Deep Learning (CS324)

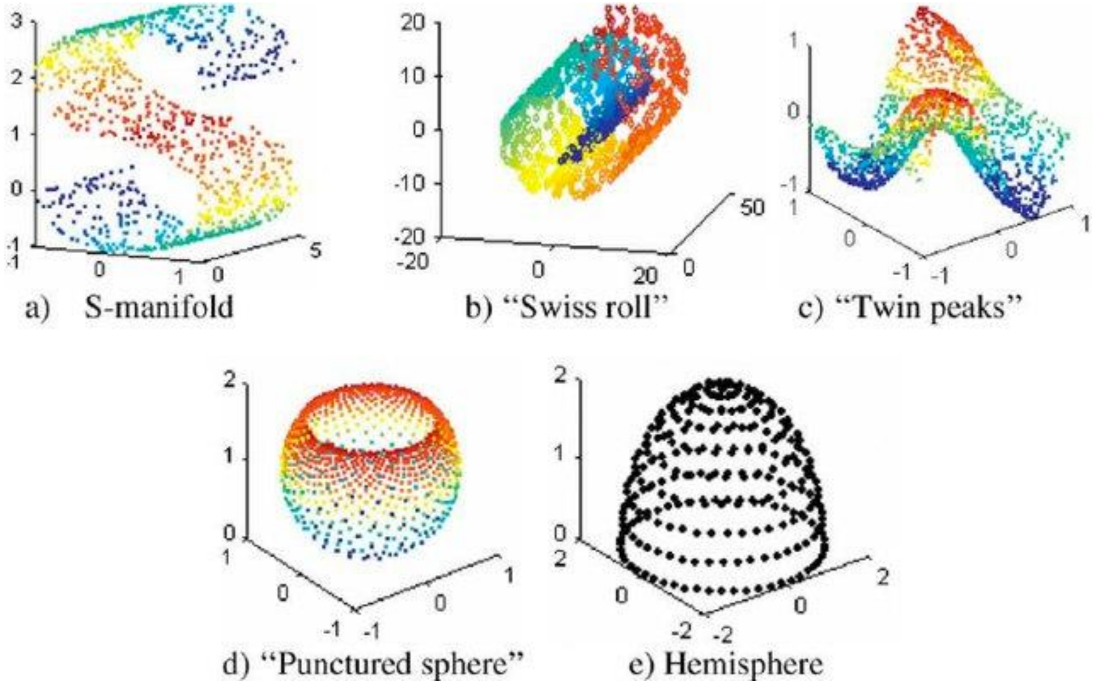
7. Autoencoders*

Prof. Jianguo Zhang
SUSTech

*Based on http://www.deeplearningbook.org chapters 5.11.3, 13.5 & 14 + sources mentioned in slides

Manifold hypothesis

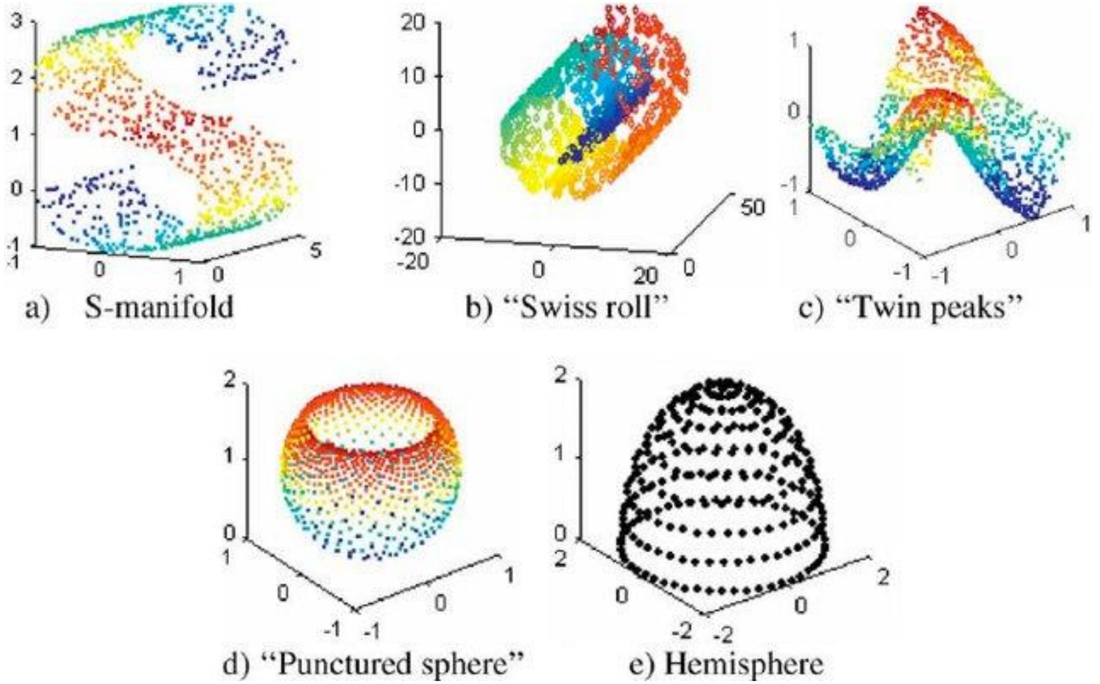
- Real-world data lives in a low-dimensional non-linear manifold



Karbauskaitė et al. "Topology preservation measures in the visualization of manifold-type multidimensional data", 2009

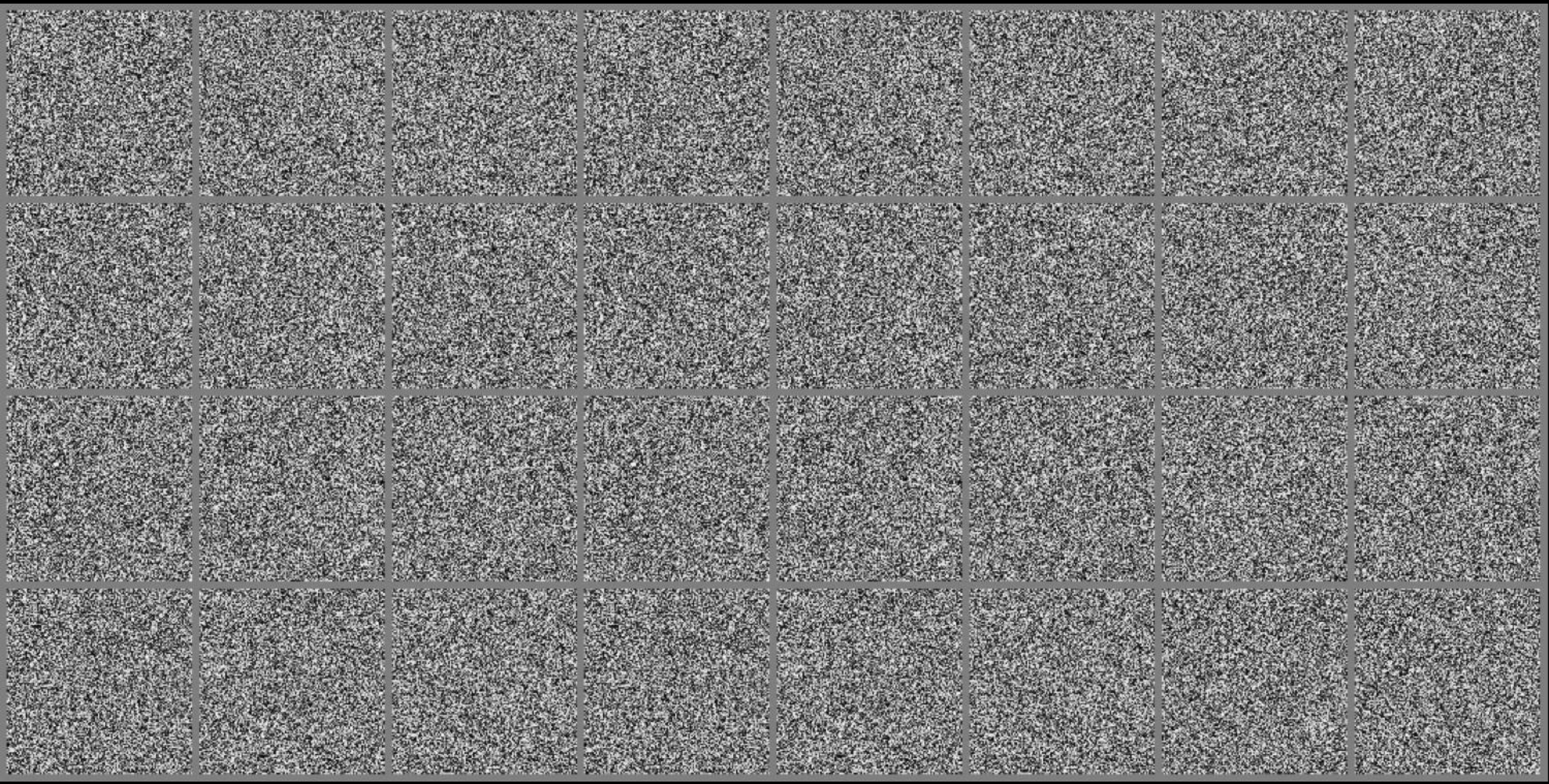
Manifold hypothesis

- In other words, the data is concentrated with high probability in a small non-linear region of space



Karbauskaitė et al. "Topology preservation measures in the visualization of manifold-type multidimensional data", 2009

Uniformly sampled images from the space of 256x256 pixels



Real-world data example: images of faces

QMUL faces dataset: 133 facial images covering a view of +/-90 degrees in yaw and +/-30 degrees in tilt at 10 degrees increment

