CS545 Machine Learning for Signal Processing

Features Part I – Principal Component Analysis

4 September 2023

Today's lecture

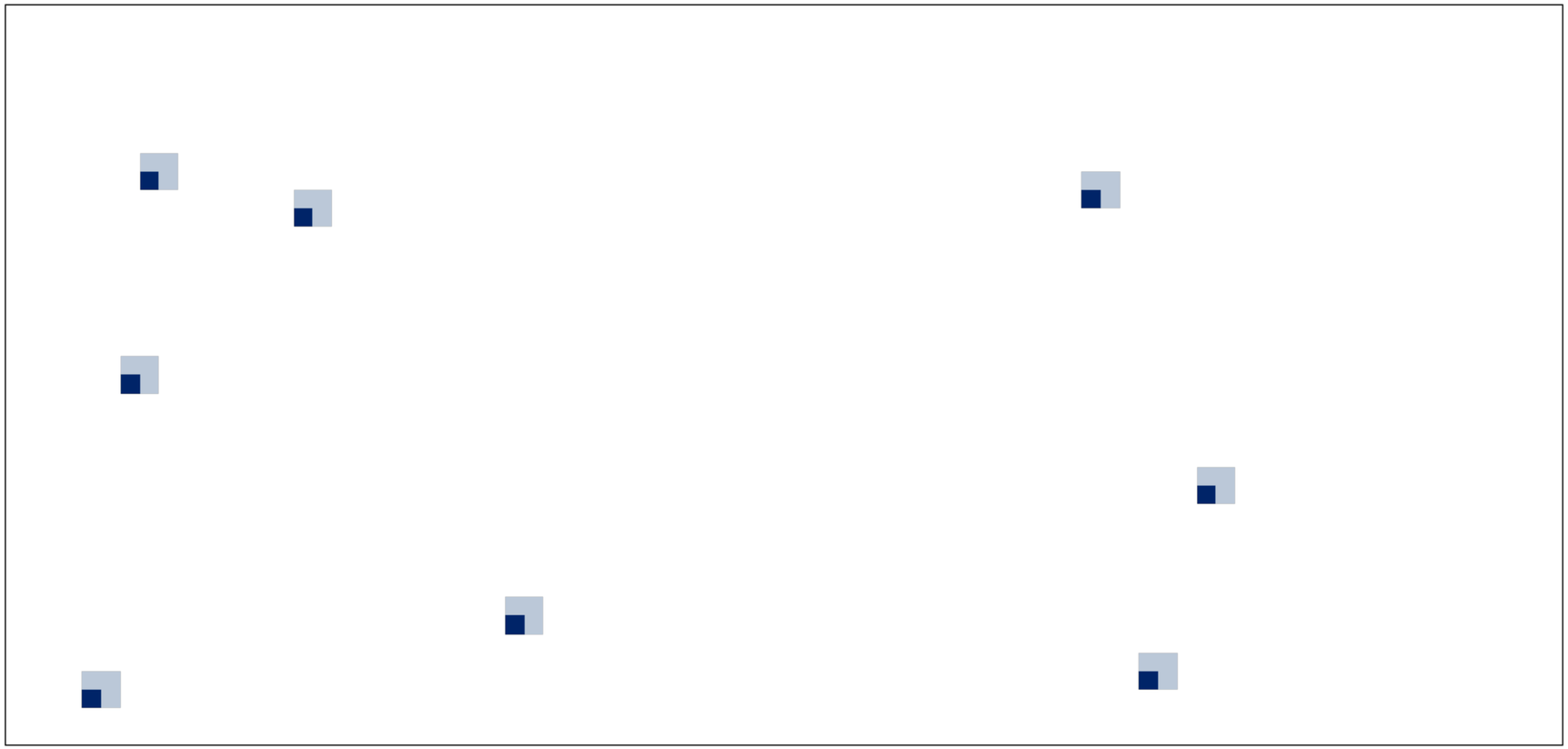
- Adaptive Feature Extraction
  - Can we automatically learn features from data?
    - And not look into perception for clues
- Principal Component Analysis
  - How, why, when, which

A dual goal

- Find a way to describe the data
  - The features part
- Reduce redundancy in the data
  - A side effect of “proper” features

