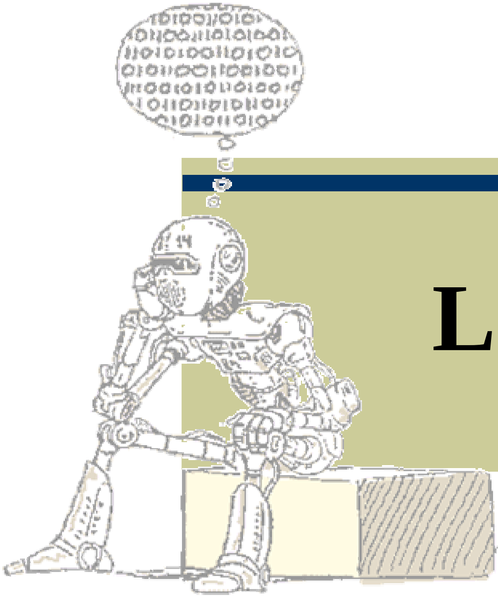


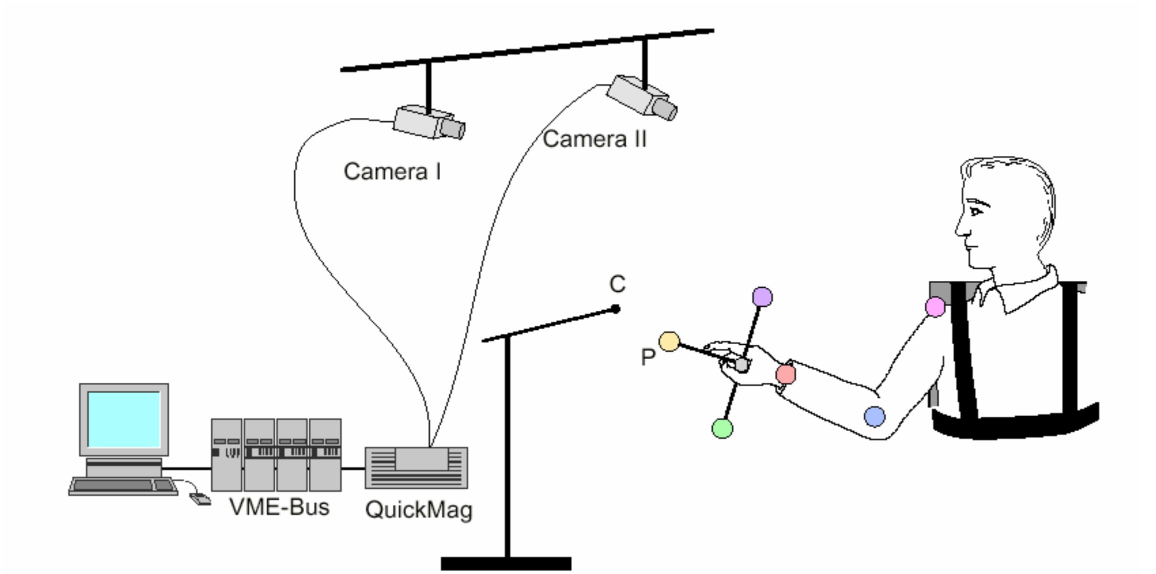
Lecture VIII– Dim. Reduction (I)

Contents:

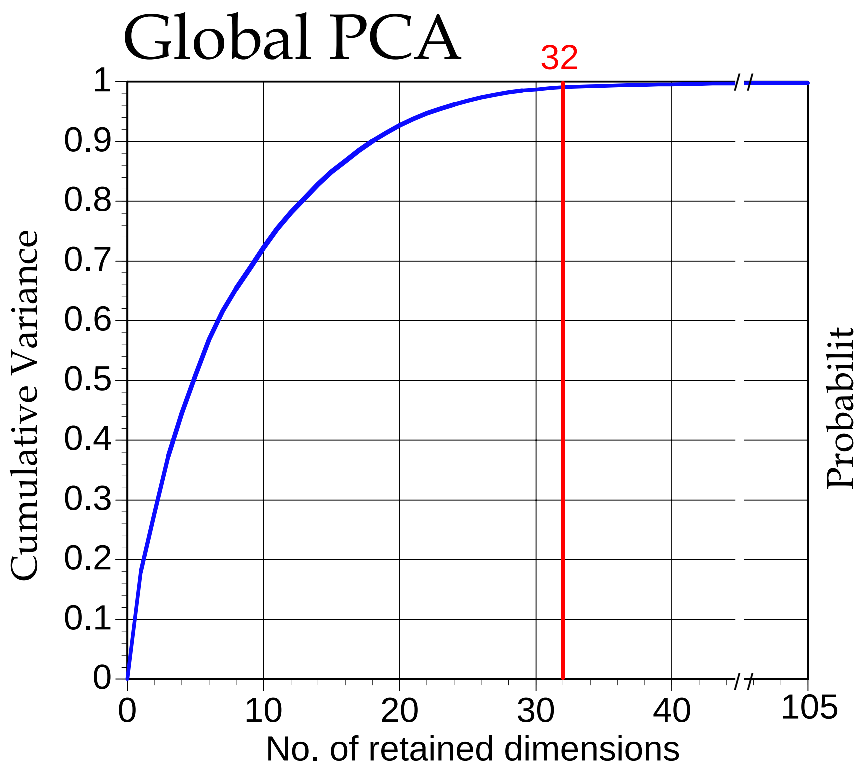
- Subset Selection & Shrinkage
  - Ridge regression, Lasso
- PCA, PCR, PLS

