MSA220 Project 3, Survey on non-linear dimensionality reduction method

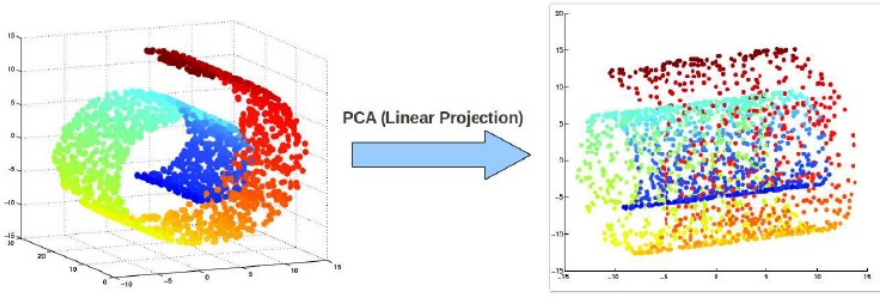
Thomas Elsken
thomas.elsken@uni-muenster.de

Elouan Tardivel
elouan@student.chalmers.se

1 Abstract

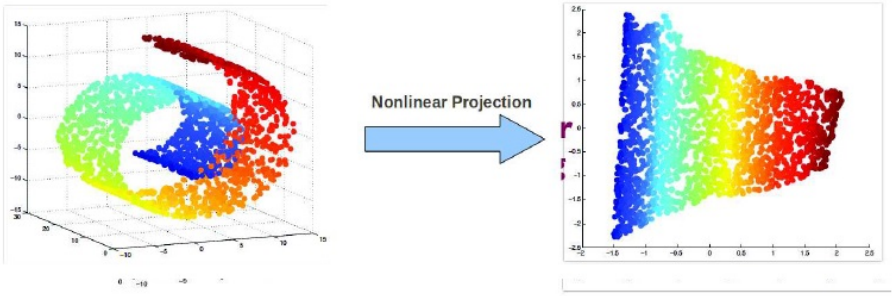
PCA is widely applied in research areas like Statistics and Machine Learning to deal with high dimensional data. However, the dimensionality reduction is done by linear projection and therefore not suitable for data whose structure is non linear. For example, consider the Swiss roll in figure 1. To find an appropriate lower dimensional representation we need methods who are capable of exploring the non linear structure, see figure 2.
In this report we will recap the most popular non linear dimensionality reduction methods.

Figure 1: PCA applied to Swiss Roll. Reference: [5]



Some Notation: The original data is denoted by $x_1, \dots, x_n$, where $x_i \in \mathbb{R}^p$ and the lower dimensional data we want to deduce by $\psi_1, \dots, \psi_n$, $\psi_i \in \mathbb{R}^k$.

Figure 2: Non-linear dimensionality reduction applied to Swiss roll. Reference: [5]



2 Kernel PCA

The idea of Kernel PCA is applying ordinary PCA in a transformed vector space. To avoid too much computational effort (the dimension of the transformed vector space might be larger than the original one, possibly even infinite dimensional) and caring about a good transformation, Kernel PCA makes use of the 'kernel trick', that is widely applied in Statistics and Machine Learning. In what follows is a short description.

Suppose an algorithm you want to apply depends only on scalar products of data $x_1, \ldots, x_n \in \mathbb{R}^p$. If we wanted to apply the algorithm in a transformed space $\mathbb{V} = \phi(\mathbb{R}^p)$ with data $\phi_1, \ldots, \phi_n$, we don't really need to know the ϕ_i's since the algorithm only depends on scalar products. It would be sufficient to have a function $k : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ with $k(x, y) = \phi(x)^t \phi(y)$. Such a function is called kernel function. Ergo we can run the algorithm on the transformed data without actually having to compute them.
Let us go back to ordinary PCA and see how to utilize this idea. For simplicity we **assume zero-mean data in the original space as well as in the transformed space during the whole section**, i.e.

$$\frac{1}{n}\sum_{i=1}^{n} x_i = 0 = \frac{1}{n}\sum_{i=1}^{n} \phi_i.$$

Given the sample covariance matrix in the original space $\mathbb{R}^p$.