Example case

- Describe this data

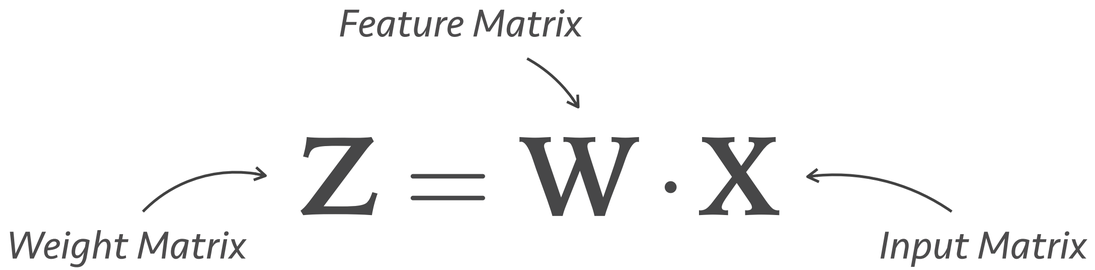


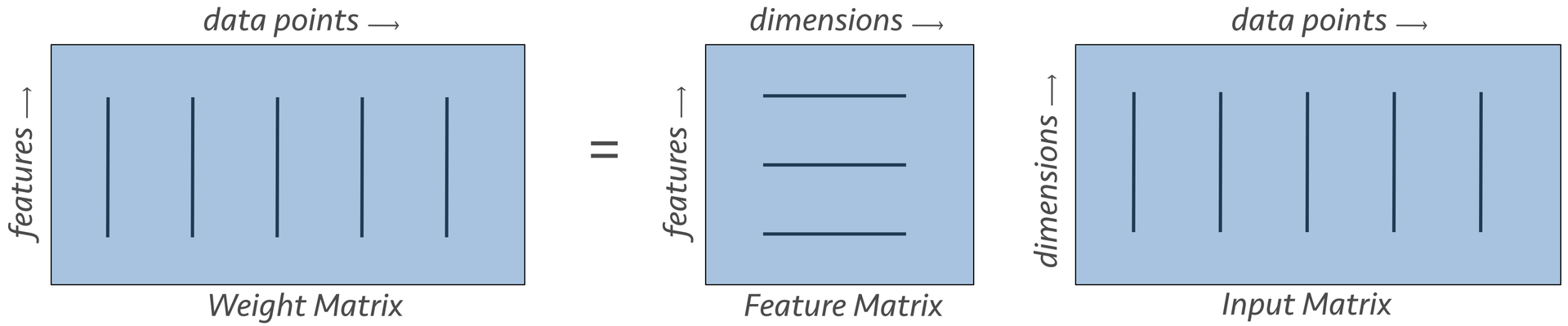
A “good feature”

- “Simplifies” the explanation of the input
  - Represents common patterns
  - Once defined, makes the input description simpler