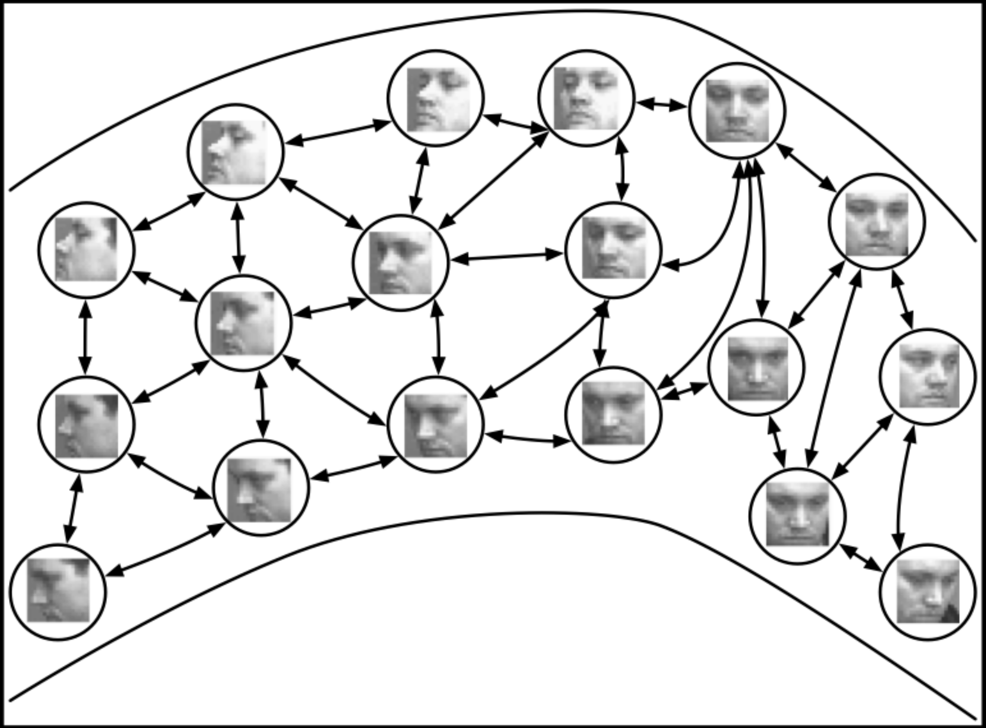
Real-world data example: images of faces

Human face has about **50 muscles**, **3 cartesian coordinates** (translations) and **3 Euler angles** (rotations), so the manifold of faces of a person has less than **56 dimensions**

Real-world data example: images of faces

PROBLEM: how to find these dimensions/coordinates automatically?

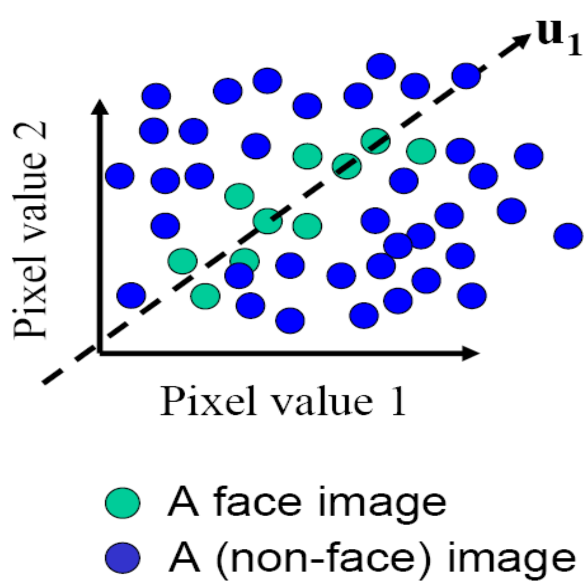
Real-world data example: images of faces



PROBLEM: how to find these dimensions/coordinates automatically?

Principal Component Analysis

- The goal is to find the directions that best explain the variations of the data
- To find several directions (a subspace) such that the variance of the projected data in the subspace is maximized.



Principal Component Analysis

- Given: N data points $\mathbf{x_1}, \ldots, \mathbf{x_N}$ in R^d
- Choose an unit vector $\mathbf{u}$ in R^d that captures the most data variance
- Therefore, we could construct a new set of features by projecting the data onto those directions:

$$u(\mathbf{x}_i) = \mathbf{u}^T(\mathbf{x}_i - \boldsymbol{\mu})$$

($\boldsymbol{\mu}$: mean of data points)

Principal Component Analysis

- Direction that maximizes the variance of the projected data:

Maximize $$\frac{1}{N}\sum_{i=1}^{N} \underbrace{\mathbf{u}^{\mathrm{T}}(\mathbf{x}_i - \mu)}_{\text{Projection of data point}}(\mathbf{u}^{\mathrm{T}}(\mathbf{x}_i - \mu))^{\mathrm{T}}$$ subject to ||**u**||=1

$$= \mathbf{u}^{\mathrm{T}} \underbrace{\left[1/N \sum_{i=1}^{N} (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^{\mathrm{T}} \right]}_{\text{Covariance matrix of data}} \mathbf{u}$$

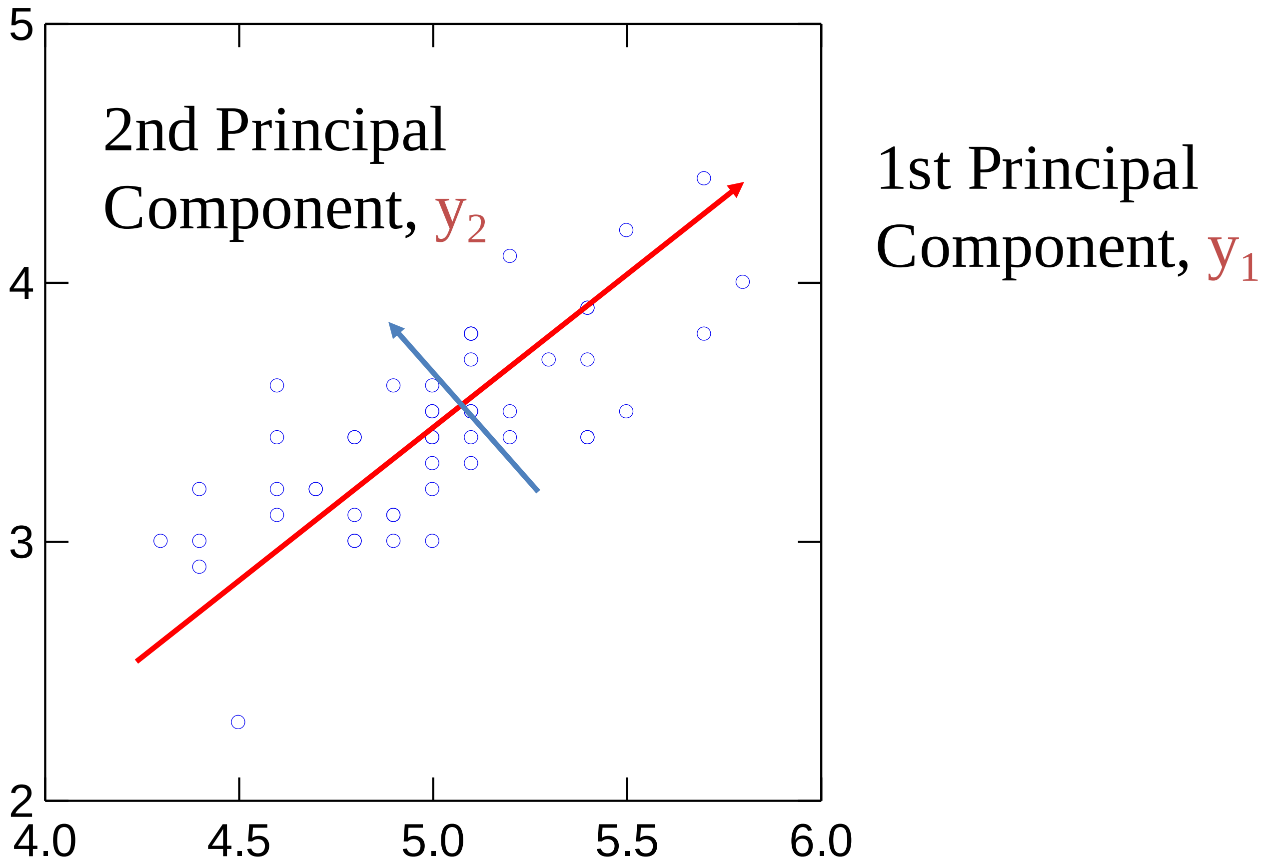
$$= \mathbf{u}^{\mathrm{T}} \Sigma \mathbf{u}$$

The direction that maximizes the variance is the eigenvector associated with the largest eigenvalue of Σ

