3F8: Inference

Dimensionality reduction

José Miguel Hernández–Lobato and Richard E. Turner
Department of Engineering
University of Cambridge

Lent Term

What is dimensionality reduction?

A **type of problem** in machine learning requiring

- to identify **patterns** and **regularities** in the data, related to
- finding a $\underbrace{\text{low dimensional representation}}_{\text{manifold}}$ of **relevant information**.

Example: rotation and shrinking transformations of images with the digit 4.

The resulting manifold is highly non-linear,

but can be unfolded and mapped into a **latent** plane.

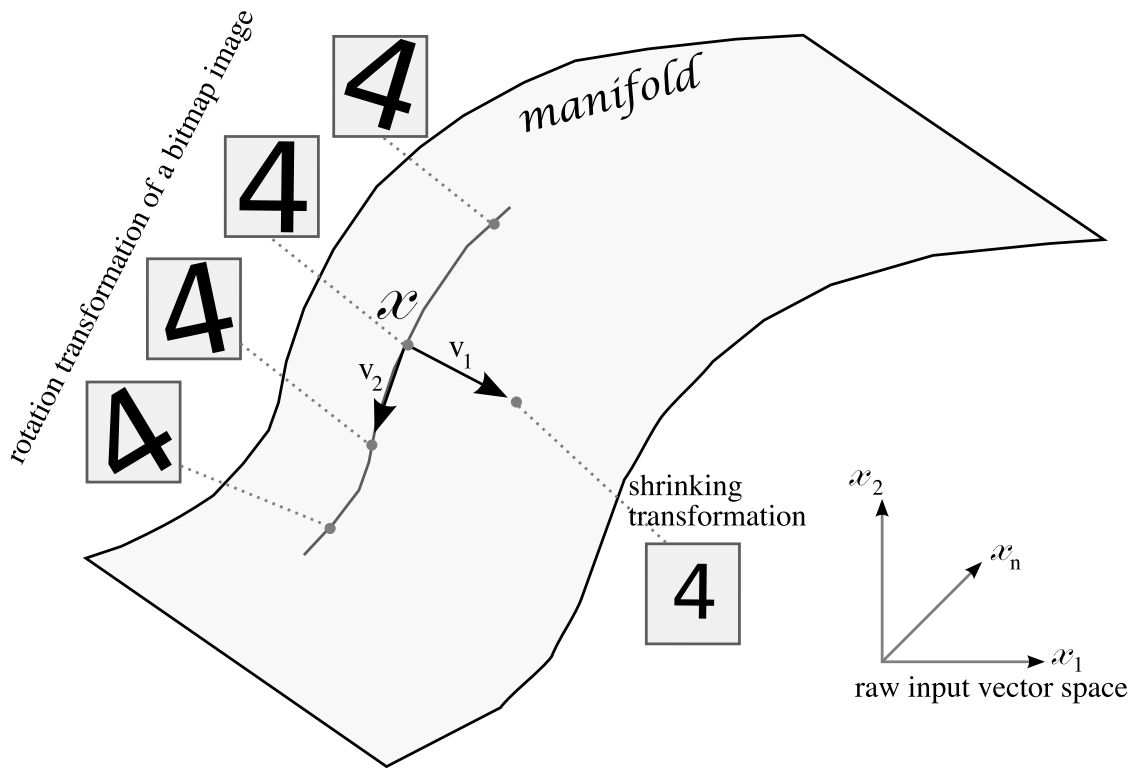
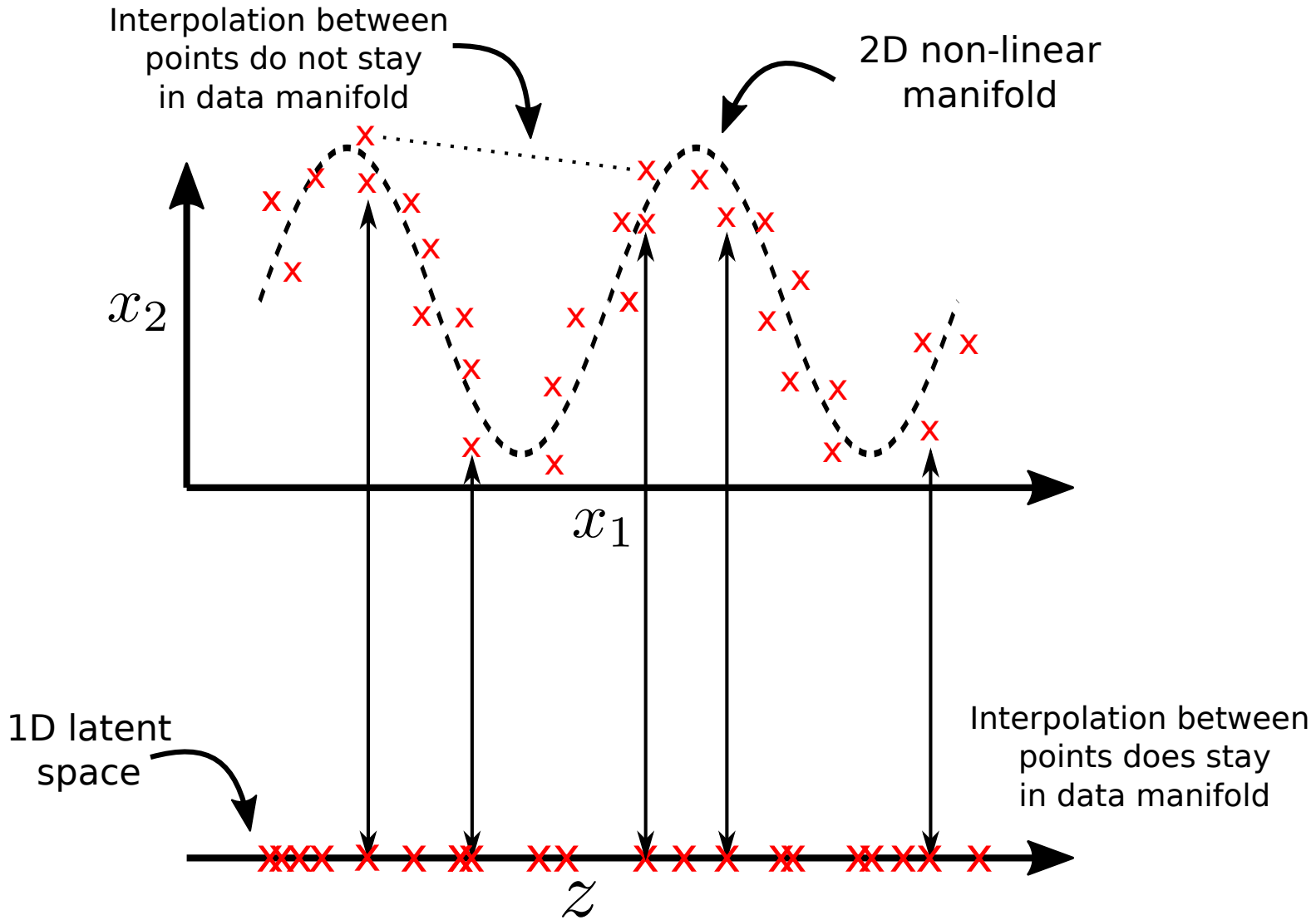