- How do we define these abstract qualities?
  - On to the math …

Linear features

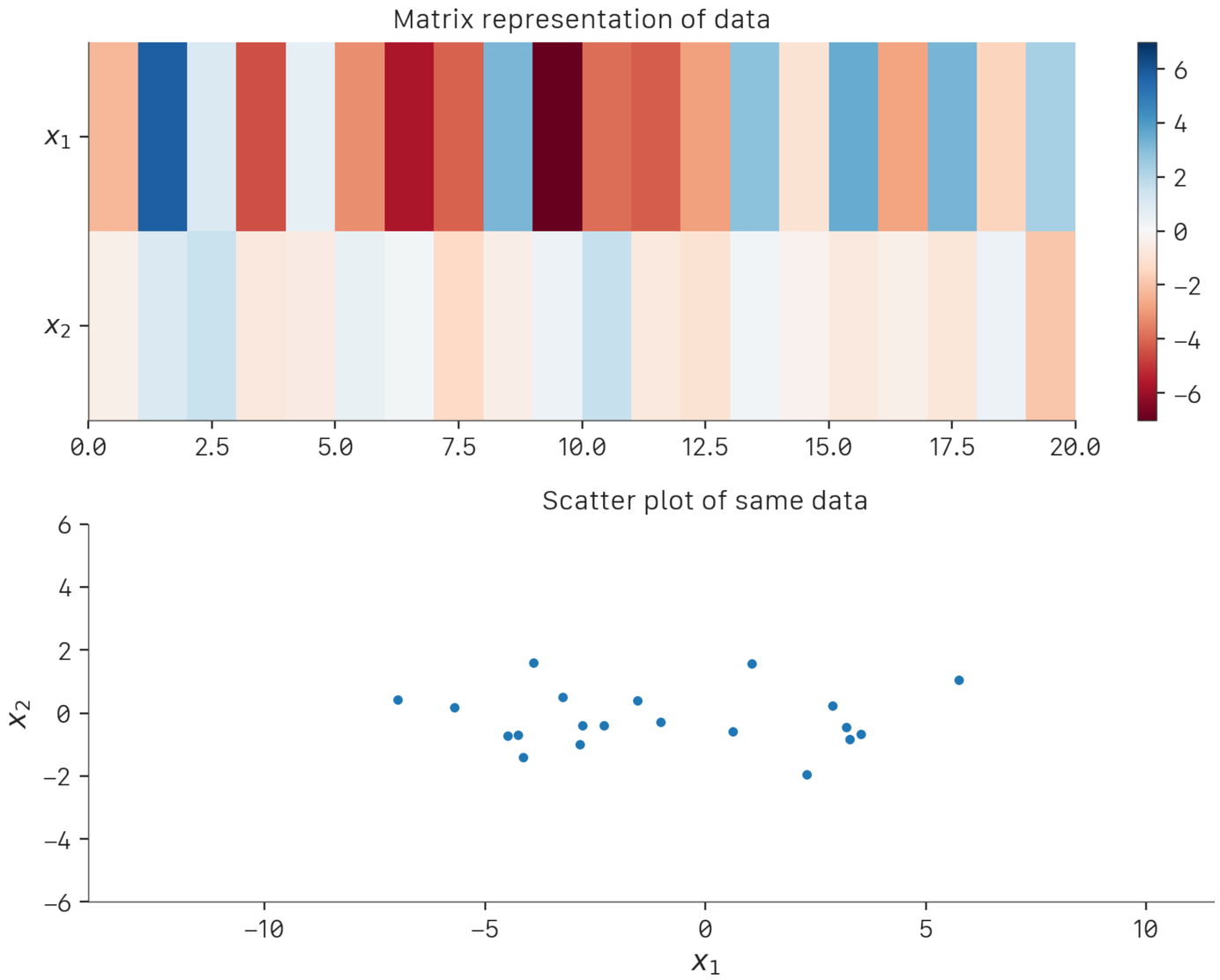




A 2D case

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

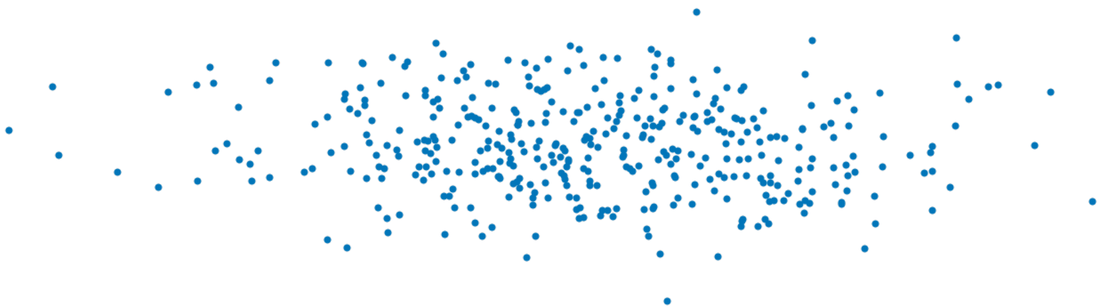
$$\begin{bmatrix} \mathbf{z}_1^\top \\ \mathbf{z}_2^\top \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \end{bmatrix}$$



* *remember, all vectors are column vectors in this class*
For now let's assume zero mean $\mathbf{X}$

Defining a goal

- Desirable feature features
  - Describe the input concisely
  - Avoid feature redundancy



- How do we define these?

One way to proceed

- Concise representation
  - Minimize similarity of the two weight dimensions
    - i.e. $\mathbf{z}_1$ and $\mathbf{z}_2$ should be “different”
- Avoiding feature redundancy
  - Same thing for features!
    - $\mathbf{w}_1$ and $\mathbf{w}_2$ should be “different”

$$\mathbf{Z} = \mathbf{W} \cdot \mathbf{X} = \begin{bmatrix} \mathbf{z}_1^\top \\ \mathbf{z}_2^\top \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \end{bmatrix}$$

One way to proceed

- Concise representation
  - Minimize similarity of the two weight dimensions
    - *Decorrelate*: $\mathbf{z}_1^\top \cdot \mathbf{z}_2 = 0$
- Avoiding feature redundancy
  - Same thing for features!
    - *Decorrelate*: $\mathbf{w}_1^\top \cdot \mathbf{w}_2 = 0$

$$\mathbf{Z} = \mathbf{W} \cdot \mathbf{X} = \begin{bmatrix} \mathbf{z}_1^\top \\ \mathbf{z}_2^\top \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \end{bmatrix}$$

Expressing it concisely

- Use the *covariance matrix*

Technically, we divide by $N-1$ but we simplify for readability

$$\text{cov}(\mathbf{Z}) = \begin{bmatrix} \mathbf{z}_1^\top \cdot \mathbf{z}_1 & \mathbf{z}_1^\top \cdot \mathbf{z}_2 \\ \mathbf{z}_2^\top \cdot \mathbf{z}_1 & \mathbf{z}_2^\top \cdot \mathbf{z}_2 \end{bmatrix} /N = \mathbf{Z} \cdot \mathbf{Z}^\top /N$$

  - $\{i, j\}$ element is the dot product of $\{i, j\}$-dimension pair
    - We are assuming zero-mean vectors $\mathbf{z}_i$
    - Represents all dimension combinations at once

Diagonalizing the covariance

- Diagonalizing the covariance suppresses co-activation across dimensions
  - if $\mathbf{z}_1$ is high, $\mathbf{z}_2$ won't be, etc.

$$\text{cov}(\mathbf{Z}) = \begin{bmatrix} \mathbf{z}_1^\top \cdot \mathbf{z}_1 & \mathbf{z}_1^\top \cdot \mathbf{z}_2 \\ \mathbf{z}_2^\top \cdot \mathbf{z}_1 & \mathbf{z}_2^\top \cdot \mathbf{z}_2 \end{bmatrix} / N = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \mathbf{I}$$

Problem definition

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

$$(\mathbf{W} \cdot \mathbf{X}) \cdot (\mathbf{W} \cdot \mathbf{X})^\top = N\,\mathbf{I} \Rightarrow \mathbf{Z} \cdot \mathbf{Z}^\top = N\,\mathbf{I}$$

How do we solve this?

