# Machine Learning

Winter 2011/12

Roland Memisevic

Lecture 6, Nov. 28, 2011

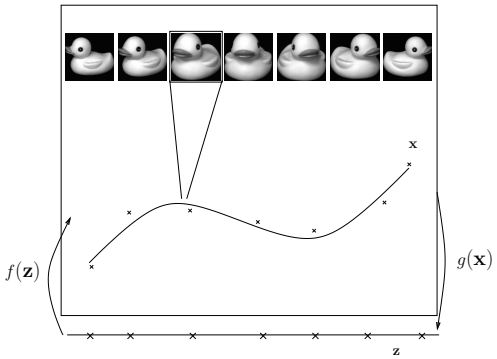


# Outline

- Continuous latent variables
- Principal Components Analysis (PCA)
- Probabilistic PCA
- Non-linear continuous latent variables



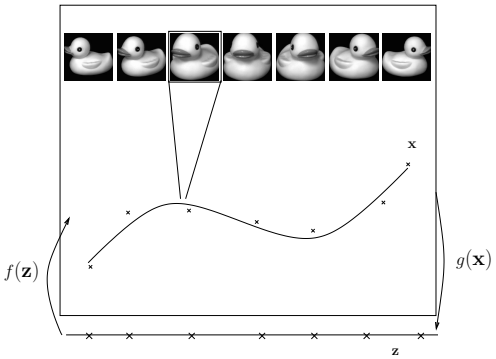
# Latent variables




- Reminder: We can use latent variables to model structure that is "hidden" in a given set of observations.
- In the previous lecture we looked at discrete latent variables. In this lecture we shall look at *continuous* latent variables. This is like *regression* with missing outputs (or inputs).

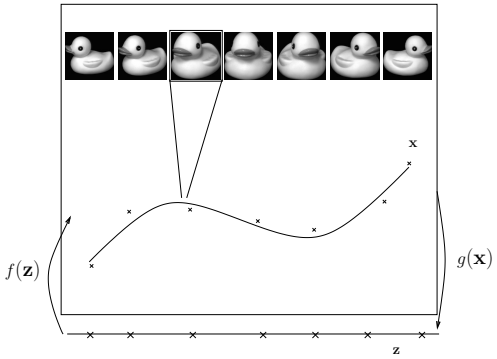


# Latent variables




- When the dimensionality $M$ of the continuous latent variable is smaller than the dimensionality $D$ of the data, then we can think of the data as being distributed along some lower-dimensional *manifold* in the dataspace.
- Learning the manifold is known as *dimensionality reduction*.

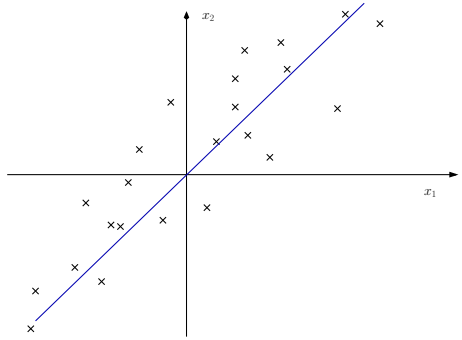
# Latent variables




- In many applications, we not only need the latent codes $\mathbf{z}$ for the data, but also:
  1. A **backward mapping** $g(\mathbf{x})$, that we can apply to new data after training.
  2. A **forward mapping** $f(\mathbf{z})$, with which we can "fantasize" new data.



# Principal Components Analysis (PCA)




- If we assume the manifold to be *linear*, learning gets simple and it can done in closed form.
- We can in this case think of the manifold as a lower-dimensional *subspace*.
- Learning amounts to finding the optimal subspace. Inference amounts to projecting data onto the subspace.

