder network and then through the encoder network (1). The whole system is called an ‘‘autoencoder’’ and is depicted in Fig. 1.

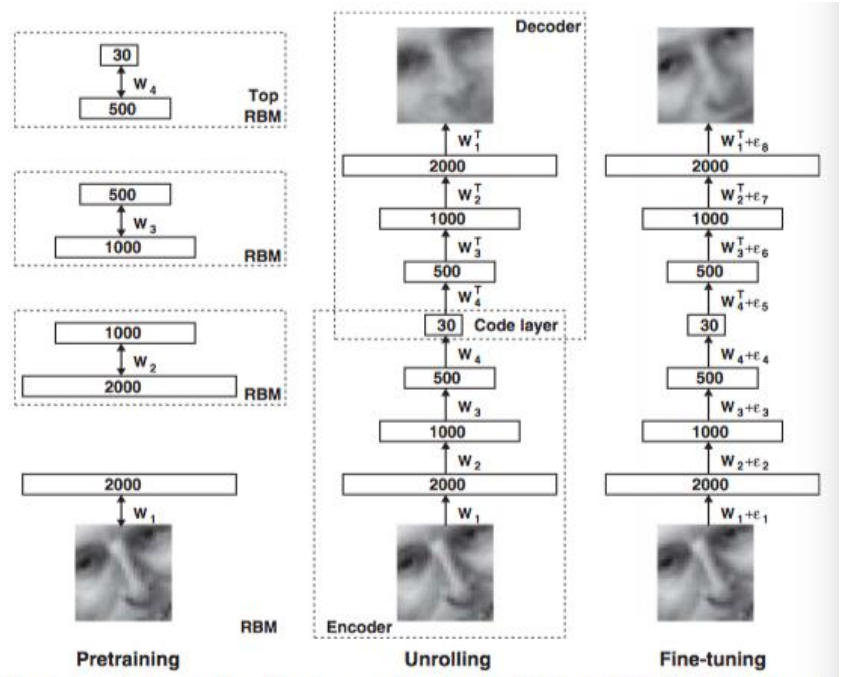


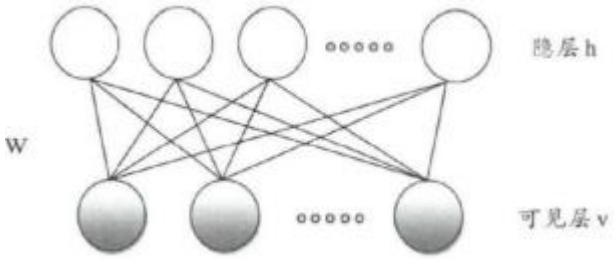
Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBMs are "unrolled" to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.

It is difficult to optimize the weights in nonlinear autoencoders that have multiple hidden layers (2－4).

With large initial weights, autoencoders typically find poor local minima; with small initial weights, the gradients in the early layers are tiny, making it infeasible to train autoencoders with many hidden layers. （初始化太小，会导致神经元的输入过小，随着层数的不断增加，会出现信号消失的问题）

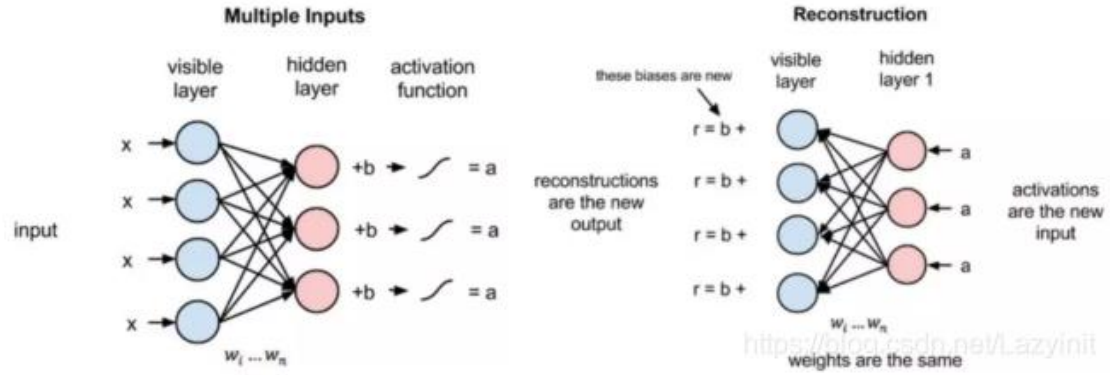
If the initial weights are close to a good solution, gradient descent works well, but finding such initial weights requires a very different type of algorithm that learns one layer of features at a time. （找到这样的一个初始化权重需要对每一层尝试许多类型的算法 ）

We introduce this "pretraining" procedure for binary data, generalize it to real-valued data, and show that it works well for a variety of data sets.



