Dimension reduction

Emma Yu

Feb 8, 2017

Austin ACM SIGKDD
Advanced Machine Learning with Python (Session 13)

Today’s slides and the notebook:
https://github.com/ExLupi/ACM_SIGKDD_dimension_reduction

Outline

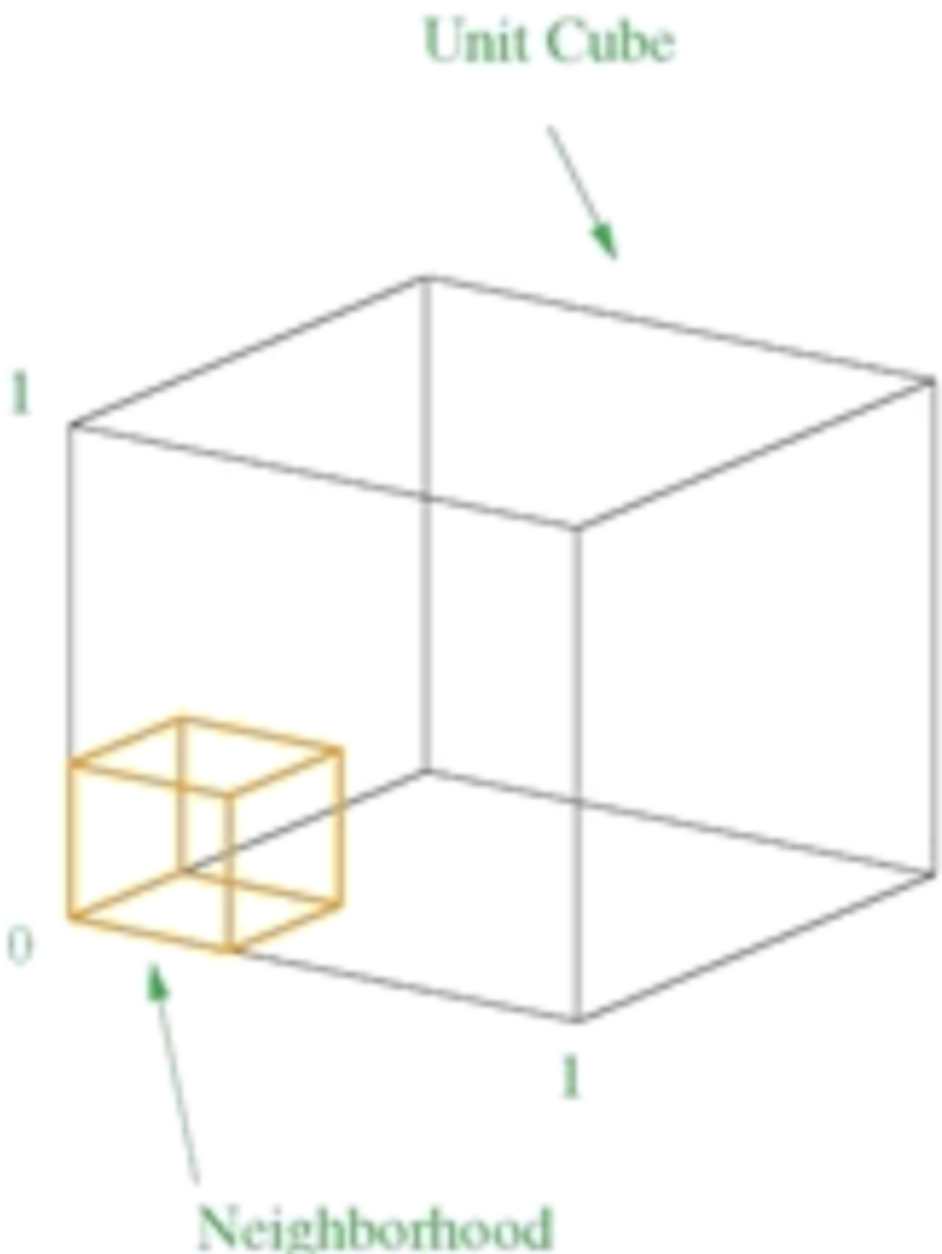
- Why dimension reduction?
- Principal Component Analysis (PCA)
- PCA demo with the MNIST dataset
- Other techniques — Probabilistic PCA, LDA, Kernel PCA and Autoencoders

What are the problems we face dealing with high-dimensional data?

Problems dealing with high-dimensional data

- Expensive to store and expensive to compute (e.g. larger matrixes to inverse)
- Easier to overfit the data (too many degrees of freedom)
- Hard to visualize (we live in a 3D world)
- The curse of dimensionality — it's hard to find structures in high-dimensional data