Figure: Y. Bengio, Foundations and Trends in Machine Learning, 2(1), 1–127, 2009.

Example

A one dimensional manifold in 2D space is unfolded into a latent real line:



Dimensionality reduction methods allow us to map (x_1, x_2) to z and z to (x_1, x_2).

More examples

Linear operations in latent space produce results within the data manifold.

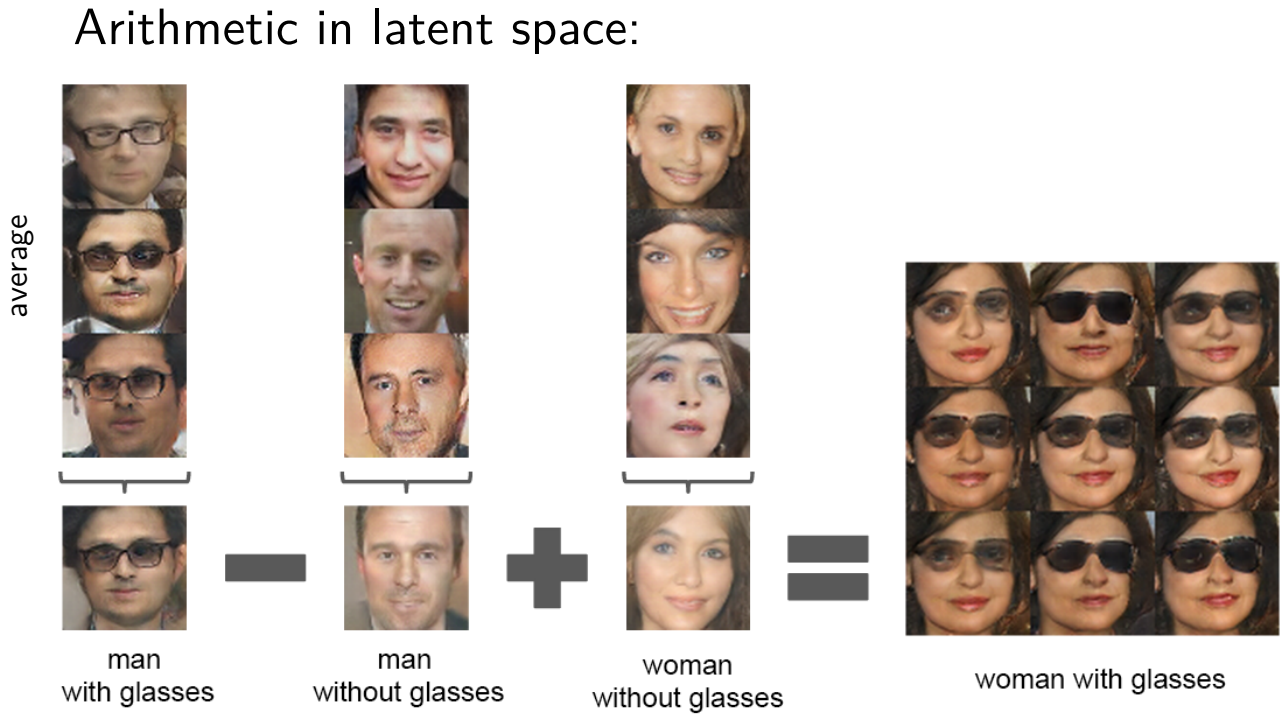


Figure: A. Radford, L. Metz, S. Chintala. ICLR 2016.

More examples

Interpolation in latent space:

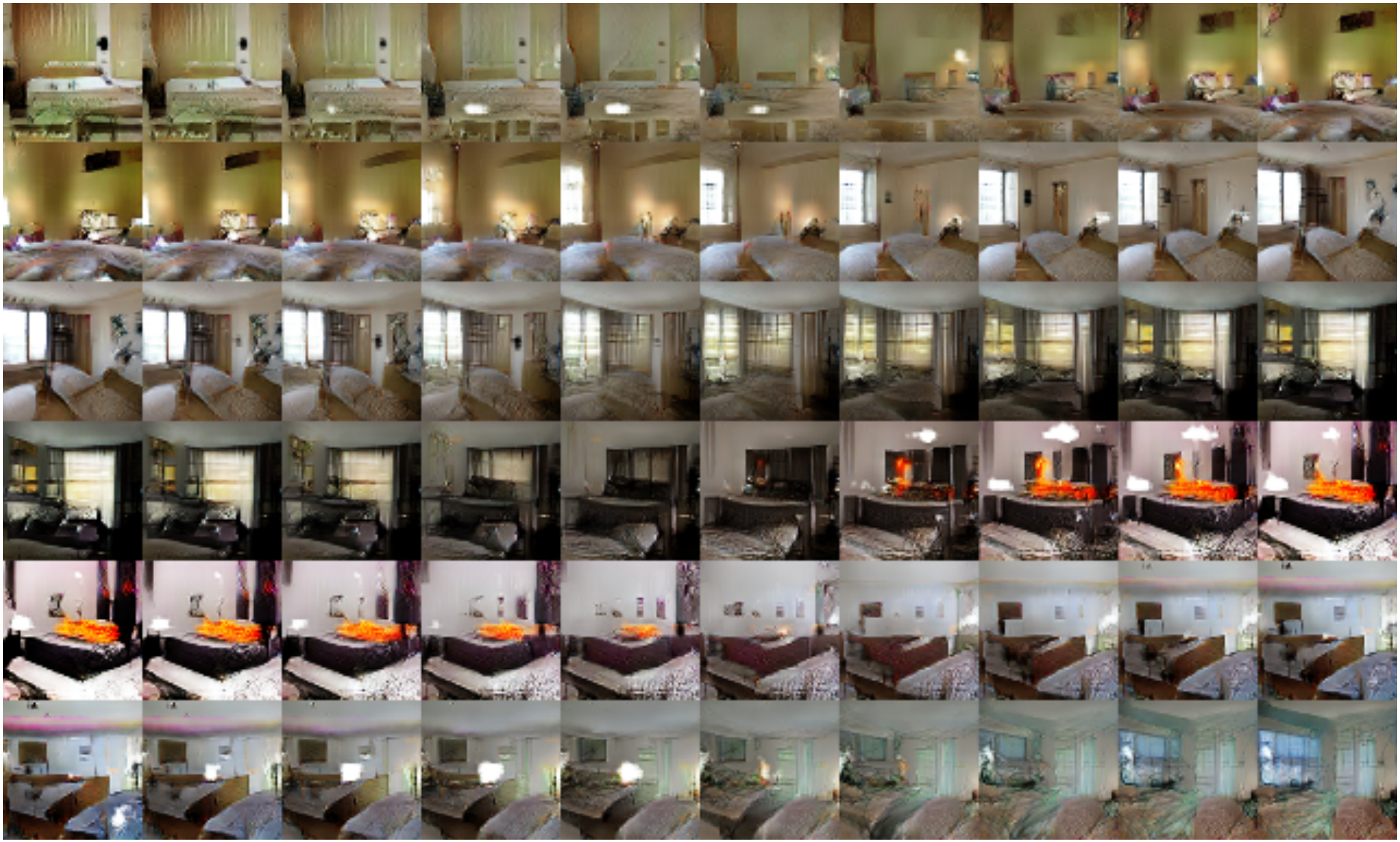


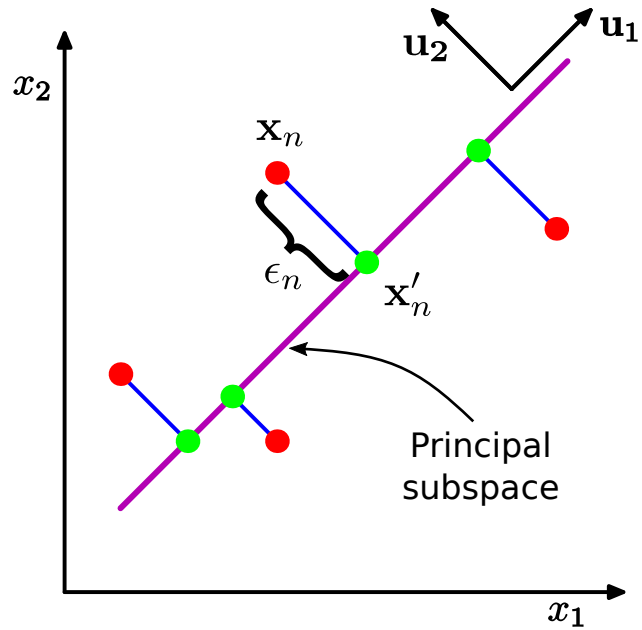
Figure: A. Radford, L. Metz, S. Chintala. ICLR 2016.

Principal component analysis (PCA)

Linear dimensionality reduction method.

Finds the project that **minimises the square reconstruction error**.

We assume we are given a dataset $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, with $\mathbf{x}_n \in \mathbb{R}^D$, $\sum_{n=1}^{N} \mathbf{x}_n = \mathbf{0}$.



By using the **orthonormal** basis $\{\mathbf{u}_i\}_{i=1}^{D}$

$$\mathbf{x}_n = \underbrace{\sum_{i=1}^{M} \mathbf{x}_n^{\mathsf{T}} \mathbf{u}_i \mathbf{u}_i}_{\mathbf{x}'_n} + \underbrace{\sum_{i=M+1}^{D} \mathbf{x}_n^{\mathsf{T}} \mathbf{u}_i \mathbf{u}_i}_{\epsilon_n},$$

where $\mathbf{x}'_n$ is the projected vector and ϵ_n is the reconstruction error.

$\mathbf{u}_1, \ldots, \mathbf{u}_D$ are called the **principal component vectors**.

$\mathbf{x}_n^{\mathsf{T}}\mathbf{u}_1, \ldots, \mathbf{x}_n^{\mathsf{T}}\mathbf{u}_D$, $n = 1, \ldots, N$, are called the **principal component scores**.

PCA cost function

Given the reconstruction errors

$$\epsilon_n = \sum_{i=M+1}^{D} \mathbf{x}_n^{\mathsf{T}} \mathbf{u}_i \mathbf{u}_i \,, \quad n = 1, \ldots, n \,,$$

PCA finds $\mathbf{u}_1, \ldots, \mathbf{u}_D$ by minimising the sum of square errors:

