Development of Complex Curricula for Molecular Bionics and Infobionics Programs within a consortial* framework**

Consortium leader

PETER PAZMANY CATHOLIC UNIVERSITY

Consortium members

SEMMELWEIS UNIVERSITY, DIALOG CAMPUS PUBLISHER

The Project has been realised with the support of the European Union and has been co-financed by the European Social Fund ***

**Molekuláris bionika és Infobionika Szakok tananyagának komplex fejlesztése konzorciumi keretben

***A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

Digital- and Neural Based Signal Processing & Kiloprocessor Arrays

Digitális- neurális-, és kiloprocesszoros architektúrákon alapuló jelfeldolgozás

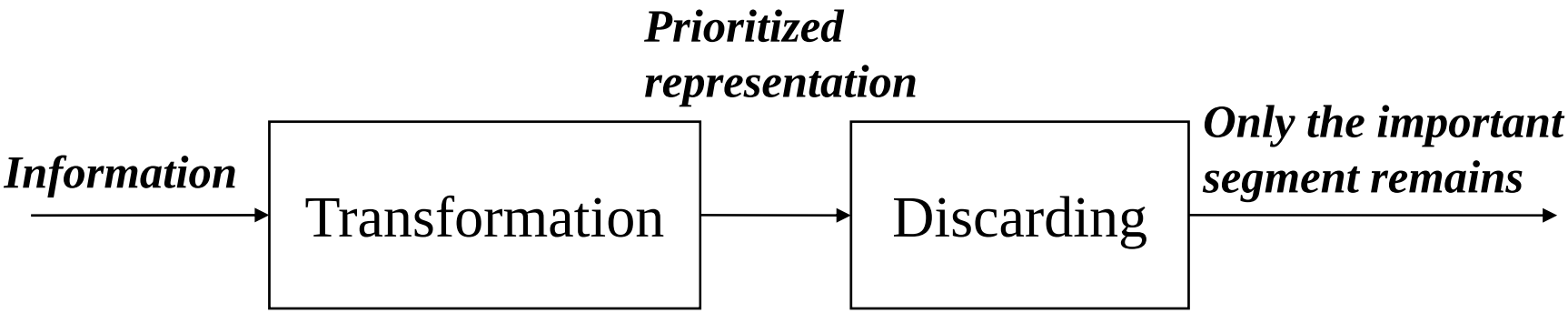
Principal Component Analysis

Főkomponens analízis

András Oláh, Gergely Treplán, Dávid Tisza

Contents

- Data compression
- Principal Component analysis (PCA or KLT)
  - Notation
  - Summary of the algorithm
  - Problems, extensions
- Hebbian PCA (The Oja algorithm)
- Experiments
- Problems



Examples of well known transformations:

- DCT - Discrete Cosine Transformation (JPEG,MPEG)
- KLT - Karhunen-Loève Transformation

Notation

- Scalars are noted with normal letters
- Vectors are noted with bold face lower case.
- Matrices are noted with bold face upper case.
  - Random data vector (e.g. video frame): $\mathbf{x}$
  - Correlation matrix: $\mathbf{R} = E\left(\mathbf{x}\mathbf{x}^T\right); R_{ij} = E\left(x_i x_j\right)$
- Vectors are defined to be column vectors:
  $\dim(\mathbf{x}) = N \times 1$

- i[th] element of a vector (scalar): x_i
- (i,j)[th] element of a matrix: $R_{i,j}$
- i[th] row of a matix: $\mathbf{R}_{i,:}$
- j[th] column of a matix: $\mathbf{R}_{:,j}$
- Submatrix (eg. first K rows and all columns): $\mathbf{R}_{1:K,:}$
- i[th] element of a set (eg. a set of eigenvectors)

$$\mathbf{s}^{(i)} \in S_{\mathbf{R}} = \{\mathbf{x} : \mathbf{x} \text{ is an Eigen vector of } \mathbf{R}\}$$

- Expectation operator, if X is a discrete random variable: $E(X) := \sum_i x_i p(x_i)$

- Dot product: $< \mathbf{a}, \mathbf{b} >= \mathbf{a}^T \cdot \mathbf{b} = \sum_{i=1}^{N} a_i \cdot b_i$
- Correlation matix:

$$\mathbf{R} = E(\mathbf{x} \cdot \mathbf{x}^T), \quad \dim(\mathbf{R}) = N \times N$$

- Diagonal extension operator:

$$\mathbf{a} = [a_1, a_2, \ldots, a_N]^T$$

$$\mathbf{A} = diag(\mathbf{a}) := \begin{bmatrix} a_1 & 0 & \cdots & 0 \\ 0 & a_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & a_N \end{bmatrix}$$

- eigenvectors and eigenvalues of $\mathbf{R}$

$$\mathbf{R}\cdot\mathbf{s}^{(i)} = \lambda_i\,\mathbf{s}^{(i)}, \quad i = 1,\ldots,N$$

- Set of Eigen equations: $\mathbf{R}\cdot\mathbf{S} = \mathbf{S}\cdot\mathbf{\Lambda}$

$$\mathbf{S} = \left[\mathbf{s}^{(1)}, \mathbf{s}^{(2)}, \ldots, \mathbf{s}^{(N)}\right]$$

$$\mathbf{\Lambda} = diag\left(\left[\lambda_1, \lambda_2, \ldots, \lambda_N\right]^T\right)$$

- Properties of the eigenvectors:

$$<\mathbf{s}^{(i)}, \mathbf{s}^{(j)}> = \begin{cases} 1 \,|\, i = j \\ 0 \,|\, i \neq j \end{cases}$$ orthonormal bases, so $\mathbf{S}^T\cdot\mathbf{S} = \mathbf{I}$

Basis transformations back and forth to the eigenvectors:

- Coder, transforming to the eigenvectors basis:

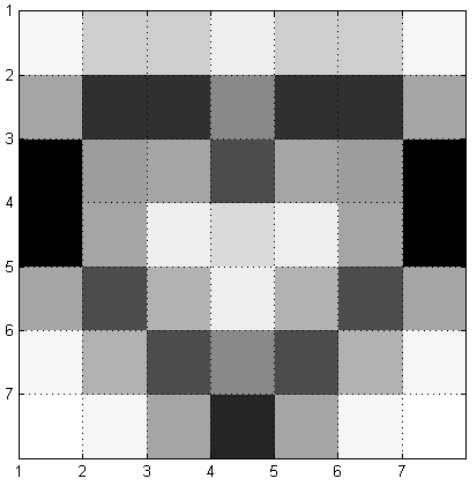
$$\mathbf{y}=\mathbf{S}^T\cdot\mathbf{x},\quad y_i=(\mathbf{S}_{:,i})^T\cdot\mathbf{x}=(\mathbf{s}^{(i)})^T\cdot\mathbf{x}=\sum_{n=1}^{N}s_n^{(i)}x_n$$

- Decoder, transforming from the eigenvector basis: $\mathbf{x}=\mathbf{S}\cdot\mathbf{y}$

$$x_i=(\mathbf{S}_{i,:})^T\cdot\mathbf{y}=\left(\left[s_i^{(1)},s_i^{(2)},\ldots,s_i^{(N)}\right]\right)\mathbf{y}=\sum_{n=1}^{N}s_i^{(n)}y_n$$

PCA

- Suppose we have an element of a data set, which is a grayscale image map now. data frame at time instant k:



- Generate the data vector from it, by reading it column wise: **x**[k]

