Principal Component Analysis

CS498

Today's lecture

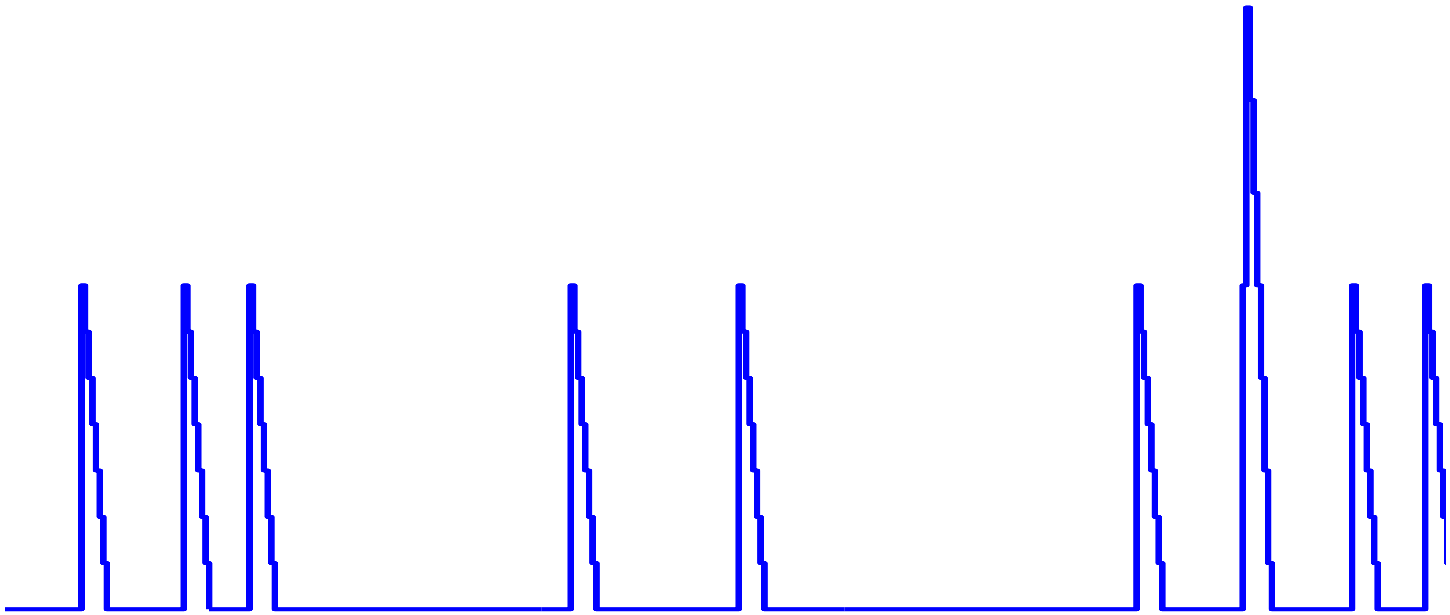
- Adaptive Feature Extraction
- Principal Component Analysis
  - How, why, when, which

A dual goal

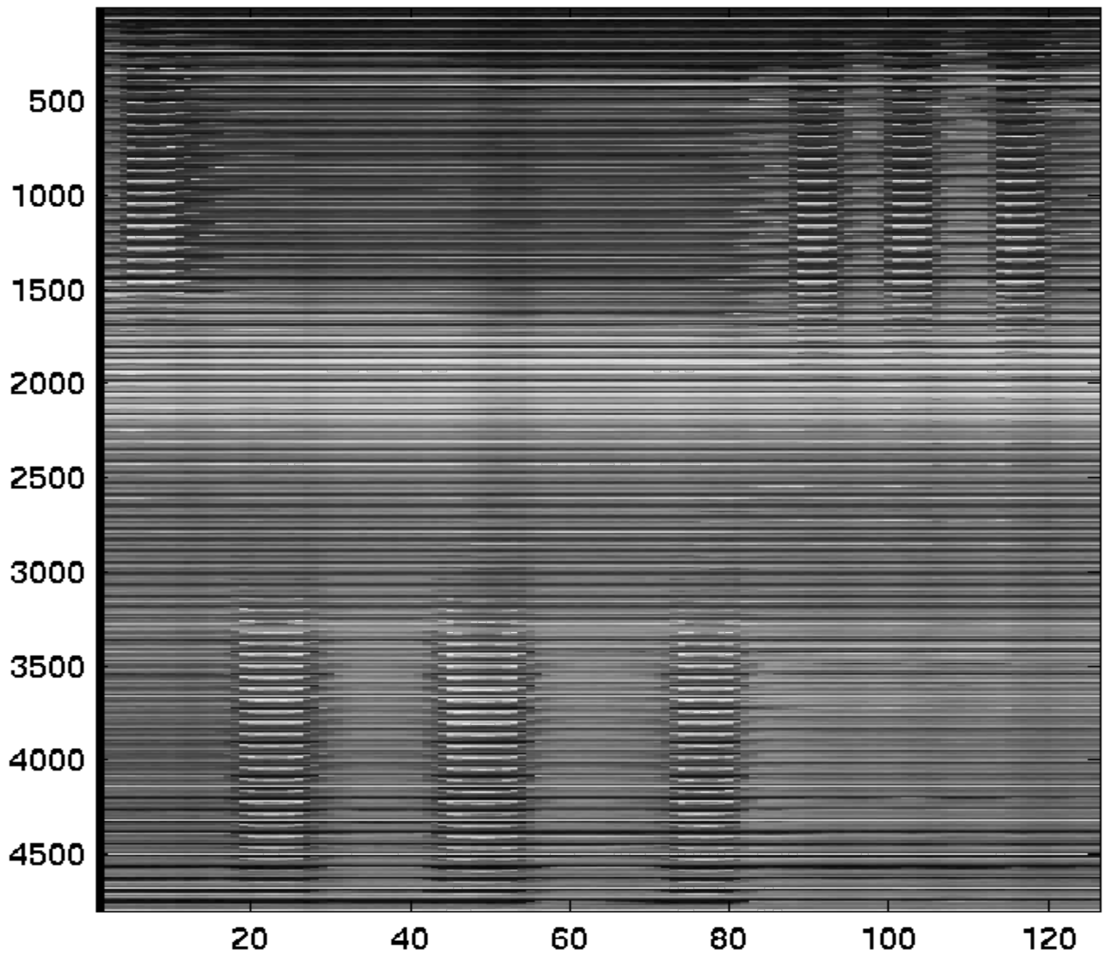
- Find a good representation
  - The features part
- Reduce redundancy in the data
  - A side effect of "proper" features

Example case

- Describe this input



What about now?