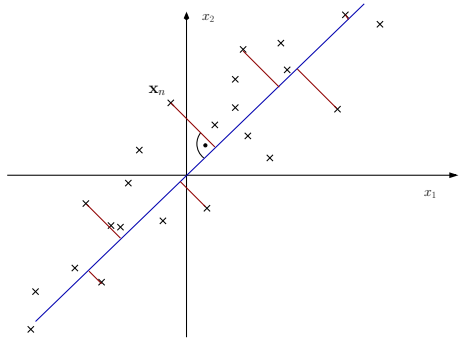


# Principal Components Analysis

- Learning the linear manifold is known as Principal Components Analysis (PCA).
- PCA can be derived by maximizing the *variance* of the set of points that we would get by projecting our training data, $\{\mathbf{x}_n\}_{n=1,\ldots,N}$, onto that subspace.
- Equivalently, one can maximize the *average distance* between the projections and the original points:

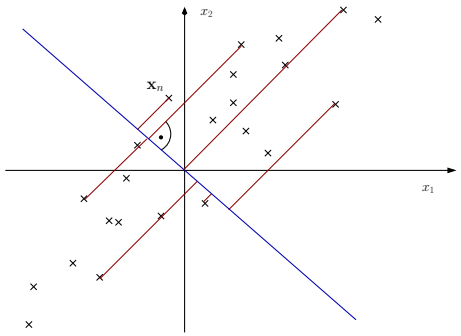


# Principal Components Analysis




- The variance along the manifold is **large**.
- The average projection error is **small**.

## Principal Components Analysis




- The variance along the manifold is **small**.
- The average projection error is **large**.



## Principal Components Analysis

- Since PCA involves optimizing a subspace, it is common to work under the assumption that the data is *mean-centered*:

$$\frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n = \mathbf{0}$$

- (Forgetting to mean-center your data is a common mistake when implementing PCA in practice!)
- To derive PCA, it is useful to define an *orthonormal basis* for the lower-dimensional subspace, consisting of vectors

$$\mathbf{u}_1, \ldots, \mathbf{u}_M \qquad (\text{with } M < D)$$

- PCA amounts to *learning* this basis.



## Principal Components Analysis

- It is convenient to stack the basis-vectors column-wise in a matrix $\mathbf{U}$, as this allows us to write the forward and backward mappings in a convenient way:

### Projecting data (backward mapping)

- The optimal coefficients that approximate $\mathbf{x}$ within the subspace are given by

$$\mathbf{z} = \mathbf{U}^{\mathrm{T}}\mathbf{x}$$

### Reconstructing data (forward mapping)

- The approximation $\tilde{\mathbf{x}}$ of $\mathbf{x}$ is given by

$$\tilde{\mathbf{x}} = \mathbf{U}\mathbf{z} = \mathbf{U}\mathbf{U}^{\mathrm{T}}\mathbf{x}$$



## Principal Components Analysis

- To learn the subspace, minimize the reconstruction error:

$$E(\mathbf{U}) = \sum_n \|\mathbf{x}_n - \tilde{\mathbf{x}}_n\|^2 = \sum_n \|\mathbf{x}_n - \mathbf{U}\mathbf{U}^{\mathrm{T}}\mathbf{x}_n\|^2$$

(under the constraint $\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}$ !)

- To solve the problem, it is convenient to stack data row-wise in matrix $\mathbf{X}$, and to rewrite the objective function as a *quadratic form* in $\mathbf{U}$:

$$\begin{aligned} E(\mathbf{U}) &= \|\mathbf{X}^{\mathrm{T}} - \mathbf{U}\mathbf{U}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\|_F^2 \\ &= \mathrm{Tr}((\mathbf{X}^{\mathrm{T}} - \mathbf{U}\mathbf{U}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}})^{\mathrm{T}}(\mathbf{X}^{\mathrm{T}} - \mathbf{U}\mathbf{U}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}})) \\ &= \mathrm{Tr}(\mathbf{X}\mathbf{X}^{\mathrm{T}}) - \mathrm{Tr}(\mathbf{U}^{\mathrm{T}}\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{U}) \\ &= -\mathrm{Tr}(\mathbf{U}^{\mathrm{T}}\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{U}) + \mathrm{const} \end{aligned}$$

- Optimizing a quadratic form under an orthonormality constraint is a common exercise in linear algebra:

## Principal Components Analysis

### Optimizing quadratic forms

- The maximizer of
$$\mathrm{Tr}(\mathbf{U}^{\mathrm{T}}\mathbf{A}\mathbf{U})$$
subject to
$$\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}$$
(where $\mathbf{U}$ is $D \times M$) is given by the matrix whose columns are the eigenvectors of $\mathbf{A}$ corresponding to the $M$ largest eigenvalues.