- The data usually can be modeled as a random vector variable **x**, because we are dealing with processes assuming quasi stationarity and ergodicity. E.g. consecutive image frames from a video flow are usually „similar”.
- Thus in a time window we examine the corr. matrix of the process:

$$\mathbf{R} = E\left\{\mathbf{x}\mathbf{x}^T\right\} = \begin{bmatrix} E\{x_1x_1\} & E\{x_1x_2\} & . & E\{x_1x_N\} \\ E\{x_2x_1\} & E\{x_2x_2\} & . & E\{x_2x_N\} \\ . & . & . & . \\ E\{x_Nx_1\} & E\{x_Nx_2\} & . & E\{x_Nx_N\} \end{bmatrix}$$

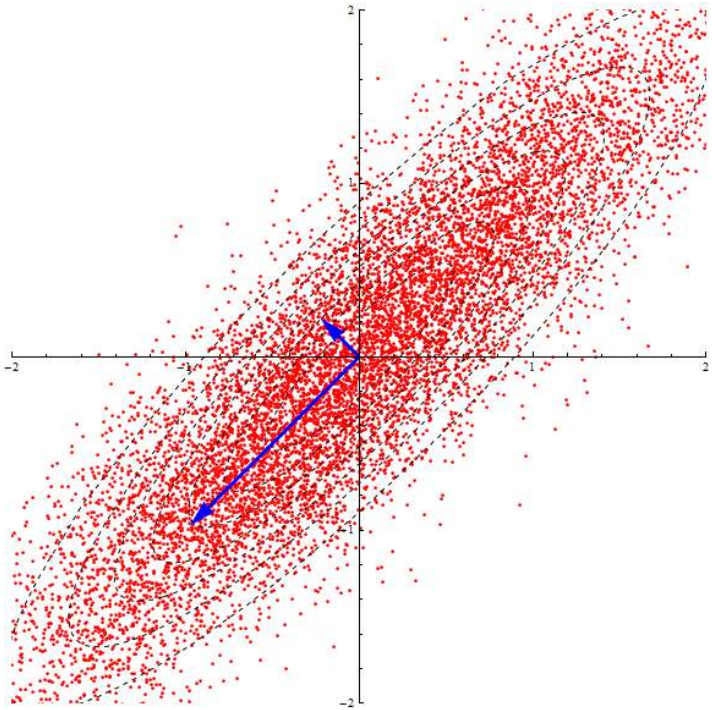
$$\dim(\mathbf{R}) = N \times N$$

- The data vector random is assumed to have 0 mean. $E(\mathbf{x}) = \mathbf{0}$
- Compute the eigenvectors and eigenvalues of the correlation matrix **R** and sort it in an decreasing order (by importance):

$$\mathbf{R}\cdot\mathbf{s}^{(i)} = \lambda_i \, \mathbf{s}^{(i)}, \quad i = 1, \ldots, N$$

$$\begin{array}{ccccccccccccc} \lambda_1 & \geq & \lambda_2 & \geq & \lambda_3 & \geq \ldots \geq & \lambda_M & \geq & \lambda_{M+1} & \geq \ldots \geq & \lambda_N \\ \updownarrow & & \updownarrow & & \updownarrow & & \updownarrow & & \updownarrow & & \updownarrow \\ \mathbf{s}^{(1)} & & \mathbf{s}^{(2)} & & \mathbf{s}^{(3)} & & \mathbf{s}^{(M)} & & \mathbf{s}^{(M+1)} & & \mathbf{s}^{(N)} \end{array}$$

- Eigenvectors point to orthogonal directions in each dimension which tries to capture the independent variance directions of the distribution
- Eigenvalues are proportional to the magnitude of variances



Eigenvalue scaled eigenvectors:
blue arrows

Iso curves of the distribution:
black dashed contours

Sample points of the distribution:
red dots

$$X \sim N\left(\begin{bmatrix}0\\0\end{bmatrix},\begin{bmatrix}1 & 0.9\\0.9 & 1\end{bmatrix}\right)$$

$$\mathbf{S}=\begin{bmatrix}-0.7071 & -0.7071\\-0.7071 & 0.7071\end{bmatrix},\mathbf{\Lambda}=\begin{bmatrix}1.9 & 0\\0 & 0.1\end{bmatrix}$$

- Eigen vectors correspond to the directions of the standard deviations of the data understood as a sample set of a multidimensional random process.
- Eigen values correspond to the magnitude of the standard deviations.
- If we use a hypothesis that this linear quantity (correlation) captures real world “importance” then we can **sort** the dimensions along “importance” and
- **compress** by discarding “not really important” dimensions.

- We can transform the data frame to the new basis system in a lossless fashion:

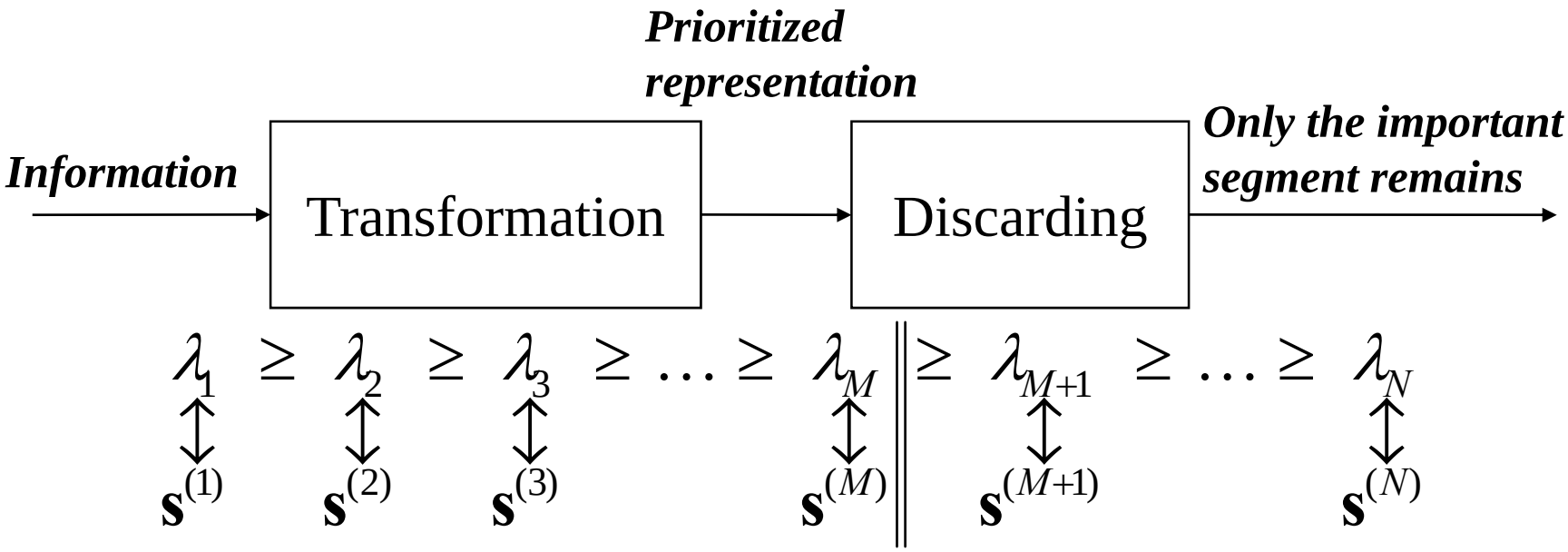
$$\text{Given } \mathbf{R},\ \text{get all } \lambda_i, \mathbf{s}^{(i)}, i = 1, \ldots, N$$