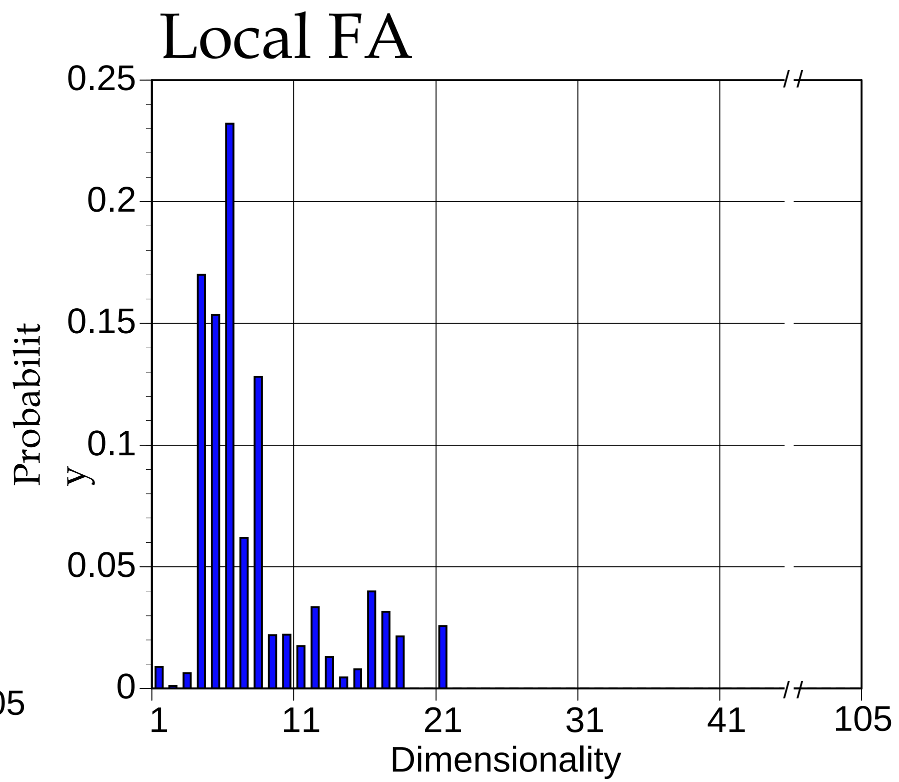
Data From Human Movement

- Measure arm movement and full-body movement of humans and anthropomorphic robots
- Perform local dimensionality analysis with a growing variational mixture of factor analyzers



Dimensionality of Full Body Motion





About 8 dimensions in the space formed by joint positions, velocities, and accelerations are needed to model an inverse dynamics model

Dimensionality Reduction

- **Goals:**
  - fewer dimensions for subsequent processing
  - better numerical stability due to removal of correlations
  - simplify post processing due to “advanced statistical properties” of pre-processed data
  - don’t lose important information, only redundant information or irrelevant information
  - perform dimensionality reduction spatially localized for nonlinear problems (e.g., each RBF has its own local dimensionality reduction)

Subset Selection & Shrinkage Methods

Subset Selection

Refers to methods which selects a set of variables to be included and discards other dimensions based on some optimality criterion. Does regression on this reduced dimensional input set. ***Discrete method*** –variables are either selected or discarded

- leaps and bounds procedure (Furnival & Wilson, 1974)
- Forward/Backward stepwise selection

Shrinkage Methods

Refers to methods which reduces/shrinks the redundant or irrelevant variables in a more continuous fashion.

