

10.3 Non-linear Dimensionality Reduction

Machine Learning 1: Foundations

Marius Kloft (TUK)

- 1 What is Dimensionality Reduction?
- 2 Linear Dimensionality Reduction
- 3 Non-linear Dimensionality Reduction
 - Kernel PCA
 - Autoencoders

To Kernelize PCA ...

... we apply the representer theorem (see kernel lecture) to the PCA solution $W_* = (\mathbf{w}_1^*, \dots, \mathbf{w}_k^*)$:

$$\forall j = 1, \dots, k \quad \exists \boldsymbol{\alpha}_j = (\alpha_{1j}, \dots, \alpha_{nj})^\top \in \mathbb{R}^n :$$

$$\mathbf{w}_j^* = \sum_{i=1}^n \alpha_{ij} \phi(\mathbf{x}_i) = \phi(X) \boldsymbol{\alpha}_j,$$

or, more compactly:

$$W_* = \phi(X) \boldsymbol{\alpha},$$

where we employ the notation:

$$\blacktriangleright \phi(X) := (\phi(\mathbf{x}_1), \dots, \phi(\mathbf{x}_n)), \quad \boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_k)$$

So the data in the dimensionality-reduced coordinate system is:

$$\tilde{X} := W_*^\top \phi(X) = \boldsymbol{\alpha}_*^\top K$$

Since the data lies in a high-dim. space, the original coord. sys. is not interesting here, unless we use a lin. kernel.

Kernel PCA (KPCA)

Thus, using $\mathbf{w}_j = \phi(X)\alpha_j$, we have:

$$\begin{aligned} & \max_{W \in \mathbb{R}^{d \times k}} \sum_{j=1}^k \mathbf{w}_j^\top \phi(X) \phi(X)^\top \mathbf{w}_j \\ &= \max_{\alpha \in \mathbb{R}^{n \times k}} \sum_{j=1}^k \alpha_j^\top \underbrace{\phi(X)^\top \phi(X) \phi(X)^\top \phi(X)}_{=K^2} \alpha_j \end{aligned}$$

(subject to $\alpha_j^\top K \alpha_j = 1 \ \forall j$)

One can show (homework):

Theorem

The KPCA solution $\alpha_* = (\alpha_1^*, \dots, \alpha_k^*)$ is given by the k largest Eigenvectors of the (centered) kernel matrix K .

What is actually the centered kernel matrix?

Centered Kernel Matrix

PCA requires that the data is centered as a preprocessing step:

$$\blacktriangleright \hat{\boldsymbol{\mu}} := \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i, \quad \forall i : \mathbf{x}_i \leftarrow \mathbf{x}_i - \hat{\boldsymbol{\mu}}$$

Similar, KPCA requires data that is centered **in feature space**:

$$\blacktriangleright \hat{\boldsymbol{\mu}} := \frac{1}{n} \sum_{i=1}^n \phi(\mathbf{x}_i), \quad \forall i : \phi(\mathbf{x}_i) \leftarrow \phi(\mathbf{x}_i) - \hat{\boldsymbol{\mu}}$$

Definition

The **centered kernel matrix** is the kernel matrix computed on the data that has been centered in feature space.

One can show (homework):

Theorem

The centered kernel matrix \tilde{K} can be computed from the (uncentered) kernel matrix K by:

$$\tilde{K} = \left(I - \frac{\mathbf{1}\mathbf{1}^\top}{n}\right) K \left(I - \frac{\mathbf{1}\mathbf{1}^\top}{n}\right)$$

KPCA Algorithm

```
1: function KPCA(parameter  $k$ , kernel matrix  $K \in \mathbb{R}^{n \times n}$ )  
2:   center the kernel matrix:  $K \leftarrow (I - \frac{\mathbf{1}\mathbf{1}^\top}{n})K(I - \frac{\mathbf{1}\mathbf{1}^\top}{n})$   
3:   compute  $k$  largest eigenvectors  $\alpha = (\alpha_1, \dots, \alpha_k)$  of  $K$   
   (e.g., in MATLAB:  $[\text{foo}, \alpha] = \text{eig}(K)$ )  
4:   compute:  $\tilde{X} := \alpha^\top K$   
5:   return dimensionality-reduced data:  $\tilde{X} \in \mathbb{R}^{k \times n}$   
6: end function
```