下面讲RBM https://zhuanlan.zhihu.com/p/36529237

An ensemble of binary vectors (e.g., images) can be modeled using a two-layer network called a "restricted Boltzmann machine" (RBM) (5, 6) in which stochastic, binary pixels are connected to stochastic, binary feature detectors using symmetrically weighted connections. The pixels correspond to "visible" units of the RBM because their states are observed; the feature detectors correspond to "hidden" units.



在前向传递过程中，给定权重的情况下 RBM 会使用输入x来预测节点的激活值a，或者输出的概率p(a|x; w) -> p(h|x,w)。h1 {0,1} P(h=1|v,w) P(h=0|v,w)

在反向传播的过程中，当激活值作为输入并输出原始数据的预测时，RBM 尝试在给定激活值 a 的情况下估计输入 x 的概率，它具有与前向传递过程中相同的权重参数。这第二个阶段可以被表达为 p(x|a; w)-> p(x|h,w)。

这两个概率估计将共同得到关于输入 x 和激活值 a 的联合概率分布（给定 a 时 x 的概率以及给定 x 时 a 的概率，可以根据 RBM 两层之间的共享权重而确定）p(x, a) -> p(v,h)。

E(v,1| theta), E(v,0| theta)

$$E(\mathbf{v},\mathbf{h}\,|\,\theta) = -\sum_{i=1}^{n} a_i v_i - \sum_{j=1}^{m} b_j h_j - \sum_{i=1}^{n}\sum_{j=1}^{m} v_i W_{ij} h_j$$

$$P(\mathbf{v},\mathbf{h}\,|\,\theta) = \frac{\mathrm{e}^{-E(\mathbf{v},\mathbf{h}\,|\,\theta)}}{Z(\theta)}, \quad Z(\theta) = \sum_{\mathbf{v},\mathbf{h}} \mathrm{e}^{-E(\mathbf{v},\mathbf{h}\,|\,\theta)},$$

能量函数：随机神经网络中，引入了能量函数。能量函数是描述整个系统状态的一种测度。系统概率分布p(v,h)越集中，系统的能量越小。反之，系统概率分布p(v,h)越趋于均匀分布，则系统的能量越大。能量函数的最小值，对应于系统的最稳定状态。

随机神经网络的核心思想就是在网络中加入概率因素，网络并不是确定的向能量函数减小的方向演化，而是以一个较大概率向这个方向演化，以保证正确的迭代方向，能量函数增大的概率也存在，以防止陷入局部极小值。

A joint configuration(构造) (v, h) of the visible and hidden units has an energy (7) given by

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i\in\text{pixels}} b_i v_i - \sum_{j\in\text{features}} b_j h_j - \sum_{i,j} v_i h_j w_{ij} \qquad (1)$$

where v_i and h_j are the binary states of pixel i and feature j, b_i and b_j are their biases, and w_{ij} is the weight between them.
The network assigns a probability to every possible image via this energy function, as explained in (8)supporting material.

Q: probability 是什么概率？

The probability that the model assigns to a visible vector, v is

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

The probability of a training image can be raised by adjusting the weights and biases to lower the energy of that image and to raise the energy of similar, "confabulated"(虚构的) images that the network would prefer to the real data.
通过调节权重以及偏差，来降低真实image的能量，提高虚构image的能量，使得网络更加倾向于那些真实的数据。

Given a training image, the binary state h_j of each feature detector j is set to 1 with probability $\sigma(b_j + \sum_i v_i w_{ij})$, where $\sigma(x)$ is the logistic function 1/[1 + exp (-x)], b_j is the bias of j, v_i is the state of pixel i, and w_{ij} is the weight between i and j.
Once binary states have been chosen for the hidden units, a "confabulation" is produced by setting each v_i to 1 with probability $\sigma(b_i + \sum_j h_j w_{ij})$, where b_i is the bias of i.
The states of the hidden units are then updated once more so that they represent features of the confabulation.

