10.3 Non-linear Dimensionality Reduction

Machine Learning 1: Foundations

Marius Kloft *(TUK)*

To Kernelize PCA ...

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

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

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

or, more compactly:

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

where we employ the notation:

- $\phi(X) := (\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_1)), \quad \boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_k)$

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

Last video

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

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 $\mathbf{w}_j = \phi(X)\boldsymbol{\alpha}_j$, we have:

$$\max_{W \in \mathbb{R}^{d \times k}} \sum_{j=1}^{k} \mathbf{w}_j^\top \phi(X)\phi(X)^\top \mathbf{w}_j$$

?

Kernel PCA (KPCA)

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

$$\max_{W\in\mathbb{R}^{d\times k}} \sum_{j=1}^{k} \mathbf{w}_j^\top \phi(X)\phi(X)^\top \mathbf{w}_j$$

$$= \max_{\boldsymbol{\alpha}\in\mathbb{R}^{n\times k}} \sum_{j=1}^{k} \boldsymbol{\alpha}_j^\top \underbrace{\overbrace{\phi(X)^\top\phi(X)}^{=K}\overbrace{\phi(X)^\top\phi(X)}^{=K}}_{=K^2} \boldsymbol{\alpha}_j$$

Kernel PCA (KPCA)

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

$$\max_{W \in \mathbb{R}^{d\times k}} \sum_{j=1}^{k} \mathbf{w}_j^\top \phi(X)\phi(X)^\top \mathbf{w}_j$$

$$= \max_{\boldsymbol{\alpha} \in \mathbb{R}^{n\times k}} \sum_{j=1}^{k} \boldsymbol{\alpha}_j{}^\top \underbrace{\phi(X)^\top \phi(X)\phi(X)^\top \phi(X)}_{=K^2} \boldsymbol{\alpha}_j$$

(subject to $\boldsymbol{\alpha}_j^\top K \boldsymbol{\alpha}_j = 1 \ \forall j$)

Kernel PCA (KPCA)

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

$$\max_{W \in \mathbb{R}^{d\times k}} \sum_{j=1}^{k} \mathbf{w}_j^\top \phi(X)\phi(X)^\top \mathbf{w}_j$$

$$= \max_{\boldsymbol{\alpha} \in \mathbb{R}^{n\times k}} \sum_{j=1}^{k} \boldsymbol{\alpha}_j{}^\top \underbrace{\phi(X)^\top \phi(X)\phi(X)^\top \phi(X)}_{=K^2} \boldsymbol{\alpha}_j$$

$$(\text{subject to } \boldsymbol{\alpha}_j^\top K \boldsymbol{\alpha}_j = 1 \ \forall j)$$

One can show (homework):

Kernel PCA (KPCA)

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

$$\max_{W \in \mathbb{R}^{d\times k}} \sum_{j=1}^{k} \mathbf{w}_j^\top \phi(X)\phi(X)^\top \mathbf{w}_j$$

$$= \max_{\boldsymbol{\alpha} \in \mathbb{R}^{n\times k}} \sum_{j=1}^{k} \boldsymbol{\alpha}_j^\top \underbrace{\phi(X)^\top \phi(X)\phi(X)^\top \phi(X)}_{=K^2} \boldsymbol{\alpha}_j$$

$$(\text{subject to } \boldsymbol{\alpha}_j^\top K \boldsymbol{\alpha}_j = 1 \ \forall j)$$

One can show (homework):

Theorem

The KPCA solution $\boldsymbol{\alpha}_* = (\boldsymbol{\alpha}_1^*, \ldots, \boldsymbol{\alpha}_k^*)$ is given by the k largest Eigenvectors of the (centered) kernel matrix K.