Implementation Issue

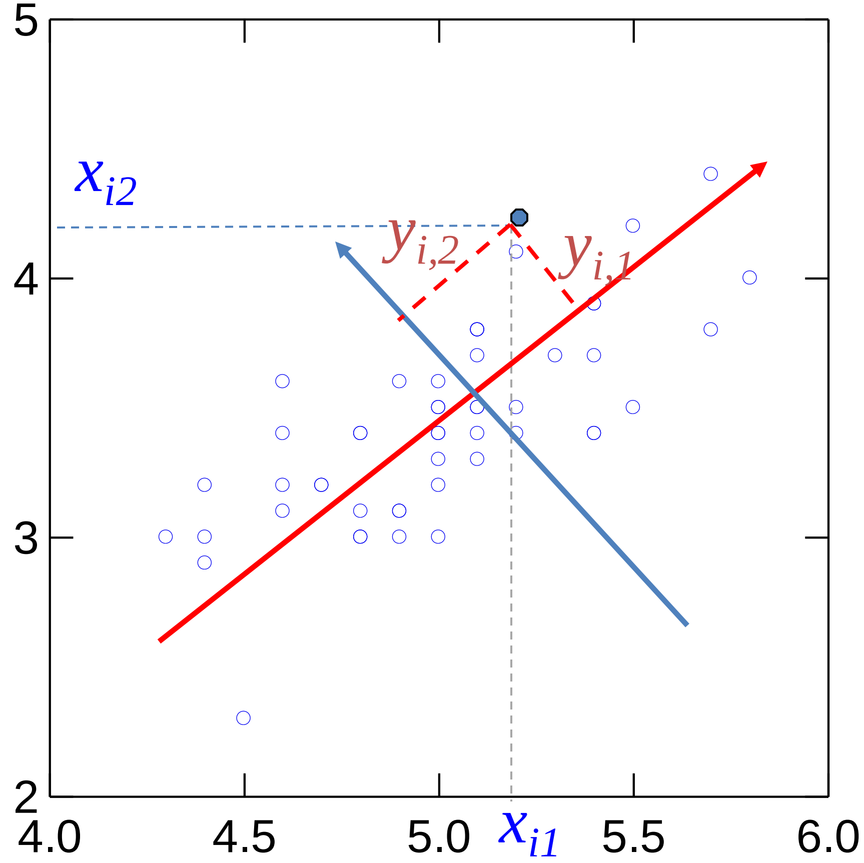
- Covariance matrix is huge (d^2 for d dimensions)
- But typically # examples << d
- Simple trick
  - $\mathbf{X}$ is matrix of centralised training data (each row is an observation)
  - Solve for eigenvectors $\mathbf{u}$ of $\mathbf{X}\mathbf{X}^T$ instead of $\mathbf{X}^T\mathbf{X}$
  - Then $\mathbf{X}^T\mathbf{u}$ is eigenvector of covariance $\mathbf{X}^T\mathbf{X}$
  - May need to normalize (to get unit length vector)

Principal Component Analysis



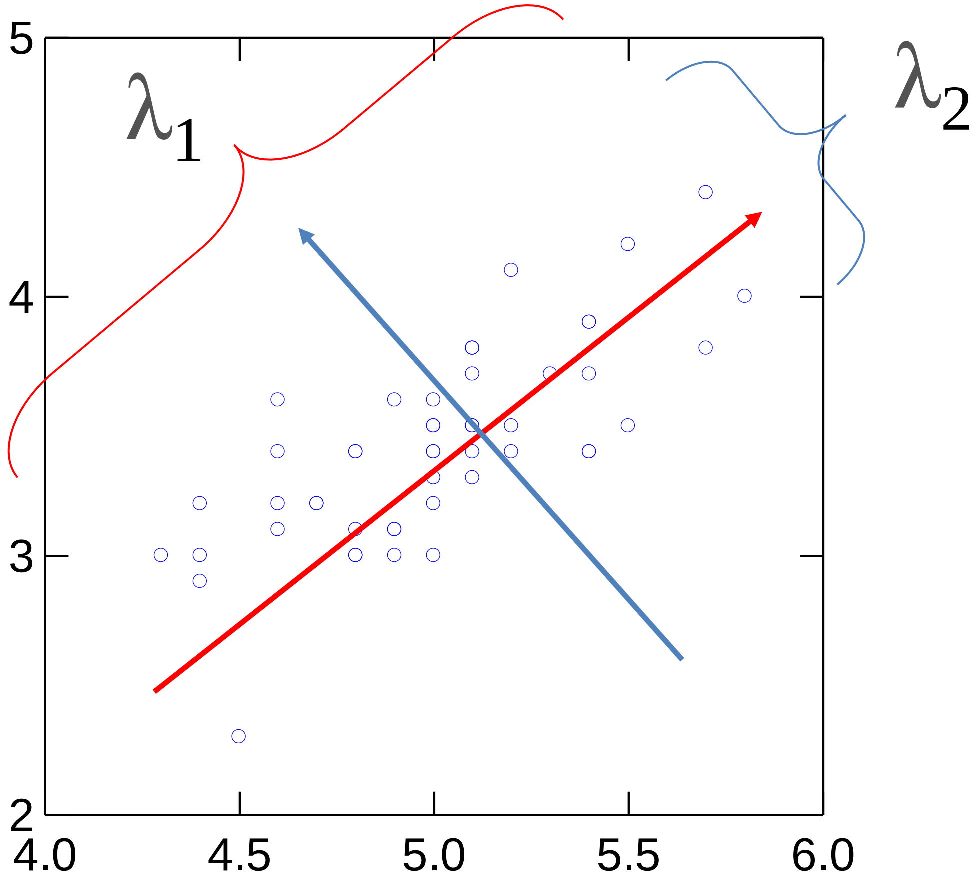
The meaning of directions -- Eigen vectors

PCA Scores



The geometrical meaning of projection on those directions

PCA Eigenvalues

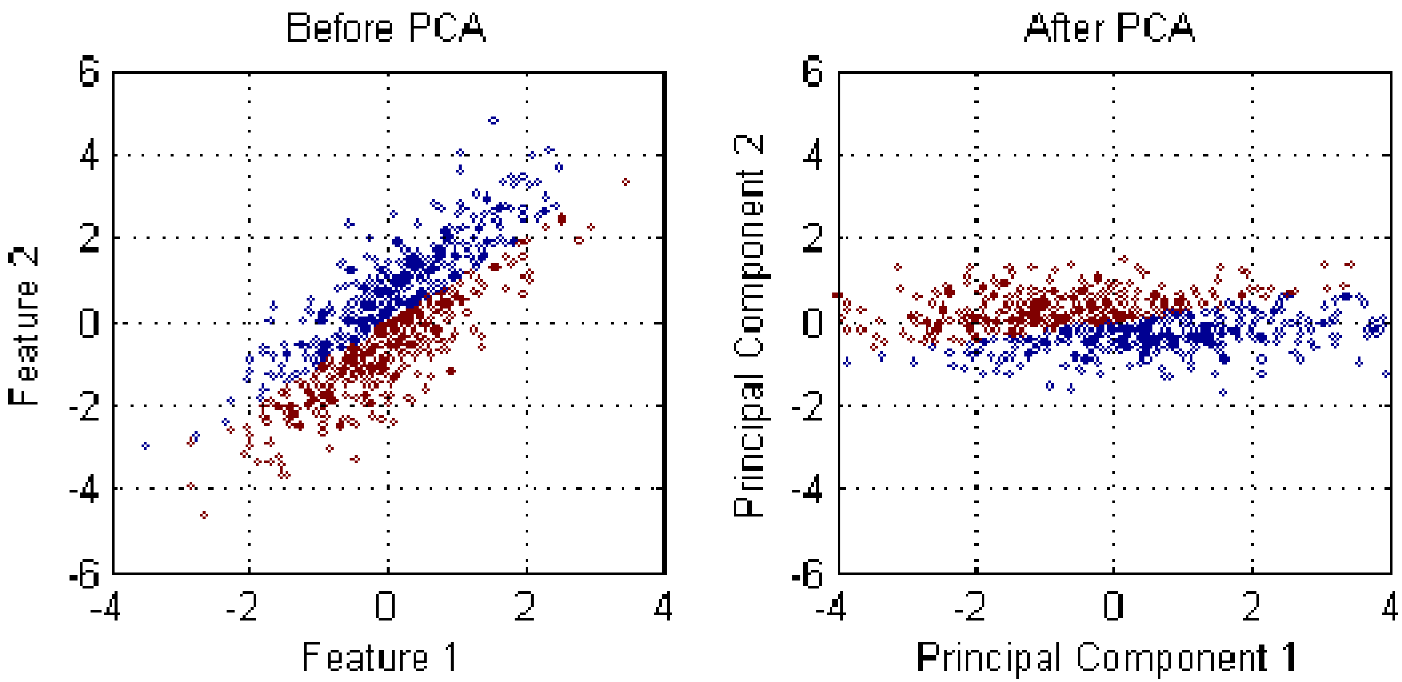


PCA - Summary

- PCA defines an orthogonal linear transformation from an input **x** to a representation *z = f(x)*
- More formally, if **X** is the where each row corresponds to a data sample, $\mathbf{T} = \mathbf{X}\mathbf{W}$ denotes the full principal components decomposition, where **W** is the matrix whose columns are the eigenvectors of $\mathbf{X}^T\mathbf{X}$
- To reduce the dimensionality of the input we use only the first *k* eigenvectors, which yields $\mathbf{T}_k = \mathbf{X}\mathbf{W}_k$

Principal components analysis

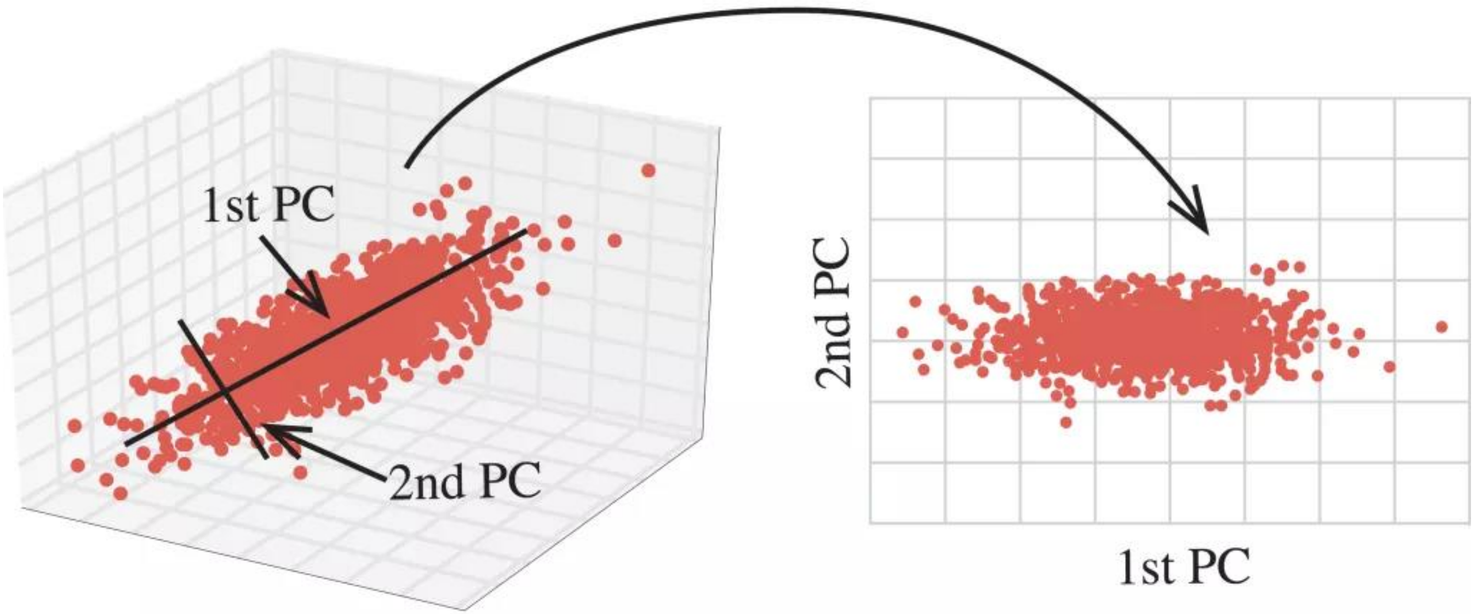
- PCA defines an orthogonal linear transformation from an input **x** to a representation $z = f(x)$