- Any ideas?

$$(\mathbf{W} \cdot \mathbf{X}) \cdot (\mathbf{W} \cdot \mathbf{X})^\top = N\,\mathbf{I}$$

Solving for diagonalization

$$(\mathbf{W} \cdot \mathbf{X}) \cdot (\mathbf{W} \cdot \mathbf{X})^\top = N\,\mathbf{I} \Rightarrow$$

$$\Rightarrow \mathbf{W} \cdot \mathbf{X} \cdot \mathbf{X}^\top \cdot \mathbf{W}^\top = N\,\mathbf{I}$$

$$\Rightarrow \mathbf{W} \cdot \mathrm{cov}(\mathbf{X}) \cdot \mathbf{W}^\top = \mathbf{I}$$

Solving for diagonalization

- Covariance matrices are positive definite
  - And square and symmetric
  - Which makes them factorized by:

$$\mathbf{C} \cdot \mathbf{U} = \mathbf{U} \cdot \mathbf{\Lambda} \Rightarrow \mathbf{U}^\top \cdot \mathbf{C} \cdot \mathbf{U} = \mathbf{\Lambda}$$

  - Where $\mathbf{U}$ has the eigenvectors of $\mathbf{C}$ in its columns
  - $\mathbf{\Lambda} = \mathrm{diag}(\lambda_i)$, where λ_i are the eigenvalues of $\mathbf{C}$
    - Where λ_i will be real and positive-valued

Solving for diagonalization

Eigenanalysis *Problem to solve*

What we have: $\mathbf{U}^\top \cdot \mathbf{C} \cdot \mathbf{U} = \mathbf{\Lambda}$ $\mathbf{W} \cdot \mathrm{cov}(\mathbf{X}) \cdot \mathbf{W}^\top = \mathbf{I}$

Substitute $\mathbf{C} = \mathrm{cov}(\mathbf{X})$: $\mathbf{U}^\top \cdot \mathrm{cov}(\mathbf{X}) \cdot \mathbf{U} = \mathbf{\Lambda}, \quad \mathbf{U}, \mathbf{\Lambda} = \mathrm{eig}(\mathbf{X})$

Get the $\mathbf{I}$ *on the right:* $\mathbf{\Lambda}^{-1/2} \cdot \mathbf{U}^\top \cdot \mathrm{cov}(\mathbf{X}) \cdot \mathbf{U} \cdot \mathbf{\Lambda}^{-1/2} = \mathbf{I}$

Solve for $\mathbf{W}$ *:* $\mathbf{W} = \mathbf{\Lambda}^{-1/2} \cdot \mathbf{U}^\top = \begin{bmatrix} \sqrt{\lambda_1} & 0 \\ 0 & \sqrt{\lambda_2} \end{bmatrix}^{-1} \cdot \mathbf{U}^\top$

Solving for diagonalization

- The solution is a product of the eigenvectors $\mathbf{U}$ and the square root of the eigenvalues $\mathbf{\Lambda}$ of $\mathrm{cov}(\mathbf{X})$

$$\mathbf{W} \cdot \mathrm{cov}(\mathbf{X}) \cdot \mathbf{W}^\top = \mathbf{I} \Rightarrow$$

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

Another solution

- That was not the only solution

- Consider this one: $\mathbf{W} \cdot \mathbf{X} \cdot (\mathbf{W} \cdot \mathbf{X})^\top = N\,\mathbf{I} \Rightarrow$

$$\mathbf{W} \cdot \mathbf{X} \cdot \mathbf{X}^\top \cdot \mathbf{W}^\top = N\,\mathbf{I} \Rightarrow$$

$$\mathbf{W} = \sqrt{N}\left(\mathbf{X} \cdot \mathbf{X}^\top\right)^{-1/2}$$

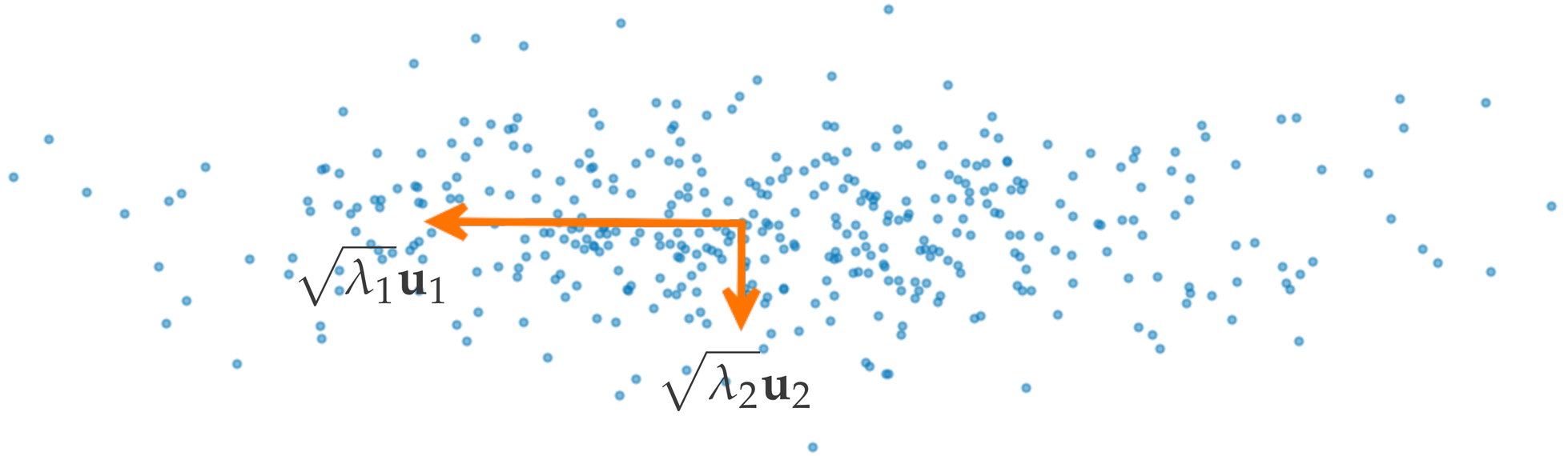
How do you take the matrix square root? $\longrightarrow$

$$\mathbf{W} = \mathbf{U} \cdot \mathbf{S}^{-1/2} \cdot \mathbf{V}^\top$$

$$[\mathbf{U}, \mathbf{S}, \mathbf{V}] = \mathrm{SVD}\left(\mathbf{X} \cdot \mathbf{X}^\top\right)$$