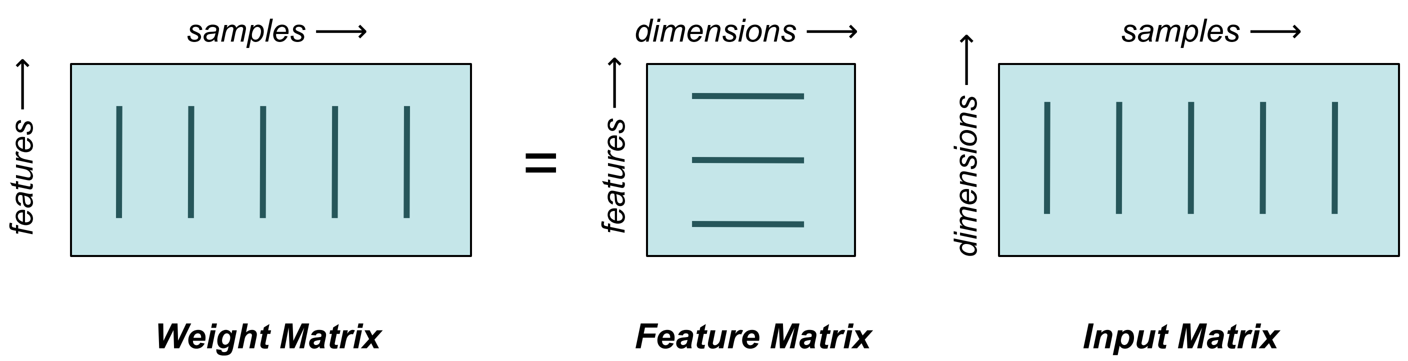


A “good feature”

- “Simplify” the explanation of the input
  - Represent repeating patterns
  - When defined makes the input simpler
- How do we define these abstract qualities?
  - On to the math …

Linear features

$$\mathbf{Z} = \mathbf{W}\mathbf{X}$$



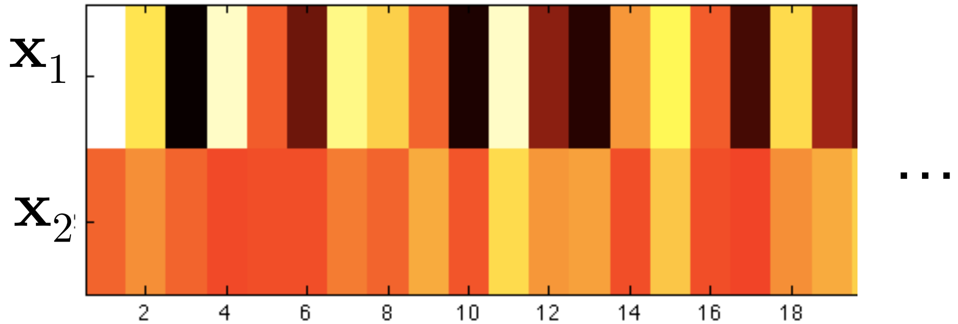
ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

A 2D case

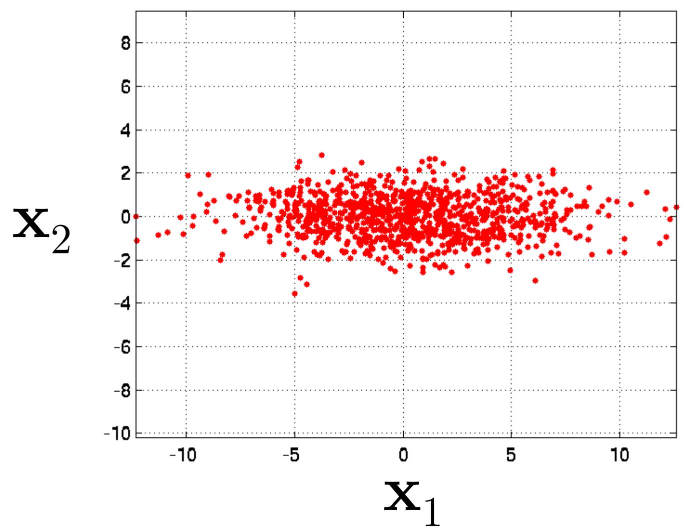
$$\mathbf{Z} = \mathbf{W}\mathbf{X} = $$

$$= \begin{bmatrix} \mathbf{z}_1^T \\ \mathbf{z}_2^T \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \end{bmatrix} \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \end{bmatrix}$$

Matrix representation of data



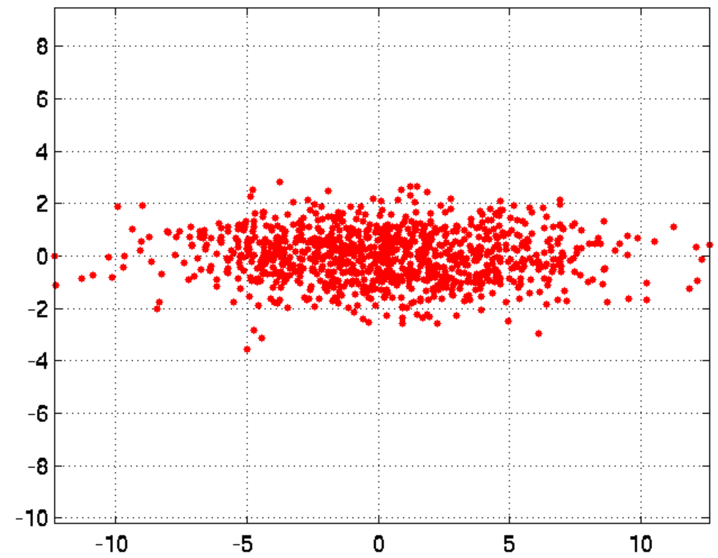
Scatter plot of same data



Defining a goal

- Desirable feature features
  - Give “simple” weights
  - Avoid feature similarity

- How do we define these?



One way to proceed

- "Simple weights"
  - Minimize relation of the two dimensions
- "Feature similarity"
  - Same thing!

One way to proceed

- "Simple weights"
  - Minimize relation of the two dimensions
  - Decorrelate: $\mathbf{z}_1^T \mathbf{z}_2 = 0$

- "Feature similarity"
  - Same thing!
  - Decorrelate: $\mathbf{w}_1^T \mathbf{w}_2 = 0$

Diagonalizing the covariance

- Covariance matrix

$$\mathrm{Cov}\left(\mathbf{z}_1, \mathbf{z}_2\right) = \begin{bmatrix} \mathbf{z}_1^T\mathbf{z}_1 & \mathbf{z}_1^T\mathbf{z}_2 \\ \mathbf{z}_2^T\mathbf{z}_1 & \mathbf{z}_2^T\mathbf{z}_2 \end{bmatrix} / N$$

- Diagonalizing the covariance suppresses cross-dimensional co-activity
  - if $\mathbf{z}_1$ is high, $\mathbf{z}_2$ won't be, etc

$$\mathrm{Cov}\left(\mathbf{z}_1, \mathbf{z}_2\right) = \begin{bmatrix} \mathbf{z}_1^T\mathbf{z}_1 & \mathbf{z}_1^T\mathbf{z}_2 \\ \mathbf{z}_2^T\mathbf{z}_1 & \mathbf{z}_2^T\mathbf{z}_2 \end{bmatrix} / N = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \mathbf{I}$$

Problem definition

- For a given input $\mathbf{X}$
- Find a feature matrix $\mathbf{W}$
- So that the weights decorrelate

$$\left(\mathbf{W}\mathbf{X}\right)\left(\mathbf{W}\mathbf{X}\right)^T = N\mathbf{I} \Rightarrow \mathbf{Z}\mathbf{Z}^T = N\mathbf{I}$$

How do we solve this?