Kernel PCA (KPCA)

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

$$\max_{W \in \mathbb{R}^{d \times k}} \sum_{j=1}^{k} \mathbf{w}_j^\top \phi(X)\phi(X)^\top \mathbf{w}_j$$

$$= \max_{\boldsymbol{\alpha} \in \mathbb{R}^{n \times k}} \sum_{j=1}^{k} \boldsymbol{\alpha}_j^\top \underbrace{\phi(X)^\top \phi(X)\phi(X)^\top \phi(X)}_{=K^2} \boldsymbol{\alpha}_j$$

(subject to $\boldsymbol{\alpha}_j^\top K \boldsymbol{\alpha}_j = 1 \ \forall j$)

One can show (homework):

Theorem

The KPCA solution $\boldsymbol{\alpha}_* = (\boldsymbol{\alpha}_1^*, \ldots, \boldsymbol{\alpha}_k^*)$ 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:

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

Centered Kernel Matrix

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

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

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

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

Centered Kernel Matrix

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

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

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

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

Definition

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

Centered Kernel Matrix

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

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

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

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

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 $\widetilde{K}$ can be computed from the (uncentered) kernel matrix K by:

$$\widetilde{K} = (I - \frac{\mathbf{1}\mathbf{1}^\top}{n})K(I - \frac{\mathbf{1}\mathbf{1}^\top}{n}) \quad \succ 0$$

KPCA Algorithm

function KPCA(parameter k, kernel matrix $K \in \mathbb{R}^{n \times n}$)

  center the kernel matrix: $K \leftarrow (I - \frac{\mathbf{1}\mathbf{1}^\top}{n}) K (I - \frac{\mathbf{1}\mathbf{1}^\top}{n})$

  compute k largest eigenvectors $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_k)$ of K
(e.g., in MATLAB: [foo,α] = eig(K))

  compute: $\widetilde{X} := \boldsymbol{\alpha}^\top K$

  return dimensionality-reduced data: $\widetilde{X} \in \mathbb{R}^{k \times n}$

end function

KPCA Algorithm

function KPCA(parameter k, kernel matrix $K \in \mathbb{R}^{n \times n}$)
  center the kernel matrix: $K \leftarrow (I - \frac{\mathbf{11}^\top}{n})K(I - \frac{\mathbf{11}^\top}{n})$
  compute k largest eigenvectors $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_k)$ of K
 (e.g., in MATLAB: [foo,α] = eig(K))
  compute: $\widetilde{X} := \boldsymbol{\alpha}^\top K$
  return dimensionality-reduced data: $\widetilde{X} \in \mathbb{R}^{k \times n}$
end function

Note:

- KPCA can be even useful for linear kernels, when $d > n$

KPCA Algorithm

function KPCA(parameter k, kernel matrix $K \in \mathbb{R}^{n \times n}$)
    center the kernel matrix: $K \leftarrow (I - \frac{\mathbf{1}\mathbf{1}^\top}{n})K(I - \frac{\mathbf{1}\mathbf{1}^\top}{n})$
    compute k largest eigenvectors $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_k)$ of K
(e.g., in MATLAB: [foo,α] = eig(K))
    compute: $\widetilde{X} := \boldsymbol{\alpha}^\top K$
    return dimensionality-reduced data: $\widetilde{X} \in \mathbb{R}^{k \times n}$
end function

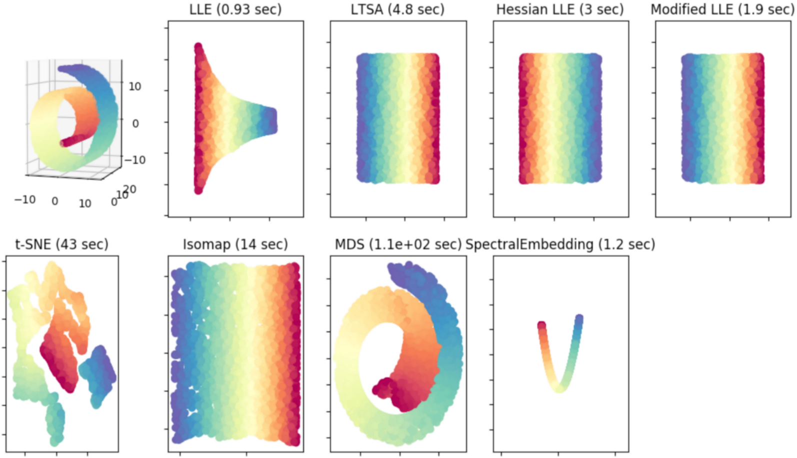
Note:

- 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):

$$\hat{X} = W_* \widetilde{X} = \phi(X)\boldsymbol{\alpha}_* \boldsymbol{\alpha}_*^\top K,$$

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**

Eigenfaces is Nice...