- Ridge Regression
- Lasso
- Derived Input Direction Methods – PCR, PLS

Ridge Regression

Ridge regression shrinks the coefficients by imposing a penalty on their size. They minimize a penalized residual sum of squares :

$$\hat{w}^{ridge} = \arg\min_{w} \left\{ \sum_{i=1}^{N} (t_i - w_0 - \sum_{j=1}^{M} x_{ij} w_j)^2 + \lambda \sum_{j=1}^{M} w_j^2 \right\}$$

Complexity parameter controlling amount of shrinkage

Equivalent representation:

$$\hat{w}^{ridge} = \arg\min_{w} \left\{ \sum_{i=1}^{N} (t_i - w_0 - \sum_{j=1}^{M} x_{ij} w_j)^2 \right\}$$

$$subject\ \ to\ \sum_{j=1}^{M} w_j^2 \le s$$

Ridge regression (cont'd)

Some notes:

- when there are many correlated variables in a regression problem, their coefficients can be poorly determined and exhibit high variance e.g. a wildly large positive variable can be canceled by a similarly largely negative coefficient on its correlated cousin.
- The bias term is not included in the penalty.
- When the inputs are orthogonal, the ridge estimates are just a scaled version of the least squares estimate

Matrix representation of criterion and it's solution:

$$J(\mathbf{w},\lambda) = \frac{1}{2}\left(\mathbf{t} - \mathbf{X}\mathbf{w}\right)^T \left(\mathbf{t} - \mathbf{X}\mathbf{w}\right) + \lambda \mathbf{w}^T \mathbf{w}$$

$$\hat{\mathbf{w}}^{ridge} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^T\mathbf{t}$$

Ridge regression (cont'd)

Ridge regression shrinks the dimension with least variance the most.

Shrinkage Factor :

$$Each\ \ direction\ \ is\ \ shrunk\ \ by\ \frac{d_j^2}{(d_j^2+\lambda)},$$

[Page 62: Elem. Stat. Learning]

$$where\ \ d_j^2\ \ refers\ \ to\ \ the\ \ corresponding\ \ eigen\ \ value.$$

Effective degree of freedom :

$$\mathrm{df}(\lambda) = tr[\mathbf{X}(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T] = \sum_{j=1}^{M}\frac{d_j^2}{(d_j^2+\lambda)}$$

[Page 63: Elem. Stat. Learning]

$$Note: \mathrm{df}(\lambda) = M\ \ if\ \ \lambda = 0\ (no\ \ regularization)$$

Lasso

Lasso is also a shrinkage method like ridge regression. It minimizes a penalized residual sum of squares :

$$\hat{w}^{lasso} = \arg\min_{w} \left\{ \sum_{i=1}^{N} (t_i - w_0 - \sum_{j=1}^{M} x_{ij} w_j)^2 + \lambda \sum_{j=1}^{M} \left| w_j \right| \right\}$$

Equivalent representation:

$$\hat{w}^{lasso} = \arg\min_{w} \left\{ \sum_{i=1}^{N} (t_i - w_0 - \sum_{j=1}^{M} x_{ij} w_j)^2 \right\}$$

$$subject\ \ to \sum_{j=1}^{M} \left| w_j \right| \leq s$$

The L_2 ridge penalty is replaced by the L_1 lasso penalty. This makes the solution non-linear in **t.**

Derived Input Direction Methods

These methods essentially involves transforming the input directions to some low dimensional representation and using these directions to perform the regression.

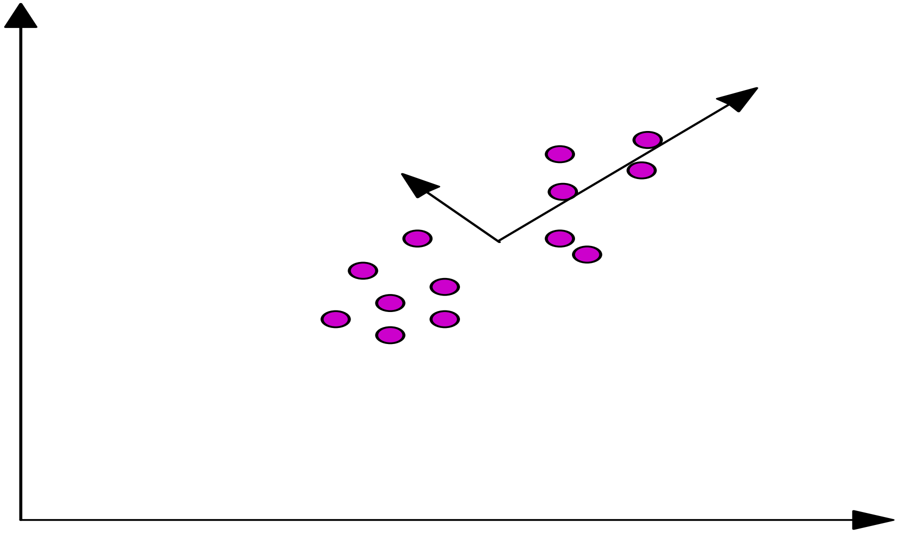
Example Methods

- **Principal Components Regression (PCR)**
  - Based on input variance only
- **Partial Least Squares (PLS)**
  - Based on input-output correlation and output variance