Source: Li et al., A Robust Deep Model for Improved Classification of AD/MCI Patients, 2015

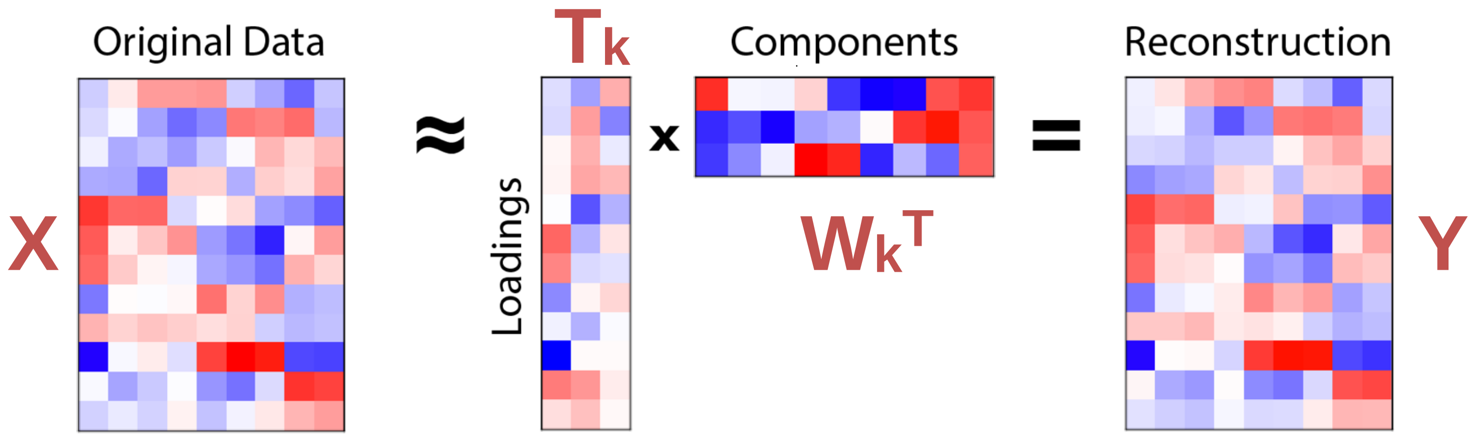
Principal components analysis

- PCA defines an orthogonal linear transformation from an input **x** to a representation $z = f(x)$



Source: http://blog.kaggle.com/2017/04/10/exploring-the-structure-of-high-dimensional-data-with-hypertools-in-kaggle-kernels/

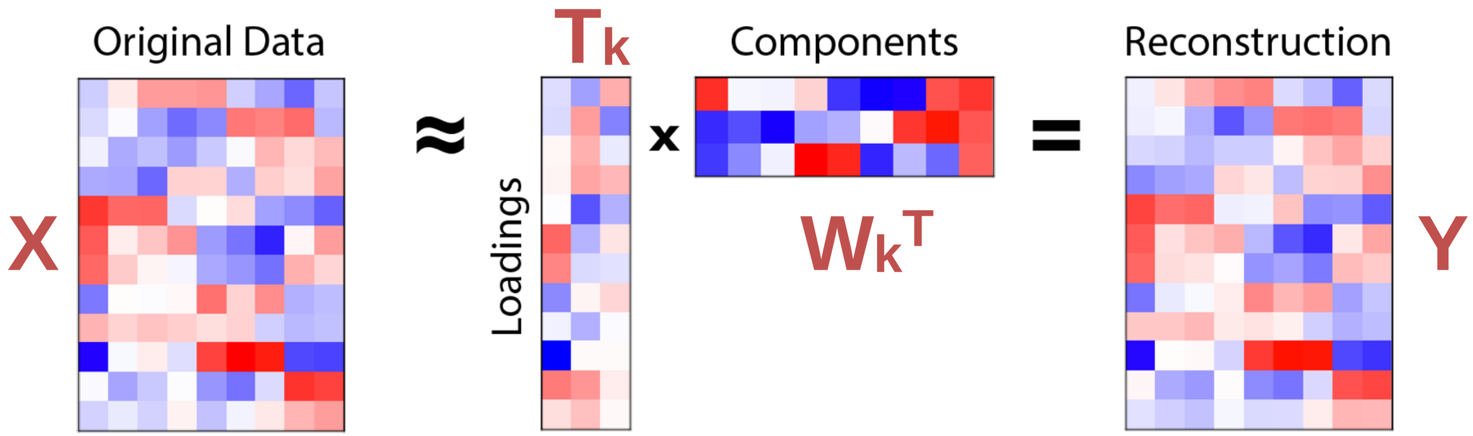
Input reconstruction with PCA



$$Y = T_k W_k^T$$

Source: http://alexhwilliams.info/itsneuronalblog/2016/03/27/pca/

Input reconstruction with PCA

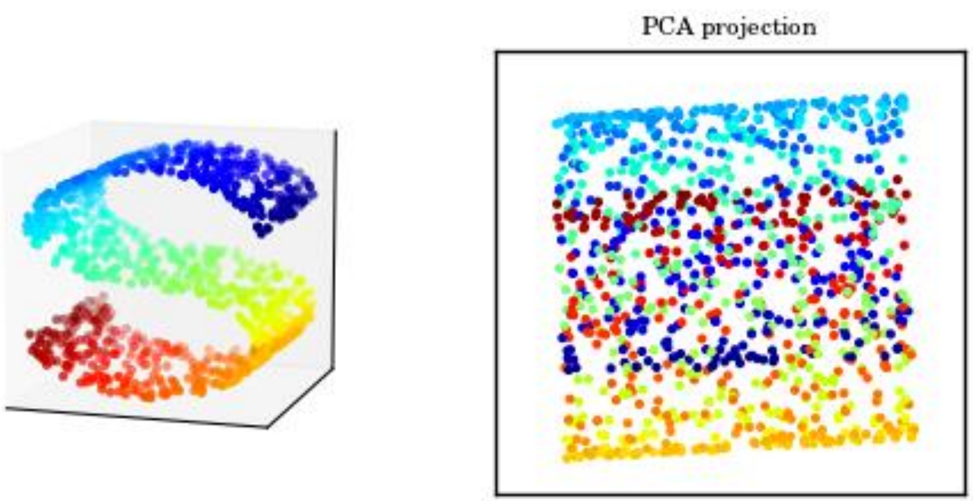


PCA minimises the l2 distance between **X** and **Y**

Source: http://alexhwilliams.info/itsneuronalblog/2016/03/27/pca/

PCA limitations

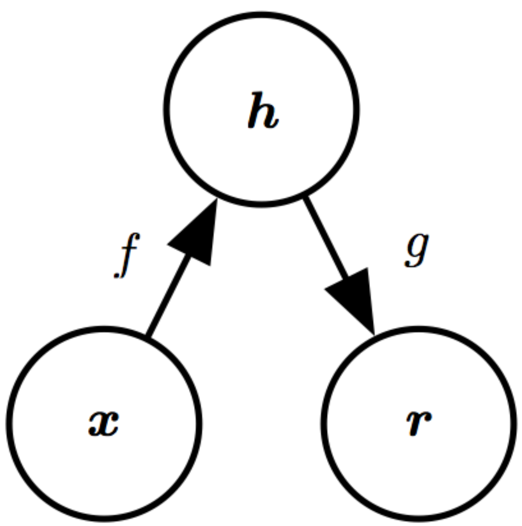
- However our ability to capture the structure of the data in the original space is limited by the type of transformations we consider (orthogonal linear)