- Any ideas?

$$\left(\mathbf{W}\mathbf{X}\right)\left(\mathbf{W}\mathbf{X}\right)^T = N\mathbf{I}$$

Solving for diagonalization

$$\left(\mathbf{W}\mathbf{X}\right)\left(\mathbf{W}\mathbf{X}\right)^T = N\mathbf{I} \Rightarrow$$

$$\Rightarrow \mathbf{W}\mathbf{X}\mathbf{X}^T\mathbf{W}^T = N\mathbf{I} \Rightarrow$$

$$\Rightarrow \mathbf{W}\mathrm{Cov}\left(\mathbf{X}\right)\mathbf{W}^T = \mathbf{I}$$

Solving for diagonalization

- Covariance matrices are positive definite
  - Therefore symmetric
    - have orthogonal eigenvectors and real eigenvalues
  - and are factorizable by:

$$\mathbf{U}^T\mathbf{A}\mathbf{U} = \Lambda$$

  - Where $\mathbf{U}$ has eigenvectors of $\mathbf{A}$ in its columns
  - $\Lambda = \mathrm{diag}(\lambda_i)$, where λ_i are the eigenvalues of $\mathbf{A}$

Solving for diagonalization

- The solution is a function of the eigenvectors $\mathbf{U}$ and eigenvalues $\mathbf{\Lambda}$ of $\mathrm{Cov}(\mathbf{X})$

$$\mathbf{W}\,\mathrm{Cov}\left(\mathbf{X}\right)\mathbf{W}^T = \mathbf{I} \Rightarrow$$

$$\Rightarrow \mathbf{W} = \begin{bmatrix} \sqrt{\lambda_1} & 0 \\ 0 & \sqrt{\lambda_2} \end{bmatrix}^{-1} \mathbf{U}^T$$

So what does it do?

- Input data covariance: $\mathrm{Cov}\left(\mathbf{X}\right) \approx \begin{bmatrix} 14.9 & 0.05 \\ 0.05 & 1.08 \end{bmatrix}$
- Extracted feature matrix: $\mathbf{W} \approx \begin{bmatrix} 0.12 & -30.4 \\ -8.17 & -0.03 \end{bmatrix} / N$
- Weights covariance: $\mathrm{Cov}(\mathbf{WX}) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Another solution

- This is not the only solution to the problem
- Consider this one:

$$\left(\mathbf{W}\mathbf{X}\right)\left(\mathbf{W}\mathbf{X}\right)^{T}=N\mathbf{I}\Rightarrow$$

$$\mathbf{W}\mathbf{X}\mathbf{X}^{T}\mathbf{W}^{T}=N\mathbf{I}\Rightarrow$$

$$\mathbf{W}=\left(\mathbf{X}\mathbf{X}^{T}\right)^{-1/2}$$

Another solution

- This is not the only solution to the problem
- Consider this one:

$$\left(\mathbf{WX}\right)\left(\mathbf{WX}\right)^T = N\mathbf{I} \Rightarrow$$

$$\mathbf{WXX}^T\mathbf{W}^T = N\mathbf{I} \Rightarrow$$

$$\mathbf{W} = \left(\mathbf{XX}^T\right)^{-1/2} \Rightarrow$$

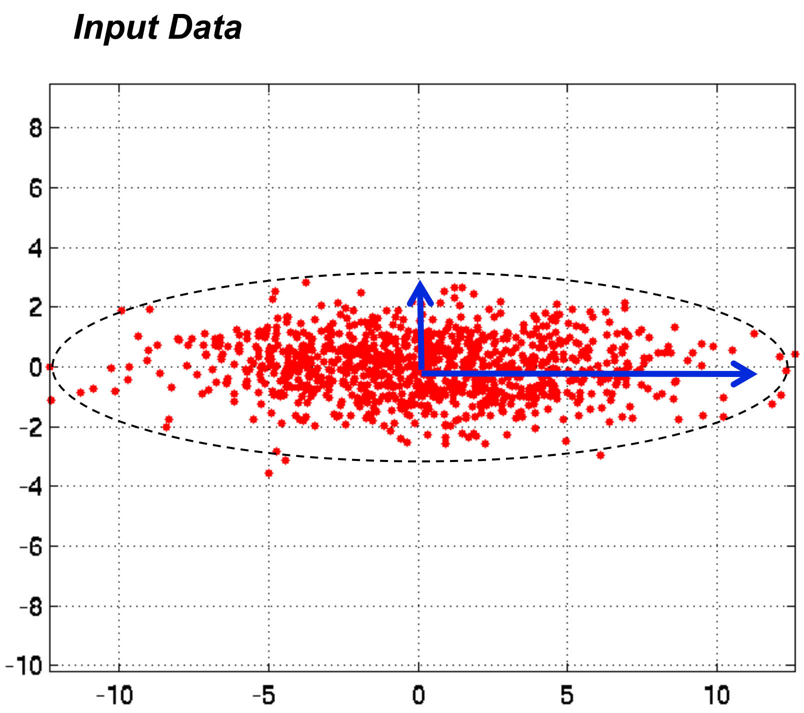
$$\mathbf{W} = \mathbf{US}^{-1/2}\mathbf{V}^T$$

$$[\mathbf{U},\mathbf{S},\mathbf{V}] = \mathrm{SVD}\left(\mathbf{XX}^T\right)$$

- Similar but out of scope for now

Decorrelation in pictures

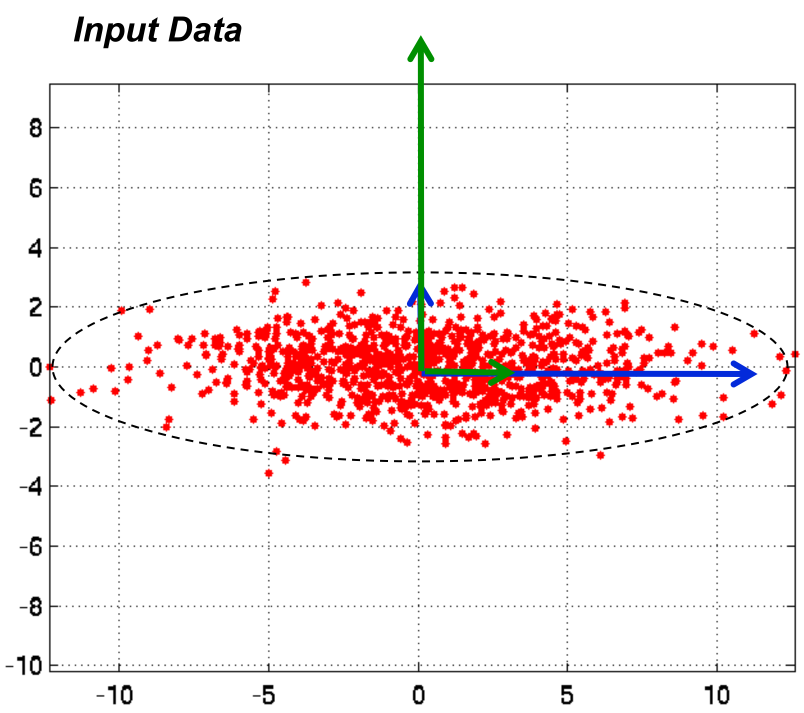
- An implicit Gaussian assumption
  - N-D data has N directions of variance



ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

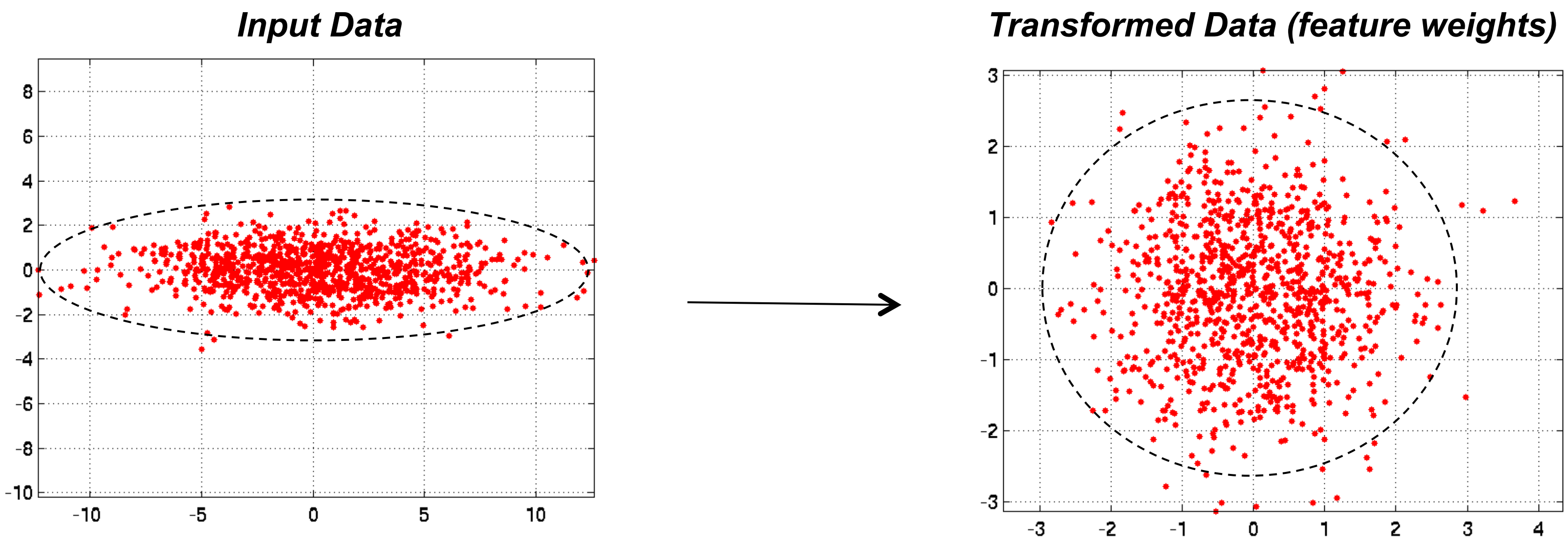
Undoing the variance

- The decorrelating matrix $\mathbf{W}$ contains two vectors that normalize the input's variance



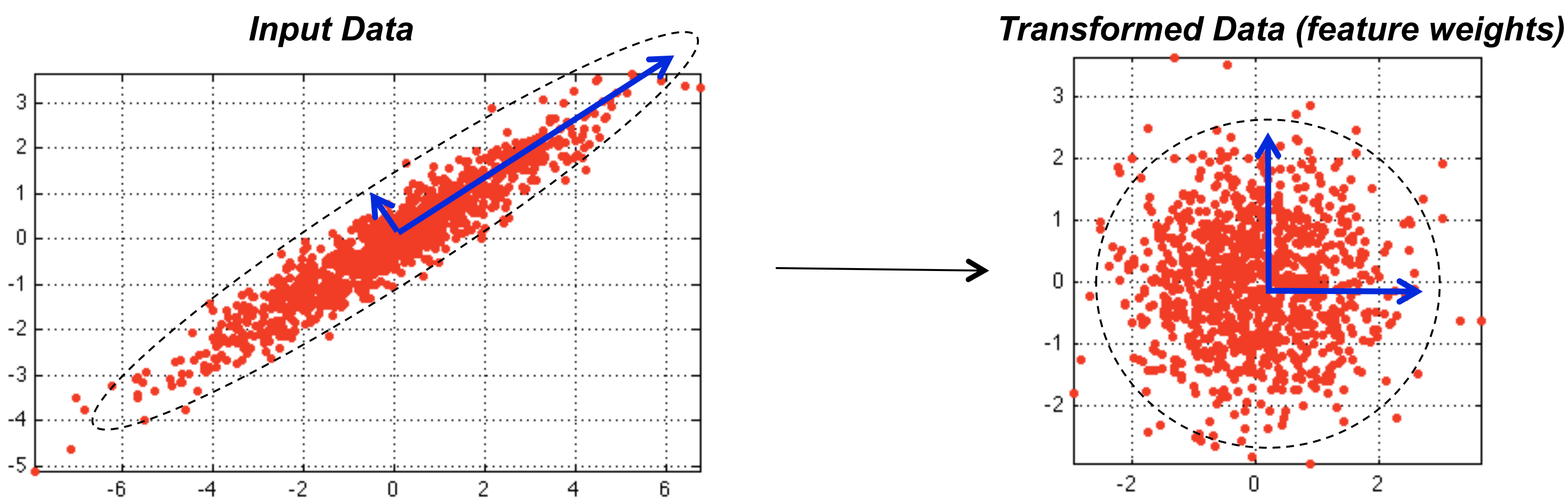
Resulting transform

- Input gets scaled to a well behaved Gaussian with unit variance in all dimensions