$$\begin{aligned}
\text{cost}(\{\mathbf{u}_i\}_{i=1}^{D}) &= \frac{1}{N} \sum_{n=1}^{N} \epsilon_n^{\mathsf{T}} \epsilon_n = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=M+1}^{D} \sum_{j=M+1}^{D} \mathbf{x}_n^{\mathsf{T}} \mathbf{u}_i \mathbf{u}_i^{\mathsf{T}} \mathbf{u}_j \mathbf{u}_j^{\mathsf{T}} \mathbf{x}_n \\
&= \frac{1}{N} \sum_{n=1}^{N} \sum_{i=M+1}^{D} \mathbf{x}_n^{\mathsf{T}} \mathbf{u}_i \mathbf{u}_i^{\mathsf{T}} \mathbf{x}_n = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=M+1}^{D} \mathbf{u}_i^{\mathsf{T}} \mathbf{x}_n \mathbf{x}_n^{\mathsf{T}} \mathbf{u}_i \\
&= \sum_{i=M+1}^{D} \mathbf{u}_i^{\mathsf{T}} \left(\frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^{\mathsf{T}} \right) \mathbf{u}_i = \sum_{i=M+1}^{D} \mathbf{u}_i^{\mathsf{T}} \hat{\mathbf{S}} \mathbf{u}_i \,, \qquad (1)
\end{aligned}$$

where $\hat{\mathbf{S}} = N^{-1} \sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^{\mathsf{T}}$ is the covariance matrix for the data.

We minimise (1) subject to $\mathbf{u}_1, \ldots, \mathbf{u}_D$ having **unit norm**. This requires to use the method of **Lagrange multipliers**!

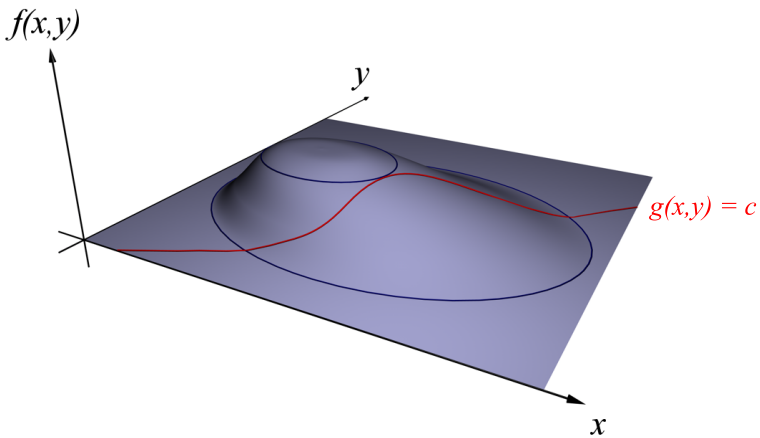
The method of Lagrange multipliers

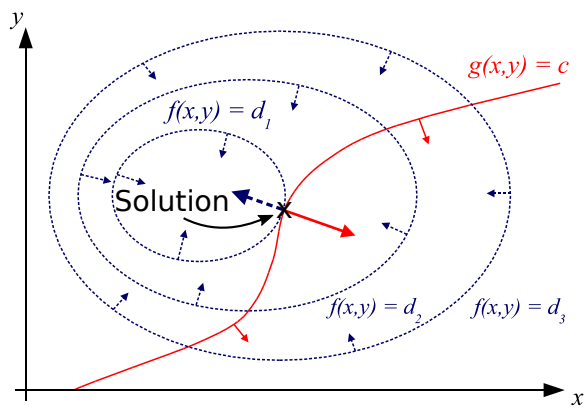
Allows us to solve optimization problems with **equality constraints**.
For example,

maximize $f(x, y)$
subject to $g(x, y) = c$.

$g(x, y) = c$ specifies a contour line of $g(x, y)$.

Gradient vectors are orthogonal to contour lines!





At the solution, the gradients of $f(x, y)$ and $g(x, y)$ are aligned!

Therefore, the solution should satisfy $\nabla_{x,y} f(x, y) = -\lambda \nabla_{x,y} g(x, y)$ and $g(x, y) = c$.

Achieved by maximising the **Lagrangian** function

$$\mathcal{L}(x, y, \lambda) = f(x, y) + \lambda(g(x, y) - c),$$

with respect to x, y and λ, where λ is called the **Lagrange multiplier**.

In particular, $\nabla_{x,y,\lambda} \mathcal{L}(x, y, \lambda) = \mathbf{0} \Leftrightarrow \nabla_{x,y} f(x, y) = -\lambda \nabla_{x,y} g(x, y)$ and $g(x, y) = c$.

PCA solution

We use **Lagrange multipliers** to guarantee that the $\{\mathbf{u}_i\}_{i=1}^D$ are normalized, that is, that they satisfy $\mathbf{u}_i^\mathsf{T}\mathbf{u}_i = 1$ for $i = 1, \ldots, D$. The resulting Lagrangian is

$$\text{cost}'(\{\mathbf{u}_i\}_{i=1}^D) = \sum_{i=M+1}^{D} \left[\mathbf{u}_i^\mathsf{T}\hat{\mathbf{S}}\mathbf{u}_i + \lambda_i(1 - \mathbf{u}_i^\mathsf{T}\mathbf{u}_i)\right] .$$

Solution: $\hat{\mathbf{S}}\mathbf{u}_i = \lambda_i\mathbf{u}_i$. The $\mathbf{u}_i$ and λ_i are **eigenvectors** and **eigenvalues** of $\hat{\mathbf{S}}$.