Source: http://www.astroml.org/book_figures/chapter7/fig_S_manifold_PCA.html

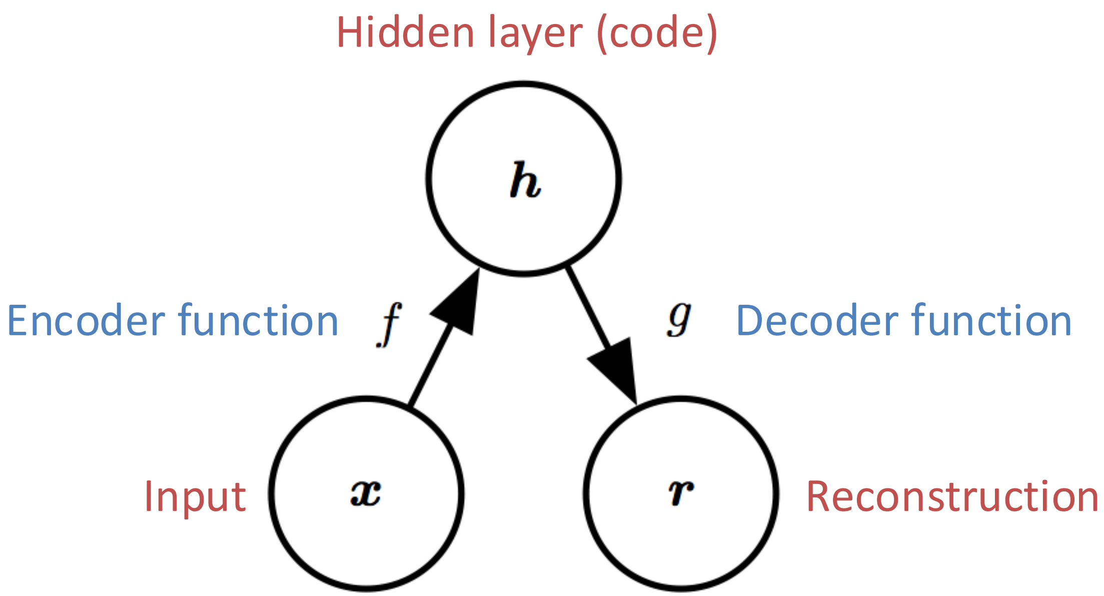
Autoencoders

- An **autoencoder** is neural networks that is trained to copy its input to its output
- It can generalise PCA by accounting for non-linear transformations



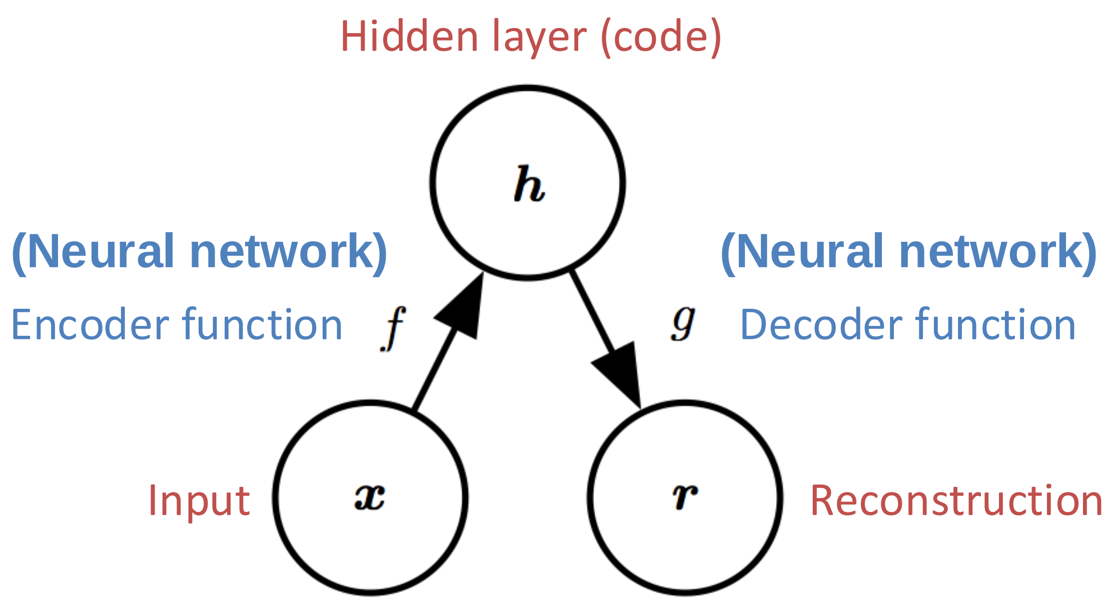
Autoencoders

- An **autoencoder** is neural networks that is trained to copy its input to its output
- It can generalise PCA by accounting for non-linear transformations



Autoencoders

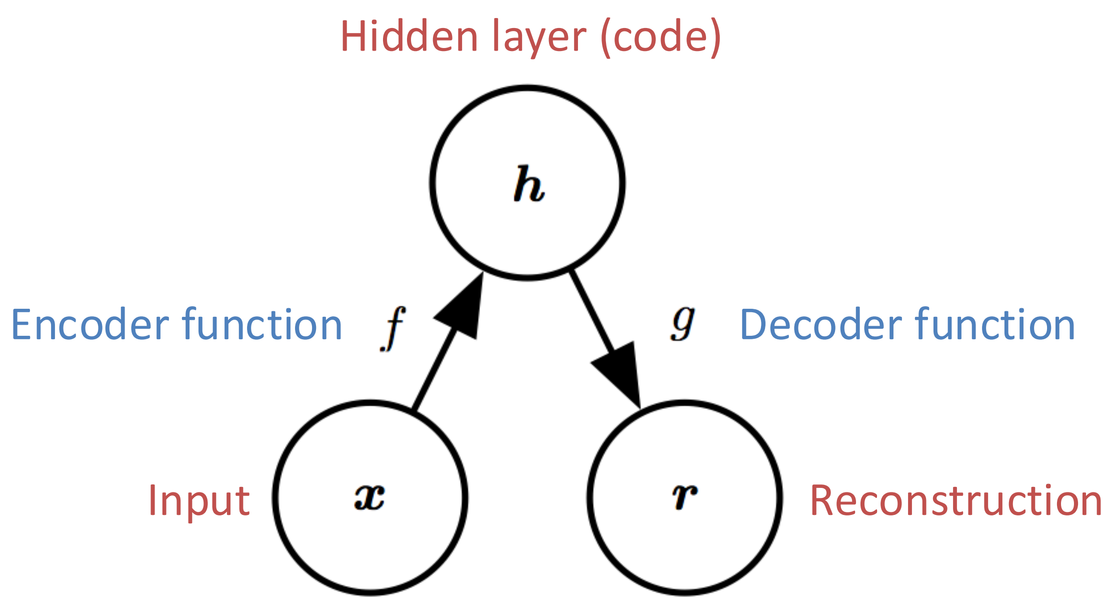
- An **autoencoder** is neural networks that is trained to copy its input to its output
- It can generalise PCA by accounting for non-linear transformations



Autoencoders

- We want to force the code to have a lower dimensionality than the input
- We want to learn a low-dimensional representation which captures the salient features of the input

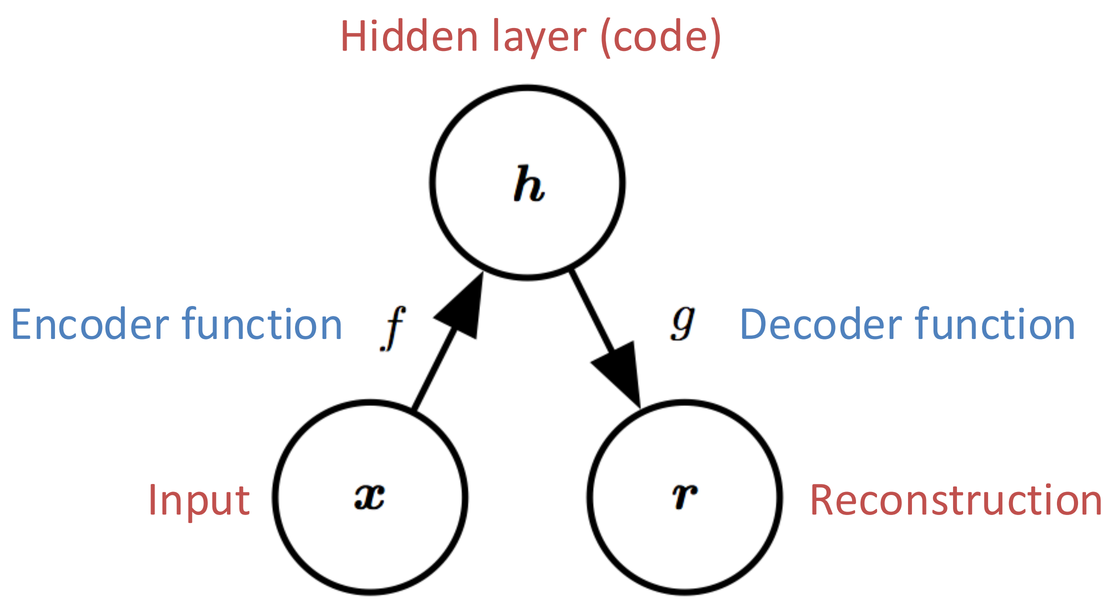
突出的



Autoencoders

- Minimise the loss/reconstruction error

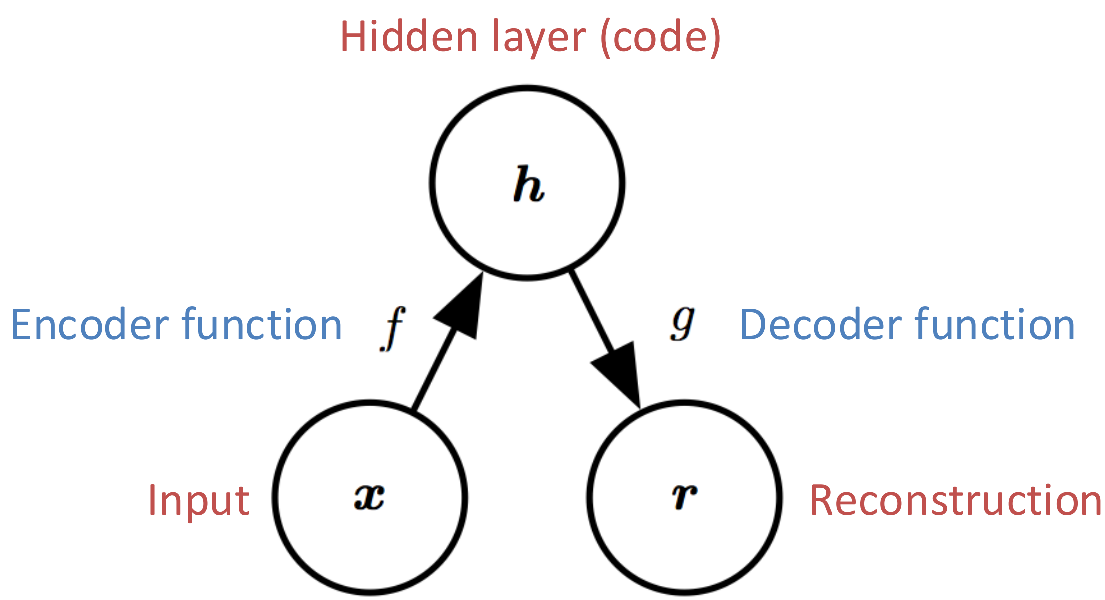
$$\frac{1}{N}\sum_{n=1}^{N} L(\boldsymbol{x}_n, \boldsymbol{r}_n) = \frac{1}{N}\sum_{n=1}^{N} L(\boldsymbol{x}_n, g(f(\boldsymbol{x}_n)))$$