The Curse of Dimensionality



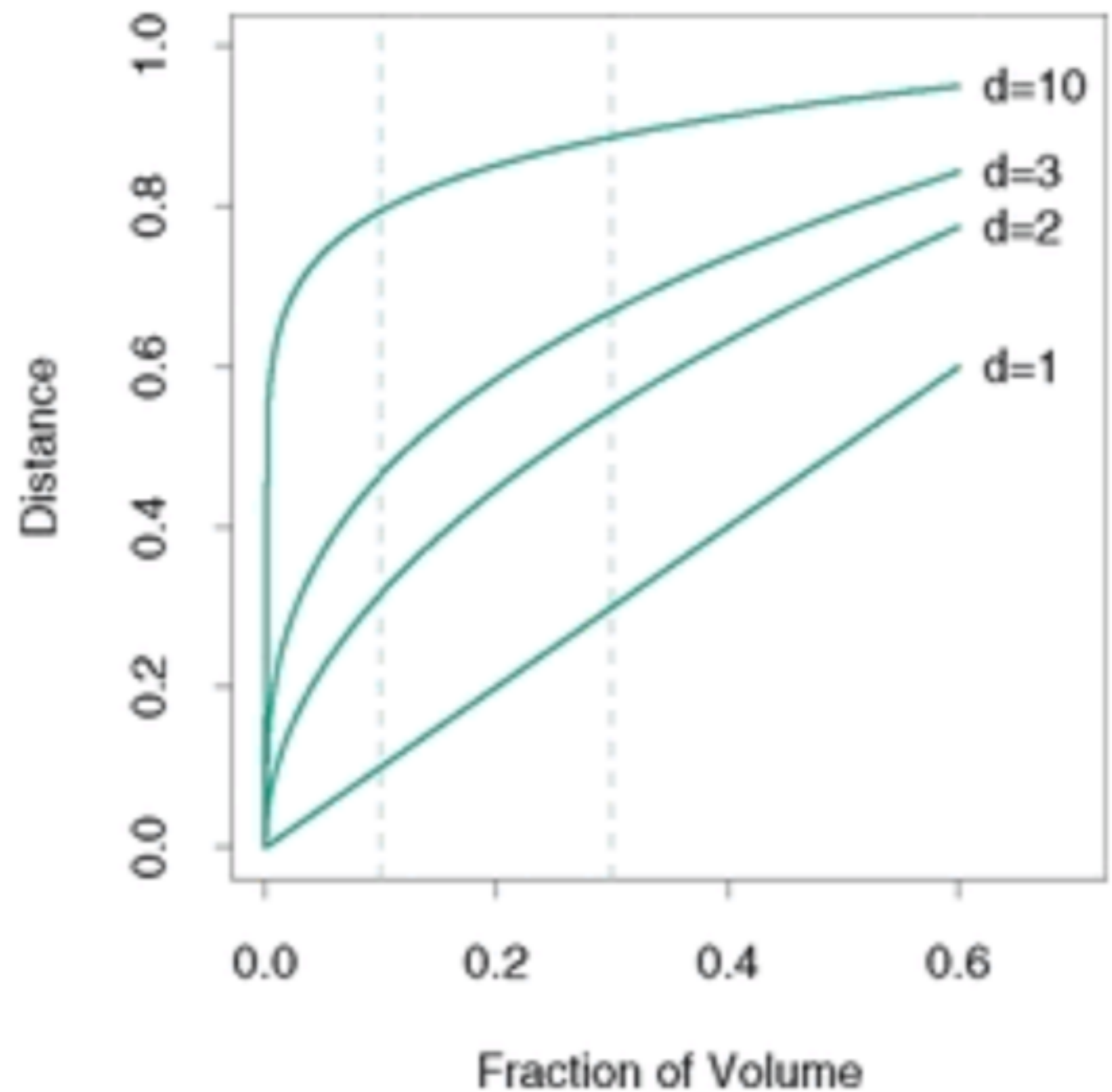


Image credit: Dana Ballard, lecture notes

The Curse of Dimensionality

Distance-based algorithms (e.g. KNN) do not work well in high dimensional space

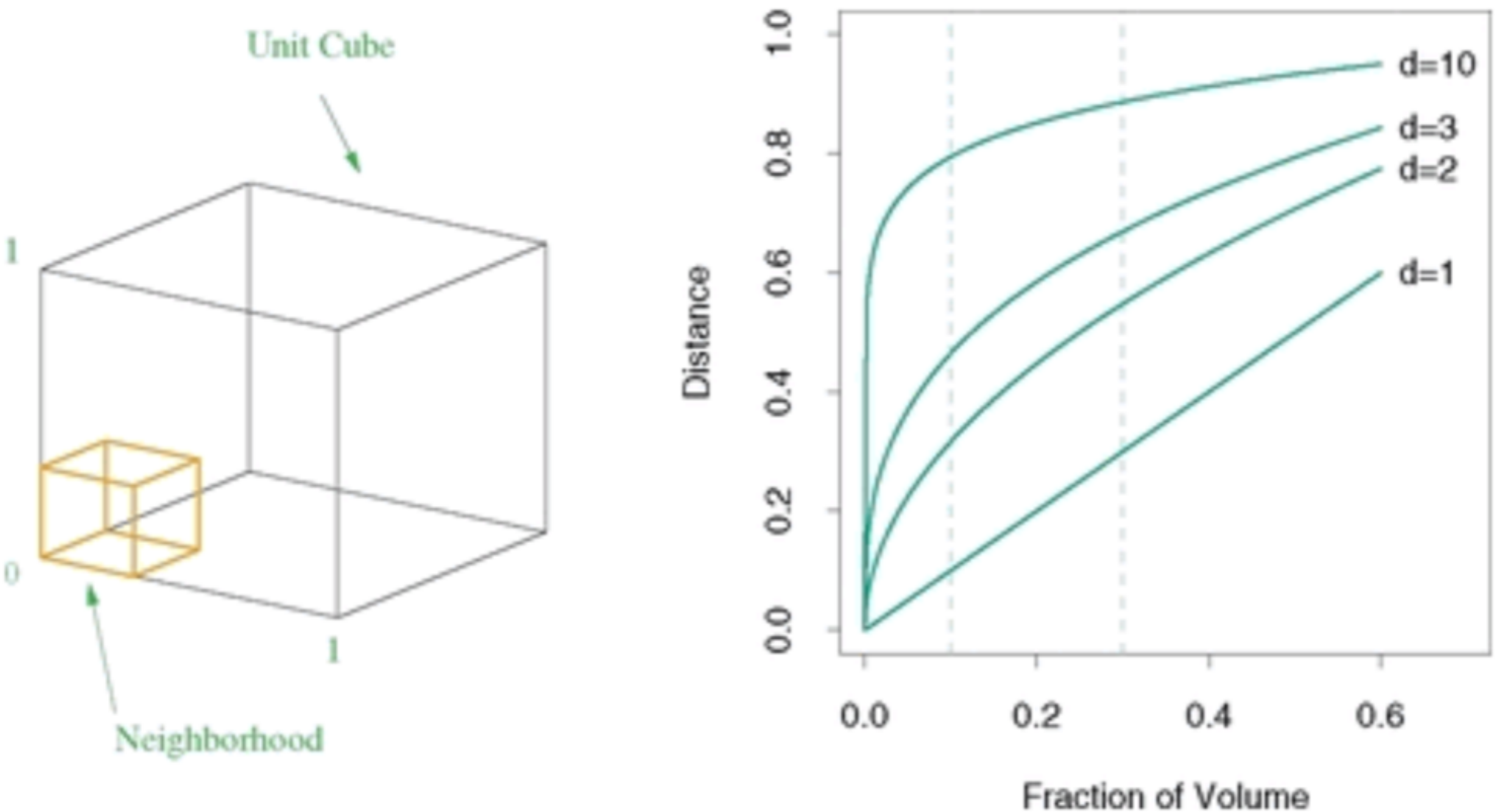


Image credit: Bishop 2006

The actual data usually lies near a low-dimension manifold

- We **moved and rotated** a 100*100 pixel image of 3 to make this dataset. Each image is represented by a point in the **10,000-dimensional** data space. What is the actual dimension of data?