When we replace this solution in the original objective, we obtain

$$\text{cost}(\{\mathbf{u}_i\}_{i=1}^D) = \sum_{i=M+1}^{D} \mathbf{u}_i^\mathsf{T}\hat{\mathbf{S}}\mathbf{u}_i = \sum_{i=M+1}^{D} \lambda_i \, . \tag{2}$$

From (2), we observe that the solution is given by the $\{\mathbf{u}_i\}_{i=1}^D$ such that

- $\mathbf{u}_{M+1}, \ldots, \mathbf{u}_D$ are eigenvectors of $\hat{\mathbf{S}}$ with the $D - M$ smallest eigenvalues.
- $\mathbf{u}_1, \ldots, \mathbf{u}_M$ are eigenvectors of $\hat{\mathbf{S}}$ with the M largest eigenvalues.

Therefore, PCA also maximises **the variance of the projected data**.

How to choose M?

No single best solution, but we can **sort** eigenvalues and **visualise** them.

Then choose M to be the value beyond which there is an **inflection point**.

The remaining eigenvalues are small with eigenvectors that capture just **noise**.

However, the inflection point is not always clear!

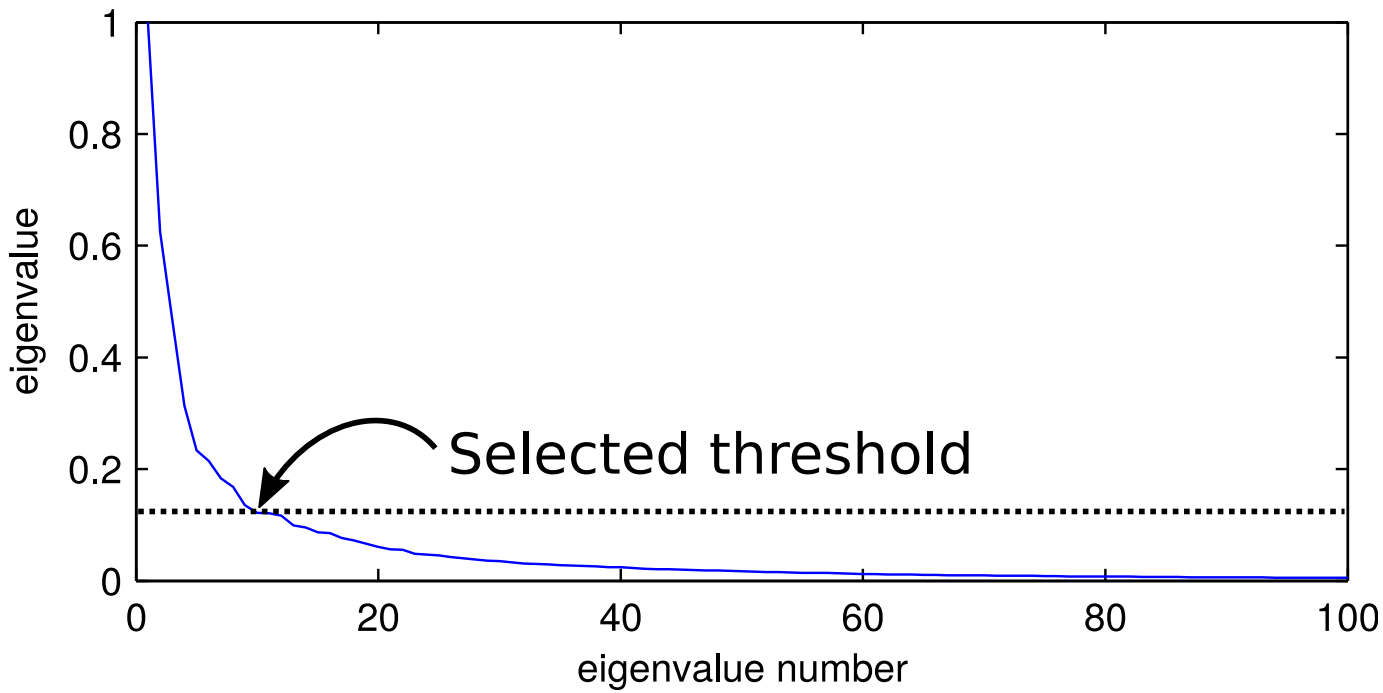


Figure: D. Barber. *Bayesian Reasoning and Machine Learning*, 2007.