$$C_{\mathbb{R}^p} = \frac{1}{n}\sum_{i=1}^{n} x_i x_i^t,$$

ordinary PCA looks for the eigenvectors corresponding to large eigenvalues of this matrix, representing the principal components and using them to project

the data. Whereas in the transformed space we would do the same things with respect to

$$C_{\mathbb{V}} := \frac{1}{n}\sum_{i=1}^{n}\phi_i\phi_i^t,$$

i.e. look for $v \in \mathbb{V}, \lambda \in \mathbb{R}$ with

$$Cv = \lambda v.$$

From the above equation we get

$$v = \sum_{i=1}^{n} a_i\phi_i$$

with $a_i = \frac{1}{n\lambda}\phi_i^t v$. Plugging this back in the eigenvector equation yields

$$\frac{1}{n}\sum_{i=1}^{n}\phi_i\phi_i^t\sum_{j=1}^{n}a_j\phi(x_j) = \lambda\sum_{i=1}^{n}a_i\phi_i.$$

Multiplying this equation by $\phi(x_k)^t$ from the left for some k and rearranging some terms gives

$$\frac{1}{n}\sum_{i=1}^{n}\phi(x_k)^t\phi_i\sum_{j=1}^{n}a_j\phi_i^t\phi(x_j) = \lambda\sum_{i=1}^{n}a_i\phi(x_k)_{\phi_i}^t,$$

which can also be formualted as

$$\frac{1}{n}\sum_{i=1}^{n}k(x_k,x_i)\sum_{j=1}^{n}a_jk(x_i,x_j) = \lambda\sum_{i=1}^{n}a_ik(x_k,x_i).$$

Doing this for $k = 1,\ldots,n$ gives n equations. In compact form:

$$K^2a = \tilde{\lambda}Ka, \tag{1}$$

with the kernel matrix $K = (k_{i,j})_{i,j=1}^n, k_{i,j} = k(x_i,x_j), a = (a_1,\ldots,a_n)^t$ and $\tilde{\lambda} = n\lambda$. Now clearly any solution of

$$Ka = \tilde{\lambda}a \tag{2}$$

is also a solution of 1. But that is it! We have found a n-dimensional eigenvalue problem that corresponds to the eigenvalue problem we had to solve when applying ordinary PCA in $\mathbb{V}$. So Kernel PCA solves 2 and uses the solutions as principal components to construct the lower dimensional representation $\psi_1,\ldots,\psi_n$.

For further details (e.g. for handling non-zero-mean data) see [1]. Also have a look at [5].

3 Manifold methods

3.1 Principal Curves

Remember that one way to derive the principal components $v_1, \dots v_m$ in ordinary PCA is to minimize

$$\sum_{i=1}^{n} ||x_i - \sum_{j=1}^{k} x_i^t v_j v_j||^2,$$

where k is the desired reduced dimension. To get a non-linear generalization, an obvious idea is to replace $\sum_{j=1}^{k} x_i^t v_j v_j$ (i.e. the linear projection of x_i) by an arbitrary curve $f(t) : [a, b] \to \mathbb{R}^p$, yielding the optimization problem

$$\min_{f \in G} \sum_{i=1}^{n} ||x_i - f(\lambda(x_i))||^2$$

with the projection index $\lambda(x) : \mathbb{R}^p \to [a, b]$ defined by

$$\lambda(x) := \sup_t \{t : ||x - f(t)|| = \inf_{\tilde{t}} ||x - f(\tilde{t})\}, \tag{3}$$

i.e. the largest value t for which $f(t)$ is closest to x. G is some suitable space of curves. Now think of the data as samples from a RV X with probability density function h. Then, a reasonable extension of equation 3.1 would be to minimize the squared distance functional

$$D^2(f) := \mathbb{E}_h ||X - f(t(X))||^2, \quad D^2 : G \to \mathbb{R},$$

A necessary condition for a function $f : \mathbb{R}^n \to \mathbb{R}$ having a minimum at x_{min} is $\frac{d}{d\epsilon} f(x_{min} + \epsilon \tilde{x}) \mid_{\epsilon=0} = 0$ for every $\tilde{x}$, which simplifies to $\nabla f(x_{min}) = 0$. One can derive an analogous condition for minimizing a functional, which is in our case

$$\frac{d}{d\epsilon} D^2(f + \epsilon g) \mid_{\epsilon=0} = 0 \quad \text{for every } g \in G. \tag{4}$$

Definition 3.1. *A curve f is called a principal curve of a RV X with PDF h if*

$$f(t) = \mathbb{E}_h[X | \lambda(X) = t].$$

The following theorem justifies the above definition.
Theorem 3.2. *A curve f is a principal curve iff it satisfies equation 4.*