The change in a weight is given by

$$\Delta w_{ij} = \varepsilon\left(\langle v_i h_j\rangle_{\text{data}} - \langle v_i h_j\rangle_{\text{recon}}\right) \qquad (2)$$

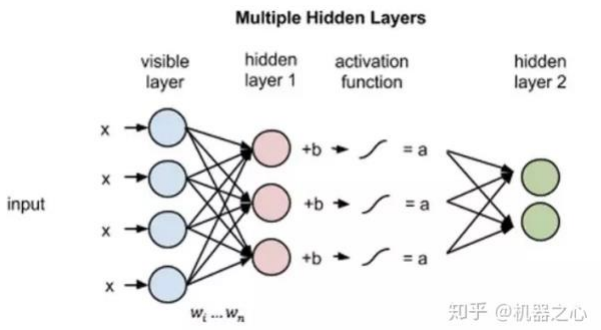
where ε is a learning rate, $\langle v_i h_j\rangle_{\text{data}}$ is the fraction of times that the pixel i and feature detector j are on together when the feature detectors are being driven by data, and $\langle v_i h_j\rangle_{\text{recon}}$ is the corresponding fraction for confabulations.
A simplified version of the same learning rule is used for the biases.
The **learning works well** even though it is **not exactly following the gradient of the log probability** of the training data (6).

Algorithm 1. k-step contrastive divergence

Input: RBM $(V_1, \ldots, V_m, H_1, \ldots, H_n)$, training batch S
Output: gradient approximation Δw_{ij}, Δb_j and Δc_i for $i = 1, \ldots, n$, $j = 1, \ldots, m$

init $\Delta w_{ij} = \Delta b_j = \Delta c_i = 0$ for $i = 1, \ldots, n$, $j = 1, \ldots, m$
forall the $\boldsymbol{v} \in S$ **do**
  $\boldsymbol{v}^{(0)} \leftarrow \boldsymbol{v}$
  for $t = 0, \ldots, k-1$ **do**
    for $i = 1, \ldots, n$ **do** sample $h_i^{(t)} \sim p(h_i \mid \boldsymbol{v}^{(t)})$
    for $j = 1, \ldots, m$ **do** sample $v_j^{(t+1)} \sim p(v_j \mid \boldsymbol{h}^{(t)})$
  for $i = 1, \ldots, n$, $j = 1, \ldots, m$ **do**
    $\Delta w_{ij} \leftarrow \Delta w_{ij} + p(H_i = 1 \mid \boldsymbol{v}^{(0)}) \cdot v_j^{(0)} - p(H_i = 1 \mid \boldsymbol{v}^{(k)}) \cdot v_j^{(k)}$
    $\Delta b_j \leftarrow \Delta b_j + v_j^{(0)} - v_j^{(k)}$
    $\Delta c_i \leftarrow \Delta c_i + p(H_i = 1 \mid \boldsymbol{v}^{(0)}) - p(H_i = 1 \mid \boldsymbol{v}^{(k)})$

A single layer of binary features is not the best way to model the structure in a set of images. **After learning one layer of feature detectors**, we can **treat their activities**—when they are being driven by the data—**as data for learning a second layer of features.**
The first layer of feature detectors then become the visible units for learning the next RBM. This layer-by-layer learning can be repeated as many times as desired.



It can be shown that **adding an extra layer always improves a lower bound on the log probability???** that the model assigns to the training data, provided **the numbe**

r of feature detectors per layer does not decrease and their weights are initialized correctly (9).
This bound does not apply when the higher layers have fewer feature detectors, but the layer-by-layer learning algorithm is nonetheless a very effective way to pretrain the weights of a deep autoencoder.

Each layer of features captures strong, high-order correlations between the activities of units in the layer below.
For a wide variety of data sets, this is an efficient way to progressively reveal low-dimensional, nonlinear structure.

After pretraining multiple layers of feature detectors, the model is "unfolded" (Fig. 1) to produce encoder and decoder networks that initially use the same weights.

