

# Autoencoders and PCA

F. Noé<sup>1</sup>

Deep Learning Classes, FU Berlin 2018

# Data and Linear Projections

- Data set  $\mathbf{X} \in \mathbb{R}^{T \times n}$

$$\mathbf{X} = \begin{bmatrix} x_1(t=1) & \cdots & x_D(t=1) \\ \vdots & & \vdots \\ x_1(t=T) & \cdots & x_D(t=T) \end{bmatrix}.$$

- Assume that this data is empirically mean-free, i.e.

$$\sum_{t=1}^T x_i(t) = 0 \quad \forall i.$$

If that is not the case, we remove the mean and store it for later use.

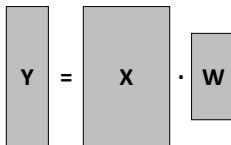
- **Objective:** find vectors  $\mathbf{w} \in \mathbb{R}^n$  that define linear projections

$$\mathbf{y} = \mathbf{X}\mathbf{w}$$

and are in some sense optimal (different methods arise from different optimality criteria).

- **Generalization:** search projection matrix  $\mathbf{W} \in \mathbb{R}^{n \times m}$  with  $m \leq n$  which projects the data onto a linear subspace:

$$\mathbf{Y} = \mathbf{XW}.$$



- $m < n$ :  $\mathbf{W}$  performs a dimension reduction (often  $m \ll n$ ).
- Generally, dimension reduction involves lossy compression, i.e. information in the neglected dimensions is discarded.

# Data and Linear Projections

- **Generalization:** search projection matrix  $\mathbf{W} \in \mathbb{R}^{n \times m}$  with  $m \leq n$  which projects the data onto a linear subspace:

$$\mathbf{Y} = \mathbf{XW}.$$

- Find  $\mathbf{W}$  by minimizing loss or maximizing score.
- **Idea:** find vectors  $\mathbf{W}_m = [\mathbf{w}_1, \dots, \mathbf{w}_m]$  that ...
  - maximize the **explained variance**, i.e. the variance of projected data  $(\mathbf{y}_1, \dots, \mathbf{y}_m)$ . For a given direction  $\mathbf{y} = \mathbf{y}_i$ , using the fact that data is mean-free, we have:

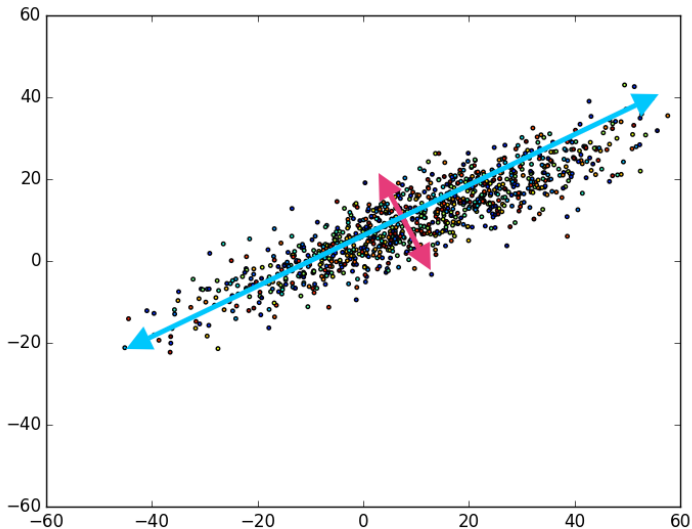
$$\max_t \sum_t y_t^2 = \max_t \sum_t \|\mathbf{X}\mathbf{w}\|^2$$

- minimize the **reconstruction error**, i.e. the difference between the original data  $\mathbf{X}$  and the reconstructed data  $\mathbf{XWW}^\top$ :

$$\min \|\mathbf{X} - \mathbf{XWW}^\top\|_F^2$$

# Principal Component Analysis (PCA)

Example



# Principal Component Analysis (PCA)

**First principal component:** we seek the direction along which the data has the maximum variance:

$$\mathbf{w}_1 = \arg \max_{\|\mathbf{w}\|=1} \{\|\mathbf{X}\mathbf{w}\|^2\} = \arg \max_{\|\mathbf{w}\|=1} \left\{ \mathbf{w}^\top \mathbf{X}^\top \mathbf{X} \mathbf{w} \right\}$$