(Rough) proof. Applying dominated convergence theorem, one can show

$$\frac{d}{d\epsilon} D^2(f + \epsilon g) = \mathbb{E}_h[\frac{d}{d\epsilon} ||X - f + \epsilon g||^2],$$

i.e. interchanging integration and differentiation. Noticing

$$\begin{aligned}||X-(f+\epsilon g)||^2 &= ||(X-f)-\epsilon g)||^2 \\ &= ||X-f||^2+\epsilon^2||g||^2-2\epsilon(X-f)^t g\end{aligned}$$

we compute using some properties of conditional expectations

$$\begin{aligned}\frac{d}{d\epsilon}D^2(f(t)+\epsilon g(t))\mid_{\epsilon=0} &= \mathbb{E}_h\big[\frac{d}{d\epsilon}\big(||X-f(t)||^2+\epsilon^2||g(t)||^2-2\epsilon(X-f(t))^t g(t)\big)\mid_{\epsilon=0}\big] \\ &= -2\mathbb{E}_h\big[(X-f(t))^t g(t)\big] \\ &= -2\mathbb{E}_h\Big[\big(\mathbb{E}[X|\lambda(X)=t]-f(t)\big)^t g(t)\Big].\end{aligned}$$

Now, if f is a principal curve, the above equation becomes 0, i.e. a principal curve satisfies equation 4.
On the other side suppose

$$-2\mathbb{E}_h\Big[\big(\mathbb{E}[X|\lambda(X)=t]-f(t)\big)^t g(t)\Big]=0$$

for every g. Then by some version of the Fundamental lemma of calculus of variations we have $\mathbb{E}[X|\lambda(X)=t]=f(t)$ a.s..

For a strict proof see [9].

□

The authors of [9] suggest the following iterative algorithm to compute a principal curve.
3.3 (Hastie Stuetzle Principal Curve algorithm (HSPC))**.** *Initialize* $f^{(0)}(t)=\bar{x}+vt$*, where* v *is the first linear principal component.*
Iterate (denoting the iteration number by j):

- *(Projection step) Compute the projection index (see equation 3)* $\lambda^{(j)}(x)$ *w.r.t* $f^{(j)}$*.*
- *(Expectation step) Update f:* $f^{(j+1)}(t)=\mathbb{E}_h[X|\lambda^{(j)}(x_i)=t]$

Stop, if changes in $D^2(f)$ *are below some threshold.*

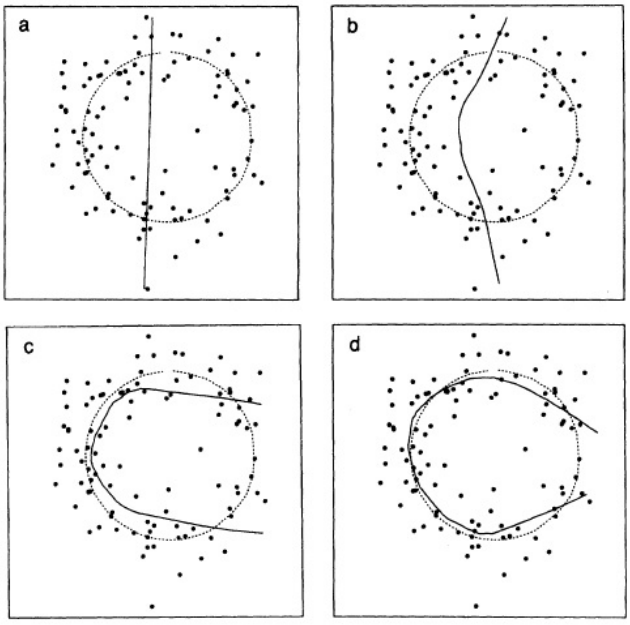
Figure 3 illustrates the algorithm.

Again, for further details see the original paper [9]. Also have a look at [10], where extensions of the algorithm are described.

3.2 Characteristic preserving methods

In this subsection we consider methods that try to preserve some characteristics of the data.

Figure 3: Application of HSPC algorithm to noisy data lying on a circle. 4 iterations (a,b,c,d). Reference: [9]



3.2.1 Mutlidimensional Scaling

Mutlidimensional Scaling (MDS) tries to preserve pairwise euclidean distances in the original data. The ordinary MDS (also known as metric MDS) algorithm yields a **linear dimensionality reduction**, in fact it yields the same results as PCA. Although this is a linear method we introduce it because it will help to understand some of the following methods.

3.4 (metric MDS algorithm)**.**

- *Compute the squared distance matrix* $D^2 = (d_{i,j}^2)_{i,j=1}^n$, *where* $d_{i,j} = ||x_i - x_j||_2$
- *Apply double centering:* $B = -\frac{1}{2} C D^2 C$ *using the centering matrix*

$$C = I - \frac{1}{n} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} (1, \ldots, 1)^{\,1}$$

- *Compute the k largest eigenvalues* $\lambda_1, \ldots, \lambda_k$ *with corresponding eigenvec-*

[1] Multiplying the centering matrix with a vector has the same effect as subtracting the mean of the components from every component.

tors $v_1, \ldots, v_k$.