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

- **deep learning**

How deep can we go in dimensionality reduction?

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

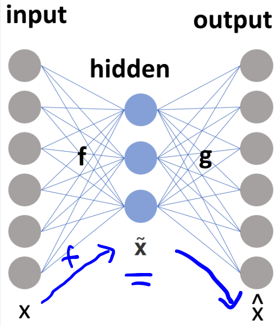
An **autoencoder** (AE) is defined as:

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

?

where:

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



The dimensionality-reduced data is:

- in original coord. system: $\hat{X} := g_{\tilde{W}}(f_W(X)) \in \mathbb{R}^{d \times n}$
- in k-dim. coordinate system: $\widetilde{X} := f_W(X) \in \mathbb{R}^{k \times n}$

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

We denote here $f_W(X) := (f_W(\mathbf{x}_1), \ldots, f_W(\mathbf{x}_n))$ 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 $f_W(\mathbf{x}) := \underline{W^\top \mathbf{x}}$
  and the decoder $g_{\tilde{W}}(\mathbf{x}) := \underline{\tilde{W}^\top \mathbf{x}}$

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: $\tilde{W} = W^\top$

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: $\widetilde{X} = W^\top X$ and $\hat{X} = WW^\top X$

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: $\widetilde{X} = W^\top X$ and $\hat{X} = WW^\top X$

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

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: $\widetilde{X} = W^\top X$ and $\hat{X} = WW^\top X$

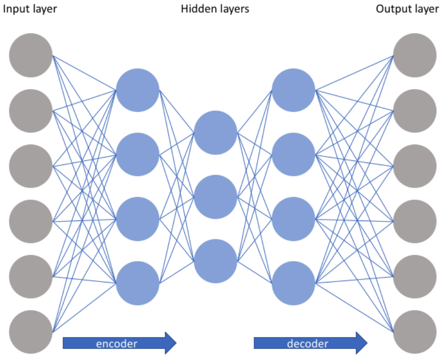
The AE solution is of the same form as for PCA!

Note, however:

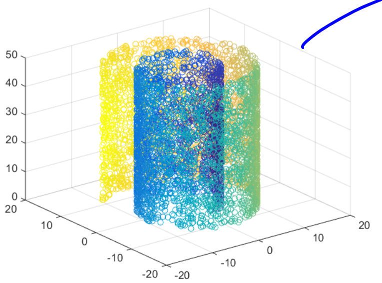
- An AE does not enforce orthonormality of W
- So the solution W_*^{AE} of an AE will be similar to—but not exactly the same as—the solution W_*^{PCA} of PCA

Deep Autoencoders

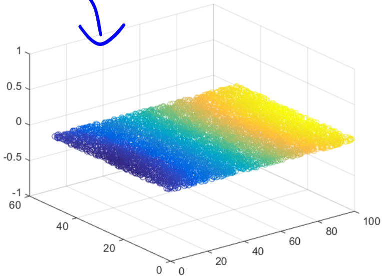
Of course, we can use multiple hidden layers...



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)

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

Goodfellow et al:
D.L.

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

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

with model

- $f(\mathbf{x}) = \langle \mathbf{w}, \phi(\mathbf{x})\rangle + b$ for regression and classification
- $f(\mathbf{x}) = \left\|\phi(\mathbf{x}) - \mathbf{w}\mathbf{w}^\top\phi(\mathbf{x})\right\|^2$ 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 $XX^\top$
  - 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