$$\left.\begin{array}{lll} \lambda_1 & \mathbf{s}^{(1)} & \Rightarrow \quad y_1 = \mathbf{x}^T \mathbf{s}^{(1)} \\ \lambda_2 & \mathbf{s}^{(2)} & \Rightarrow \quad y_2 = \mathbf{x}^T \mathbf{s}^{(2)} \\ \vdots & \vdots & \vdots \\ \lambda_i & \mathbf{s}^{(i)} & \Rightarrow \quad y_i = \mathbf{x}^T \mathbf{s}^{(i)} \\ \vdots & \vdots & \vdots \\ \lambda_N & \mathbf{s}^{(N)} & \Rightarrow \quad y_N = \mathbf{x}^T \mathbf{s}^{(N)} \end{array}\right\} \Rightarrow \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_N \end{bmatrix}$$

$$\text{Transformed data: } \mathbf{y} = \left(\mathbf{x}^T \cdot \mathbf{S}\right)^T = \mathbf{S}^T \cdot \mathbf{x}$$

$$\text{original data: } \mathbf{x} = \mathbf{S} \cdot \mathbf{y}$$

- We want to discard the „unimportant” components to achieve compression:



- Instead of using all eigenvectors, just use the first M to get an approximate of **y**

- The approximate of the transformed data:

$$\left.\begin{array}{lll} \lambda_1 & \mathbf{s}^{(1)} & \Rightarrow \quad y_1 = \mathbf{x}^T \mathbf{s}^{(1)} \\ \lambda_2 & \mathbf{s}^{(2)} & \Rightarrow \quad y_2 = \mathbf{x}^T \mathbf{s}^{(2)} \\ \vdots & \vdots & \vdots \\ \lambda_M & \mathbf{s}^{(M)} & \Rightarrow \quad y_M = \mathbf{x}^T \mathbf{s}^{(M)} \end{array}\right\} \Rightarrow \hat{\mathbf{y}} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix},$$

$\hat{\mathbf{S}} = \mathbf{S}_{:,1:M}$ the first M eigenvectors, arranged in a matrix

compressed, transformed data: $\hat{\mathbf{y}} = \left(\mathbf{x}^T \cdot \hat{\mathbf{S}}\right)^T = \hat{\mathbf{S}}^T \cdot \mathbf{x}$

decompressed, lossy data recovery: $\hat{\mathbf{x}} = \hat{\mathbf{S}}^T \cdot \hat{\mathbf{y}}$

PCA compression

$\dim(\mathbf{x}) = N$ $\dim(\hat{\mathbf{y}}) = M$ $\dim(\hat{\mathbf{x}}) = N$

$N >> M$

Compression algorithm: $\hat{\mathbf{y}} = \hat{\mathbf{S}}^T \cdot \mathbf{x}$

Decompression algorithm: $\hat{\mathbf{x}} = \hat{\mathbf{S}} \cdot \hat{\mathbf{y}}$

The PCA is the optimal lossy representation in the terms of Mean Square Error:

$\mathbf{E}\left\|\mathbf{x}-\hat{\mathbf{x}}\right\|_2^2 < \varepsilon$: QoS criterion

$$\mathbf{E}\left\|\mathbf{x}-\hat{\mathbf{x}}\right\|_2^2 = \mathbf{E}\left\|\mathbf{x}-\hat{\mathbf{S}}\hat{\mathbf{y}}\right\|_2^2 = \mathbf{E}\left[(\mathbf{x}-\hat{\mathbf{S}}\hat{\mathbf{y}})^T(\mathbf{x}-\hat{\mathbf{S}}\hat{\mathbf{y}})\right] = \mathbf{E}\left[(\mathbf{x}^T-\hat{\mathbf{y}}^T\hat{\mathbf{S}}^T)(\mathbf{x}-\hat{\mathbf{S}}\hat{\mathbf{y}})\right] =$$

$$= \mathbf{E}\left[\mathbf{x}^T\mathbf{x}-\hat{\mathbf{y}}^T\underbrace{\hat{\mathbf{S}}^T\mathbf{x}}_{\hat{\mathbf{y}}}-\underbrace{\mathbf{x}^T\hat{\mathbf{S}}}_{\hat{\mathbf{y}}^T}\hat{\mathbf{y}}+\hat{\mathbf{y}}^T\underbrace{\hat{\mathbf{S}}^T\hat{\mathbf{S}}}_{\mathbf{I}}\hat{\mathbf{y}}\right] = \mathbf{E}\left[\mathbf{x}^T\mathbf{x}-2\hat{\mathbf{y}}^T\hat{\mathbf{y}}+\hat{\mathbf{y}}^T\hat{\mathbf{y}}\right] =$$

$$= \mathbf{E}\left[\mathbf{x}^T\mathbf{x}\right]-\mathbf{E}\left[\hat{\mathbf{y}}^T\hat{\mathbf{y}}\right] = \sum_{i=1}^{N}\lambda_i - \sum_{i=1}^{M}\lambda_i = \sum_{i=M+1}^{N}\lambda_i$$

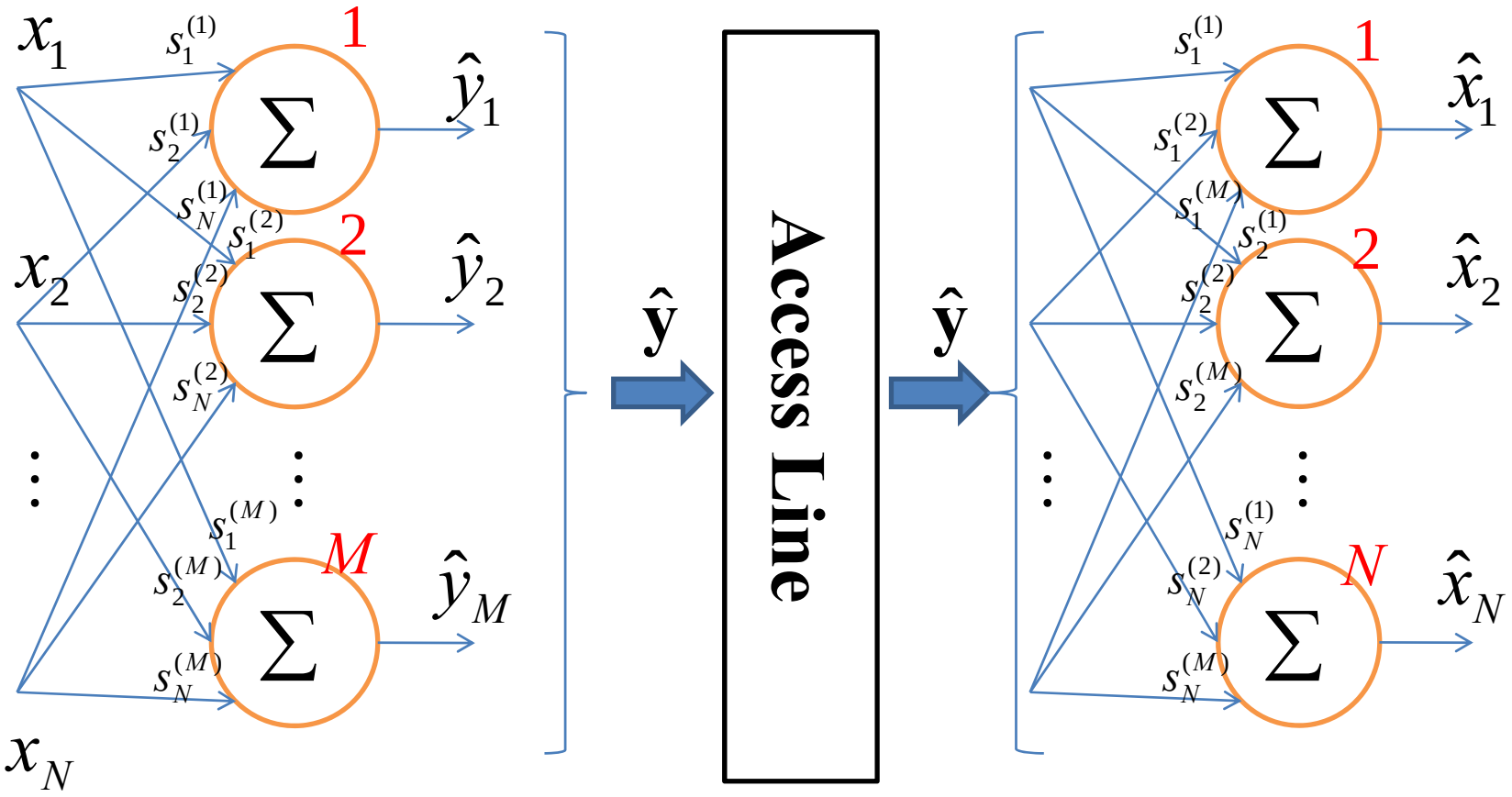
→ Minimum because eigenvalues are ordered in a monotone decreasing sequence