- *Compute the new data representation*

$$\begin{pmatrix} \psi_1^t \\ \vdots \\ \psi_n^t \end{pmatrix} = V\Lambda^{1/2},$$

where V *is the matrix of eigenvectors and* Λ *the diagonal matrix with* $diag(\Lambda) = (\lambda_1, \ldots \lambda_k)$.

3.2.2 Isometric Feature Mapping

Isometric Feature Mapping (Isomap) is very similar to MDS - the only difference is to choose a more appropriate distance matrix in the beginning. Have a look at figure 4. The data obviously lies on a manifold. Consider the two data points A and B. The dashed line represents the shortest route in $\mathbb{R}^3$ (euclidean distance in $\mathbb{R}^3$) between them, whereas the non-dashed line represents the shortest route from A to B on the manifold - the so called geodesic distance. Since we want to investigate the structure of the manifold it should be a good idea to use the geodesic distance instead of the euclidean. Unfortunately it is hard to compute. Isomap approximates the geodesic distance as follows: Create a weighted graph whose nodes corresponds to a data point. For each point, find its L nearest neighbours (L is a hyperparameter) based on euclidean distance. If x_i and x_j are L-nearest neighbours, add an edge between them in the graph with weight equal to euclidean distance. Then, for an arbitrary pair of points the geodesic distance is approximated by its shortest path in the graph, which can be computed by e.g.the Djikstras algorithm or the Floyd-Warshall algorithm.

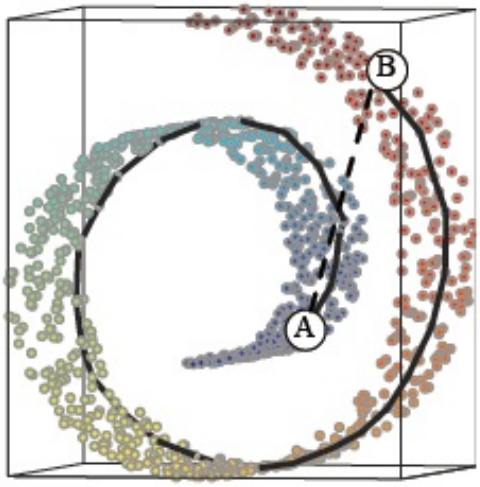
In summary

3.5 (Algorithm: Isomap)**.**

- *For each data point, find its L nearest neighbours based on euclidean distance.*
- *Create a graph* $G = (V, E)$*, where* $V = (x_1, \ldots, x_n)$ *and* $(x_i, x_j) \in E \Leftrightarrow x_i, x_j$ *are L nearest neighbours.*
- *Assign weights* $w_{i,j}$ *to each edge* (x_i, x_j)*:* $w_{i,j} = ||x_i - x_j||_2$.
- *For every pair of nodes* x_i, x_j*, compute the shortest path* $d_{i,j}$ *in the graph.*
- *For* $D^2 = (d_{i,j}^2)_{i,j=1}^n$*, apply double centering:* $B = -\frac{1}{2}CD^2C$ *using the centering matrix*

$$C = I - \frac{1}{n}\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}(1, \ldots, 1)$$

Figure 4: Swiss roll. Euclidean distance (dashed line) vs. geodesic distance from A to B. Reference: [2]



- *Compute the k largest eigenvalues* $\lambda_1, \ldots, \lambda_k$ *with corresponding eigenvectors* $v_1, \ldots, v_k$.
- *Compute the new data representation*

$$\begin{pmatrix} \psi_1^t \\ \vdots \\ \psi_n^t \end{pmatrix} = V\Lambda^{1/2},$$

where V *is the matrix of eigenvectors and* Λ *the diagonal matrix with* $diag(\Lambda) = (\lambda_1, \ldots \lambda_k)$.

For further details see [1], [2] and [5].

3.2.3 Locally Linear Embedding

Locally Linear Embedding assumes - as the name reveals - a local linearity in the data and tries to preserve this when projecting to a lower dimensional subspace. As well as Isomap, the algorithm starts with finding L nearest neighbours for each data point. Denote the neighbourhood set of x_i by $N(x_i)$. By the local linearity assumption, one should then by able to write

$$x_i \approx \sum_{j \in N(x_i)} w_{i,j} x_j$$

for some weights $w_{i,j}$. These weights are found by solving the least-squares problem

$$\begin{aligned} W = \underset{W\in\mathbb{R}^{n\times n}}{\arg\min}\, E_X(W) = \underset{W\in\mathbb{R}^{n\times n}}{\arg\min} \sum_{i=1}^{n} ||x_i - \sum_{j\in N(x_i)} w_{i,j}x_j||_2^2 \\ \text{subject to } \sum_{j\in N(x_i)}^{L} w_{i,j} = 1 \text{ for every } i, \\ w_{i,j} = 0 \text{ for every } i,j : j \notin N(x_i) \end{aligned} \tag{5}$$

The first set of constraints ensure E_X to be invariant under data translation $x_i \to x_i + z$.