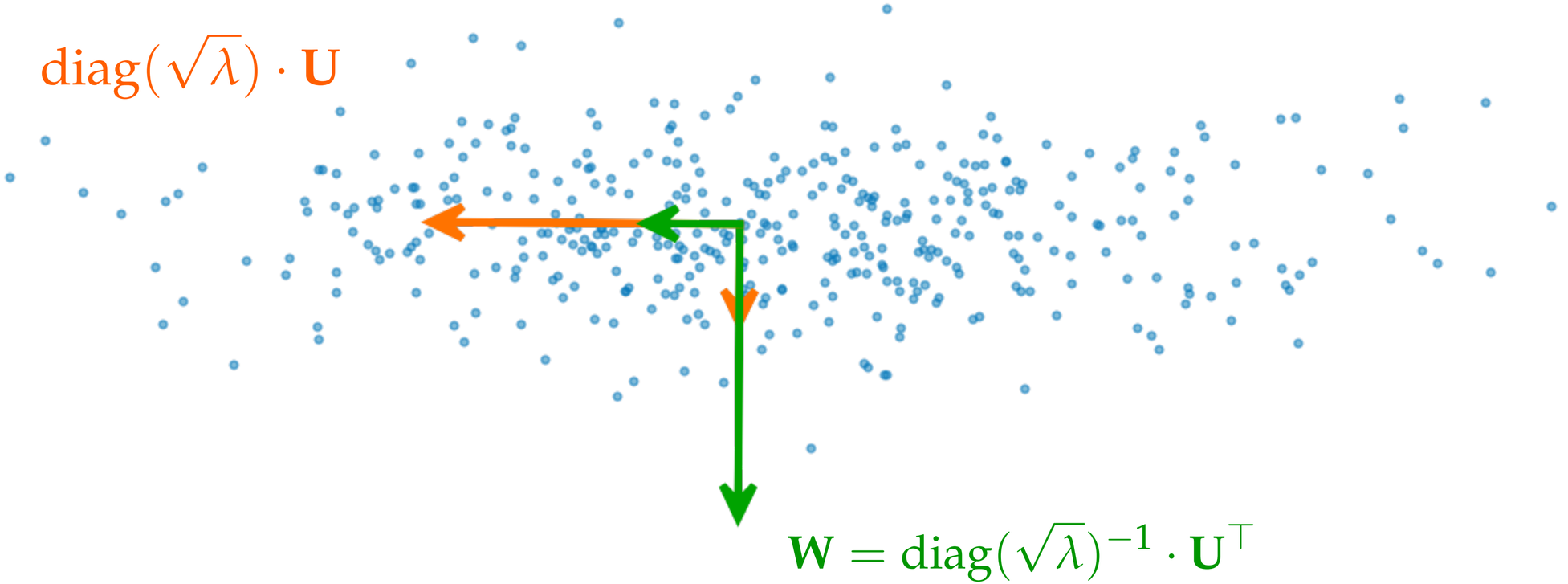
In pictures

- The eigenanalysis results are interpretable
  - Eigenvectors $\mathbf{u}_i$ show orthogonal directions of maximal variance
  - Eigenvalues λ_i show the variance of each direction