Autoencoders

- Backprop and SGD to find the optimal parameters

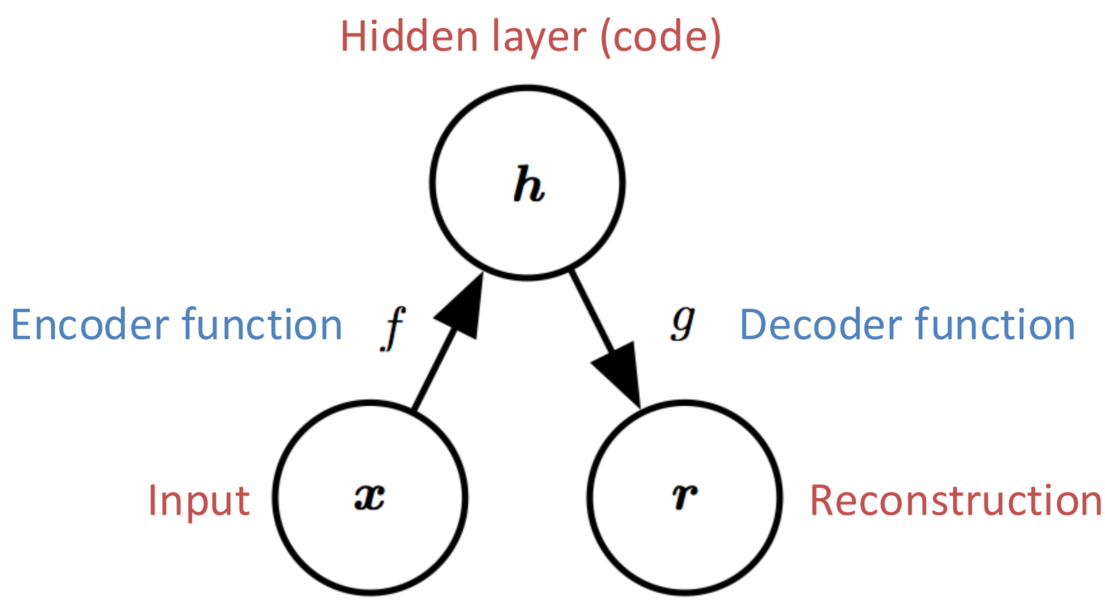
$$\frac{1}{N}\sum_{n=1}^{N} L(\boldsymbol{x}_n, \boldsymbol{r}_n) = \frac{1}{N}\sum_{n=1}^{N} L(\boldsymbol{x}_n, g(f(\boldsymbol{x}_n)))$$



Autoencoders

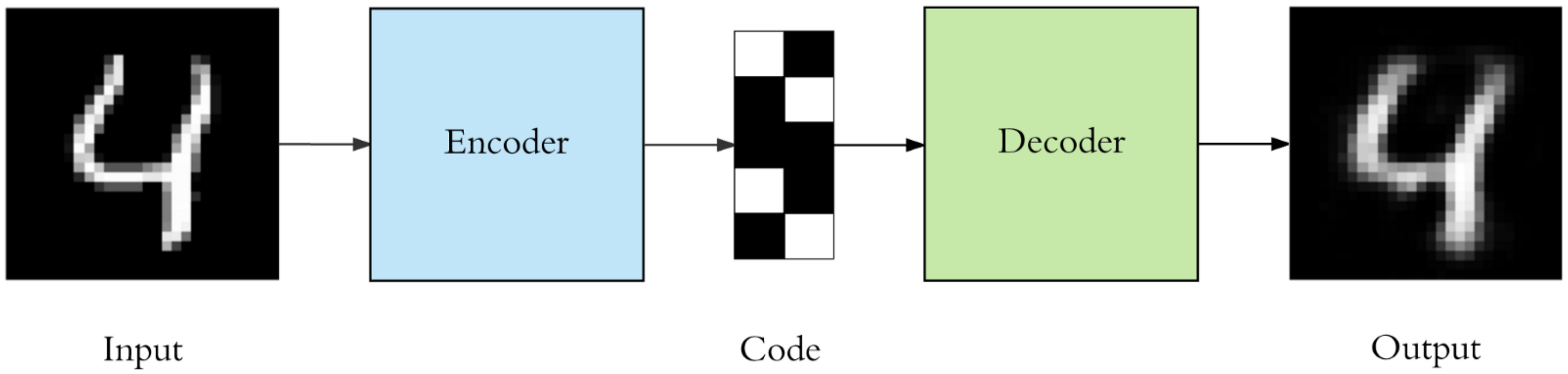
- With l2 norm and *f*, *g* linear we recover PCA

$$\frac{1}{N}\sum_{n=1}^{N} L(\boldsymbol{x}_n, \boldsymbol{r}_n) = \frac{1}{N}\sum_{n=1}^{N} L(\boldsymbol{x}_n, g(f(\boldsymbol{x}_n)))$$



Autoencoders

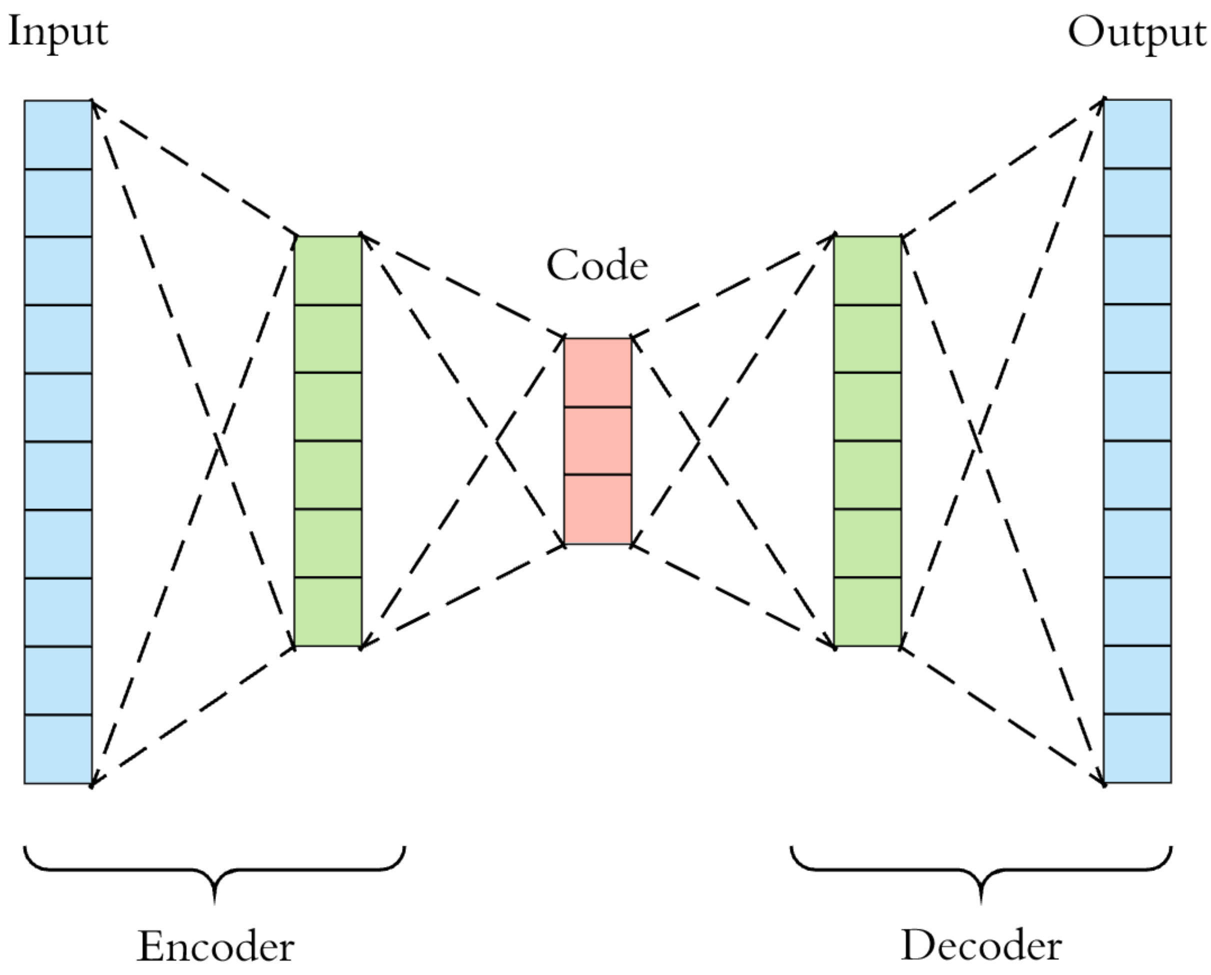
Example: one-layer encoder + one-layer decoder



$$f(\boldsymbol{x}) = \mathrm{ReLU}(W\boldsymbol{x} + \boldsymbol{b}), \quad g(\boldsymbol{x}) = \sigma(V f(\boldsymbol{x}) + \boldsymbol{c})$$