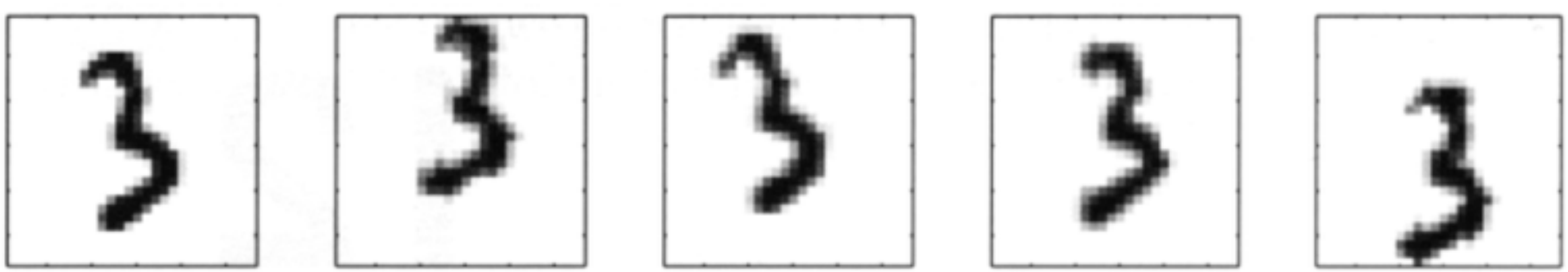


Image credit: Bishop 2006

The actual data usually lies near a low-dimension manifold

- We **moved and rotated** a 100*100 pixel image of 3 to make this dataset. Each image is represented by a point in the **10,000-dimensional** data space. What is the actual dimension of data?

3 - x, y and rotation

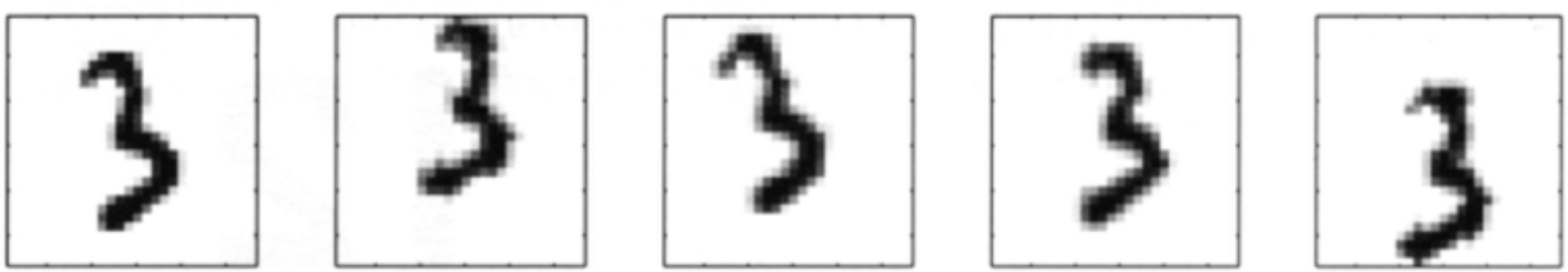


Image credit: Bishop 2006

Dimension reduction:
Deriving a set of degrees of freedom which can be used to best re-express the dataset