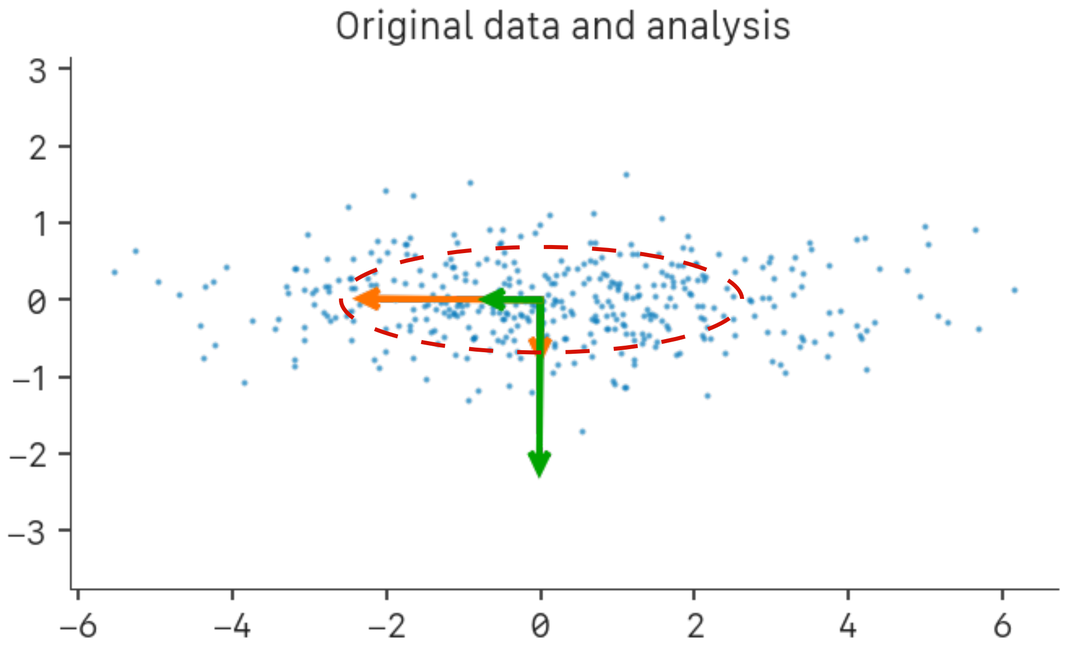
Decorrelation in pictures

- The decorrelating matrix $\mathbf{W}$ "undoes" these variances
  - Due to inversion, shrinks wide dimensions, expands narrow ones



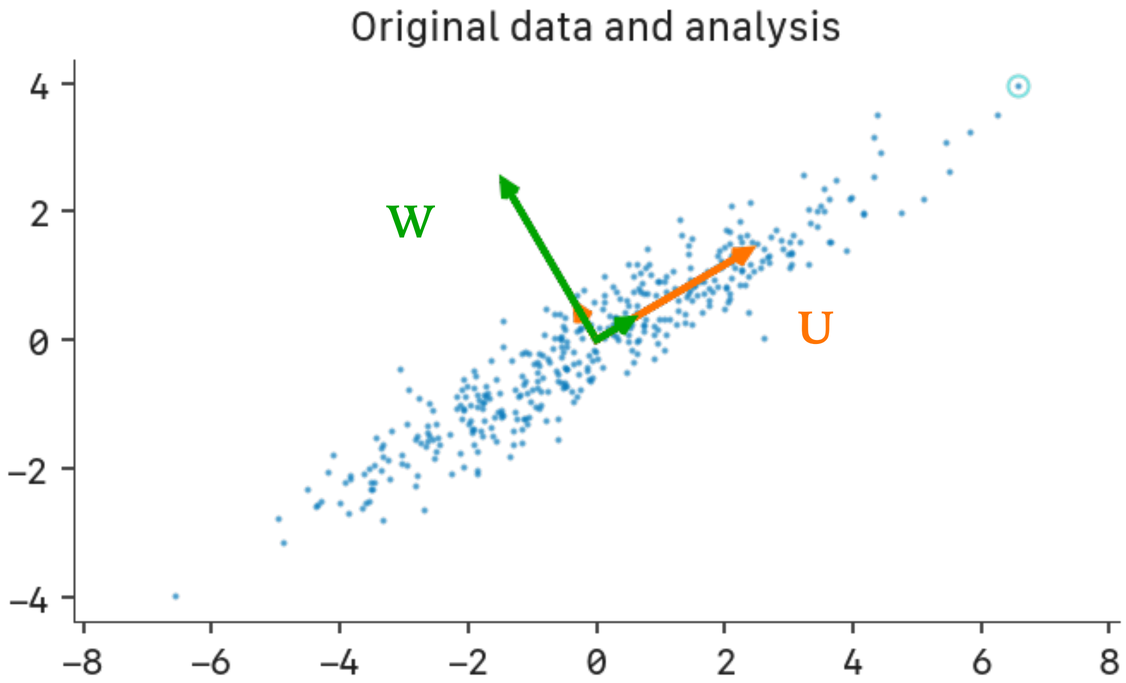
Resulting transformation

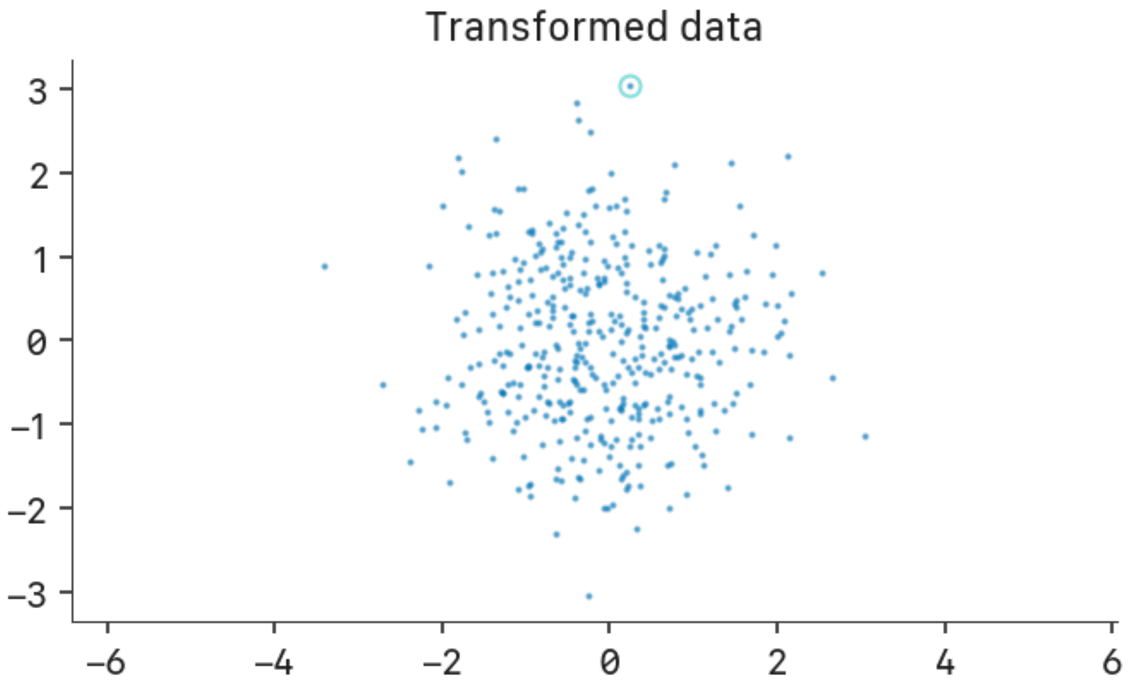
- Data get's scaled to have unit covariance