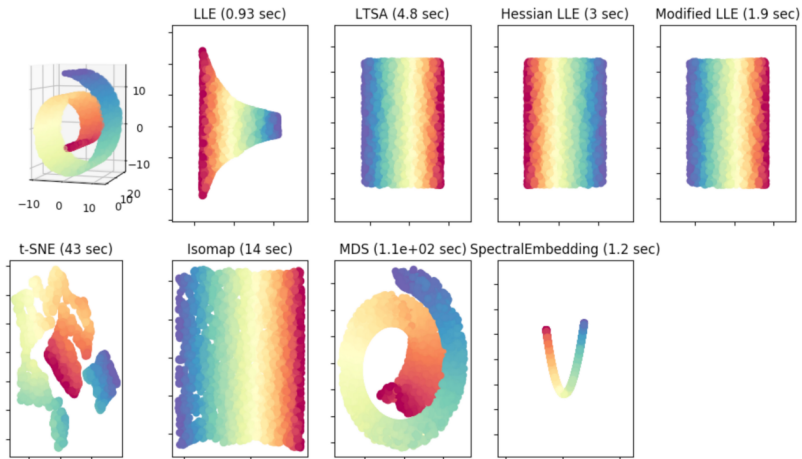
Note:

- ▶ KPCA can be even useful for linear kernels, when $d > n$
- ▶ In this case, the projected data in the original coordinate system is (assuming centered X and K):

$$\hat{X} = W_* \tilde{X} = \phi(X) \alpha_* \alpha_*^\top K,$$

Outlook

There are many other non-linear dimensionality reduction methods:



Next, we will look into one of them: autoencoders

Eigenfaces is Nice...

... but the state of the art in image processing is:

- ▶ **deep learning**

How deep can we go in dimensionality reduction?

The default method for dimensionality reduction in DL is:

- ▶ **autoencoders** (AEs)

Autoencoders (AEs)

Definition

An **autoencoder** (AE) is defined as:

$$\min_{W, \tilde{W}} \sum_{i=1}^n \|\mathbf{x}_i - g_{\tilde{W}}(f_W(\mathbf{x}_i))\|^2,$$

where:

- ▶ $f_W : \mathbb{R}^d \rightarrow \mathbb{R}^k$ is a neural network (called **encoder**)
- ▶ $g_{\tilde{W}} : \mathbb{R}^k \rightarrow \mathbb{R}^d$ is a neural network (called **decoder**)

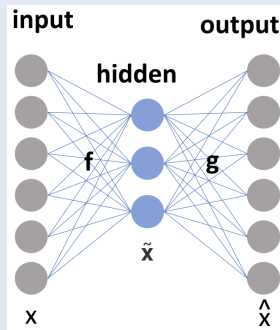
The dimensionality-reduced data is:

- ▶ in original coord. system:
- ▶ in k -dim. coordinate system:

$$\hat{X} := g_{\tilde{W}}(f_W(X)) \in \mathbb{R}^{d \times n}$$

$$\tilde{X} := f_W(X) \in \mathbb{R}^{k \times n}$$

$\tilde{\mathbf{x}} := f_W(\mathbf{x})$ is the **code** and $\hat{\mathbf{x}} := g_{\tilde{W}}(f_W(\mathbf{x}))$ the **reconstruction** of \mathbf{x} .



We denote here $f_W(X) := (f_W(\mathbf{x}_1), \dots, f_W(\mathbf{x}_n))$ and analogue for $g_{\tilde{W}} \circ f_W$.