Is there another basis, which is a linear combination of the original basis, that preserves the variance of our dataset?

Is there another basis, which is a linear combination of the original basis, that preserves the variance of our dataset?

Yes! It's made of principal components!

Principal Component Analysis

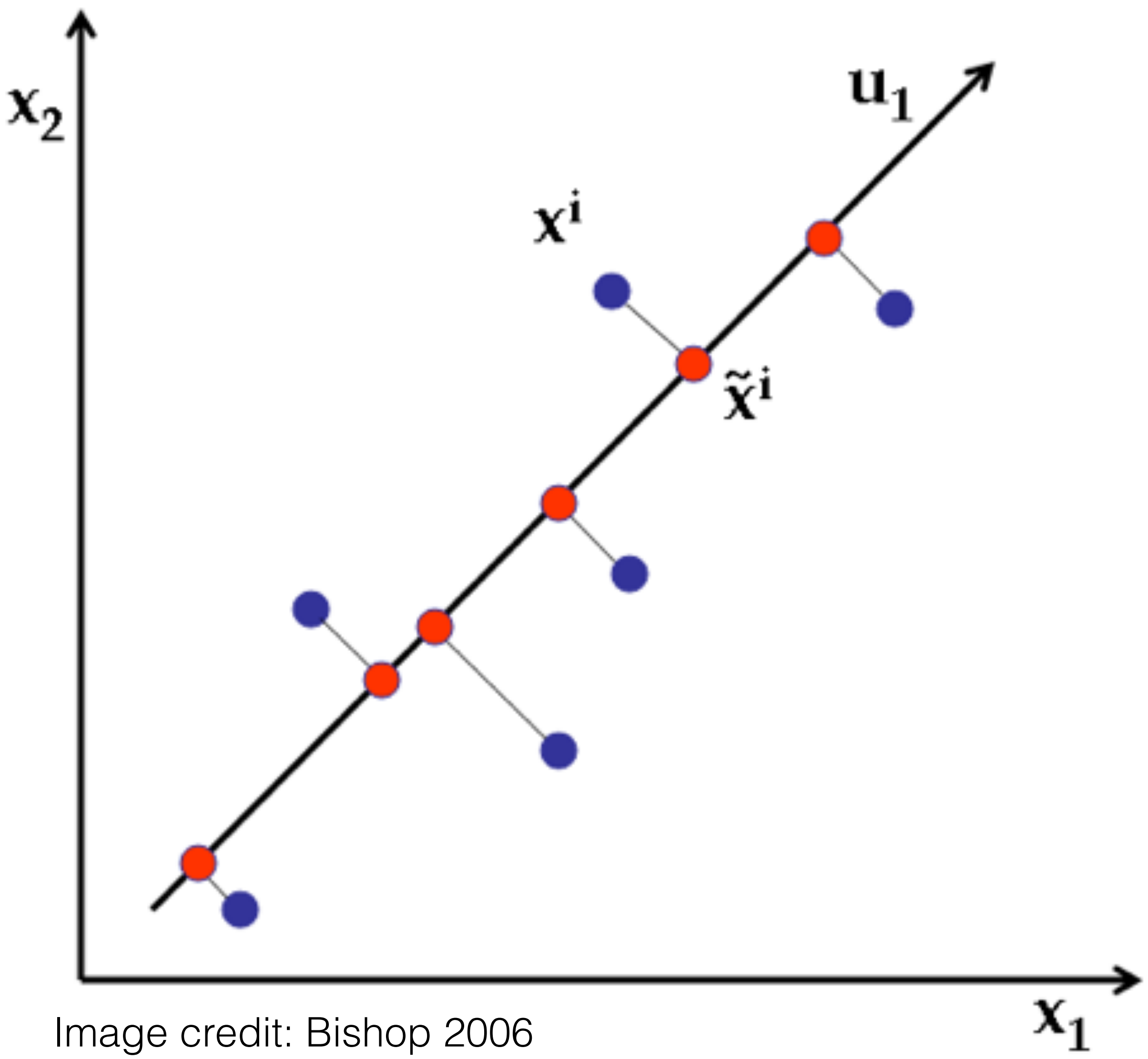


Image credit: Bishop 2006

- maximizes the variance of projected data
- minimizes the average projection cost (mean squared reconstruction error)

- Also known as KLT transform, empirical orthogonal functions, quasi harmonic modes (Wikipedia)

How to find the principal components?