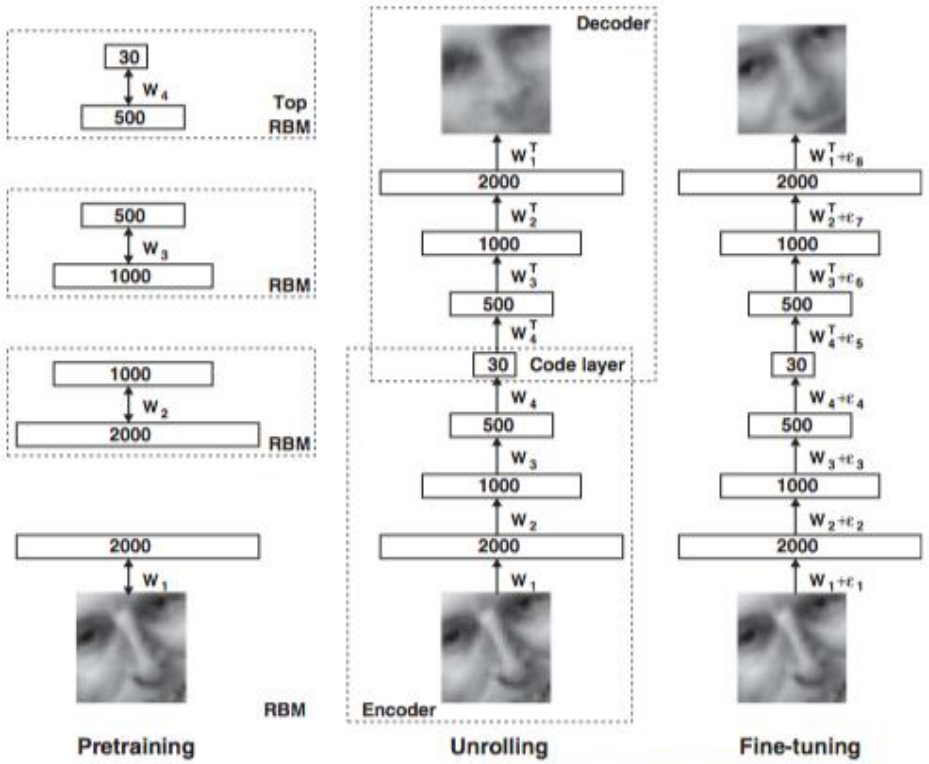


Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBMs are "unrolled" to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.

Q: 什么是deterministic, real-valued probabilities ？

The global fine-tuning stage then replaces stochastic activities by deterministic, real-valued probabilities and uses backpropagation through the whole autoencoder to fine-tune the weights for optimal reconstruction.

For continuous data, the hidden units of the first-level RBM remain binary, but the visible units are replaced by linear units with Gaussian noise (10).
If this noise has unit variance, the stochastic update rule for the hidden units remains the same and the update rule for visible unit i is to sample from a Gaussian with unit variance and mean $b_i + \sum_j h_j w_{ij}$.

In all our experiments, the visible units of every RBM had real-valued activities, which were in the range [0, 1] for logistic units.

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:

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

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.

While training higher level RBMs, the visible units were set to the activation probabilities of the hidden units in the previous RBM, but the hidden units of every RBM except the top one had stochastic binary values.

The hidden units of the top RBM had stochastic real-valued states drawn from a unit variance Gaussian whose mean was determined by the input from that RBM's logistic visible units.

This allowed the low-dimensional codes to make good use of continuous variables and facilitated comparisons with PCA. Details of the pretraining and fine-tuning can be found in (8).

To demonstrate that our pretraining algorithm allows us to fine-tune deep networks efficiently, we trained a very deep autoencoder on a synthetic data set containing images of "curves" that were generated from three randomly chosen points in two dimensions (8).
For this data set, the true intrinsic dimensionality is known, and the relationship between the pixel intensities and the six numbers used to generate them is highly nonlinear.
The pixel intensities lie between 0 and 1 and are very non-Gaussian, so we used logistic output units in the autoencoder, and the fine-tuning stage of the learnin

g **minimized** the **cross-entropy error** $[-\sum_i p_i \log \hat{p}_i - \sum_i (1-p_i)\log(1-\hat{p}_i)]$, where p_i is the intensity of pixel i and $p_i\hat{}$ is the intensity of its reconstruction.