- So we can find the principal components by performing an **eigen-decomposition of the data covariance matrix**!



## Principal Components Analysis

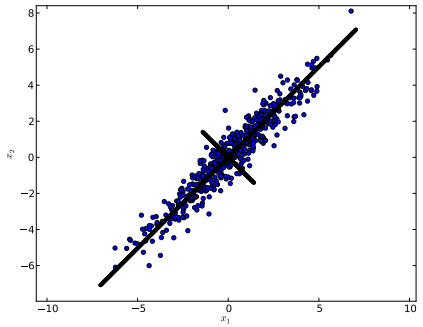
### Summary: Computing principal components

1. Mean-center the data.
2. Compute the covariance matrix $\mathbf{C} = \frac{1}{N}\mathbf{X}\mathbf{X}^{\mathrm{T}}$.
3. Perform an eigen-decomposition of $\mathbf{C}$.
4. Sort the eigen-vectors according to the size of their eigenvalues.
5. Stack the leading $M$ eigen-vectors in a matrix $\mathbf{U}$.

- $\mathbf{U}^{\mathrm{T}}$ now defines the forward mapping, $\mathbf{U}$ defines the backward mapping.



## Principal Components Analysis




- A two-dimensional dataset and the two principal components.
- Projections onto the leading eigenvectors preserve most of the variability in the data. So PCA performs *lossy compression*.



## PCA and Whitening

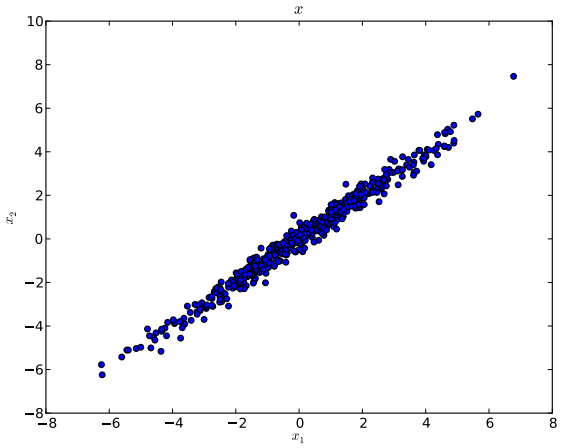
- The components of the latent data representation $\mathbf{Z}$ are uncorrelated (that is, $\mathbf{Z}$ has a diagonal covariance matrix):
$$\begin{aligned}\frac{1}{N}\sum_n \mathbf{z}_n\mathbf{z}_n^{\mathrm{T}} &= \frac{1}{N}\sum_n \mathbf{U}^{\mathrm{T}}\mathbf{x}_n\mathbf{x}_n^{\mathrm{T}}\mathbf{U} \\ &= \mathbf{U}^{\mathrm{T}}\big(\frac{1}{N}\sum_n \mathbf{x}_n\mathbf{x}_n^{\mathrm{T}}\big)\mathbf{U} \\ &= \mathbf{U}^{\mathrm{T}}\mathbf{C}\mathbf{U} \\ &= \mathbf{L}\end{aligned}$$
where the diagonal matrix $\mathbf{L}$ contains the eigenvalues of $\mathbf{C}$ on its diagonal.
- (The last step follows from the eigenvalue definition: $\mathbf{C}\mathbf{u}_i = \lambda_i\mathbf{u}_i$)