- Define a unit vector in the D-dimensional space $\mathbf{u}_1$
- Projected value of a data point onto this direction is $\mathbf{u}_1^{\mathrm{T}}\mathbf{x}_n$. Variance of projected data is $\mathbf{u}_1^{\mathrm{T}}\mathbf{S}\mathbf{u}_1$, where S is the covariance matrix $\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \overline{\mathbf{x}})(\mathbf{x}_n - \overline{\mathbf{x}})^{\mathrm{T}}.$
- Goal: to maximize $\mathbf{u}_1^{\mathrm{T}}\mathbf{S}\mathbf{u}_1$ under the constraint $\mathbf{u}_1^{\mathrm{T}}\mathbf{u}_1 = 1$.
- Solve the optimization problem, and we get:

$$\mathbf{S}\mathbf{u}_1 = \lambda_1\mathbf{u}_1$$

Eigenvalues and Eigenvectors

Eigenvalue

$$A\mathbf{x} = \lambda\mathbf{x}$$

Eigenvector

$$A = \begin{bmatrix} 2 & 0 \\ 0 & -1 \end{bmatrix} \quad \mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Eigenvalue

$$A\mathbf{x}_1 = \begin{bmatrix} 2 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix} = 2\begin{bmatrix} 1 \\ 0 \end{bmatrix} = 2\mathbf{x}_1$$

Eigenvector

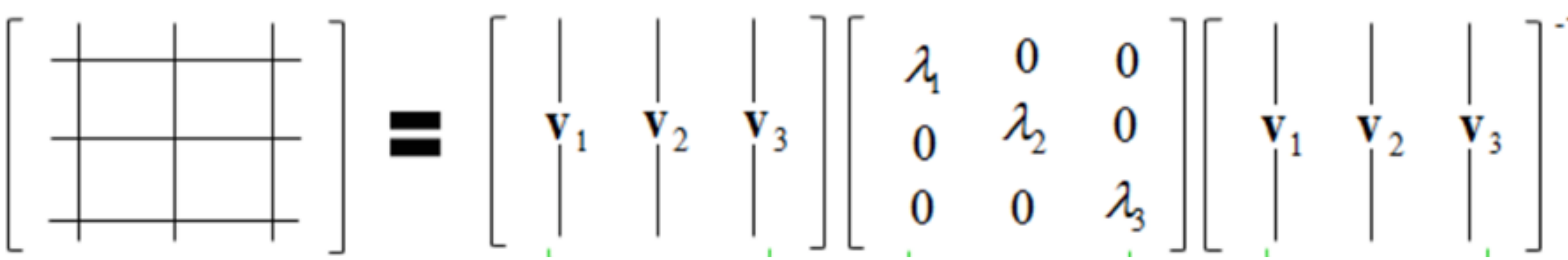


Image credit: datinker.com

- Principal components are the largest eigenvectors of the covariance matrix

- Value of distortion = sum of all the other eigenvalues

- Cost of full eigenvector decomposition - $O(D^3)$

- PCA assumes the data is centered at zero and is sensitive to relative scaling - one wants to standardize the data first!

- PCA prefers multiple medium sized errors than one large error => sensitive to outliers

- It works great for unlabeled data. For labeled data LDA sometimes works better

- The projection to the two/three largest principal components is commonly used for visualization

Demo time!

PCA and SVD

- Singular value decomposition (SVD) calculates the **singular vectors** of the **data matrix**
- PCA finds the principal components by solving for the **eigenvectors** of the **covariance matrix**
- The covariance matrix is simply $\mathbf{X}^\top\mathbf{X}/(n-1)$
- PCA is usually solved by SVD to avoid calculating the covariance matrix

PCA and SVD

- SVD factors a matrix into $\mathbf{M} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^*$
- $\mathbf{U}$ and $\mathbf{V}^*$ are two unitary matrixes and $\boldsymbol{\Sigma}$ is a diagonal matrix
- The right singular vectors $\mathbf{V}$ are simply the principal directions (eigenvectors of the covariance matrix)
- The diagonal terms in $\boldsymbol{\Sigma}$ are related to the eigenvalues of the covariance matrix via $\lambda_i = s_i^2/(n-1)$

Probabilistic PCA

- PCA is also the maximum likelihood solution of a probabilistic latent variable model