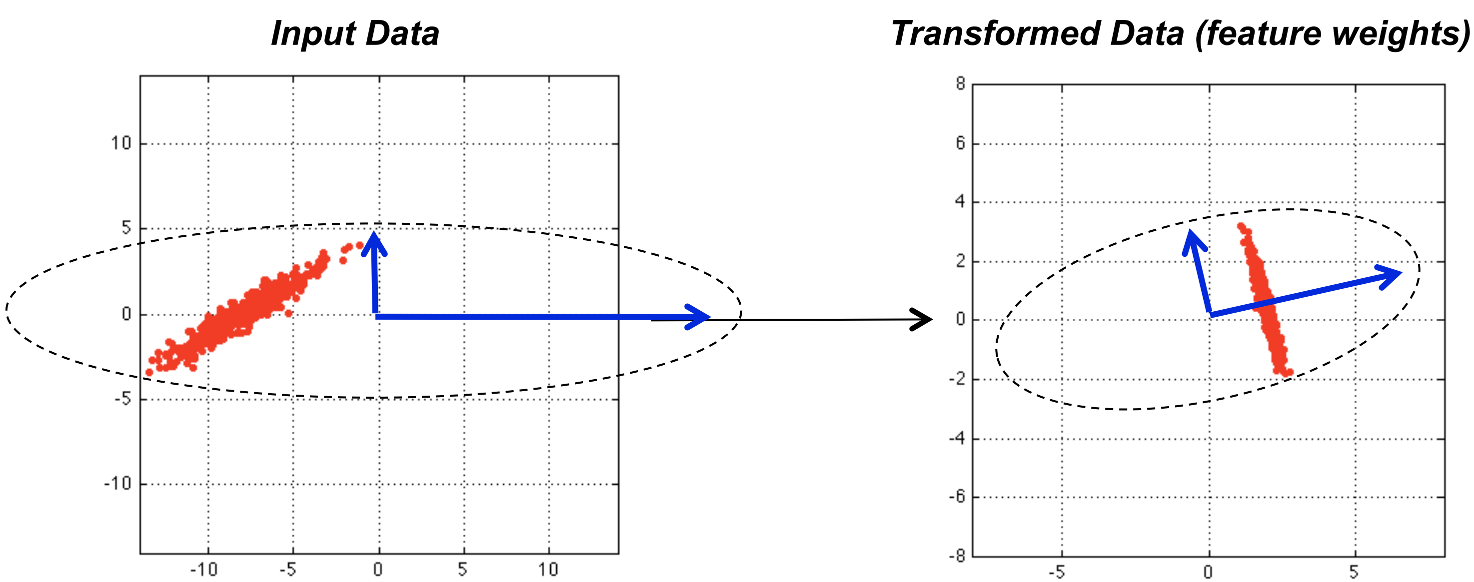
A more complex case

- Having correlation between two dimensions
  - We still find the directions of maximal variance
  - But we also rotate in addition to scaling



One more detail

- So far we considered zero-mean inputs
  - The transforming operation was a rotation
- If the input mean is not zero bad things happen!
  - Make sure that your data is zero-mean!



Principal Component Analysis

- This transform is known as PCA
  - The features are the principal components
    - They are orthogonal to each other
    - And produce orthogonal (white) weights
  - Major tool in statistics
    - Removes dependencies from multivariate data

- Also known as the KLT
  - Karhunen-Loeve transform

A better way to compute PCA

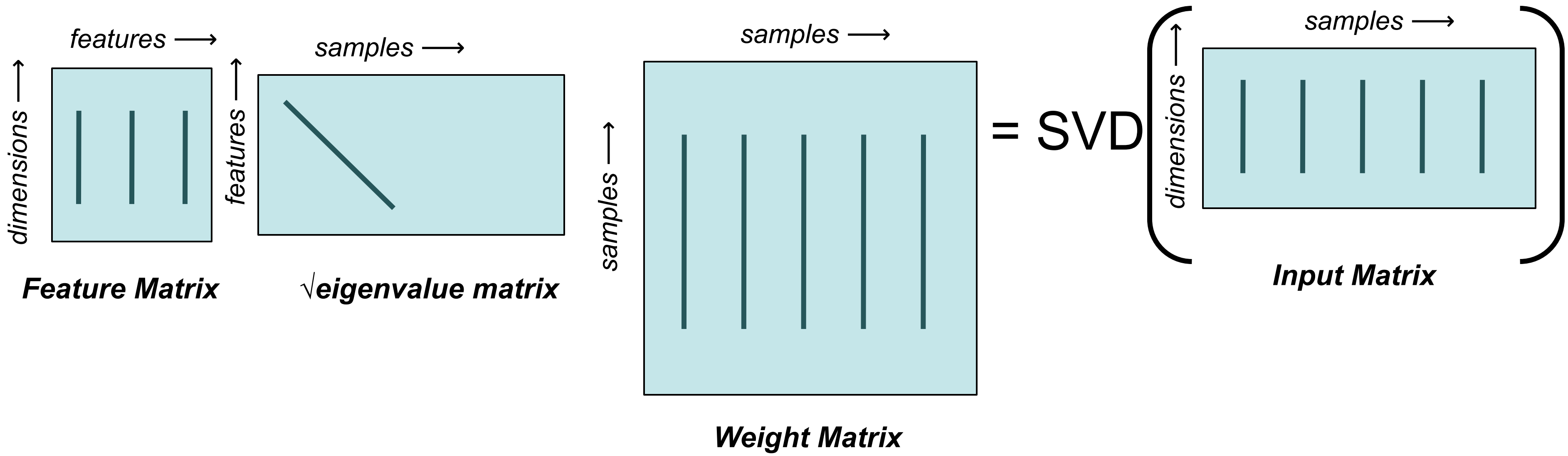
- The Singular Value Decomposition way

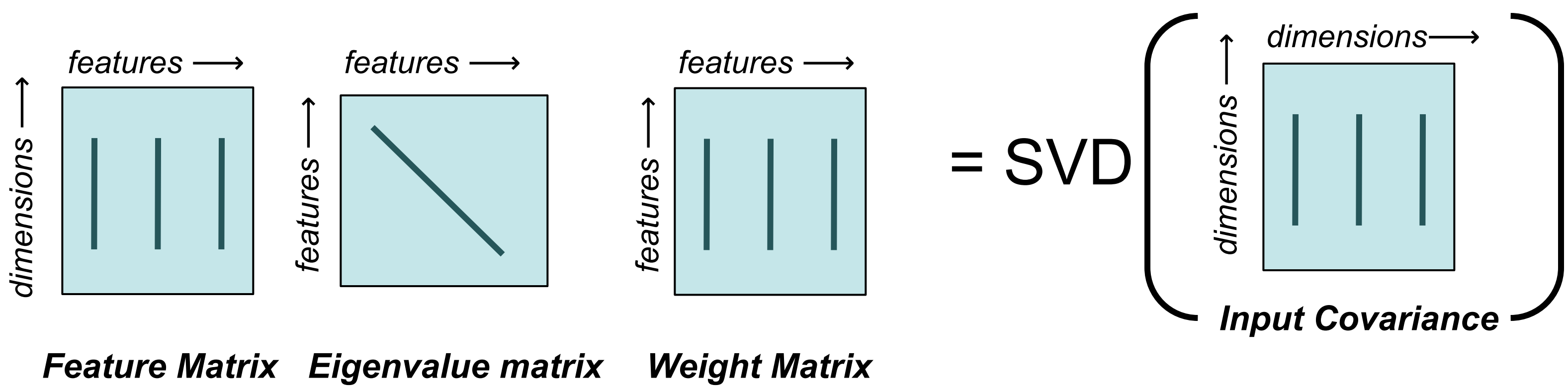
$$[\mathbf{U}, \mathbf{S}, \mathbf{V}] = \mathrm{SVD}(\mathbf{A}) \Rightarrow \mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$

- Relationship to eigendecomposition
  - In our case (covariance input $\mathbf{A}$), $\mathbf{U}$ and $\mathbf{S}$ will hold the eigenvectors/values of $\mathbf{A}$