- The components we discard in the course of compression contribute as much as their corresponding eigenvalues.

$$\mathbf{E}\left\|\mathbf{x}-\hat{\mathbf{x}}\right\|_2^2=\sum_{i=1}^{N}\lambda_i-\sum_{i=1}^{M}\lambda_i=\sum_{i=M+1}^{N}\lambda_i$$

- The mean square error is the sum of the eigenvalues corresponding to the discarded components. We drop the minimum contribution due to the monoton decreasing order of eigenvalues.

PCA architecture



PCA compression overview

We assume that **R** is given for the process we are dealing with and **R** remains the same over time. (stationarity)

1. Solve the Eigen problem for **R** to get the first M largest eigenvalues and the corresponding eigenvectors based on the quality requirement.
2. Assemble the coding/decoding matrix.
3. Use the coder to encode the data (matrix-vector multiplication), and send through the access line.
4. Decode the data with the decoding matrix.

Difficulties of direct implementing the PCA for video compression

- Nobody tells us what is **R** of an underlying video flow.
- **R** does not remain „still” for the whole time
- Computing **eigenvalues** and **eigenvectors** are computationally complex
- Practically the normalization of the variances in the original data set along each dimension before the PCA transformation is useful to get meaningful relative importance values after the transformation.

Difficulties of direct implementing the PCA for video compression

- Storage problems:
  - N: dim(**x**)=640x480=307200 per color channel
  - NxN: dim(**R**)=dim(E(**x** $\mathbf{x}^T$))=307200x307200
  - If a float (32 bit) used to hold an element of **R**,
  - sizeof(**R**)~350GB per color channel in memory to hold **R** or an estimate of **R**

Estimation of the correlation matrix

- Using the assumed ergodicity of the process and estimate in a time window of length K

$$\lim_{K\to\infty}\frac{1}{K}\sum_{k=1}^{K}\mathbf{x}[k]\mathbf{x}[k]^T=\mathbf{R}\Rightarrow\frac{1}{K}\sum_{k=1}^{K}\mathbf{x}[k]\mathbf{x}[k]^T=\hat{\mathbf{R}}$$

- From time to time we have to reestimate the empirical correlation matrix.
- Note that we still assume quasi stationarity for a K length time window

Overcoming the storage problem - Introducing the kernel based PCA

- We are using the fact that $\hat{\mathbf{R}}$ was constructed using K outer products, so $rank(\hat{\mathbf{R}}) = K$
- Using the basis transform:

$$\mathbf{s}^{(i)} = \mathbf{X} \cdot \boldsymbol{\xi}^{(i)} = \left[\mathbf{x}[1], \cdots, \mathbf{x}[K]\right] \cdot \boldsymbol{\xi}^{(i)}$$

- We can rewrite:

$$\hat{\mathbf{R}}\mathbf{s}^{(i)} = \lambda_i \mathbf{s}^{(i)}, \dim(\hat{\mathbf{R}}) = N \times N$$

$$\boxed{\mathbf{G} \cdot \boldsymbol{\xi}^{(i)} = K\lambda_i \boldsymbol{\xi}^{(i)}}, \dim(\mathbf{G}) = K \times K << N \times N$$

Kernel based PCA

$$\mathbf{s}^{(i)} = \mathbf{X} \cdot \underline{\xi}^{(i)} = \left[\mathbf{x}[1] \cdots \mathbf{x}[K]\right] \cdot \underline{\xi}^{(i)}, \quad \hat{\mathbf{R}} = \frac{1}{K} \sum_{k=1}^{K} \mathbf{x}[k]\mathbf{x}[k]^T$$

$$\hat{\mathbf{R}}\mathbf{s}^{(i)} = \lambda_i \mathbf{s}^{(i)}$$

$$\sum_{k=1}^{K} \mathbf{x}[k]\mathbf{x}[k]^T \sum_{l=1}^{K} \xi_l^{(i)} \mathbf{x}[l] = \lambda_i \sum_{m=1}^{K} \xi_m^{(i)} \mathbf{x}[m] \quad \Big| \rightarrow K \cdot \mathbf{x}[n]^T \cdot$$

$$\sum_{l=1}^{K} \xi_l^{(i)} \sum_{k=1}^{K} \mathbf{x}[n]^T \mathbf{x}[k]\mathbf{x}[k]^T \mathbf{x}[l] = K\lambda_i \sum_{m=1}^{K} \xi_m^{(i)} \mathbf{x}[n]^T \mathbf{x}[m]$$

$$\sum_{l=1}^{K} \xi_l^{(i)} \sum_{k=1}^{K} \mathbf{G}_{n,k} \mathbf{G}_{k,l} = K\lambda_i \sum_{m=1}^{K} \xi_m^{(i)} \mathbf{G}_{n,m}$$

$$\frac{1}{K} \mathbf{G}\mathbf{G} \cdot \xi^{(i)} = K\lambda_i \mathbf{G}\xi^{(i)}$$

$$\mathbf{G} \cdot \xi^{(i)} = K\lambda_i \xi^{(i)}$$

Kernel based PCA

- Storage problems:
  - N: dim(**x**)=640x480=307200 per color channel
  - *Sizeof*(**R**)~350GB per color channel in memory to hold **R**
  - If window length of K=1000 frames used

$$\mathbf{X} = [\mathbf{x}[1], \cdots, \mathbf{x}[K]]$$

$$\hat{\mathbf{R}} = \frac{1}{K}\mathbf{X}\cdot\mathbf{X}^T, \quad \dim(\hat{\mathbf{R}}) = N \times N$$

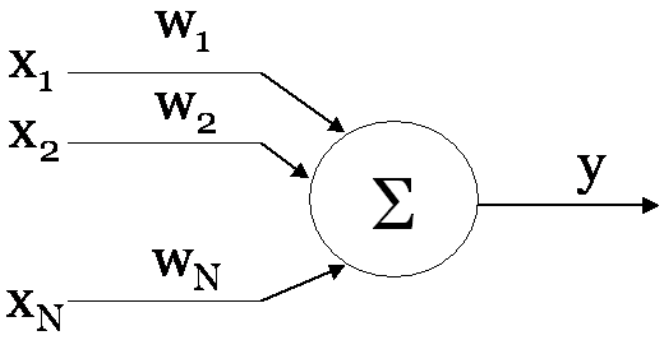
$$\mathbf{G} = \mathbf{X}^T\cdot\mathbf{X}, \quad \dim(\mathbf{G}) = K \times K$$

  - *Sizeof*(**G**)=4MB per color channel

The Oja algorithm

- The Oja algorithm can be another solution to the storage problem and an alternative for the existing effective algorithms for the computation of the EigenValue Decomposition(EVD), which is to be performed from time window to time window
- If we assume that we know $\mathbf{s}^{(i)}[k]$ can we recursively compute $\mathbf{s}^{(i)}[k+1]$ assuming it is changing slowly enough?