# PCA and Whitening

- We can obtain the *identity* as the covariance matrix for $\mathbf{Z}$, if instead of $\mathbf{U}^{\mathrm{T}}$ we use $\mathbf{L}^{-\frac{1}{2}}\mathbf{U}^{\mathrm{T}}$ to define the forward mapping.
- Data with identity covariance matrix is known as white; multiplying data by $\mathbf{L}^{-\frac{1}{2}}\mathbf{U}^{\mathrm{T}}$ as whitening.
- Whitening can be performed also without reducing the dimensionality, that is, by using $M = D$.
- This amounts to just rotating the coordinate system of the data, followed by independently "stretching" or "squeezing" the dimensions to obtain unit variance in each.
- Whitening can be thought of as a "fancier" version of the usual standardization by mean-centering and setting variances to $1$.

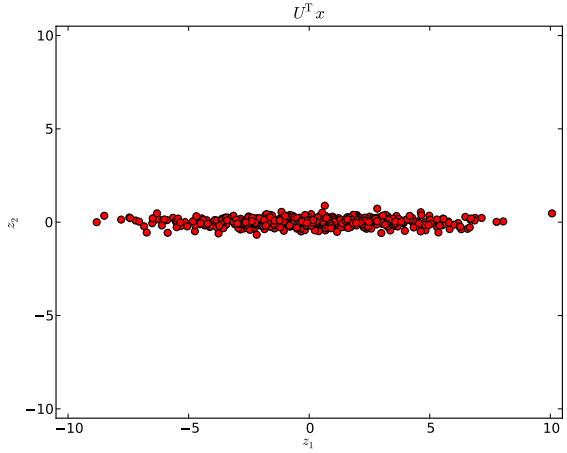


# Whitening example



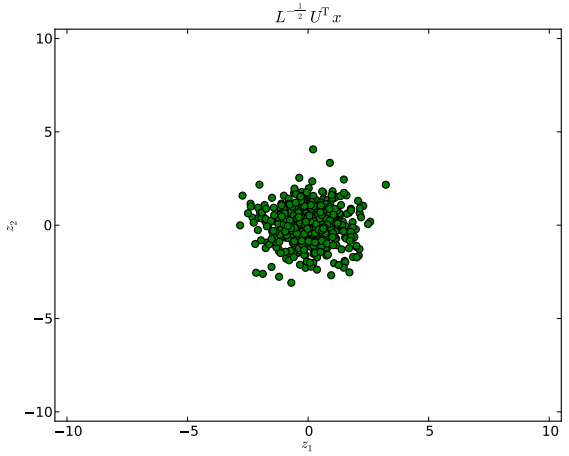



# Whitening example






# Whitening example

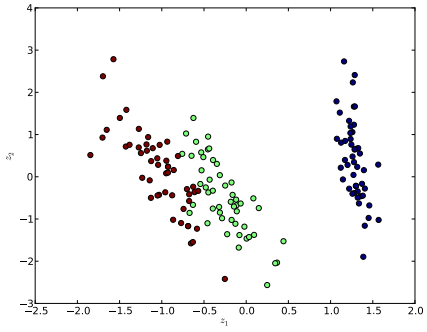


# PCA for data visualization

- Like other unsupervised learning methods, PCA has numerous applications, including compression, pre-processing, etc.
- An additional common application of dimensionality reduction is visualization of high-dimensional data (with $D > 3$):
- We project the high-dimensional data into two or three dimensions, where we can look at it using, for example, a scatter-plot.



# PCA for data visualization



- Two-dimensional PCA-projection of the classic, $4$-dimensional "iris"-dataset, whose features represent properties of Iris flowers, each belonging to one of three classes (represented by three colors in the plot).



# Probabilistic PCA

- One can define PCA also as a *probabilistic latent variable model*.
- Assume a Gaussian prior distribution

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$$

over latent variables.