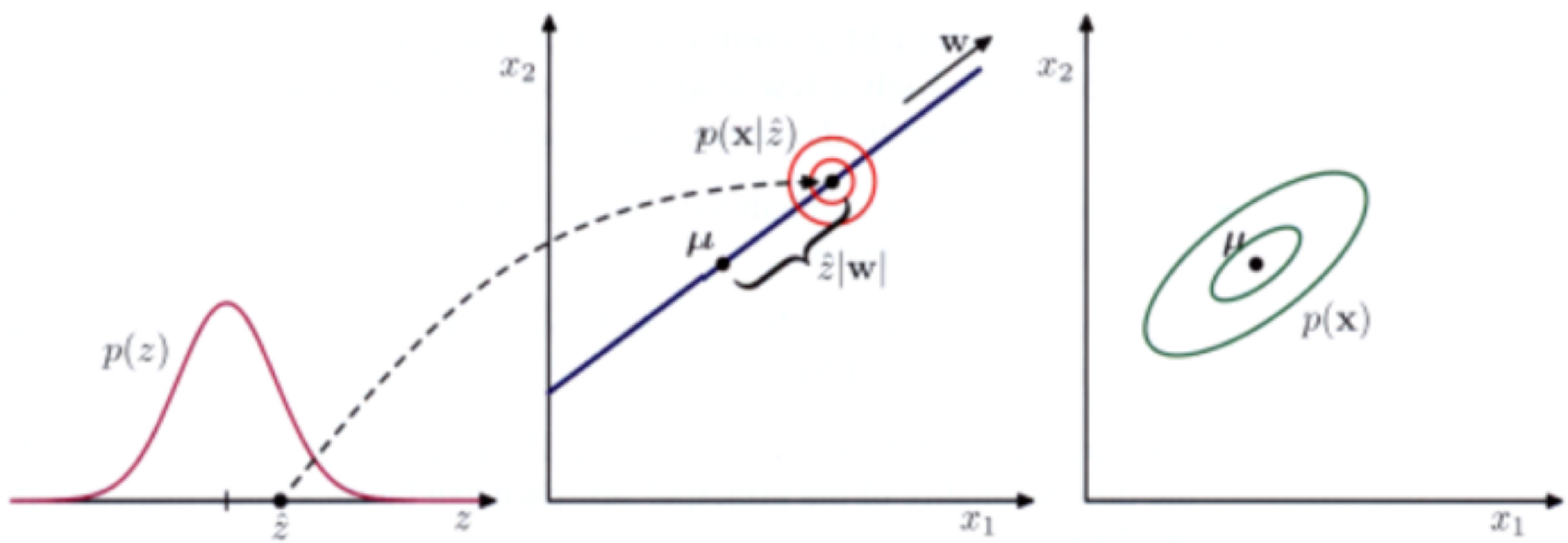


Image credit: Bishop 2006

- Combined with EM algorithm, probabilistic PCA allows us to deal with missing data

Linear Discriminant Analysis (LDA, or Fisher's linear discriminant)

- Basic idea: find a basis that maximizes (between class variance)/(within class variance)
- Similar to PCA: LDA uses a linear combination of features. All classes are assumed to be normally distributed and have equal class covariance.
- Unlike PCA: LDA models the differences between classes of data