The Oja algorithm



- The Oja algorithm is a nonlinear stochastic difference equation:

$$\mathbf{w}[k+1] = \mathbf{w}[k] + \eta\, y[k]\left[\mathbf{x}[k] - \mathbf{w}[k]\, y[k]\right]$$

$$y[k] = \varphi\left(\mathbf{w}[k]^T \cdot \mathbf{x}[k]\right), \quad \varphi(u) = u$$

$\lim_{k \to \infty} \mathbf{w}[k] = \mathbf{s}^{(1)}$, proof with Kushner Clark theorem

The Oja algorithm

- Lemma: $\max_{\mathbf{w}\in\mathbb{R}^n} \mathbf{w}^T\mathbf{R}\mathbf{w} = \lambda_1, \quad \mathbf{w} := \mathbf{s}^{(1)}$

  $s.t.\ \|\mathbf{w}\| = 1$

- Proof outline:

$$\mathbf{v} = \mathbf{S}^T\cdot\mathbf{w}, \quad \mathbf{w} = \mathbf{S}\cdot\mathbf{v}, \quad \mathbf{S}^T\mathbf{S} = \mathbf{I} \Rightarrow \mathbf{S}^T = \mathbf{S}^{-1}, \quad \mathbf{RS} = \mathbf{S}\Lambda, \quad \mathbf{S}^{-1}\mathbf{RS} = \Lambda$$

$$\mathbf{w}^T\mathbf{R}\mathbf{w} = \mathbf{v}^T\mathbf{S}^T\mathbf{R}\mathbf{S}\mathbf{v} = \mathbf{v}^T\mathbf{S}^{-1}\mathbf{R}\mathbf{S}\mathbf{v} = \mathbf{v}^T\Lambda\mathbf{v} = \sum_{i=1}^{N} v_i^2\lambda_i$$

if we choose $\mathbf{w} = \mathbf{s}^{(1)}, \mathbf{v} = \mathbf{S}^T\cdot\mathbf{s}^{(1)} = \begin{bmatrix}1 & 0 & \dots & 0\end{bmatrix}^T$

$$\mathbf{w}^T\mathbf{R}\mathbf{w}\big|_{\mathbf{w}=\mathbf{s}^{(1)}} = \sum_{i=1}^{N} v_i^2\lambda_i = \lambda_1$$

The Oja algorithm

- Using the properties of the Rayleigh quotient, that

$$\text{if } \mathbf{R}=\mathbf{R}^T, \mathbf{R}\in\mathbb{R}^{N\times N}, \max_{\mathbf{w}\in\mathbb{R}^n}\frac{\mathbf{w}^T\mathbf{R}\mathbf{w}}{\mathbf{w}^T\mathbf{w}}=\lambda_{\max}, \quad \text{if } \mathbf{w}=\mathbf{s}^{(\max)}$$

$$s.t.\ \|\mathbf{w}\|=1$$

- Starting from the Rayleigh quotient and using the Newton's method to find the maximum:

$$\mathbf{w}[k+1]=\mathbf{w}[k]+\eta\, grad_{\mathbf{w}}\left(\frac{\mathbf{w}^T\mathbf{R}\mathbf{w}}{\mathbf{w}^T\mathbf{w}}\right)=$$

$$=\mathbf{w}[k]+\eta\left[\mathbf{R}\mathbf{w}[k]-\left(\mathbf{w}[k]^T\mathbf{R}\mathbf{w}[k]\right)\mathbf{w}[k]\right]$$

The Oja algorithm

- We still need the knowledge of **R**, so we substitute: $y = \mathbf{x}^T\mathbf{w} = \mathbf{w}^T\mathbf{x}$

  (Use the simplification discussed at the LMS and at the Robins-Monroe algorithm)

$$\mathbf{w}[k+1] = \mathbf{w}[k] + \eta\left[E(\mathbf{x}\mathbf{x}^T)\mathbf{w}[k] - \left(\mathbf{w}[k]^T E(\mathbf{x}\mathbf{x}^T)\mathbf{w}[k]\right)\mathbf{w}[k]\right] =$$

$$= \mathbf{w}[k] + \eta\left[E(\mathbf{x}\underbrace{\mathbf{x}^T\mathbf{w}[k]}_{y[k]}) - E(\underbrace{\mathbf{w}[k]^T\mathbf{x}}_{y[k]}\underbrace{\mathbf{x}^T\mathbf{w}[k]}_{y[k]}\mathbf{w}[k])\right] \cong$$

$$\cong \mathbf{w}[k] + \eta y[k]\left[\mathbf{x}[k] - \mathbf{w}[k]y[k]\right]$$

- Stability can be proven by the Kushner-Clark thm

The Oja algorithm

- Extending the algorithm with the Generalized Hebbian Algorithm one can design a feed-forward neural network to catch all the eigenvectors of the correlation matrix.
- Generalized Hebbian Rule:

$$\Delta w_{ji}[k+1] = \eta \left\{ y_j[k] x_i[k] - y_j[k] \sum_{m=1}^{j} w_{mi}[k] y_m[k] \right\}$$

The Oja algorithm with the Generalized Hebbian Algorithm (GHA)

- No knowledge of **R** is needed for the iteration
- Can be implemented in a feed-forward neural network structure
- For the j[th] neuron's i[th] weight:

$$w_{ji}[k+1] = w_{ji}[k] + \eta\, y_j[k]\left(x'_{ji}[k] - w_{ji}[k]\, y_j[k]\right)$$

$$x'_{ji}[k] = x_i[k] - \sum_{m}^{j-1} w_{mi}[k]\, y_m[k]$$

note that x'_{ji} is a function of j as well

Hebbian PCA

- Hebbian learning:

$$\tilde{\mathbf{w}}[k+1] = \mathbf{w}[k] + \eta y[k]\mathbf{x}[k]$$

- Normalization:

$$\mathbf{w}[k+1] = \frac{\tilde{\mathbf{w}}[k+1]}{\left\|\tilde{\mathbf{w}}[k+1]\right\|} = \tilde{\mathbf{w}}[k+1]\left\|\tilde{\mathbf{w}}[k+1]\right\|^{-1}$$