The autoencoder consisted of an **encoder** with layers of size (28 × 28)-400-200-100-50-25-6 and **a symmetric decoder**.
The **six units in the code layer were linear** and all the **other units were logistic**.
The network was **trained on 20,000 images** and **tested on 10,000 new images**.
The autoencoder discovered how to convert each 784-pixel image into six real numbers that allow almost perfect reconstruction (Fig. 2A).
PCA gave much worse reconstructions.

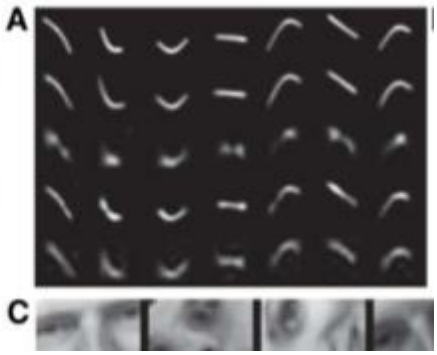


Fig. 2. (**A**) Top to bottom: Random samples of curves from the test data set; reconstructions produced by the six-dimensional deep autoencoder; reconstructions by "logistic PCA" (*8*) using six components; reconstructions by logistic PCA and standard PCA using 18 components. The average squared error per image for the last four rows is 1.44, 7.64, 2.45, 5.90. (**B**) Top

Without pretraining, the very deep **autoencoder always reconstructs the average of the training data**, even after prolonged fine-tuning(8).

Shallower autoencoders with **a single hidden layer between the data and the code** can **learn without pretraining**, but **pretraining** greatly **reduces** their total **training time** (8).
When **the number of parameters is the same**, **deep autoencoders** can produce **lower reconstruction errors** on test data **than shallow** ones, but this **advantage disappears as the number of parameters increases** (8).

Next, we used **a 784-1000-500-250-30 autoencoder to extract codes for** all the **handwritten digits in the MNIST training set** (11).
The Matlab code that we used for the pretraining and fine-tuning is available in (8).
Again, all units were logistic except for the 30 linear units in the code layer.
After fine-tuning on all **60,000 training images**, the autoencoder was **tested on 10,000 new images** and produced much **better** reconstructions than did PCA (Fig. 2B).
A two-dimensional autoencoder produced a better visualization of the data than did the first two principal components (Fig. 3).

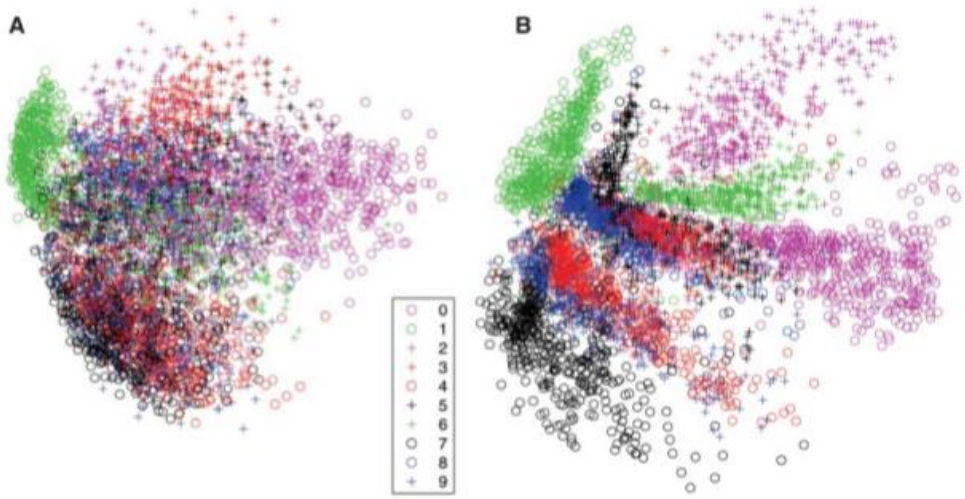


Fig. 3. (A) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (B) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (8).

We also used **a 625-2000-1000-500-30 autoencoder** with **linear input units** to **discover 30 dimensional codes for grayscale image patches** that were derived from the Olivetti face data set (12). The autoencoder clearly outperformed PCA (Fig. 2C).