Source: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

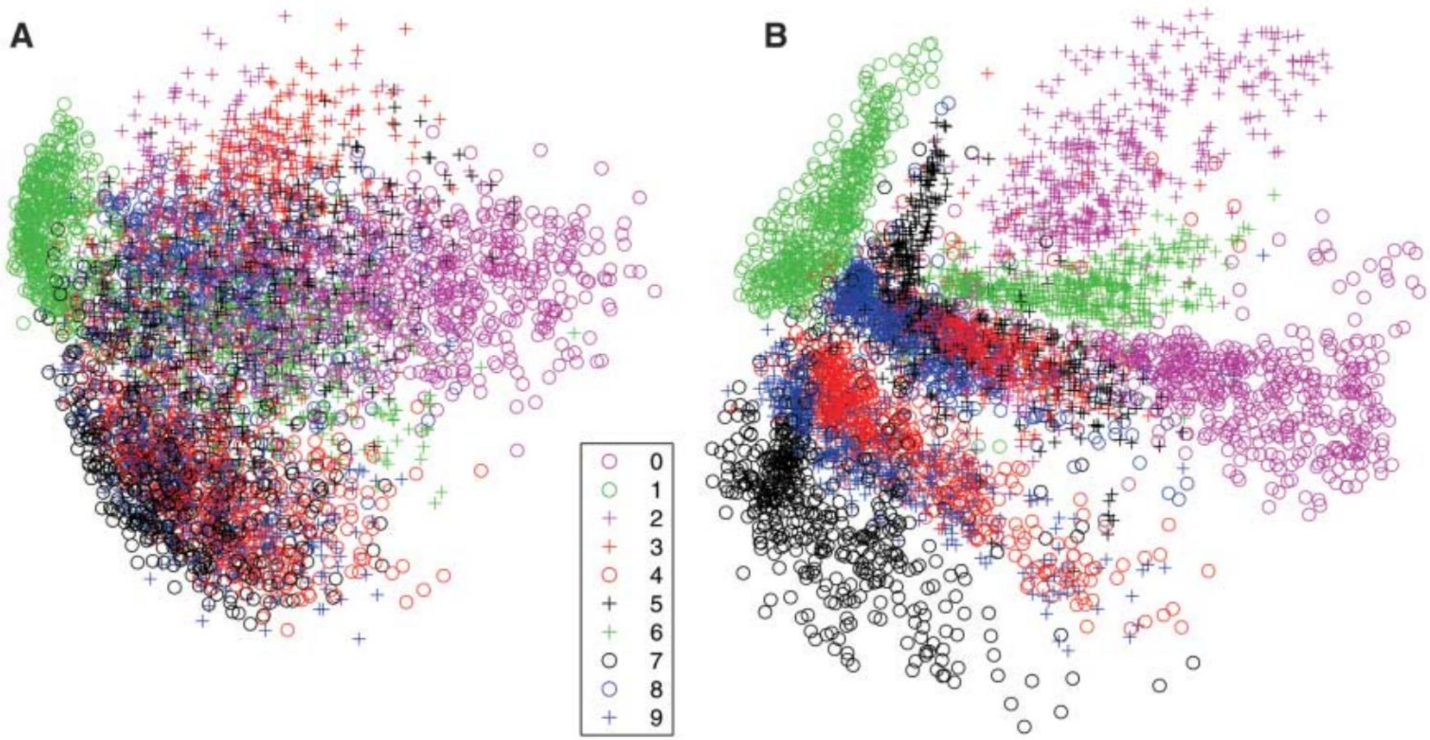
Deep autoencoders



Source: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

PCA vs autoencoders

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*).



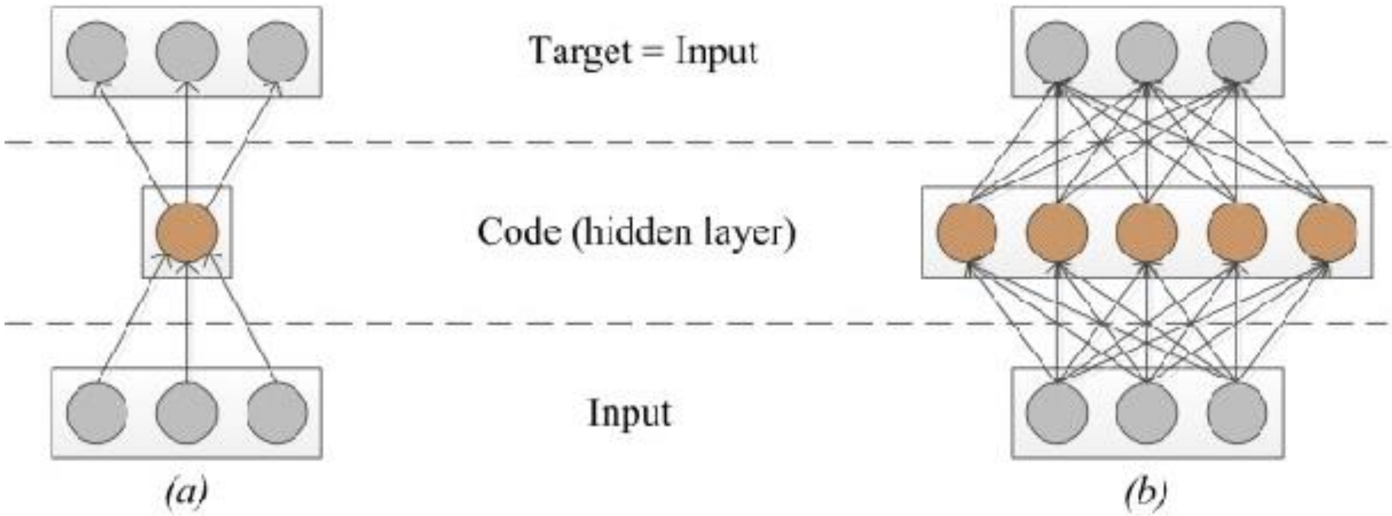
Source: https://www.cs.toronto.edu/~hinton/science.pdf

Undercomplete autoencoders

- **Undercomplete** autoencoders (what we talked about in the previous slides)
  - code has lower dimension than input
  - *f* or *g* has low capacity (e.g., linear *g*)
- If the encoder and decoder have high capacity we can learn trivial transformations
  - A one-dimensional code with a very powerful non-linear encoder could learn to represent each training example x_i with the code *i*

Overcomplete autoencoders

- **Overcomplete** autoencoders
  - h has higher dimension than x



Gavat et al., Deep Learning in Acoustic Modeling for Automatic Speech Recognition and Understanding - An Overview, 2015

Overcomplete autoencoders

- **Overcomplete** autoencoders
  - *h* has higher dimension than *x*
  - Must be regularised

$$\frac{1}{N}\sum_{n=1}^{N} L(\boldsymbol{x}_n, g(f(\boldsymbol{x}_n))) + \Omega(\boldsymbol{h})$$

- **Goal:** Encourage the model to have other useful properties besides the ability to copy its input to its output

Regularised autoencoders

- Sparse autoencoders
- Denoising autoencoders
- Contractive autoencoders

Sparse autoencoders

- Limit capacity of autoencoder by adding a cost term that penalises the code for being larger, e.g.,

$$\frac{1}{N}\sum_{n=1}^{N} L(\boldsymbol{x}_n, g(f(\boldsymbol{x}_n))) + \lambda\|\boldsymbol{h}\|_1$$

- Typically used to extract robust features or lower the dimensionality of the input data

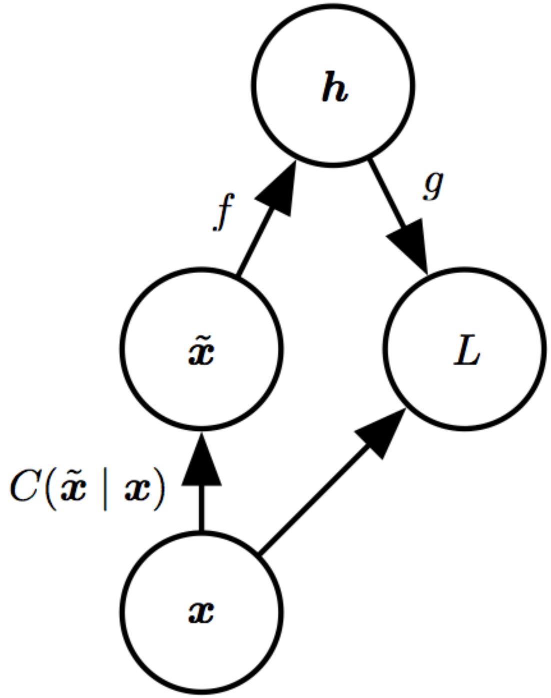
Denoising autoencoder

- Instead of adding penalty, change reconstruction error to account for the introduction of noise

$$\frac{1}{N}\sum_{n=1}^{N} L(\boldsymbol{x}_n, g(f(\tilde{\boldsymbol{x}}_n)))$$

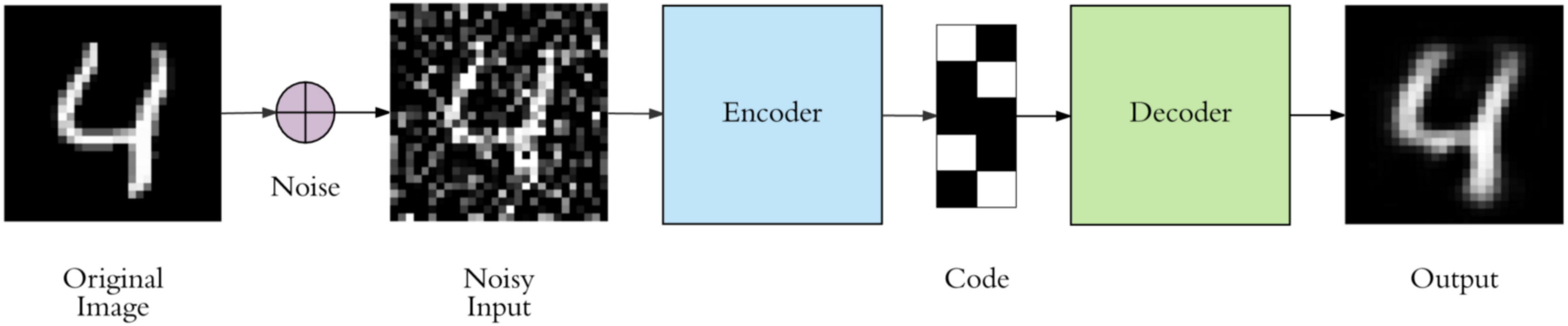
- Noise could be Gaussian or Dropout, i.e., part of the input is randomly set to 0
- The autoencoder learns a denoising map