- Expanding by Taylor series:

$$\left\|\tilde{\mathbf{w}}[k+1]\right\|^2 = \left\|\mathbf{w}[k]\right\|^2 + 2\eta y[k]\tilde{\mathbf{w}}^T[k]\mathbf{x}[k] + O\left(\eta^2\right) =$$

$$= \left\|\mathbf{w}[k]\right\|^2 + 2\eta y^2[k] + O\left(\eta^2\right)$$

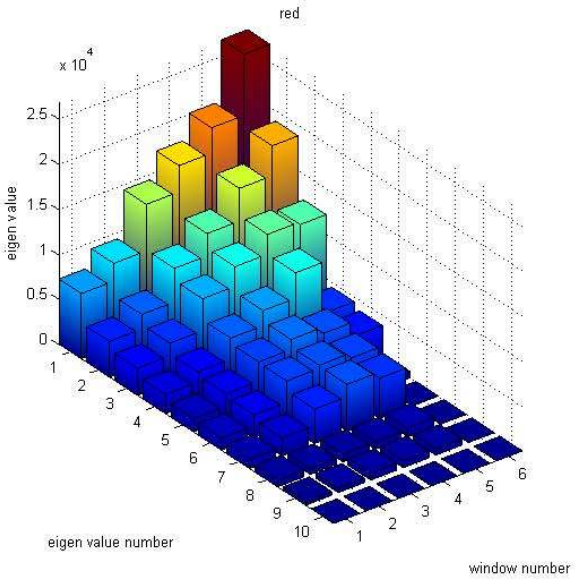
$$\left\|\tilde{\mathbf{w}}[k+1]\right\|^{-1} = 1 - \eta y^2[k] + O\left(\eta^2\right)$$

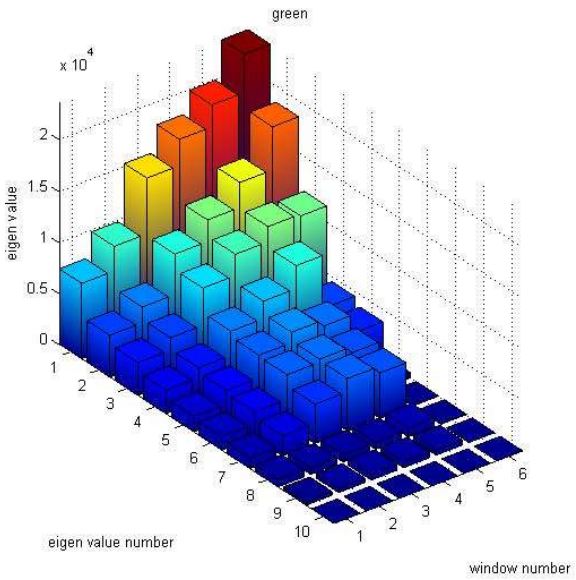
Experiments

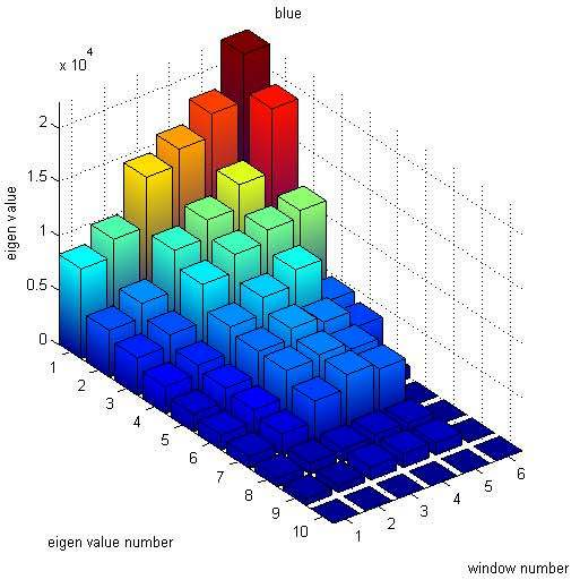
- An eigenvalue distribution of a small video stream.
- We experimented on two small video streams
  - The first illustrates a calm aquarium.
  - The second is an action scene with quick changes.
- We will present these sequences with different compression ratios using the Kernel and the Hebbian PCA

Experiments

- The video snippet consisted of 57 frames, window length of 10 was used.







Experiments

- Original:

- With 5 eigenvectors

- With all eigenvectors:

- With 2 eigenvectors:

Experiments

- Original:

- With all eigenvectors:

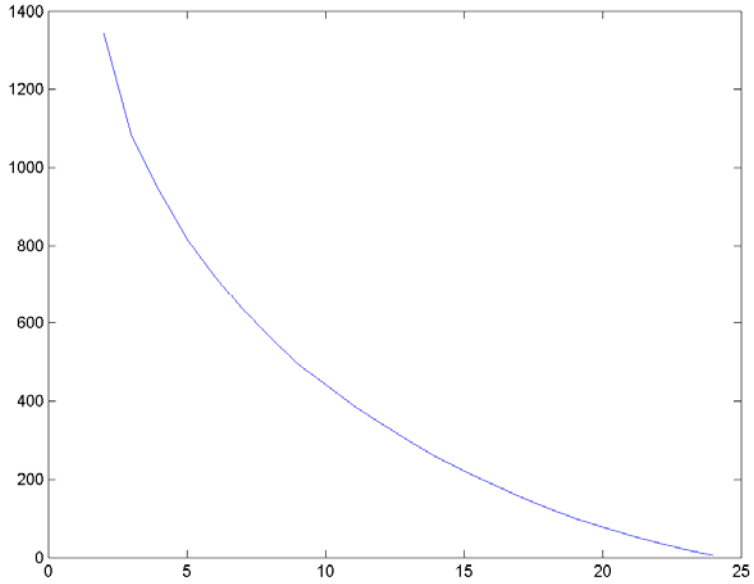
- With 20 eigenvectors:

- With 15 eigenvectors:

Experiments

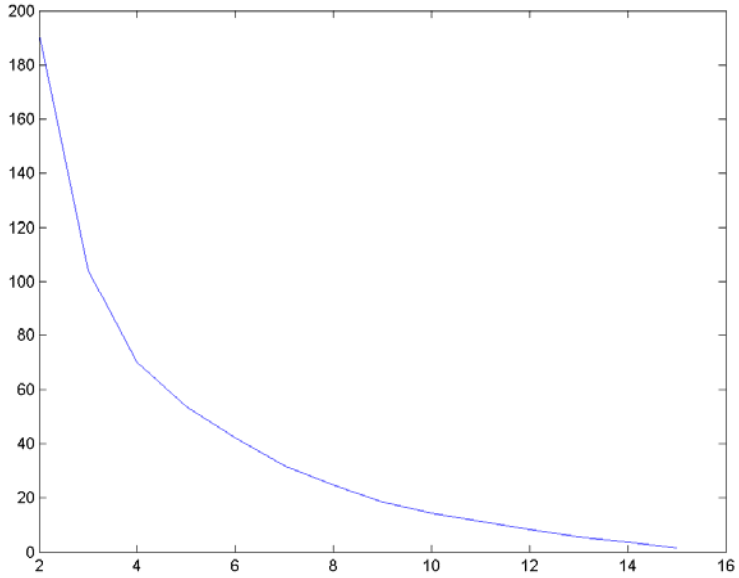
Error function: $\left\|\mathbf{x}-\hat{\mathbf{x}}\right\|^2 = \sum_{i=M+1}^{N} \lambda_i < \varepsilon$