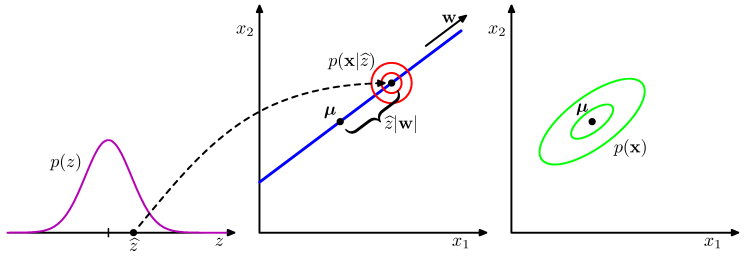
- Assume a Gaussian conditional distribution

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

over observations.



# Probabilistic PCA




- This defines a generative process, where we first draw from an $M$-dimensional Gaussian in the latent space, and then draw the observation from a $D$-dimensional Gaussian distribution, whose mean depends on the latent variable.

## Probabilistic PCA

- This process defines a probabilistic version of the forward mapping in terms of a conditional distribution!
- We can get the backward mapping using Bayes' rule:

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{\int_{\mathbf{z}} p(\mathbf{x}|\mathbf{z})p(\mathbf{z})\mathrm{d}\mathbf{z}}$$

- Plugging in the Gaussian distributions yields

$$p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\mathbf{M}^{-1}\mathbf{W}^{\mathrm{T}}(\mathbf{x}-\boldsymbol{\mu}), \sigma^{-2}\mathbf{M})$$

  with $\mathbf{M} = \mathbf{W}^{\mathrm{T}}\mathbf{W} + \sigma^2\mathbf{I}$
- (see, for example, Bishop, 2.3.3, for the conditional Gaussian derivations)



## Probabilistic PCA

- To optimize the parameters of the probabilistic model, maximize:

$$\begin{aligned} L &= \sum_n \log p(\mathbf{x}_n) \\ &= \sum_n \log \int_{\mathbf{z}_n} p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{z}_n)\ \mathrm{d}\mathbf{z}_n \end{aligned}$$

- The marginals turn out to be Gaussian, too:

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

- Thus, probabilistic PCA models data using a *constrained Gaussian* distribution, whose covariance matrix is the outer product of two low-rank matrices (plus noise).



## Probabilistic PCA

- To optimize the log-likelihood, one can use gradient-based optimization or the EM-algorithm.
- The EM-algorithm proceeds as usual: In the E-step, compute $p(\mathbf{z}_n|\mathbf{x}_n)$; in the M-step, maximize the expected complete log-likelihood.
- Unlike for Gaussian mixtures, here the posteriors $p(\mathbf{z}_n|\mathbf{x}_n)$ are Gaussians, so we represent them using mean and covariance.
- (details of the EM updates given in Bishop, page 578)
- Benefits of the probabilistic formulation are that (i) it can easily handle missing data values, (ii) it can be extended into a "mixture of PCA"-model which performs clustering and dimensionality reduction (within each cluster) at the same time, (iii) there is a fully Bayesian formulation, where parameters are integrated out.



## Factor Analysis

- Probabilistic PCA is closely related to a statistical model known as *Factor Analysis*:
- Factor analysis differs from probabilistic PCA in that it assumes a conditional Gaussian distribution

$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi})$$

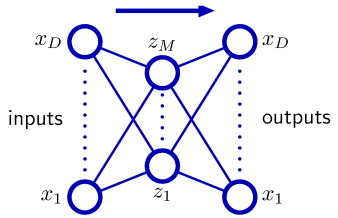
  with a diagonal (not necessarily spherical) covariance matrix $\boldsymbol{\Psi}$.
- Factor analysis uses the latent variables only to encode covariances.
- Learning is similar to probabilistic PCA.

## Non-linear continuous latent variables

- PCA is a highly common tool used in practical applications.
- But sometimes data is not distributed along a linear subspace, so the linearity assumption may be a limitation.
- (Sometimes, even the assumption of a manifold is not correct, because data may be structured in more complicated ways.)
- Neural networks provide an elegant and useful way to get beyond the linearity assumption.