- Why the SVD?
  - More stable, more robust, fancy extensions

PCA through the SVD





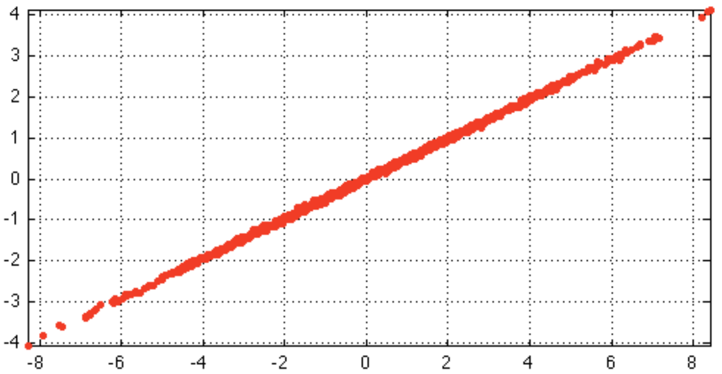
Dimensionality reduction

- PCA is great for high dimensional data

- Allows us to perform dimensionality reduction
  - Helps us find relevant structure in data
  - Helps us throw away things that won't matter

A simple example

- Two very correlated dimensions
  - e.g. size and weight of fruit
  - One effective variable
- PCA matrix here is:

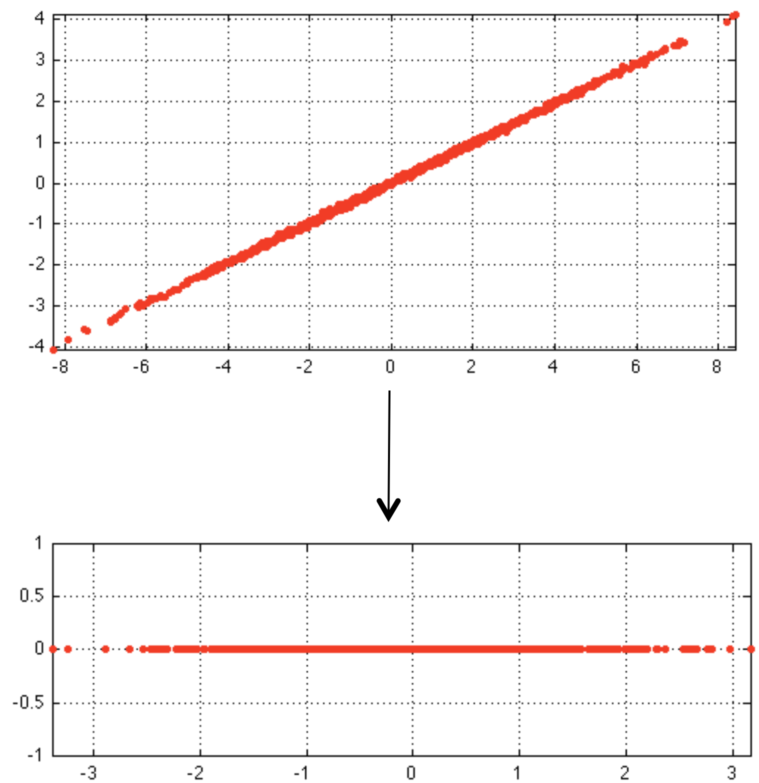


$$\mathbf{W} = \begin{bmatrix} -0.2 & -0.13 \\ -13.7 & 28.2 \end{bmatrix}$$

  - Large variance between the two components
    - about two orders of magnitude

A simple example

- Second principal component needs to be super-boosted to whiten the weights
  - maybe is it useless?
- Keep only high variance
  - Throw away components with minor contributions



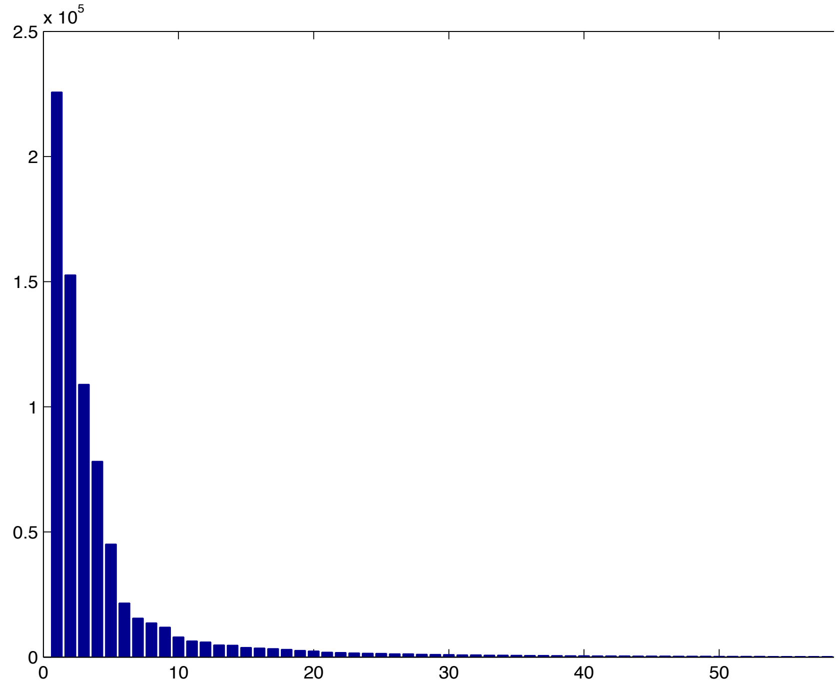
What is the number of dimensions?

- If the input was M dimensional, how many dimensions do we keep?
  - No solid answer (estimators exist but are flaky)

- Look at the singular/eigen-values
  - They will show the variance of each component, at some point it will be small

Example

- Eigenvalues of 1200 dimensional video data
  - Little variance after component 30
  - We don't need to keep the rest of the data



…

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

So where are the features?

- We strayed off-subject
  - What happened to the features?
  - We only mentioned that they are orthogonal

- We talked about the weights so far, let's talk about the principal components
  - They should encapsulate structure
  - How do they look like?