Thus  $\mathbf{w}_1$  maximizes the Rayleigh quotient

$$\mathbf{w}_1 = \arg \max \left\{ \frac{\mathbf{w}^\top \mathbf{X}^\top \mathbf{X} \mathbf{w}}{\mathbf{w}^\top \mathbf{w}} \right\}$$

For a symmetric matrix  $\mathbf{X}^\top \mathbf{X}$  the Rayleigh quotient is maximized by the largest eigenvalue of the matrix, which occurs when  $\mathbf{w}$  is the corresponding eigenvector.

# Principal Component Analysis (PCA)

**Further principal components:** The  $k$ th component can be found by subtracting the first  $k - 1$  principal components from  $\mathbf{X}$ :

$$\hat{\mathbf{X}}_k = \mathbf{X} - \sum_{s=1}^{k-1} \mathbf{X} \mathbf{w}_s \mathbf{w}_s^T$$

and then finding the loading vector which extracts the maximum variance from this new data matrix

$$\mathbf{w}_{(k)} = \arg \max \left\{ \frac{\mathbf{w}^T \hat{\mathbf{X}}_k^T \hat{\mathbf{X}}_k \mathbf{w}}{\mathbf{w}^T \mathbf{w}} \right\}$$

This gives the remaining eigenvectors of  $\mathbf{X}^T \mathbf{X}$ , with the maximum values for the quantity in brackets given by their corresponding eigenvalues.

→ Columns of  $\mathbf{W}$  can be computed as the eigenvectors of  $\mathbf{X}^T \mathbf{X}$ .

# Principal Component Analysis (PCA)

## Algorithm:

- 1 Compute the covariance matrix

$$\mathbf{C}_0 = \frac{1}{T-1} \mathbf{X}^\top \mathbf{X},$$

which is just a scaled version of  $\mathbf{X}^\top \mathbf{X}$

- 2 Solve the Eigenvalue problem:

$$\mathbf{C}_0 \mathbf{w}_i = \sigma_i^2 \mathbf{w}_i$$

with normalization  $\mathbf{w}_i^\top \mathbf{w}_i = 1$ .

- 3 Select  $m$  eigenvectors with largest eigenvalues.
- 4 Reduce dimension from  $n$  to  $m$  with  $\mathbf{W}_m = [\mathbf{w}_1, \dots, \mathbf{w}_m]$ :

$$\mathbf{Y}_m = \mathbf{X} \mathbf{W}_m.$$



# Principal Component Analysis (PCA)

## Properties

- 1 Principal components are orthogonal because  $\mathbf{C}_0$  is symmetric:

$$\mathbf{W}^\top \mathbf{W} = \mathbf{I}$$

- 2 Principal components are uncorrelated:

$$\mathbf{Y}^\top \mathbf{Y} = \mathbf{W}^\top \mathbf{X}^\top \mathbf{X} \mathbf{W} = \mathbf{W}^\top \mathbf{W} \mathbf{\Sigma}^2 \mathbf{W}^\top \mathbf{W} = \mathbf{\Sigma}^2$$

- 3 PCA eigenvalues indicate variances along principal components:

$$\sigma_i^2 = \mathbf{y}_i^\top \mathbf{y}_i$$

A method to determine  $m$  is the **cumulative explained variance**:

$$V_m = \left( \sum_{i=1}^m \sigma_i^2 \right) / \left( \sum_{i=1}^n \sigma_i^2 \right)$$

- 4 Transformation into all principal components,  $\mathbf{Y} = \mathbf{XW}$  is loss-less:

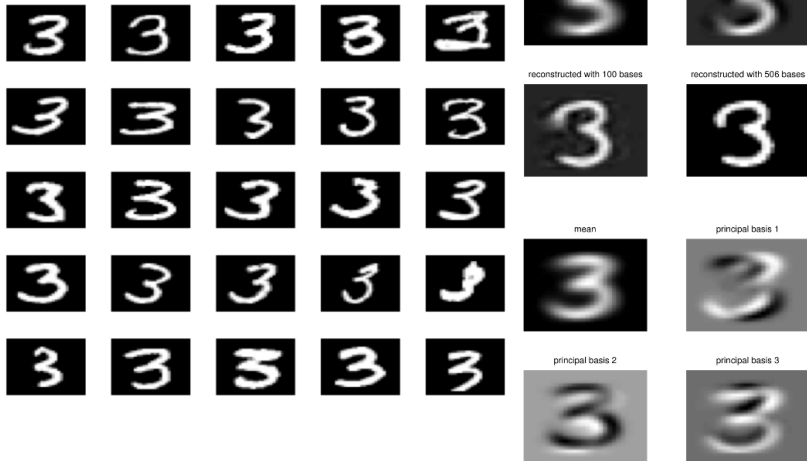
$$\mathbf{X} = \mathbf{XW} \mathbf{W}^\top$$

- 5 The projection onto  $m < n$  principal components,  $\mathbf{Y}_m = \mathbf{XW}_m$  is lossy. It minimizes the squared reconstruction error:

$$\|\mathbf{XW} \mathbf{W}^\top - \mathbf{XW}_m \mathbf{W}_m^\top\|_2^2 = \|\mathbf{X} - \mathbf{X}_m\|_2^2$$

# PCA

## Example



- Run PCA on 2429 19x19 grayscale images (CBCL data)
- Compresses the data: can get good reconstructions with only 3 components



- PCA for pre-processing: can apply classifier to latent representation
- PCA w/ 3 components obtains 79% accuracy on face/non-face discrimination in test data vs. 76.8% for m.o.G with 84 states

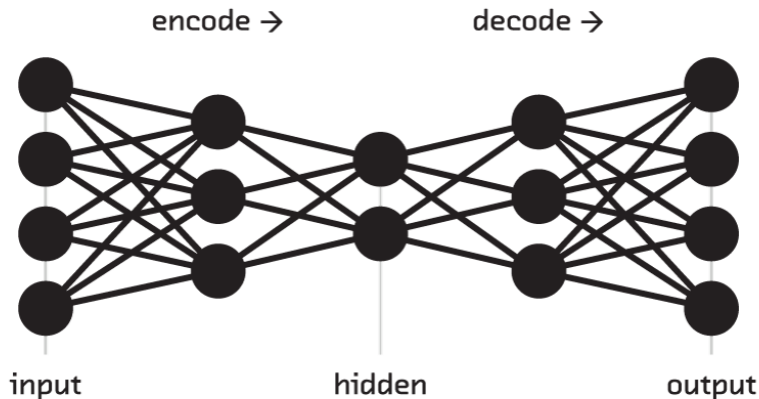
# PCA

## Eigenfaces



# Autoencoder

## Properties



Use the same loss function as in PCA:

$$C(\mathbf{X}; \theta) = \|\mathbf{X} - D(E(\mathbf{X}))\|_F^2 = \sum_{i=1}^T \|\mathbf{x}_i - D(E(\mathbf{x}_i))\|^2$$

# Autoencoder versus PCA

- Use linear encoder  $\mathbf{E} \in \mathbb{R}^{n \times m}$  and decoder  $\mathbf{D} \in \mathbb{R}^{m \times n}$ :

$$E(\mathbf{x}) = \mathbf{E}\mathbf{x}$$

$$D(\mathbf{y}) = \mathbf{D}\mathbf{y}$$

- The Autoencoder optimization problem becomes:

$$\min_{\mathbf{D}, \mathbf{E}} \sum_{i=1}^N \|\mathbf{x}_i - \mathbf{D}\mathbf{E}\mathbf{x}_i\|_F^2$$

- For choice  $\mathbf{E} = \mathbf{W}_m$  and  $\mathbf{D} = \mathbf{W}_m^\top$  the linear Autoencoder is equivalent to PCA, which provides the optimal solution.
- For nonlinear  $E$  and  $D$ , the Autoencoder performs a “nonlinear PCA”