## Auto-encoder networks




- We introduced backprop networks as a supervised learning method.
- But we can turn a backprop network into an unsupervised method, if we train it to *reconstruct its own inputs*!
- For this end, define the targets $\mathbf{t}_n$ to be *copies of the inputs* $\mathbf{x}_n$, and minimize:

$$E(\mathbf{W}) = \frac{1}{2}\|\mathbf{y}(\mathbf{x}_n, \mathbf{W}) - \mathbf{x}_n\|^2$$

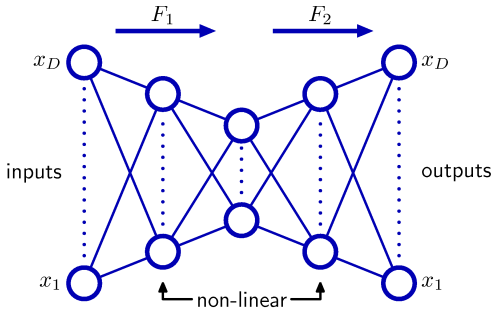


## Auto-encoder networks

- If we use a purely linear network with a small number, $M$, of hidden units, then the network will be forced to find good low-dimensional projections of the data.
- In fact, it can be shown that the first-layer weights, $\mathbf{w}$, will define a projection into the same subspace as PCA (but the components will in general not be orthonormal) (Baldi and Hornik, 1989).
- But we can use a backprop network with more hidden layers, some of which may have non-linear transfer functions.
- It is common to use an inner-most hidden layer that is linear and one non-linear layer before and after the linear layer:

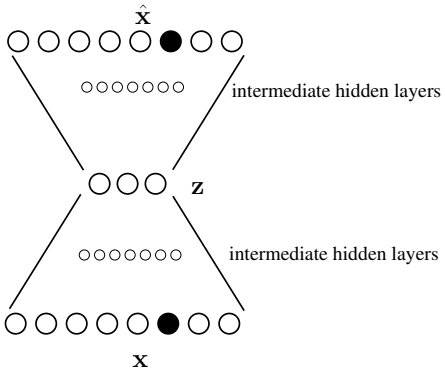


## Auto-encoder networks




- This model reconstructs its inputs by first projecting into a non-linear feature space, performing a non-orthogonal variant of linear PCA in that feature space, and then inverting the non-linear mapping.

## Auto-encoder networks




- A classic toy-application and proof-of-concept is learning of a low-dimensional code to represent a one-of-$K$ representation.



## Graph-based dimensionality reduction

- A variety of dimensionality reduction methods have been introduced in recent years, that can learn non-linear manifolds by solving a closed-form optimization problem.
- Examples include "Locally Linear Embedding", "ISOMAP", "kernel PCA", "Laplacian Eigenmaps", and there are many others.
- The idea behind these methods is to build a graph that represents neighborhood-relations between pairs of points, and then to find latent representatives $\mathbf{z}_n$ which have approximately the same neighborhood-relations.
- Potential drawbacks of most of theses methods is that they do not easily scale to large data-sets, and they do not come with the forward and backward-mappings, which are required in many applications.



## Over-complete latent variables and sparse coding

- In general, in continuous latent variable models, there are alternative ways to *constrain the capacity* of the hidden variables $\mathbf{z}$ than to make their dimensionality small.
- The most common alternative is to force the hidden representation to be *sparse*, such that most components $z_{nk}$ are approximately zero for most observations.
- This can be easily achieved with some small modifications to auto-encoder networks, but there are also many other methods that explicitly try to make latent variables sparse.
- A common, fully probabilistic sparse coding model is *Independent Components Analysis* (ICA):



## Independent Components Analysis

- ICA is based on latent variables which, unlike in PCA, are assumed to be non-Gaussian, but like in PCA they are assumed to be independent, such that

$$p(\mathbf{z}) = \prod_{j=1}^{M} p(z_j)$$

- It is common to use *heavy-tailed* distributions to define the component distributions $p(z_j)$.
- A common application is *blind source separation* (aka the "cocktail-party-problem"): Given a mixture of independent signals (imagine a set of microphones listening to a set of people who are speaking simultaneously), find the (de-)mixing matrix and the original signals.