PCA vs LDA

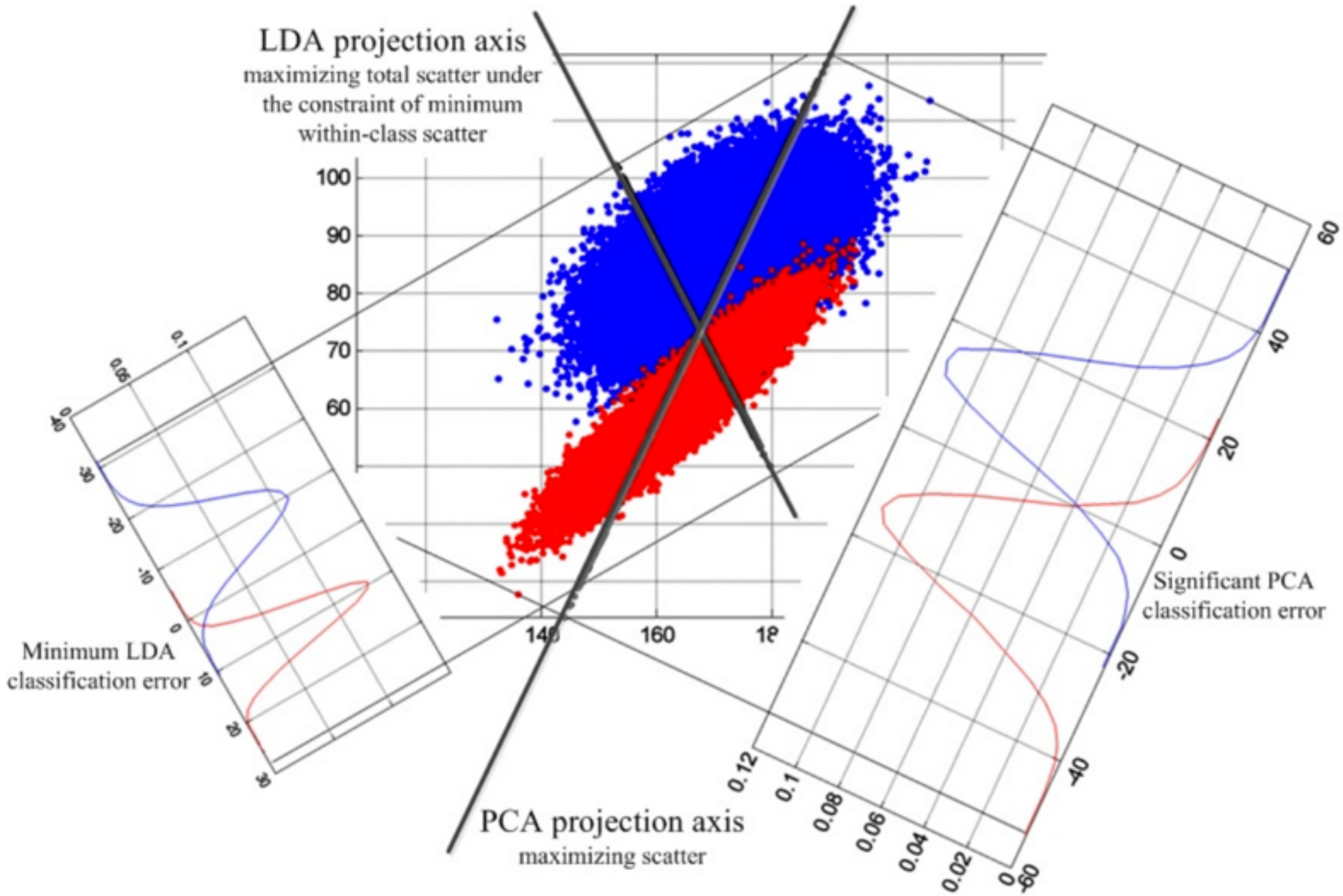


Image credit: Liam Williams's Master thesis

What if the data is not linearly separable?

Kernel PCA

- Use the Kernel trick to construct nonlinear mappings that maximize the variance in the data

- Instead of calculating the covariance matrix $\frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_i\mathbf{x}_i^{\mathsf{T}}$ Calculate the covariance of the transformed data

$$\frac{1}{m}\sum_{i=1}^{m}\Phi(\mathbf{x}_i)\Phi(\mathbf{x}_i)^{\mathsf{T}}$$

- After some algebra, it turns out one only needs to solve the eigenvalue problem for the kernel function

$$\mathbf{K}\mathbf{a}_i = \lambda_i N \mathbf{a}_i \qquad k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^{\mathrm{T}}\phi(\mathbf{x}_m)$$