Relation of AE to PCA

Consider a linear encoder and a linear decoder:

- ▶ The encoder becomes $f_W(\mathbf{x}) := W^\top \mathbf{x}$
and the decoder $g_{\tilde{W}}(\mathbf{x}) := \tilde{W}^\top \mathbf{x}$
- ▶ Can show that in the optimum: $\tilde{W} = W^\top$
- ▶ Thus: $\tilde{X} = W^\top X$ and $\hat{X} = WW^\top X$

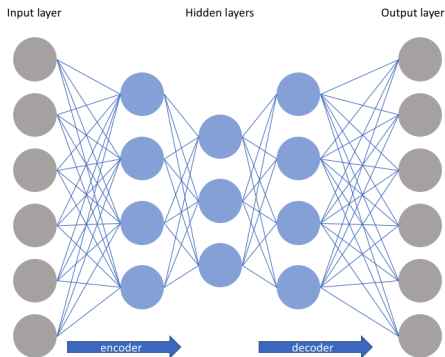
The AE solution is of the same form as for PCA!

Note, however:

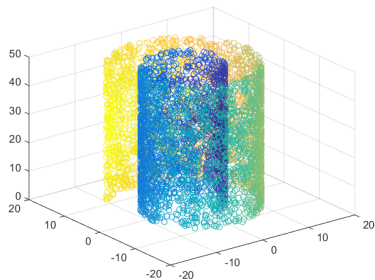
- ▶ An AE does not enforce orthonormality of W
- ▶ So the solution W_*^{AE} of an AE will be similar to—but not exactly the same as—the solution W_*^{PCA} of PCA

Deep Autoencoders

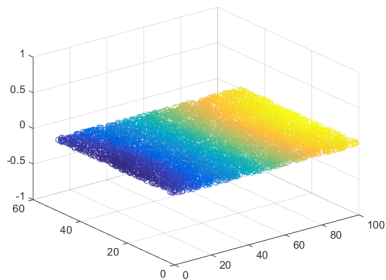
Of course, we can use multiple hidden layers...



Example: Swiss Role



original data



autoencoder

Outlook

Introducing a bottleneck $k < d$ is only one option to achieve a dimensionality reduction / compression in autoencoders

- ▶ another one is **regularization** (can have $k > d$ then)

There are various regularization techniques for autoencoders:

- ▶ Frobenius norm regularization
- ▶ Sparse autoencoders
- ▶ Denoising autoencoders
- ▶ Contractive autoencoders

Unifying View of Regr., Classif., and Dim. Reduction

$$\min_{[W,], b, \mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n l(f(\mathbf{x}_i) [y_i]) \left[+ \frac{1}{2} \sum_{l=1}^L \|W_l\|_{\text{Fro}}^2 \right],$$

with model

- ▶ $f(\mathbf{x}) = \langle \mathbf{w}, \phi(\mathbf{x}) \rangle + b$ for regression and classification
- ▶ $f(\mathbf{x}) = \|\phi(\mathbf{x}) - \mathbf{w}\mathbf{w}^T \phi(\mathbf{x})\|^2$ for dimensionality reduction

and loss

- ▶ $l(t, y) := \max(0, 1 - yt)$ for SVM (“hinge loss”)
- ▶ $l(t, y) := \ln(1 + \exp(-yt))$ for LR and ANN (“logistic loss”)
- ▶ $l(t, y) := (t - y)^2$ for regression
- ▶ $l(t) := t$ for dimensionality reduction

and feature map

- ▶ $\phi := \text{id}$ for linear SVM, linear LR, RR, and PCA.
- ▶ $\phi := \phi_K$ for kernel SVM, kernel LR, KRR, and KPCA.
- ▶ $\phi := \phi_W$ for ANN, DR, and AE.

The terms in gray brackets apply only to ANN, DR, and AE. The term in blue brackets applies only to dimensionality reduction.

Conclusion

- ▶ Unsupervised learning
 - ▶ this week: dimensionality reduction
- ▶ Most famous method: **principal component analysis (PCA)**
 - ▶ boils down to computing eigenvalues of scatter matrix XX^T
 - ▶ can be generalized to KPCA and autoencoders
- ▶ Outlook: more dimensionality reduction methods
 - ▶ optimizing different measures of closeness of the projected data to the original data
 - ▶ e.g., multidimensional scaling (MDS): tries to preserve distances among projected inputs
- ▶ Dimensionality reduction also possible in a supervised fashion (ML2: Fisher's discriminant analysis)
- ▶ Further reading: Goodfellow et al 2016, Deep Learning - Chapter: 14 Autoencoders