- Aquarium



eigenvectors

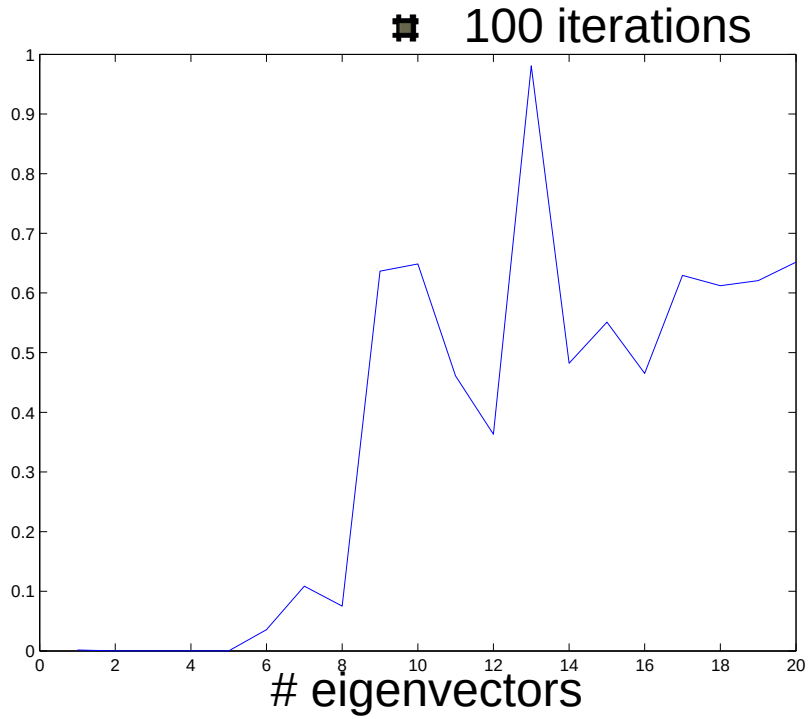
- Star Wars

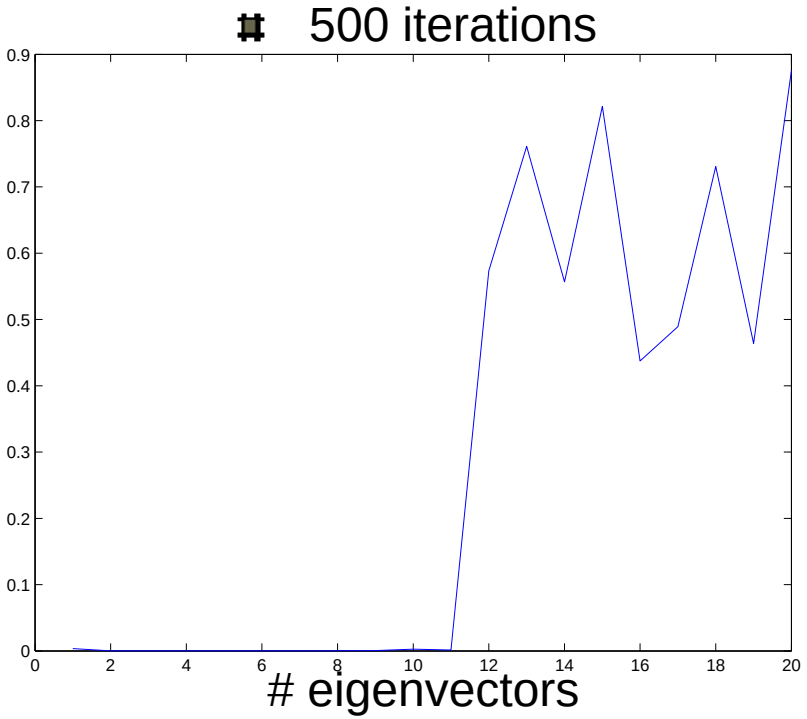


eigenvectors

Experiments

- Difference between the approximated and the true eigenvectors





Problems with solutions

1. A simple eigenvalue problem:

$$\mathbf{R} = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 1 & 0 \\ 0.5 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R}\mathbf{s}_i = \lambda_i \mathbf{s}_i \quad i = 1,2,3$$

$$\left(\mathbf{I}\lambda - \mathbf{R}\right)\mathbf{s} = 0$$

$$\det\left(\mathbf{I}\lambda - \mathbf{R}\right) = 0$$

$$\lambda_1 = 1.5 \qquad \lambda_2 = 1 \qquad \lambda_3 = 0$$

$$\mathbf{s}_1 = \begin{bmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{bmatrix} \qquad \mathbf{s}_2 = \begin{bmatrix} 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \qquad \mathbf{s}_3 = \begin{bmatrix} \frac{2}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} \end{bmatrix}$$

Problems with solutions

2. An information flow, compressed from time window to time window (e.g. video flow) has the following eigenvalues of the correlation matrix: $\lambda_m = m2^{-m}$, $\mathrm{m} = 1,\ldots,1000$
   If we want the mean square error of the PCA compression to be less or equal than 0.01, how many neurons should we implement in the coder ?

Problems with solutions

- The mean square error is defined as:

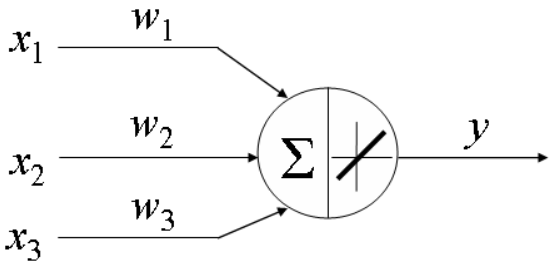
$$\mathbf{E}\left\|\mathbf{x}-\hat{\mathbf{x}}\right\|_2^2=\sum_{i=1}^{N}\lambda_i-\sum_{i=1}^{M}\lambda_i=\sum_{i=M+1}^{N}\lambda_i$$

- So at least 11 neurons must be used to achieve the desired error level

$$0.01\geq\mathbf{E}\left\|\mathbf{x}-\hat{\mathbf{x}}\right\|_2^2=\sum_{i=M+1}^{1000}\lambda_i=\sum_{i=M+1}^{1000}i2^{-i}\Bigg|_{M=11}=0.00634766$$

Problems with solutions

3. Train the underlying neuron with the Oja algorithm with the following parameters:



$$\mathbf{x} = \begin{bmatrix} 1 & -0.5 & -1 \end{bmatrix}^T$$

$$\mathbf{w}[0] = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$$

$$\eta = 0.1$$

a) Compute the weights after the 2nd iteration: $\mathbf{w}[2] = ?$

b) Can the following state be stabile? Explain your answer!

$$\mathbf{w}[\infty] = \begin{bmatrix} 2/3 & -1/3 & -2/3 \end{bmatrix}^T$$

Problems with solutions

a) Compute the weights after the 2nd iteration:

$$\mathbf{w}[k+1] = \mathbf{w}[k] + \eta y[k]\left(\mathbf{x}[k] - \mathbf{w}[k] y[k]\right)$$

$$\mathbf{x} = \begin{bmatrix} 1 & -0.5 & -1 \end{bmatrix}^{\mathrm{T}} \quad \mathbf{w}[0] = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^{\mathrm{T}} \quad \eta = 0.1$$

$$\mathbf{w}[1] = \mathbf{w}[0] + \eta y[0]\left(\mathbf{x}[0] - \mathbf{w}[0] y[0]\right)$$

$$y[0] = \mathbf{w}[0]^{T} \cdot \mathbf{x}[0] = -1$$

$$\mathbf{w}[1] = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^{\mathrm{T}} + 0.1(-1)\left(\begin{bmatrix} 1 & -0.5 & -1 \end{bmatrix}^{\mathrm{T}} - \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^{\mathrm{T}} (-1)\right)$$