PCA example

Data set with many digits 3, all centered and having the same scale.

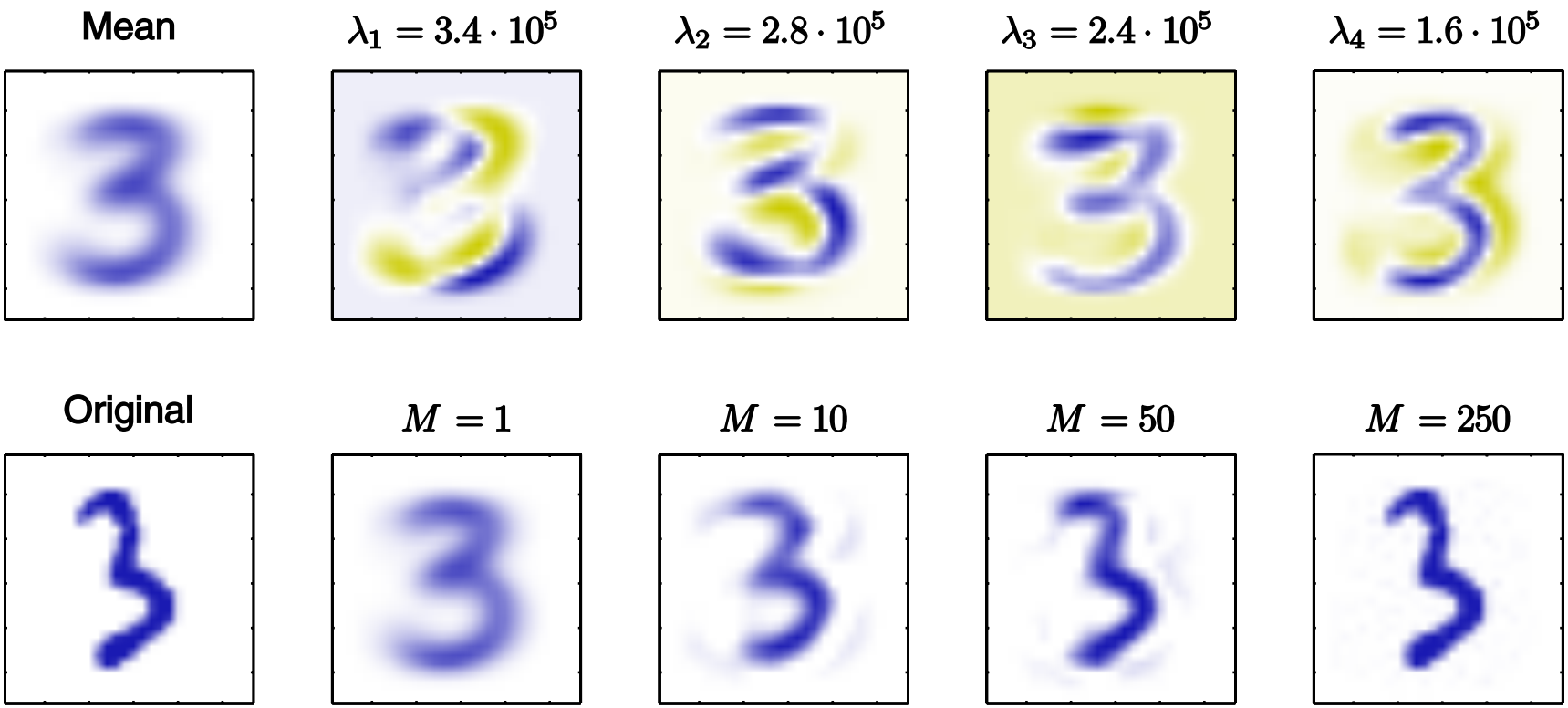


Figure: C. Bishop. *Pattern Recognition and Machine Learning*, 2006.