The embedded points $\psi = (\psi_1, \ldots, \psi_n), \psi_i \in \mathbb{R}^k$ are than found by solving

$$\begin{aligned} \psi = \underset{\psi\in\mathbb{R}^{n\times k}}{\arg\min}\, E_W(\psi) = \underset{\psi\in\mathbb{R}^{n\times k}}{\arg\min} \sum_{i=1}^{n} ||\psi_i - \sum_{j\in N(x_i)} w_{i,j}\psi_j||_2^2 \\ \text{subject to } \sum_{i=1}^{n}\psi_i = 0,\ \frac{1}{n}\sum_{i=1}^{n}\psi_i\psi_i^t = I. \end{aligned} \tag{6}$$

Without the first constraint, any translation of a solution would obviously also be a solution. To get rid of that we force a zero-mean solution (i.e. constraint 1).The second constraint eliminates the degenerated solution $\phi = 0$.
3.6 (Algorithm: LLE)**.**

- *For each data point, find its L nearest neighbours based on euclidean distance.*
- *Find the weights W describing the local linear structure, i.e. solve 5.*
- *Find the embedding best preserving this structure, i.e. solve 6.*

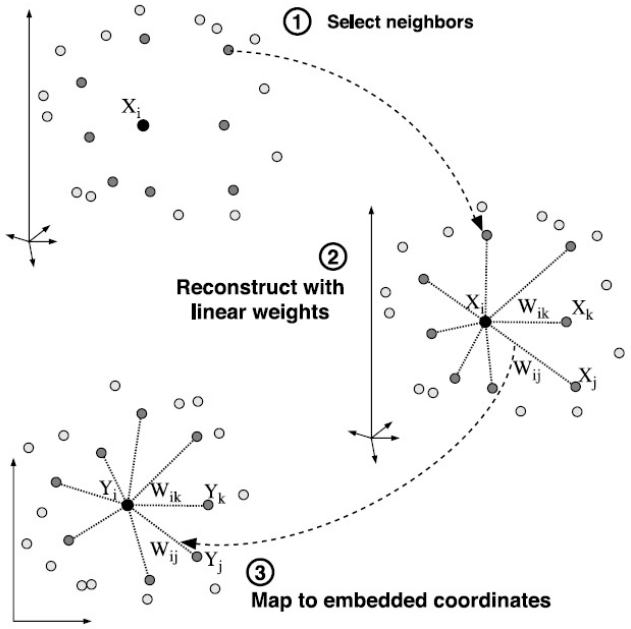
The algorithm is illustrated in figure 5.

Doing some basic calculations, one can show that the optimization 6 can be transformed into an eigenvalue problem. In fact, the embeddings $\psi_1, \ldots, \psi_n$ correspond to the eigenvectors of $(I - W)^t(I - W)$ with smallest eigenvalues. The subsection is based on [1], [2] and the original paper [6].

3.2.4 Laplacian Eigenmaps

Laplacian Eigenmaps are similar to Isomap. They also try to preserve proximity relationships in the data. However they define proximity in a different way and therefore yield other results. You will also see some similarities to the LLE method.

Figure 5: Illustration of the LLE algorithm. Reference: [6]



As for the Isomap, we start with building a weighted graph $G = (V, E)$ corresponding to the data. One can again compute L nearest neighbours to define E as described in subsection 3.2.2. An alternative suggested by the authors of the original paper is to put an edge (x_i, x_j) in the graph iff $||x_i - x_j||_2 < \epsilon$ for some threshold ϵ.
After doing that, we compute weights for the edges, where weights are interpreted as a similarity measure (i.e. small distances induce large weights), e.g.

$$w_{i,j} = \exp \frac{-||x_i - x_j||_2^2}{\sigma^2}$$

with some scaling parameter $\sigma \in \mathbb{R}$.