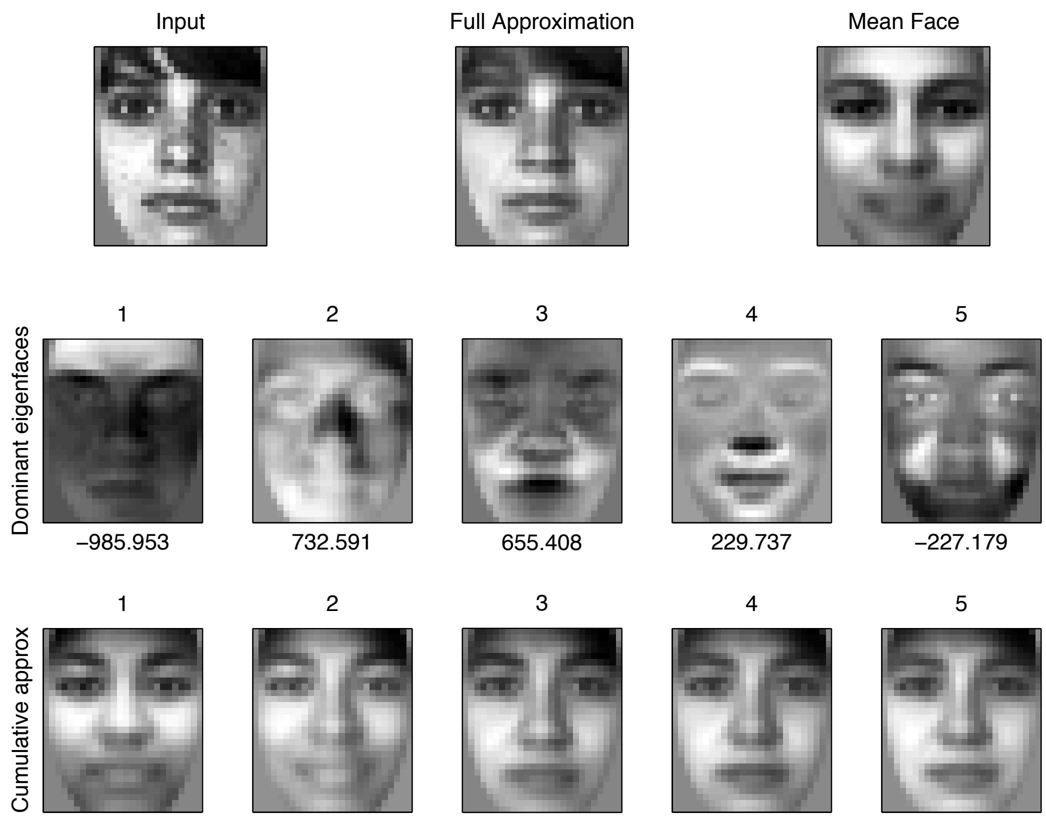
Face analysis

- Analysis of a face database
  - What are good features for faces?
- Is there anything special there?
  - What will PCA give us?
  - Any extra insight?
- Lets use MATLAB to find out …

The Eigenfaces

Low-rank model

- Instead of using 780 pixel values we use the PCA weights (here 50 and 5)



PCA for large dimensionalities

- Sometimes the data is high dim
  - e.g. videos 1280x720xT = 921,600D x T frames
- You will not do an SVD that big!
  - Complexity is $O(4m^2n + 8mn^2 + 9n^3)$
- Useful approach is the EM-PCA

EM-PCA in a nutshell

- Alternate between successive approximations
  - Start with random $\mathbf{C}$ and loop over:

$$\mathbf{Z} = \mathbf{C}^{+}\mathbf{X}$$

$$\mathbf{C} = \mathbf{X}\mathbf{Z}^{+}$$

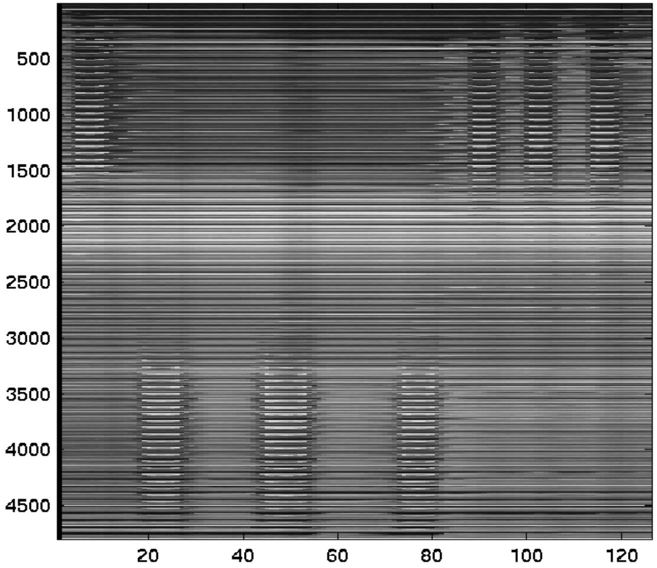
  - After convergence $\mathbf{C}$ spans the PCA space
- If we choose a low rank $\mathbf{C}$ then computations are significantly more efficient than the SVD
  - More later when we cover EM

PCA for online data

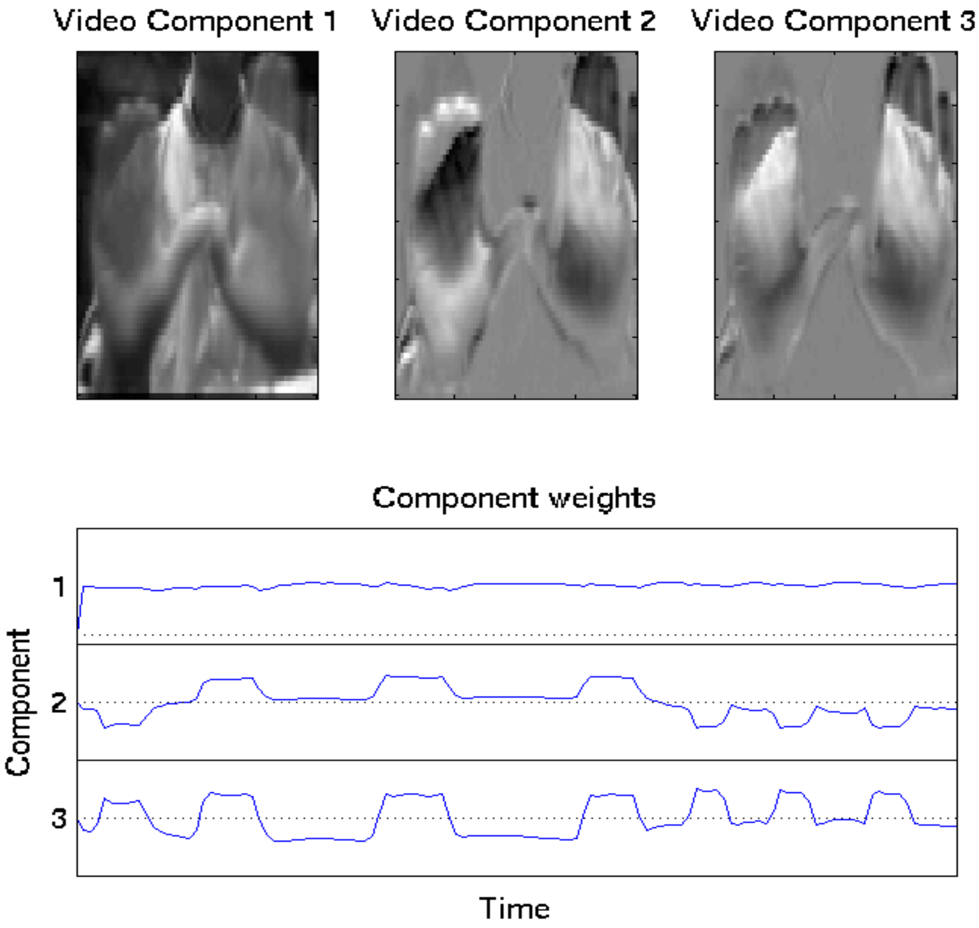
- Sometimes we have too many data samples
  - Irrespective of the dimensionality
  - e.g. long video recordings
  -
- Incremental SVD algorithms
  - Update the $\mathbf{U},\mathbf{S},\mathbf{V}$ matrices with only a small set or a single sample point
  - Very efficient updates

A Video Example

- The movie is a series of frames
  - Each frame is a data point
  - 126, 80x60 pixel frames
  - Data will be 4800x126
- We can do PCA on that



PCA Results


Video Component 1
Video Component 2
Video Component 3
Component weights
1
2
3
Component
Time

PCA for online data II

- "Neural net" algorithms
- Naturally online approaches
  - With each new datum, PC's are updated
- Oja's and Sanger's rules
  - Gradient algorithms that update $\mathbf{W}$
- Great when you have minimal resources