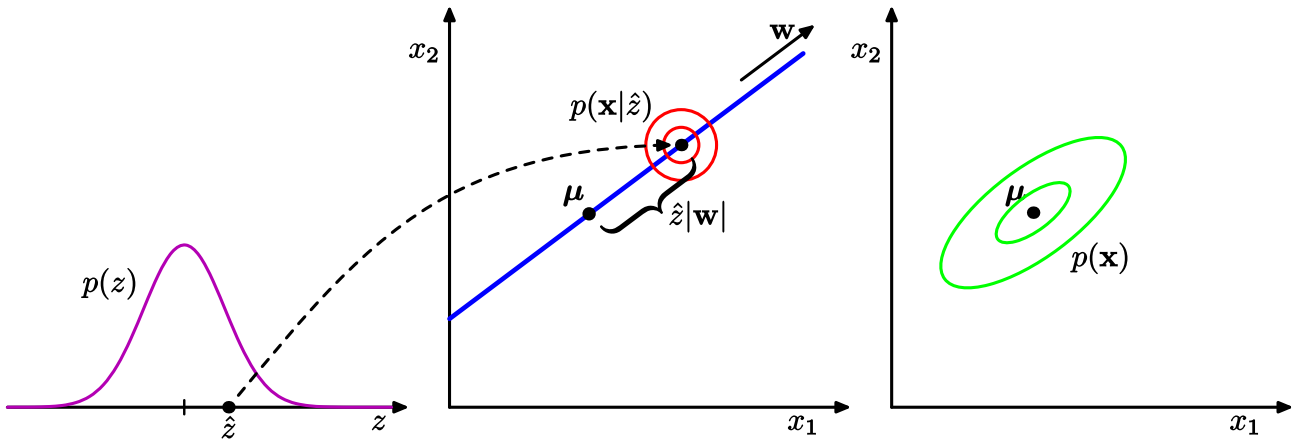
Probabilistic PCA

PCA can be formulated as the MLE of a **generative linear Gaussian model**:

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{0}|\mathbf{I}),$$
$$p(\mathbf{x}|\mathbf{W}, \mathbf{z}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma^2\mathbf{I}),$$

The data is assumed to have a **low-rank** plus diagonal covariance matrix:

$$p(\mathbf{x}|\mathbf{W}, \sigma) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{W}\mathbf{W}^{\mathsf{T}} + \sigma^2\mathbf{I}).$$



Let the columns of $\mathbf{U}$ contain the first M eigenvectors of $\hat{\mathbf{S}}$. Then

$$\sigma^2_{\text{MLE}} = \frac{1}{D-M}\sum_{i=M+1}^{D}\lambda_i. \qquad \mathbf{W}_{\text{MLE}} = \mathbf{U}\text{diag}(\sqrt{\lambda_1 - \sigma^2_{\text{MLE}}}, \ldots, \sqrt{\lambda_M - \sigma^2_{\text{MLE}}}).$$

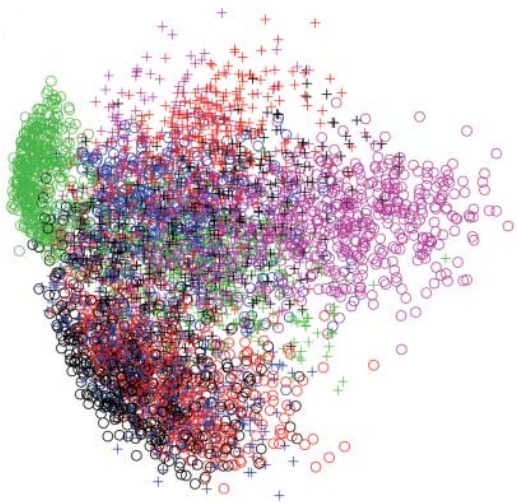
Tipping and Bishop. Journal of the Royal Statistical Society, Series B, 61(3), 1999.
Bishop. *Pattern Recognition and Machine Learning*, 2006.

Non-linear dimensionality reduction

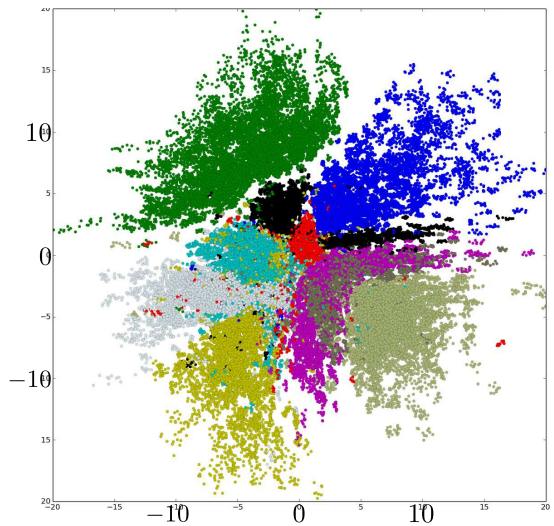
PCA fails when the **data manifold is non-linear**. The solution is to use a **non-linear generative model**.

Example: MNIST data set of handwritten digits.

PCA



Non-linear generative model



Hinton and Salakhutdinov. Science, 313, 2006.
A. Makhzani et al. ICLR Workshop 2016.