As in the case of LLE, we then set up an objective function, whose minimization yields the lower dimensional data representation:

$$E_W(\psi) = \sum_{i,j=1}^{n} (\psi_i - \psi_j)^2 w_{i,j} \tag{7}$$

Very similar points x_i, x_j (i.e. large $w_{i,j}$) force ψ_i, ψ_j to be similar. On the other hand, if x_i, x_j are not similar at all (i.e. $w_{i,j} \approx 0$), ψ_i, ψ_j do not have to be too. Hence E_W is a proper choice for our purpose. Again constraints are needed to exclude degenerated solutions and obtain some nice geometrical

properties (invariances under translation/rotation).
Why the name Laplacian eigenmaps? Again, one can show that the above optimization corresponds to an (generalized) eigenvalue problem, in fact to the problem

$$Lv = \lambda Dv, \tag{8}$$

where D is the diagonal matrix with $d_{i,i} = \sum_{j=1}^{n} w_{j,i}$ and $L = D - W$ is the Laplacian of the graph. Denoting the eigenvectors belonging to the k smallest eigenvalues (excluding the eigenvalues equal to 0) by $v_1, \dots v_k$, the embedding is given by

$$x_i \to (v_1(i), \dots, v_k(i))^t. \tag{9}$$

3.7 (Algorithm: LEM).

- *Set up a graph $G = (V, E)$ with $V = (x_1, \dots, x_n)$ and $(x_i, x_j) \in E \Leftrightarrow x_i, x_j$ are close in some sense.*
- *Compute weights $w_{i,j}$ that are interpretable as a similarity measure.*
- *Solve the generalized eigenvalue problem 8.*
- *Compute the embedding of $x_1, \dots, x_n$ as described in 9.*

Detailed explanations can be found in the original paper [4].

4 Autoencoder

4.1 Introduction to neural network

A neural network in general can be described as such. There are two fundamental entities: neurons and connections between them. These connections are directed, and each neuron has several input connections and a single output, which branches out to reach multiple other neurons. Each Neuron have an activity level determined by an associated transfer function. The branches have weights that describe the connection between neurons.

There are many different neural networks, but we will focus on feedforward neural networks (FNN). In a FNN, which is an acyclic directed graph, the information go through the network in only one direction, from the input layer to output layer. Between the input layer and the output layer there is what is called the hidden layer which gather a certain number of hidden units. In the simplest FNN you only have one hidden layer but in more complex structure the number of hidden layers can reach 150, see Microsoft team in the latest ImagetNet challenge.

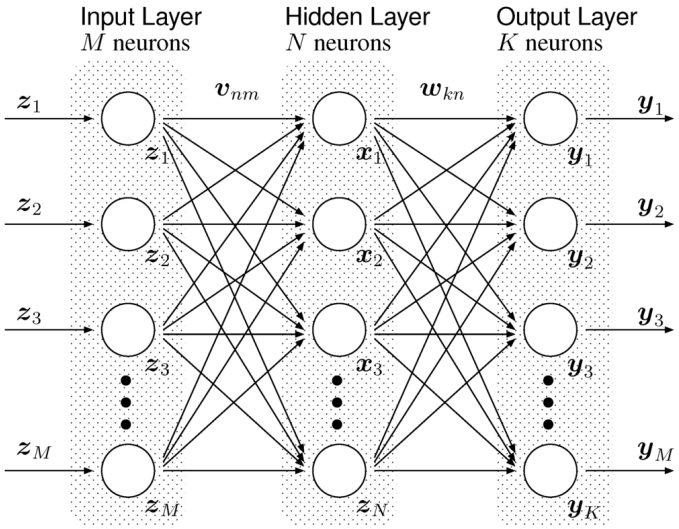


Figure 6: Multiple layers perceptron from [8] with one hidden layers

4.2 Standard Nonlinear PCA

NLPCA is based on an autoencoder neural network, which is a multi-layer perceptron with auto-associative topology. In a regular deep learning approach we are in supervised setting, the goal is to predict the output for a given input, the input is map to the output by a function $\hat{y} = f(x)$ and the learning phase implies reducing the difference between the output $\hat{y}$ and the real value y by minimising the loss function. The idea behind the autoencoder is different as we want the same output and input. Therefore, it is an unsupervised learning. It is achieved by minimising the error $E = \frac{1}{2}\hat{x} - x\|^2$. However, if the network has a high capacity the risk is to be able to completely match the input. This is indeed, useless because nothing is learned from the data, as the data is only copied perfectly. We will see that the network is constrained in some ways so that only an approximated copy of data are created.

The autoencoder is formed of two distinct parts, the encoder and the decoder which can be represented by an extraction function $\phi_{extr} : X \to Z$ and a reconstruction function $\phi_{gen} : Z \to \hat{X}$, respectively.