Principal Component Analysis

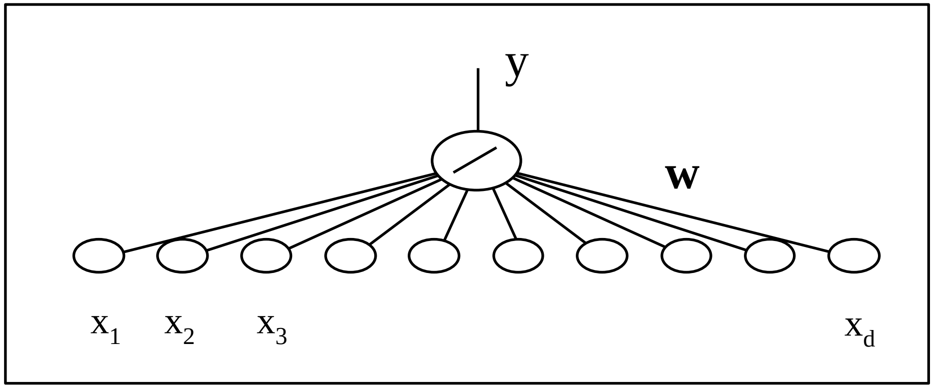
From earlier discussions:

- PCA finds the eigenvectors of the correlation (covariance) matrix of the the data
- Here, we show how PCA can be learned by bottle-neck neural networks (auto-associator) and Hebbian learning frameworks



PCA Implementation

- **Hebbian Learning**
  - obtain a measure of familiarity of a new data point
    - the more familiar a data point, the larger the output



$$\mathbf{w}^{n+1} = \mathbf{w}^{n} + \alpha y \mathbf{x}$$

Hebbian Learning

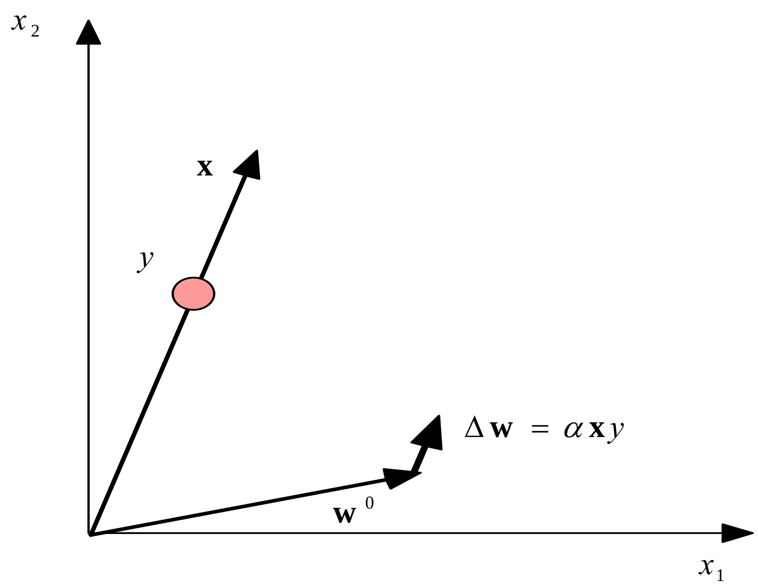
- **Properties of Hebbian Learning**
  - unstable learning rule (at most neutrally stable)
  - finds direction of maximal variance of the data

$$J = \frac{1}{2} y^2 = \frac{1}{2} \mathbf{w}^T \mathbf{x}\mathbf{x}^T \mathbf{w}$$

$$E\{J\} = E\left\{ \frac{1}{2} \mathbf{w}^T \mathbf{x}\mathbf{x}^T \mathbf{w} \right\}$$

$$= \frac{1}{2} \mathbf{w}^T E\left\{ \mathbf{x}\mathbf{x}^T \right\} \mathbf{w} = \frac{1}{2} \mathbf{w}^T \mathbf{C}\, \mathbf{w}$$

Here, **C** is the correlation matrix



Fixing the Instability (Oja's rule)

- Renormalization $$\mathbf{w}^{n+1} = \frac{\mathbf{w}^{n} + \alpha\mathbf{x}y}{\left\|\mathbf{w}^{n} + \alpha\mathbf{x}y\right\|}$$
- Oja's Rule $$\Delta\mathbf{w} = \mathbf{w}^{n+1} - \mathbf{w}^{n} = \alpha y\left(\mathbf{x} - y\mathbf{w}\right) = \alpha\left(y\mathbf{x} - y^{2}\mathbf{w}\right)$$
- Verify Oja's Rule (does it do the right thing ??)