PCA and the Fourier transform

- We've seen why sinusoids are important
  - But can we statistically justify it?
- PCA has a deep connection with the DFT
  - In fact you can derive the DFT from PCA

An example

- Let’s take a time series which is not “white”
  - Each sample is somewhat correlated with the previous one (Markov process)

$$\left[\begin{array}{ccc} x(t), & \cdots, & x(t+T) \end{array}\right]$$

- We’ll make it multidimensional

$$\mathbf{X} = \left[\begin{array}{ccc} x(t) & x(t+1) & \\ \vdots & \vdots & \cdots \\ x(t+N) & x(t+1+N) & \end{array}\right]$$

An example

- In this context, features will be repeating temporal patterns smaller than N

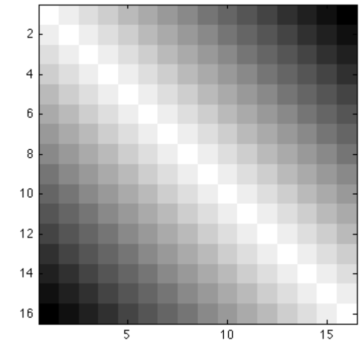
$$\mathbf{Z} = \mathbf{W} \begin{bmatrix} x(t) & x(t+1) & \\ \vdots & \vdots & \cdots \\ x(t+N) & x(t+1+N) & \end{bmatrix}$$

- If $\mathbf{W}$ is the Fourier matrix then we are performing a frequency analysis

PCA on the time series

- By definition there is a correlation between successive samples

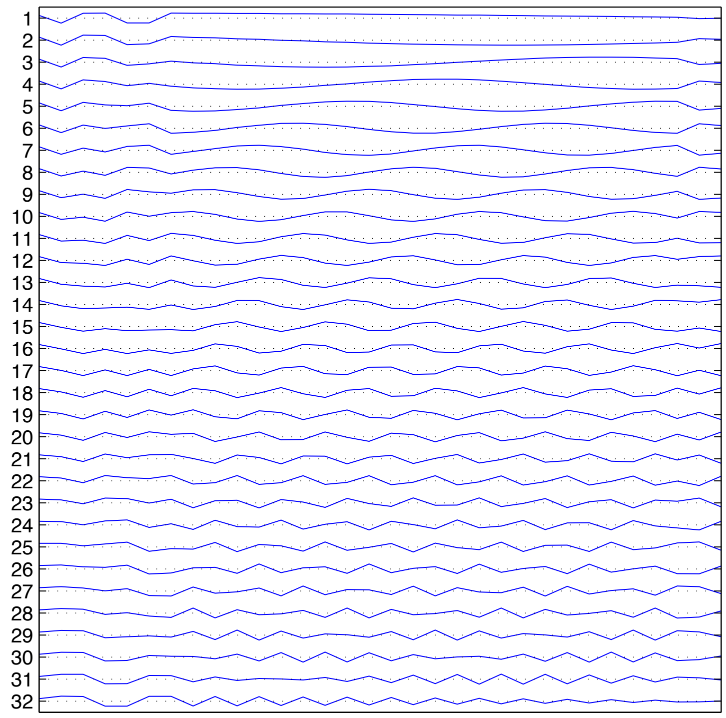
$$\mathrm{Cov}\left(\mathbf{X}\right) \approx \begin{bmatrix} 1 & 1-e & \cdots & 0 \\ 1-e & 1 & 1-e & \vdots \\ \vdots & 1-e & 1 & 1-e \\ 0 & \cdots & 1-e & 1 \end{bmatrix} =$$



- Resulting covariance matrix will be symmetric Toeplitz with a diagonal tapering towards 0

Solving for PCA

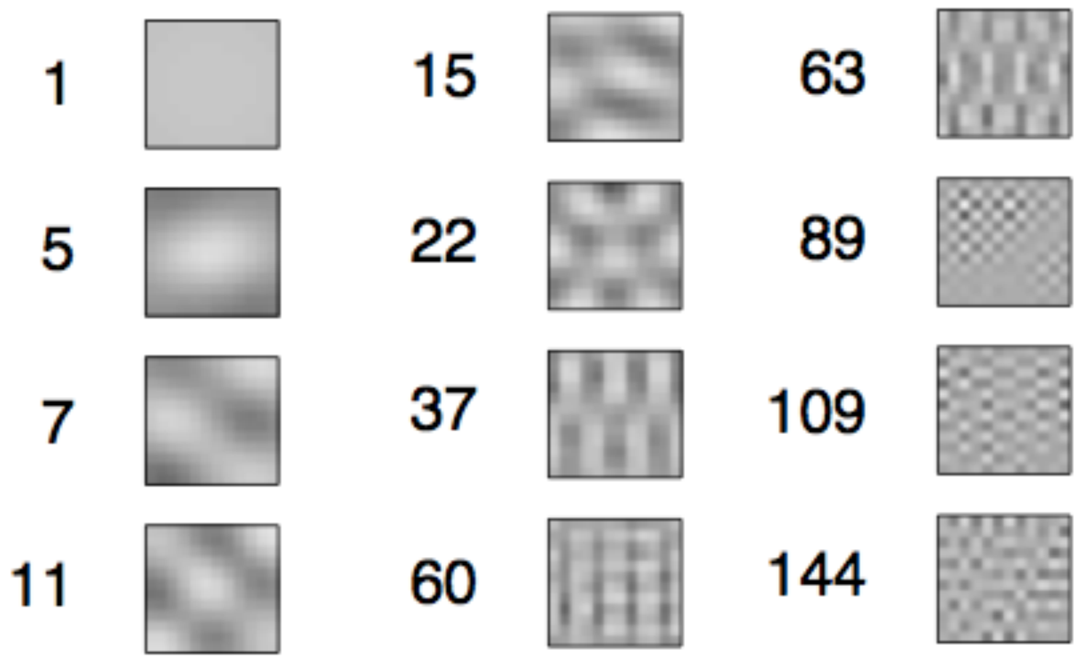
- The eigenvectors of Toeplitz matrices like this one are (approximately) sinusoids



ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

And ditto with images

- Analysis of coherent images results in 2D sinusoids



ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

So now you know

- The Fourier transform is an "optimal" decomposition for time series
  - In fact you will often not do PCA and do a DFT
- There is also a loose connection with our perceptual system
  - We kind of use similar filters in our ears and eyes (but we'll make that connection later)

Recap

- Principal Component Analysis
  - Get used to it!
  - Decorrelates multivariate data, finds useful components, reduces dimensionality
- Many ways to get to it
  - Knowing what to use with your data helps
- Interesting connection to Fourier transform

Check these out for more

- Eigenfaces
  - http://en.wikipedia.org/wiki/Eigenface
  - http://www.cs.ucsb.edu/~mturk/Papers/mturk-CVPR91.pdf
  - http://www.cs.ucsb.edu/~mturk/Papers/jcn.pdf
- Incremental SVD
  - http://www.merl.com/publications/TR2002-024/
- EM-PCA
  - http://cs.nyu.edu/~roweis/papers/empca.ps.gz