The main goal is to compress the data to reveal the meaningful components of its structure. Thus, proceed to dimension reduction by learning the useful properties of the training data. This is achieved by constraining the number of units in the middle layer (i.e the output of the encoder and input of the decoder). This middle hidden layer describes a code to represent the input. A regularisation is also needed to constrain the capacity of the network, therefore a *weight decay* term is added to the loss function $E_{total} = E + \nu \sum_i w_i^2$ to avoid

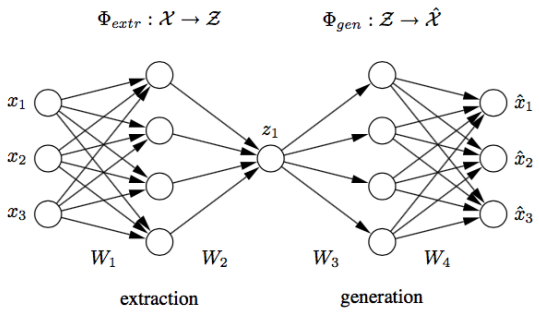


Fig. 2.2. Standard autoassociative neural network. The network output $\hat{\boldsymbol{x}}$ is required to be equal to the input $\boldsymbol{x}$. Illustrated is a 3-4-1-4-3 network architecture. Biases have been omitted for clarity. Three-dimensional samples $\boldsymbol{x}$ are compressed to one component value $\boldsymbol{z}$ in the middle by the extraction part. The inverse generation part reconstructs $\hat{\boldsymbol{x}}$ from $\boldsymbol{z}$. The sample $\hat{\boldsymbol{x}}$ is usually a noise-reduced representation of $\boldsymbol{x}$. The second and fourth layer, with four nonlinear units each, enable the network to perform nonlinear mappings. The network can be extended to extract more than one component by using additional units in the component layer in the middle

Figure 7: Simple autoencoder from [3]

large weights in the net.

4.3 Hierarchical Nonlinear PCA

The Nonlinear PCA method provides an optimal nonlinear subspace, this subspace does not have to be unique because the goal is only to operate dimensionality reduction. By contrast, it is sometime required to achieve features extraction, in that case not only the subspace matter but also the way each component appear in this subspace. That is why Scholz and Vigário [7] had introduced the hierarchical NLPCA.

By successively considering and adding new component (see figure [8]), we can express the hierarchical error function as follow :

$$E_H = E_1 + E_{1,2} + E_{1,2,3} + \Delta\Delta\Delta + E_{1,2,3,\ldots,k}$$

s

For the hierarchical condition to hold we have to search for a k-dimensional subspace of minimal mean square error (MSE) under the constraint that the (k 1)-dimensional subspace is also of minimal MSE.

4.4 Inverse Nonlinear PCA

Another interesting approach is the Inverse Model id NonLinear PCA. The classical problem implies predicting the output from the input. By contrast, the

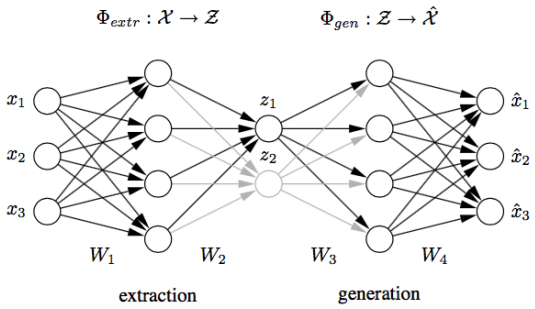


Fig. 2.3. Hierarchical NLPCA. The standard autoassociative network is hierarchically extended to perform the hierarchical NLPCA (h-NLPCA). In addition to the whole 3-4-**2**-4-3 network (grey+black), there is a 3-4-**1**-4-3 subnetwork (black) explicitly considered. The component layer in the middle has either one or two units which represent the first and second components, respectively. Both the error E_1 of the subnetwork with one component and the error of the total network with two components are estimated in each iteration. The network weights are then adapted at once with regard to the total hierarchic error $E = E_1 + E_{1,2}$

Figure 8: Hierarchical autoencoder from [3]

inverse problem tries to estimate the input that matches the best a given output. This is, therefore, a generative process which only need the reconstruction layers of the previous problem as we can see in figure 9. This method presents many assets, first, it is often easier to model the data generation process than the extraction. Secondly, because we use only the data to compute the partial error, the data can be incomplete. That allow a certain a flexibility when handling missing value as not prepossessing of the data is required. This performance can be seen on the figure [10], where the inverse model does a really good job at matching the Helix. By contrast, the basic PCA is as uninformative as performing a mean in that situation. Moreover, there are fewer weights to update, so the method is more efficient.