$$\mathbf{w}[1] = \begin{bmatrix} -0.1 & 0.05 & 1 \end{bmatrix}^{\mathrm{T}}$$

Problems with solutions

a) Compute the weights after the 2nd iteration:

$$\mathbf{x}=\begin{bmatrix}1 & -0.5 & -1\end{bmatrix}^{\mathrm{T}} \quad \mathbf{w}[1]=\begin{bmatrix}-0.1 & 0.05 & 1\end{bmatrix}^{\mathrm{T}} \quad \eta=0.1$$

$$\mathbf{w}[2]=\mathbf{w}[1]+\eta y[1]\left(\mathbf{x}[1]-\mathbf{w}[1]y[1]\right)$$

$$y[1]=\mathbf{w}[1]^{T}\cdot\mathbf{x}[1]=-1.125$$

$$\mathbf{w}[2]=\begin{bmatrix}-0.1 & 0.05 & 1\end{bmatrix}^{\mathrm{T}}+0.1\frac{-9}{8}\left(\begin{bmatrix}1 & -0.5 & -1\end{bmatrix}^{\mathrm{T}}-\begin{bmatrix}-0.1 & 0.05 & 1\end{bmatrix}^{\mathrm{T}}\frac{-9}{8}\right)$$

$$\mathbf{w}[2]\approx\begin{bmatrix}-0.1998 & 0.0999 & 0.9859\end{bmatrix}^{T}$$

Problems with solutions

b) Can the following state be stable ? Explain your answer!

$$\mathbf{w}(\infty) = \begin{bmatrix} 2/3 & -1/3 & -2/3 \end{bmatrix}^{\mathrm{T}}$$

Let us check if it can be an eigenvector: each and every eigenvector must have a unit norm and form an orthonormal basis with the others. Since this vector has the unit norm it passes this check.

Let's solve the eigen problem to be sure:

$$\mathbf{R} = \mathbf{x} \cdot \mathbf{x}^T = \begin{bmatrix} 1 & -0.5 & -1 \\ -0.5 & 0.25 & 0.5 \\ -1 & 0.5 & 1 \end{bmatrix} \qquad \det\left(\mathbf{I}\lambda - \mathbf{R}\right) = 0$$

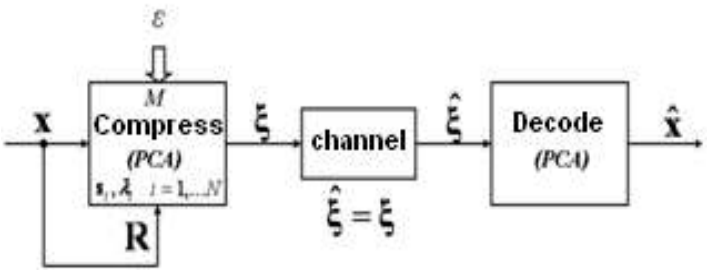
$$\lambda_1 = 2.25 \qquad \mathbf{s}^{(1)} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{2}{3} \end{bmatrix}^T$$

Problems with solutions

4. Let's assume that the correlation matrix of a weakly stationer process has the eigenvalues and vectors shown in the table below. We want to compress the process with PCA as such that the mean square error must be below the following threshold: $\varepsilon = 0.18$

   a) Give the compressed vector $\boldsymbol{\xi}$ and the restored data $\hat{\mathbf{x}}$, if a realization of the process is: $\mathbf{x} = [0.1 \quad 1 \quad 0 \quad -0.2 \quad -0.3]^T$

   b) Give the mean square error.



i	1	2	3	4	5
λ_i	0.1	0.02	0.01	0.15	0.05
$\mathbf{s}^{(i)}$	0.35034	0.34875	-0.28615	-0.82036	-0.027637
	–0.24812	–0.14365	0.72782	-0.40495	-0.47339
	0.59339	–0.79793	0.020443	-0.094387	0.043279
	–0.20137	–0.062407	0.34362	-0.26193	0.87685
	0.65041	0.46601	0.51952	0.29242	0.066293

Problems with solutions

- First rearrange the table in descending eigenvalue order.

i	1	2	3	4	5
λ_i	0.15	0.1	0.05	0.02	0.01
$\mathbf{s}^{(i)}$	-0.82036 -0.40495 -0.094387 -0.26193 0.29242	0.35034 –0.24812 0.59339 –0.20137 0.65041	-0.027637 -0.47339 0.043279 0.87685 0.066293	0.34875 –0.14365 –0.79793 –0.062407 0.46601	-0.28615 0.72782 0.020443 0.34362 0.51952

- Next we compute how many eigenvectors are needed for the coder and decoder and construct it.

$$0.18 = \varepsilon > \mathbf{E}\left\|\mathbf{x}-\hat{\mathbf{x}}\right\|_2^2 = \sum_{i=M+1}^{5} \lambda_i \Rightarrow M = 2$$

M	1	2	3	4
$E\left\|x-\hat{x}\right\|_2^2$	0.18	0.08	0.03	0.01

$$\hat{\mathbf{S}} = \begin{bmatrix} -0.82036 & 0.35034 \\ -0.40495 & -0.24812 \\ -0.094387 & 0.59339 \\ -0.26193 & -0.20137 \\ 0.29242 & 0.65041 \end{bmatrix}$$

Problems with solutions

- Let us check that the eigenvectors are proper eigenvectors:

$$\mathbf{s}^{(1)T}\cdot\mathbf{s}^{(1)}=1,\quad \mathbf{s}^{(2)T}\,\mathbf{s}^{(2)}=1,\quad \mathbf{s}^{(1)T}\,\mathbf{s}^{(2)}=7\cdot10^{-7}\approx 0 \text{ (due to truncation errors to 5 digits)}$$

- Using the coder, we transform the data to the new basis:

$$\hat{\mathbf{S}}=\begin{bmatrix} -0.82036 & 0.35034 \\ -0.40495 & -0.24812 \\ -0.094387 & 0.59339 \\ -0.26193 & -0.20137 \\ 0.29242 & 0.65041 \end{bmatrix} \qquad \mathbf{x}=\begin{bmatrix}0.1 & 1 & 0 & -0.2 & -0.3\end{bmatrix}^{\mathrm{T}}$$

$$\boldsymbol{\xi}=\hat{\mathbf{S}}^{T}\cdot\mathbf{x}=\begin{bmatrix}-0.52233\\ -0.36794\end{bmatrix}$$

- Let us compute the restored data:

$$\hat{\mathbf{x}}=\hat{\mathbf{S}}\cdot\hat{\boldsymbol{\xi}}=\begin{bmatrix}0.29959 & 0.30281 & -0.16903 & 0.2109 & -0.39205\end{bmatrix}^{T}$$

- Let us compute the empirical mean square error:

$$\left\|\mathbf{x}-\hat{\mathbf{x}}\right\|_2^2=\frac{1}{5}\sum_{i=1}^{5}\left(x_i-\widehat{x_i}\right)^2=0.14636$$

Problems

5. Design an FFNN for the compression layer and use the GHA to train the neurons for two iterations for the 4. problem.
6. Show that if the random process is not centralized (not zero mean) then the first eigenvalue is the mean and that eigenvector points to the direction of the mean.
7. Write a computer program that performs the PCA

Summary

- Data compression
- Principal Component analysis (PCA or KLT)
  - Notation
  - Summary of the algorithm
  - Problems, extensions
- Hebbian PCA (The Oja algorithm)
- Experiments
- Problems