When trained on documents, autoencoders produce codes that allow fast retrieval. We represented each of 804,414 newswire stories (13) as a vector of document-specific probabilities of the 2000 commonest word stems, and we trained a 2000-500-250-125-10 autoencoder on half of the stories with the use of the multiclass cross-entropy error function $[-\sum_i p_i \log \hat{p}_i]$ for the fine-tuning.
The 10 code units were linear and the remaining hidden units were logistic.
When the **cosine of the angle between two codes was used to measure similarity**, the **autoencoder** clearly **outperformed latent semantic analysis (LSA)** (14), a well-known document retrieval method based on PCA (Fig. 4).

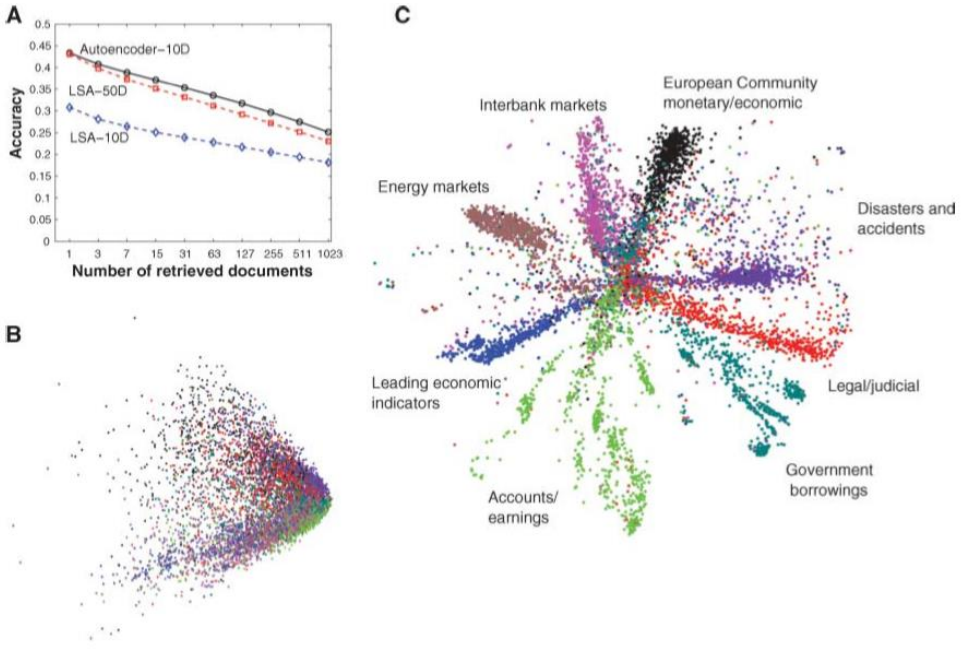


Fig. 4. (A) The fraction of retrieved documents in the same class as the query when a query document from the test set is used to retrieve other test set documents, averaged over all 402,207 possible queries. (B) The codes produced by two-dimensional LSA. (C) The codes produced by a 2000-500-250-125-2 autoencoder.

Autoencoders (8) also outperform local linear embedding, a recent nonlinear dimensionality reduction algorithm (15).

Layer-by-layer pretraining can also be used **for classification and regression.**
On a widely used version of the MNIST handwritten digit recognition task, the best reported error rates are 1.6% for randomly initialized backpropagation and 1.4% for support vector machines.
After layer-by-layer pretraining in a 784-500-500-2000-10 network, backpropagation using steepest descent and a small learning rate achieves 1.2% (8).
Pretraining helps generalization because **it ensures that most of the information in the weights comes from modeling the images.** The very limited **information in the labels** is used only to **slightly adjust the weights** found by pretraining.

It has been obvious since the 1980s that backpropagation through deep autoencoders would be very effective for nonlinear dimensionality reduction, provided that computers were fast enough, data sets were big enough, and the initial weights were close enough to a good solution. All three conditions are now satisfied.
Unlike nonparametric methods (15, 16), **autoencoders give mappings in both directions between the data and code spaces,** and they can be **applied to very large data sets because** both the **pretraining and the fine-tuning scale linearly in time and space with the number of training cases.**

创新点：(https://blog.csdn.net/Lazyinit/article/details/103556933)

运用神经网络进行降维；

概括介绍自编码器编码、解码过程;

将原有的高清图片压缩成信息量小，但又包含了图片所有特征的图片（提取最具代表性的信息），解压时再将特征图片还原成最初的图片；