A more complex case

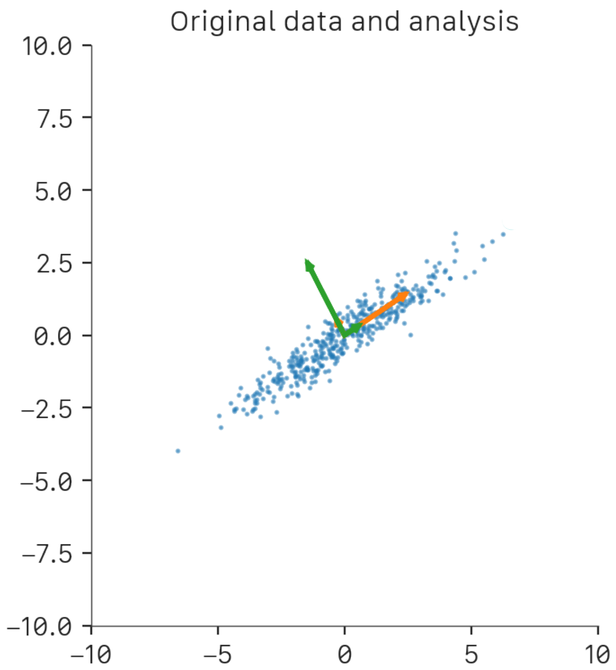
- Having correlation between two dimensions
  - We still find the directions of maximal variance
  - But now $\mathbf{W}$ rotates to line up with axes in addition to scaling