By minimising the reconstruction error we can estimate the parameter z, w of the model.

$$E(w, z) = \frac{1}{2} \sum_{n}^{N} \sum_{i}^{p} [\sum_{j}^{h} w_{ij} g(\sum_{i}^{m} w_j k z_k^n) - x_i^n]^2$$

The parameter z contains the lower dimension representation of the data. If we want the Inverse NLPCA to extract more than one component, we can increase the number of units in the input layer. The inverse NLPCA has the advantage of being able to extract components of higher nonlinear complexity compared to the standard NLPCA.

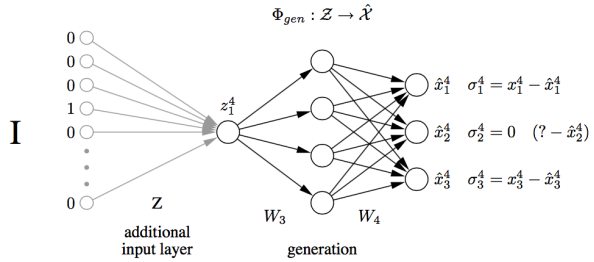


Fig. 2.5. The inverse NLPCA network. Only the second part of the autoassociative network (Figure 2.2) is needed, as illustrated by a 1-4-3 network (black). This generation part represents the inverse mapping Φ_{gen} which generates or reconstructs higher-dimensional samples $\boldsymbol{x}$ from their lower dimensional component representations $\boldsymbol{z}$. These component values $\boldsymbol{z}$ are now unknown inputs that can be estimated by propagating the partial errors σ back to the input layer $\boldsymbol{z}$. This is equivalent to the illustrated prefixed input layer (grey), where the weights are representing the component values $\boldsymbol{z}$. The input is then a (sample x sample) identity matrix I. For the 4th sample (n=4), as illustrated, all inputs are zero except the 4th, which is one. On the right, the second element x_2^4 of the 4th sample $\boldsymbol{x}^4$ is missing. Therefore, the partial error σ_2^4 is set to zero, identical to ignoring or non-back-propagating. The parameter of the model can thus be estimated even when there is missing data

Figure 9: Inverse autoencoder from [3]

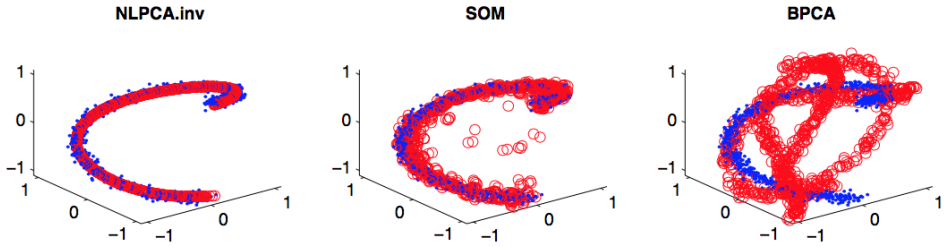


Fig. 2.7. Missing data estimation. Used is an artificial data set which describes a helical loop, plotted as dots ('·'). From each sample, one of the three values is rejected and have to be reconstructed by different missing data estimation algorithms. The reconstructed samples are plotted as circles ('○'). The inverse NLPCA identifies the nonlinear component best and hence gives a very good estimation of the missing values. SOM also gives a reasonably good estimation, but the linear approach BPCA fails on this nonlinear test set, see also Table 2.1

Figure 10: Missing Data comparison from [3]

References

[1] Christopher J.C. Burges. *Geometric Methods for Feature Extraction and Dimensional Reduction - A Guided Tour*. 2010.

[2] Fei Sha Jihun Ham Daniel D. Lee Lawrence K. Saul, Kilian Q. Weinberger. *Spectral Methods for Dimensionality Reduction*. 2006.

[3] Joachim Selbig Matthias Scholz, Martin Fraunholz. *Nonlinear Principal Component Analysis: Neural Network Models and Applications*.

[4] Partha Niyogi Mikhail Belkin. Laplacian eigenmaps for dimensionality reduction and data representation, 2002.

[5] Piyush Rai. Nonlinear dimensionality reduction, slides from machine learning course.

[6] Lawrence K. Saul Sam T. Roweis. Nonlinear dimensionality reduction by locally linear embedding, 2000.

[7] Vigário Scholz, M. *Nonlinear PCA: a new hierarchical approach*.

[8] Nobuyuki Matsui Teijiro Isokawa, Haruhiko Nishimura. Quaternionic multilayer perceptron with local analyticity, 2012.

[9] Werner Stuetzle Trevor Hastie. Principal curves, 1989.

[10] Lei Xi Uwe Kruger, Junping Zhang. *Developments and Applications of Nonlinear Principal Component Analysis – a Review*.