- The projection onto the eigenvector is simply

$$y_i(\mathbf{x}) = \sum_{n=1}^{N} a_{in} k(\mathbf{x}, \mathbf{x}_n)$$

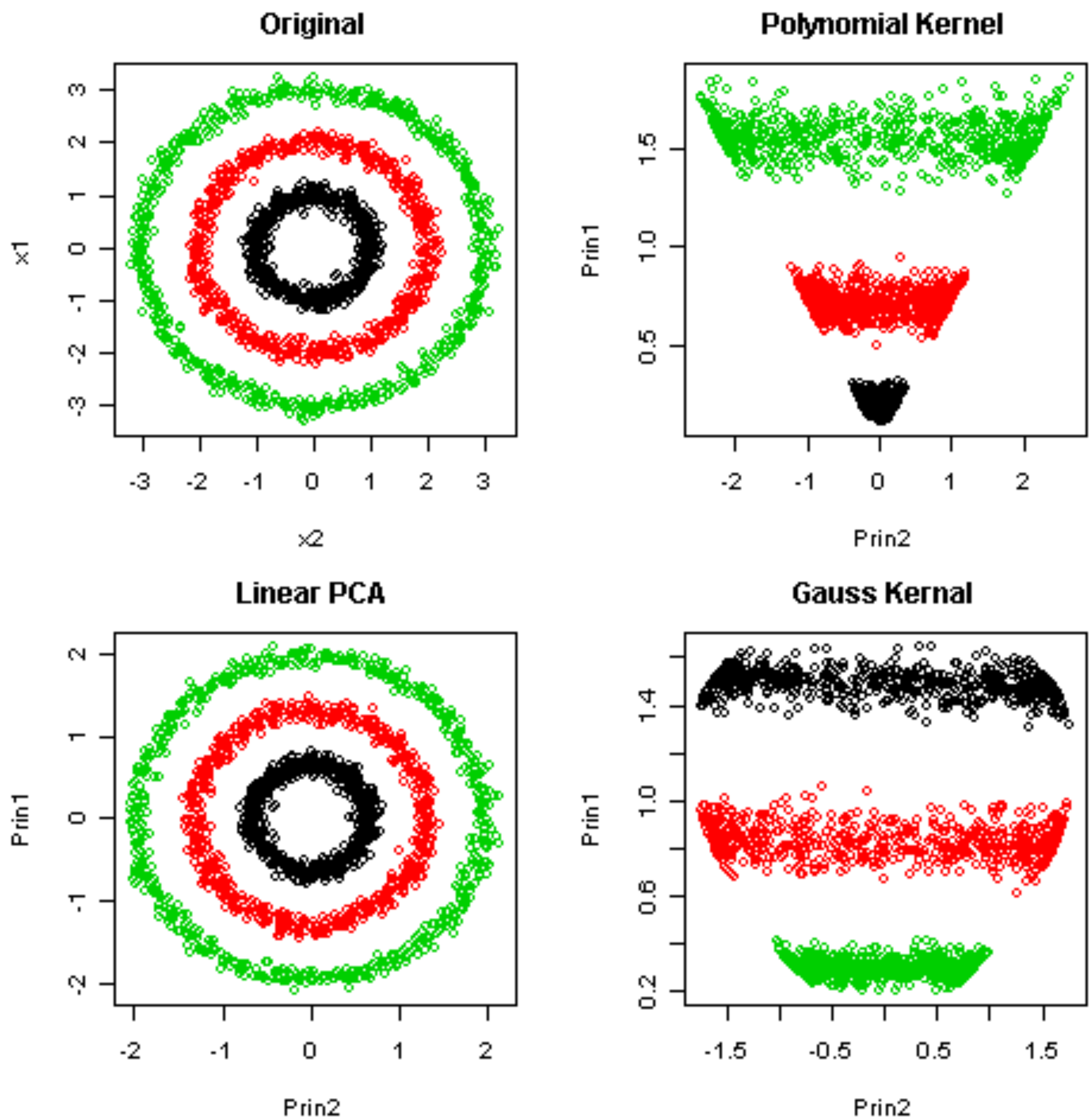
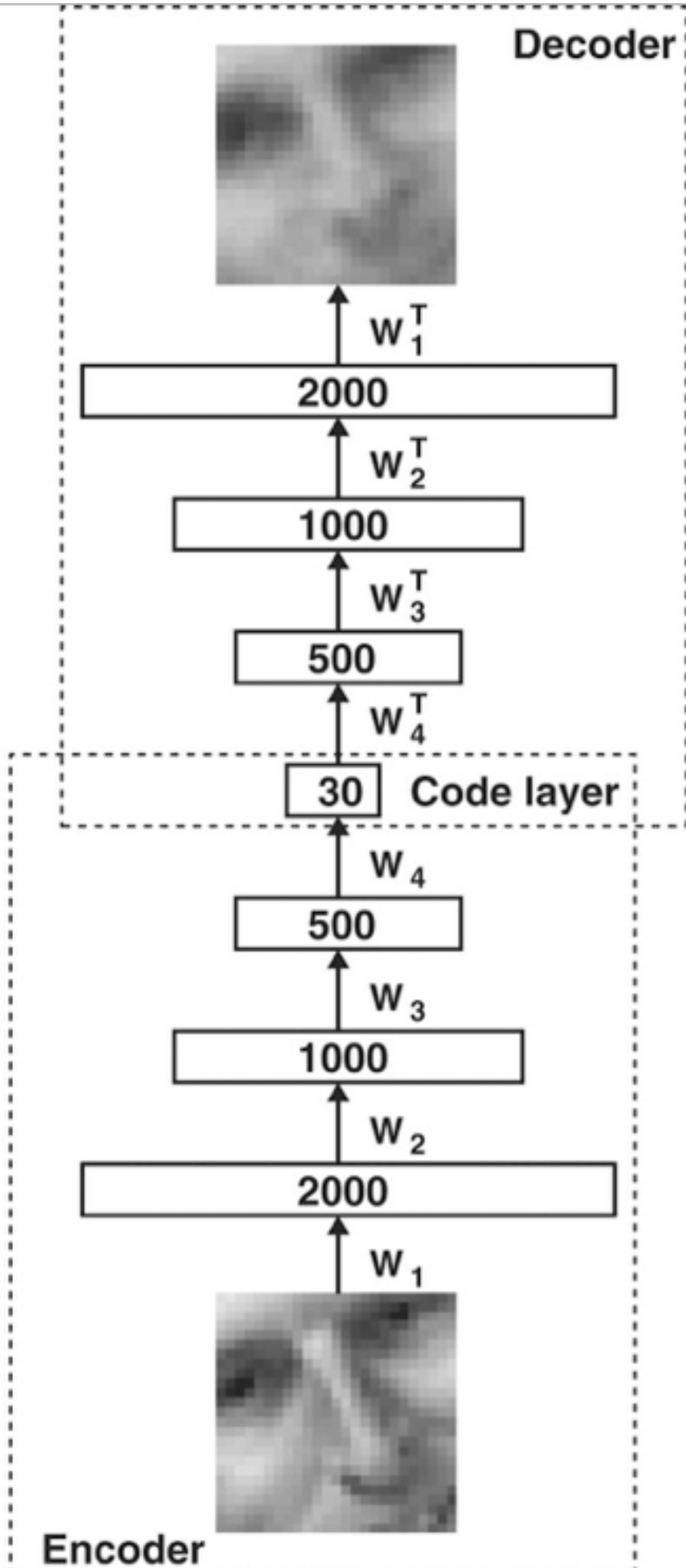


Image credit: sas-programming.com



Autoencoders

- Non-linear activation function and multiple layers!
- Train to minimize reconstruction errors
- Pre-training layer-by-layer seems to be the key

Image credit: Hinton and Salakhutdinov, Science, 2006

B

Top to bottom: original data, reconstructions from a 30-D auto encoder, 30-D logistic PCA, and 30-D standard PCA

Image credit: Hinton and Salakhutdinov, Science, 2006

Reference

- Bishop 2006, Pattern recognition and machine leaning

- Hinton and Salakhutdinov, Science, 2006, Reducing the dimensionality of data with neural networks

- Wikipedia, Quora and Stackexchange

Contact me at:
emmayuastro@gmail.com
emmayu.com