$$
\begin{aligned}
E\{\Delta\mathbf{w}\} &= E\left\{\alpha\left(\mathbf{x}y - y^{2}\mathbf{w}\right)\right\} \\
&= \alpha E\left\{\mathbf{x}\mathbf{x}^{T}\mathbf{w} - \mathbf{w}^{T}\mathbf{x}\mathbf{x}^{T}\mathbf{w}\mathbf{w}\right\} \\
&= \alpha\left(\mathbf{C}\mathbf{w} - \mathbf{w}^{T}C\mathbf{w}\mathbf{w}\right)
\end{aligned}
$$

After convergence:

$$E\{\Delta\mathbf{w}\} = 0 \Rightarrow$$

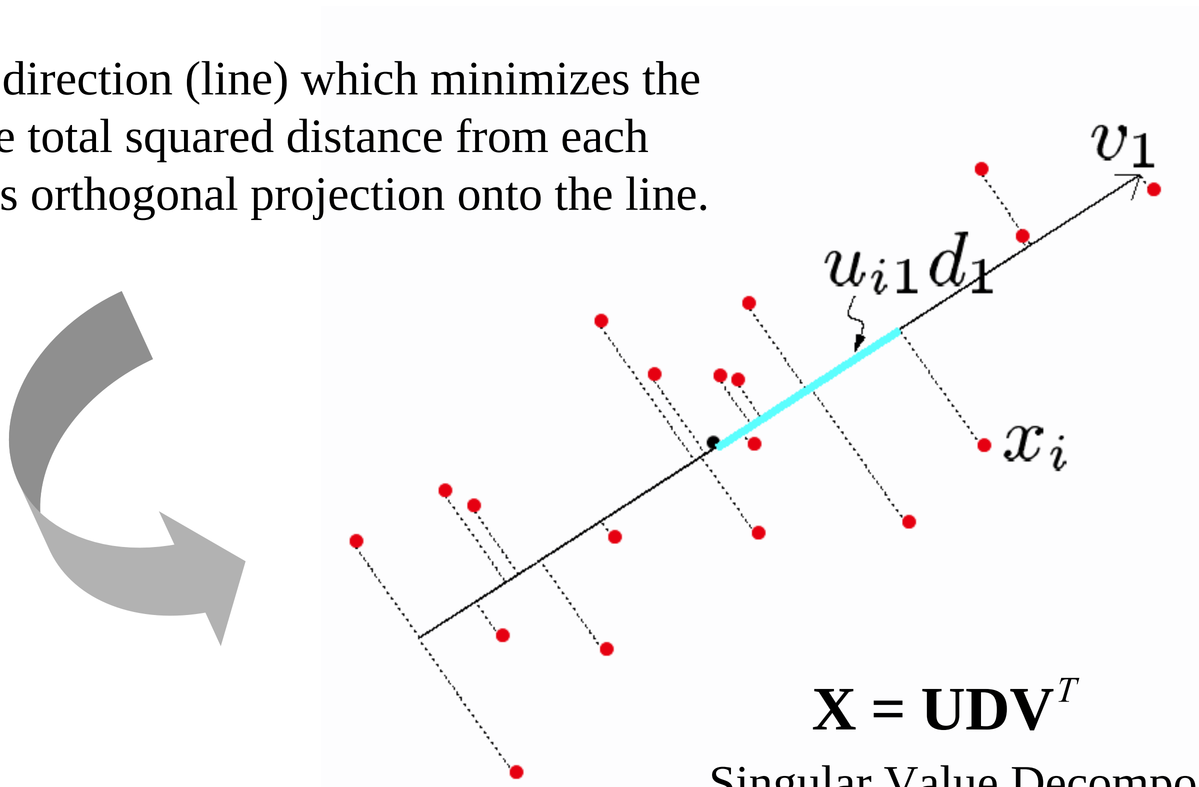
$$\mathbf{C}\mathbf{w} = \left(\mathbf{w}^{T}C\mathbf{w}\right)\mathbf{w} = \lambda\mathbf{w} \text{ , thus } \mathbf{w} \text{ is eigenvector}$$

$$\mathbf{w}^{T}\mathbf{C}\mathbf{w} = \mathbf{w}^{T}\left(\mathbf{w}^{T}C\mathbf{w}\right)\mathbf{w} = \left(\mathbf{w}^{T}C\mathbf{w}\right)\mathbf{w}^{T}\mathbf{w}$$

thus, $\mathbf{w}^{T}\mathbf{w} = 1$

Application Examples of Oja's Rule

Finds the direction (line) which minimizes the sum of the total squared distance from each point to its orthogonal projection onto the line.