# Autoencoder versus PCA

## Method Comparison



Real data

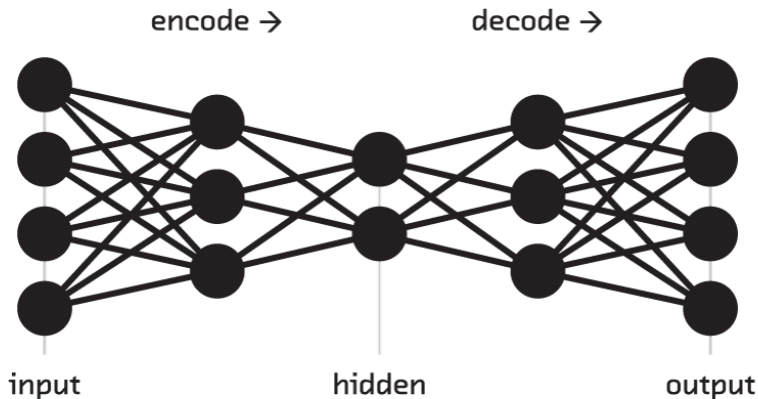
30-d deep autoencoder

30-d logistic PCA

30-d PCA

# Neural Autoencoder architectures

Dense Neural Network





# Neural Autoencoder Architectures

## Convolutional Neural Network

- **Convolution** of  $\mathbf{x} \in \mathbb{R}^n$  with  $\mathbf{w} \in \mathbb{R}^k$ , using  $k \leq n$  and “valid padding”, can be written as linear operation

$$\mathbf{y} = \mathbf{W}\mathbf{x} \quad (1)$$

with  $\mathbf{W} \in \mathbb{R}^{n-k+1 \times n}$  being a **Toeplitz matrix**:

$$\mathbf{W} = \begin{pmatrix} w_1 & \cdots & w_k & & \\ & \ddots & & \ddots & \\ & & w_1 & \cdots & w_k \end{pmatrix}$$

- Convolutions make the dimension of the data array smaller (valid padding) or leave it equal (zero padding). They cannot increase the dimension.

# Neural Autoencoder Architectures

## Convolutional Neural Network

- **Transposed convolutions:** To increase the dimension, we consider  $\mathbf{x} \in \mathbb{R}^p$ ,  $\mathbf{y} \in \mathbb{R}^q$ ,  $p \leq q$  and convolve using

$$\mathbf{y} = \mathbf{W}^T \mathbf{x}$$

where  $\mathbf{W}^T \in \mathbb{R}^{q \times p}$  increases the size of the input array and has learnable parameters:

$$\mathbf{W}^T = \begin{pmatrix} w_1 & & & \\ \vdots & \ddots & & \\ w_k & & w_1 & \\ & \ddots & \vdots & \\ & & w_k & \end{pmatrix}$$

- In practice, one does not apply Toeplitz matrices to implement convolutions because they consist mostly of zeros. Instead, we use a trick to employ the implementation of standard convolution layers to compute transposed convolutions.

# Reminder: Forward and Backward Pass

- **Symbols:**

$\mathbf{W}$ : Weight matrix

$\mathbf{x}^l$ : neural output activations at layer  $l$ ,

$\sigma(\cdot)$ : nonlinear activation function,

$\mathbf{z}^l$ : neural activations before nonlinearity,  $z_i^l = \sum_k w_{ik}^l x_k^{l-1} + b_i^l$

- **Forward pass** (assuming no bias,  $\mathbf{b}^l = \mathbf{0}$ ):

$$\mathbf{z}^{l+1} = \mathbf{W}^{l+1} \mathbf{x}^l$$

$$\mathbf{x}^{l+1} = \sigma \left( \mathbf{W}^{l+1} \mathbf{x}^l \right).$$

- **Backward pass** (using the definition of error  $e_i^l = \partial C / \partial z_i^l$ ):

$$e_i^l = \sigma'(z_i^l) \sum_j e_j^{l+1} w_{ji}^{l+1}$$

$$\mathbf{e}^l = \sigma'(\mathbf{z}^l) (\mathbf{W}^{l+1})^\top \mathbf{e}^{l+1}$$

- $\rightarrow$  Transposed convolution can be implemented by exchanging forward and backward pass.