Denoising autoencoder



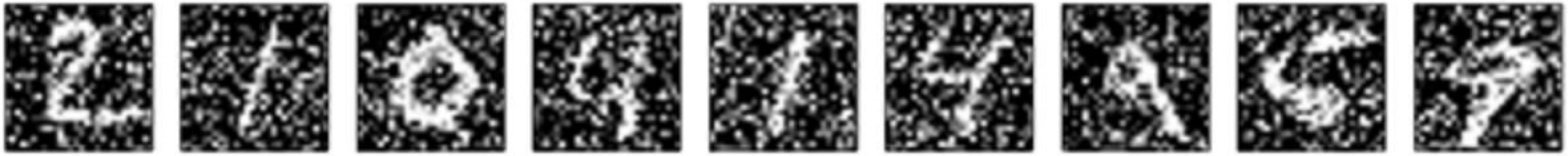
C is the corruption process which introduces noise

Denoising autoencoder



The network is forced to learn more robust representations

Denoising autoencoder



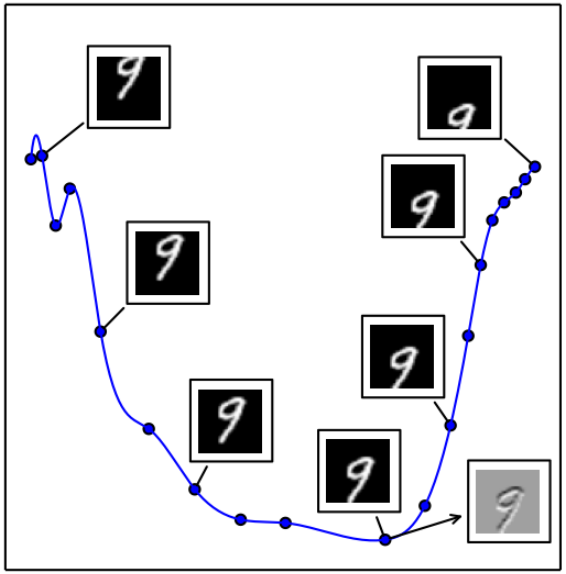
Application: image denoising

Denoising autoencoder

Application: image denoising

Autoencoders as manifold learners

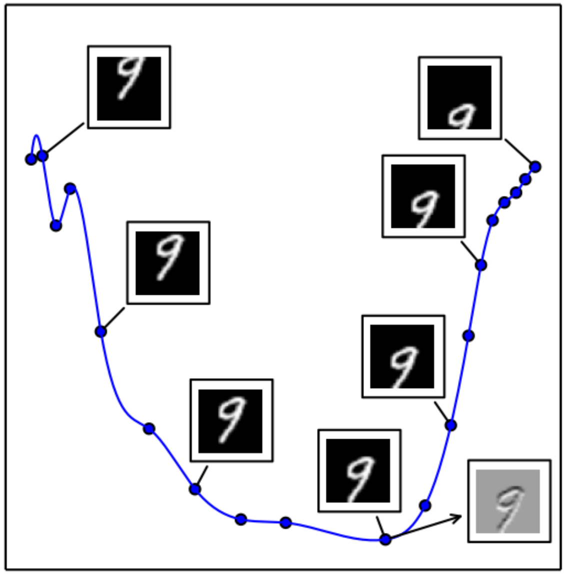
- At each point *x* of a *d*-dimensional manifold, a tangent plane is given by *d* basis vectors spanning the local directions of variation of the manifold



Grey pixel: no variation - Black/white: large variation

Autoencoders as manifold learners

- Do autoencoders learn manifold structure?



Grey pixel: no variation - Black/white: large variation

Autoencoders as manifold learners

- Training autoencoders combines two forces:
  - **Reconstruction**: learn $h = f(x)$ such that x can be recovered from it through $x = g(h)$
  - **Limited capacity**: the encoder cannot represent any possible function f
- These forces together push the hidden representation to capture information about the **structure** of the data-generating distribution

Autoencoders as manifold learners

- **Note**: the autoencoder can only afford to model the variations needed to reconstruct the training data
- If the training data concentrates near a manifold, only the variations tangent to the manifold around x need to correspond to changes in $h = f(x)$
- Autoencoders learn representation that captures a local coordinate system of the manifold

Contractive autoencoders

- More explicit formulation to learn the manifold

$$\frac{1}{N}\sum_{n=1}^{N} L(\boldsymbol{x}_n, g(f(\boldsymbol{x}_n))) + \Omega(\boldsymbol{h}, \boldsymbol{x})$$

$$\Omega(\boldsymbol{h}, \boldsymbol{x}) = \lambda \left\| \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right\|_F^2$$

- Penalises the Frobenius norm of the Jacobian of the encoder, i.e., *forces the encoder to learn a representation that doesn't change much around training samples*

Contractive autoencoders

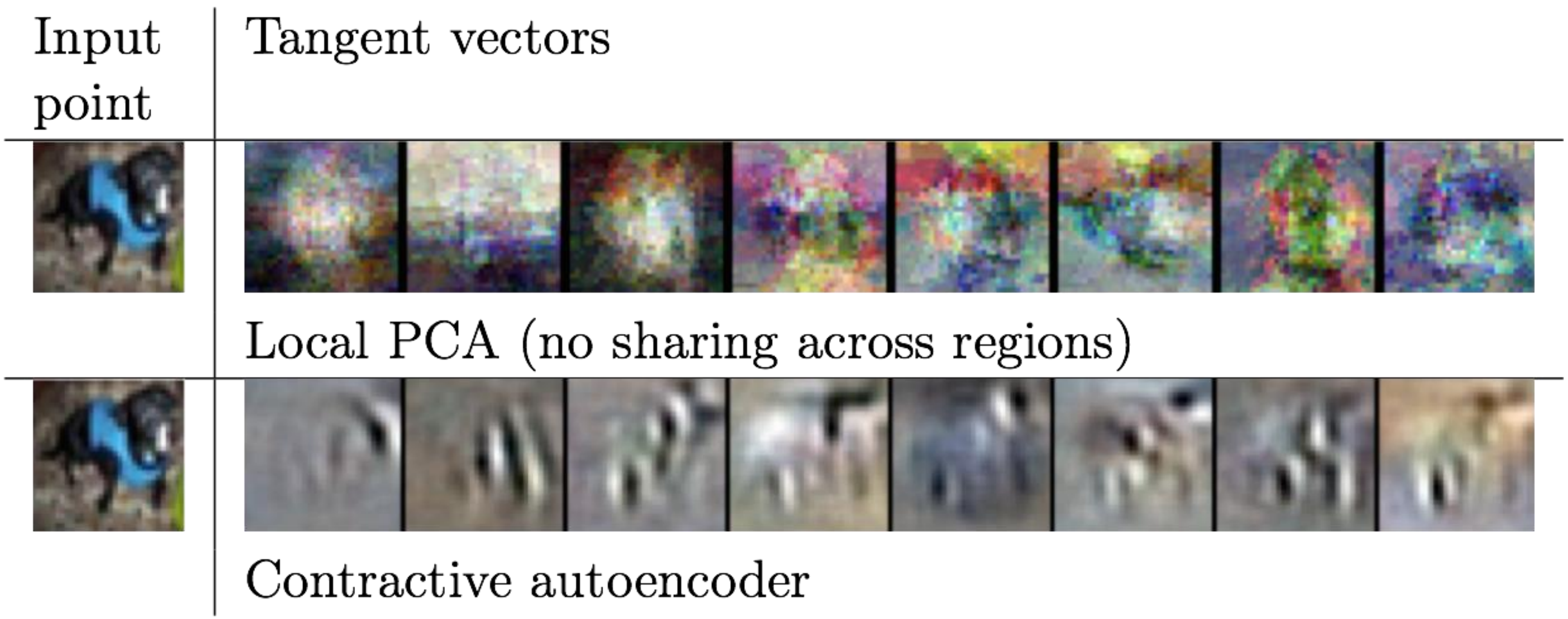


Figure 14.10: Illustration of tangent vectors of the manifold estimated by local PCA and by a contractive autoencoder. The location on the manifold is defined by the input image of a dog drawn from the CIFAR-10 dataset. The tangent vectors are estimated by the leading singular vectors of the Jacobian matrix $\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}$ of the input-to-code mapping. Although both local PCA and the CAE can capture local tangents, the CAE is able to form more accurate estimates from limited training data because it exploits parameter sharing across different locations that share a subset of active hidden units. The CAE tangent directions typically correspond to moving or changing parts of the object (such as the head or legs). Images reproduced with permission from Rifai *et al.* (2011c).

Applications of autoencoders

- Dimensionality reduction
- Image denoising
- Features extraction
- Watermarking removal
- Image colouring
    - https://github.com/baldassarreFe/deep-koalarization

Summary

- Autoencoders learn to encode and decode input via a latent space
- Regularisation is needed to learn useful representations, and it can take many forms
- The representations can be used as features or to lower the dimensionality of the input data
- Denoising and contractive autoencoders are able to learn the manifold structure of the data

Demo: https://cs.stanford.edu/people/karpathy/convnetjs/demo/autoencoder.html