$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$$

Singular Value Decomposition (SVD)

PCA : Batch vs Stochastic

- **PCA in Batch Update:**
  - just subtract the mean from the data
  - calculate eigenvectors to obtain principle components (Matlab function “eig”)

- **PCA in Stochastic Update:**
  - stochastically estimate the mean and subtract it from input data
  - use Oja’s rule on mean subtracted data

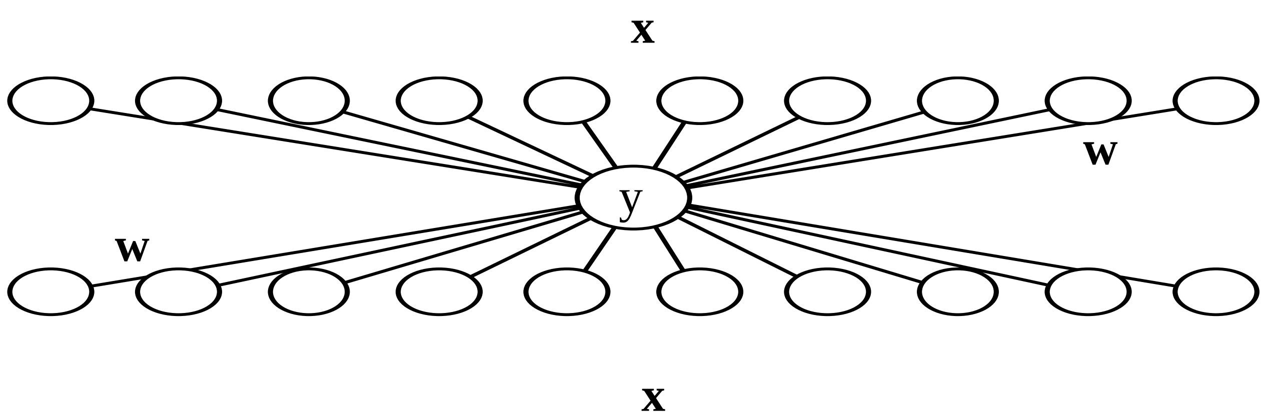
$$\bar{\mathbf{x}}^{n+1} = \frac{\bar{\mathbf{x}}^{n} n + \mathbf{x}}{n+1} = \bar{\mathbf{x}}^{n} + \frac{1}{n+1}\left(\mathbf{x} - \bar{\mathbf{x}}^{n}\right)$$

$$\bar{\mathbf{x}}^{n+1} = \bar{\mathbf{x}}^{n} + \alpha\left(\mathbf{x} - \bar{\mathbf{x}}^{n}\right)$$

Autoencoder as motivation for Oja's Rule

- Note that Oja's rule looks like a supervised learning rule
  - the update looks like a reverse delta-rule: it depends on the difference between the actual input and the back-propagated output

$$\Delta \mathbf{w} = \mathbf{w}^{n+1} - \mathbf{w}^{n} = \alpha y(\mathbf{x} - y\mathbf{w})$$



Convert unsupervised learning into a supervised learning problem by trying to reconstruct the inputs from few features!

Learning in the Autoencoder

- Minimize the cost $J = \frac{1}{2}(\mathbf{x} - \hat{\mathbf{x}})^T(\mathbf{x} - \hat{\mathbf{x}}) = \frac{1}{2}(\mathbf{x} - y\mathbf{w})^T(\mathbf{x} - y\mathbf{w})$

$$\frac{\partial J}{\partial \mathbf{w}} = -\left(\frac{\partial y}{\partial \mathbf{w}}\mathbf{w} + y\frac{\partial \mathbf{w}}{\partial \mathbf{w}}\right)^T(\mathbf{x} - y\mathbf{w})$$

$$= -\left(\frac{\partial(\mathbf{w}^T\mathbf{x})}{\partial \mathbf{w}}\mathbf{w} + y\frac{\partial \mathbf{w}}{\partial \mathbf{w}}\right)^T(\mathbf{x} - y\mathbf{w})$$

$$= -(\mathbf{x}^T\mathbf{w} + y)(\mathbf{x} - y\mathbf{w})$$

$$= -(y + y)(\mathbf{x} - y\mathbf{w})$$

$$= -2y(\mathbf{x} - y\mathbf{w})$$

$$\Delta\mathbf{w} = -\alpha\frac{\partial J}{\partial \mathbf{w}} = \alpha' y(\mathbf{x} - y\mathbf{w})$$

This is Oja's rule!

PCA with More Than One Feature

- Oja's Rule in **Multiple Dimensions**

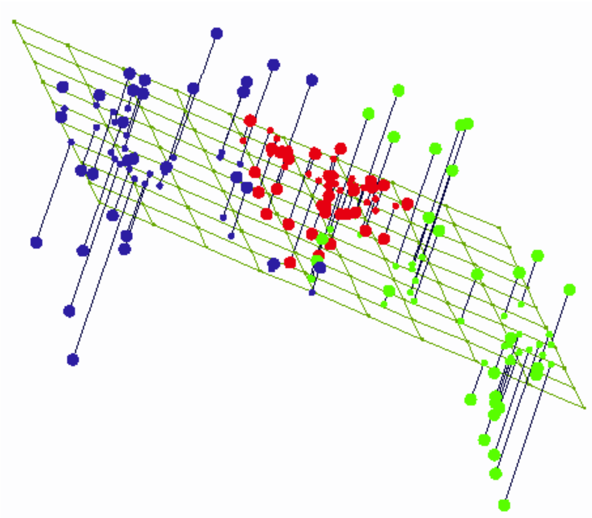
$$\mathbf{y} = \mathbf{W}\,\mathbf{x}$$
$$\hat{\mathbf{x}} = \mathbf{W}^T\mathbf{y}$$
$$\Delta\mathbf{W} = \alpha\,\mathbf{y}\left(\mathbf{x} - \mathbf{W}^T\mathbf{y}\right)^T$$

- **Sanger's Rule**: A clever trick added to Oja's rule!

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$
$$\hat{\mathbf{x}} = \mathbf{W}^T\mathbf{y}$$
$$[\Delta\mathbf{W}]_{r-th-row} = \alpha\, y_r\left(\mathbf{x} - \sum_{i=1}^{r}\left([\mathbf{W}]_{i-th-row}\right)^T y_i\right)^T$$
$$= \alpha\, y_r\left(\mathbf{x} - \left([\mathbf{W}]_{1:r-th-row}\right)^T[\mathbf{y}]_{1:r-th-row}\right)^T$$

Matlab Notation:

$$\mathbf{W}(r,:) = \alpha * \mathbf{y}(r) * (\mathbf{x} - \mathbf{W}(1:r,:)' * \mathbf{y}(1:r))$$



This rule makes the rows of **W** become the eigenvectors of the data, ordered in descending sequence according to the magnitude of the eigenvalues.

Discussion about PCA

- If data is noisy, we may represent the noise instead of the data
  - The way out: Factor Analysis (handles noise in input dimension also)
- PCA has no data generating model
- PCA has no probabilistic interpretation *(not quite true !!)*
- PCA ignores possible influence of subsequent (e.g., supervised) learning steps
- PCA is a linear method
  - Way out: Nonlinear PCA
- PCA can converge very slowly
  - Way out: EM versions of PCA
- But PCA is a very reliable method for dimensionality reduction if it is appropriate!

PCA preprocessing for Supervised Learning

- In Batch Learning Notation for a Linear Network:

$$\mathbf{U} = \left[eigenvectors \left(\frac{\sum_{n=1}^{N} \left(\mathbf{x}^n - \bar{\mathbf{x}}\right)\left(\mathbf{x}^n - \bar{\mathbf{x}}\right)^T}{N-1} \right) \right]_{\max(1:k)}$$

$$= \left[eigenvectors \left(\frac{\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}}{N-1} \right) \right]_{\max(1:k)} \quad \text{where } \tilde{\mathbf{X}} \text{ contains mean-zero data}$$

Subsequent Linear Regression for Network Weights

$$\mathbf{W} = \left(\mathbf{U}^T \tilde{\mathbf{X}}^T \tilde{\mathbf{X}} \mathbf{U}\right)^{-1} \mathbf{U}^T \tilde{\mathbf{X}}^T \mathbf{Y}$$

NOTE: Inversion of the above matrix is very cheap since it is diagonal! No numerical problems!

Problems of this pre-processing:

Important regression data might have been clipped!

Principal Component Regression (PCR)

Build the matrix X and vector y

$$\mathbf{X} = (\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{x}}_n)^T$$
$$\mathbf{t} = (t_1, t_2, \ldots, t_n)^T$$

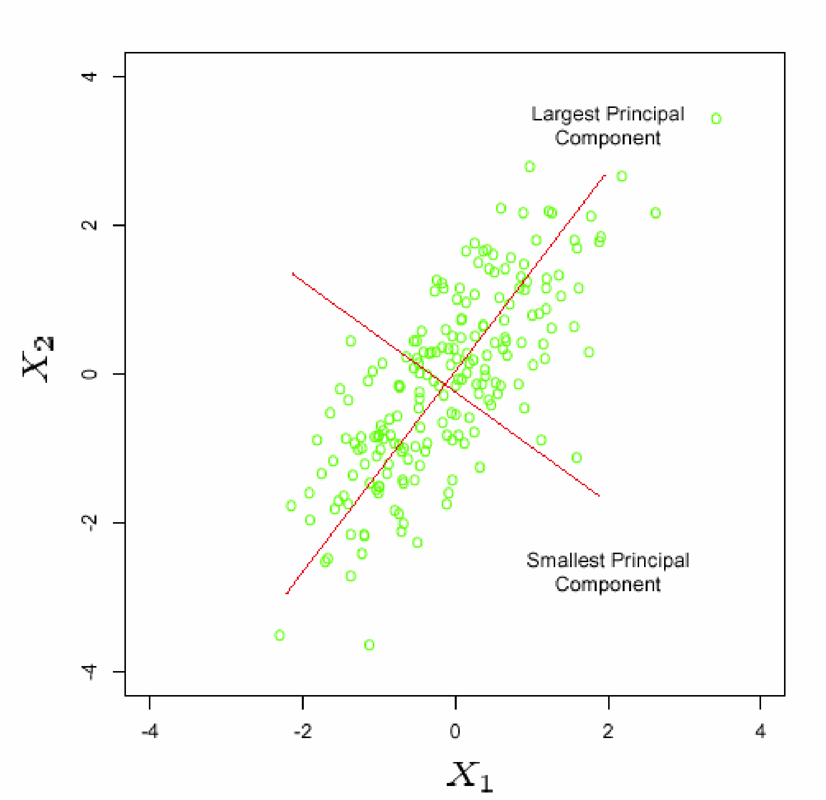
Eigen Decomposition

$$\mathbf{X}^T\mathbf{X} = \mathbf{V}\mathbf{D}^2\mathbf{V}^T$$

Compute the linear model

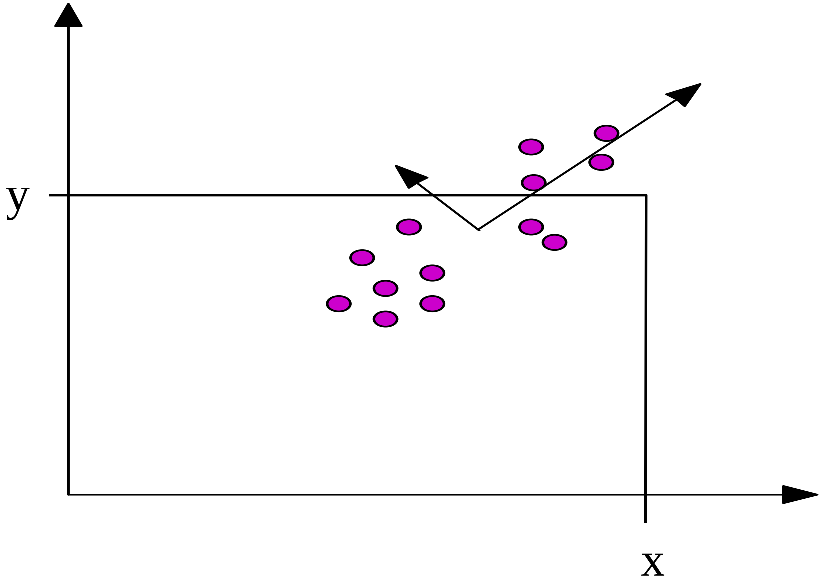
$$\mathbf{U} = \max_{eigen\,values} [\mathbf{v}_1 \mathbf{v}_2 \cdots \mathbf{v}_k]$$
$$\hat{\mathbf{w}}^{pcr} = (\mathbf{U}^T\mathbf{X}^T\mathbf{X}\mathbf{U})^{-1}\mathbf{U}^T\mathbf{X}^T\mathbf{t}$$

Data projection



PCA in Joint Data Space

- A straightforward extension to take supervised learning step into account:
  - perform PCA in joint data space
  - extract linear weight parameters from PCA results



PCA in Joint Data Space: Formalization

$$\mathbf{Z} = \begin{bmatrix} \mathbf{x} - \overline{\mathbf{x}} \\ \mathbf{y} - \overline{\mathbf{y}} \end{bmatrix}$$

$$\mathbf{U} = \left[eigenvectors \left(\frac{\sum_{n=1}^{N} \left(\mathbf{z}^n - \overline{\mathbf{z}}\right)\left(\mathbf{z}^n - \overline{\mathbf{z}}\right)^T}{N-1} \right) \right]_{\max(1:k)}$$

$$= \left[eigenvectors \left(\frac{\mathbf{Z}^T \mathbf{Z}}{N-1} \right) \right]_{\max(1:k)}$$

The Network Weights become:

$$\mathbf{W} = \mathbf{U}_x \left(\mathbf{U}_y^T - \mathbf{U}_y^T \left(\mathbf{U}_y \mathbf{U}_y^T - \mathbf{I} \right)^{-1} \mathbf{U}_y \mathbf{U}_y^T \right), \quad \text{where } \mathbf{U} = \begin{bmatrix} \mathbf{U}_x (= d \times k) \\ \mathbf{U}_y (= c \times k) \end{bmatrix}$$

Note: this new kind of linear network can actually tolerate noise in the input data! But only the same noise level in all (joint) dimensions!