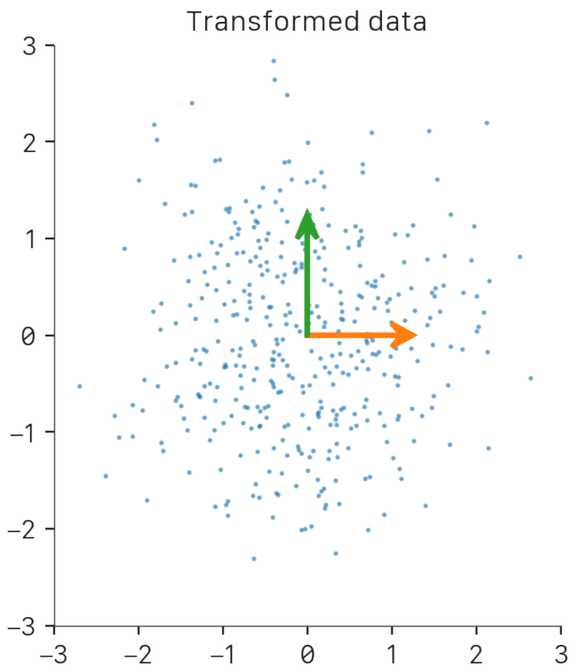


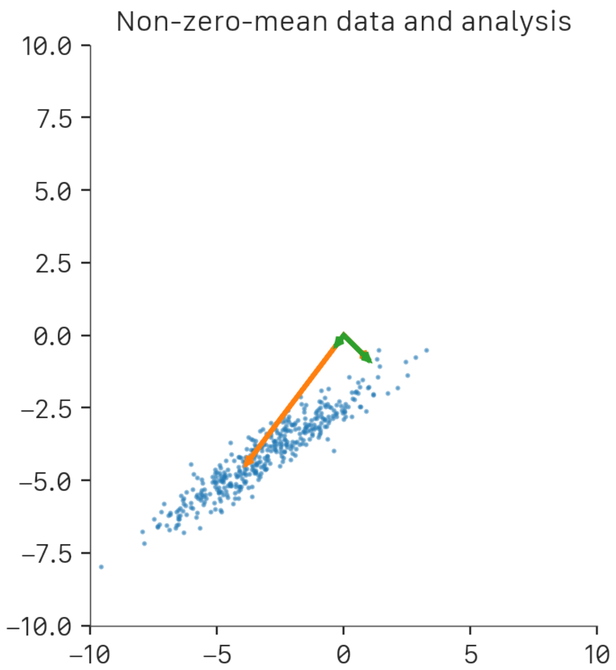


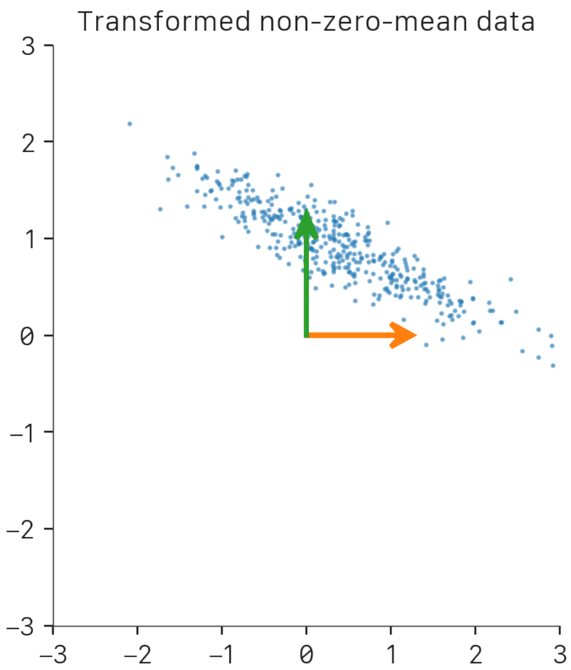
An important note

- These things only work when the inputs are zero mean!
  - We can only scale and rotate the data, not translate it!
  - Not a big deal, just remove the mean before doing this









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 across multivariate data
- Also known as the KLT
  - Karhunen-Loève Transform

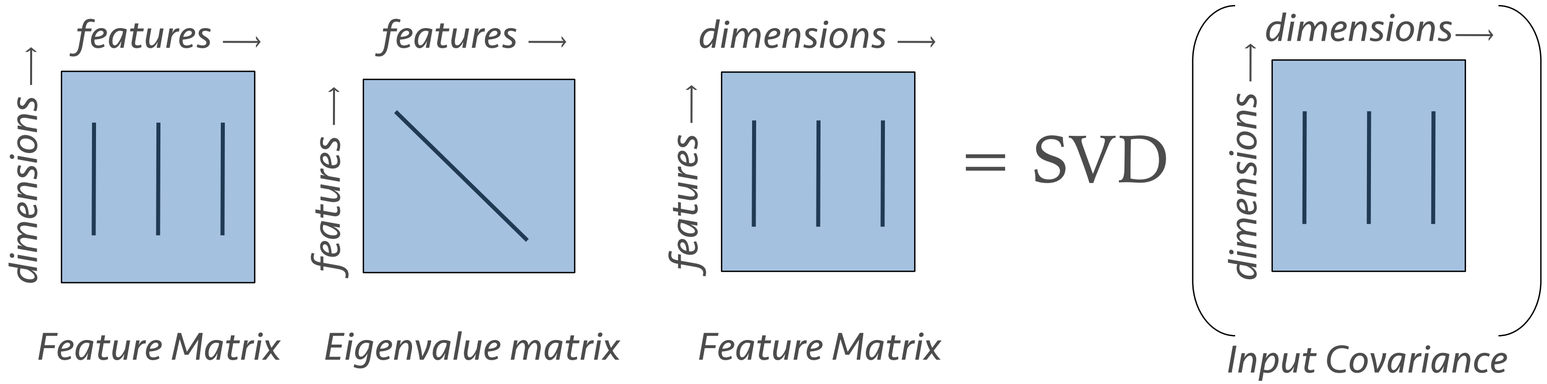
A better way to compute PCA

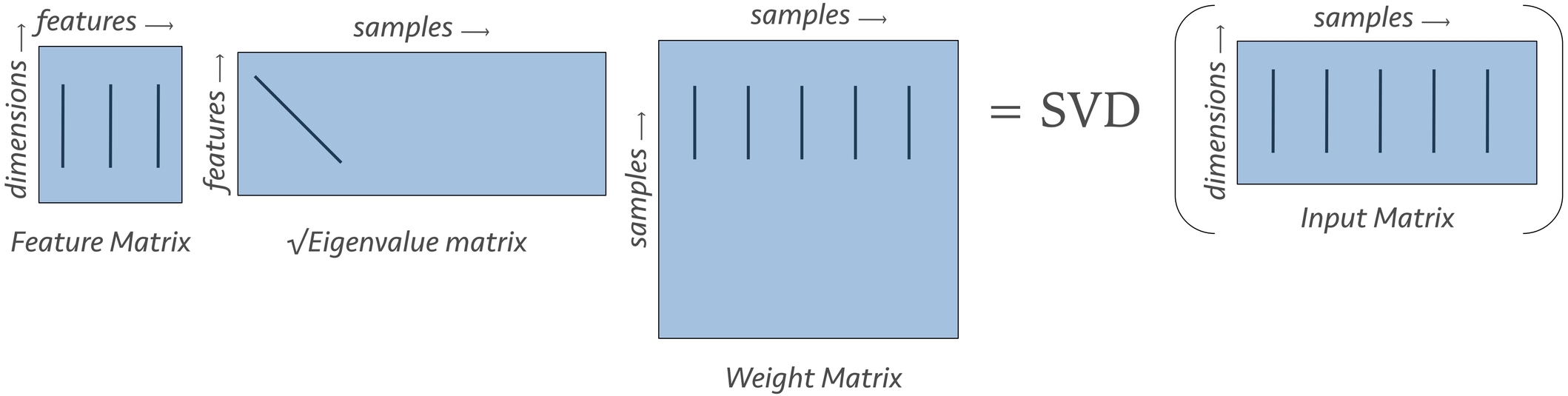
- The Singular Value Decomposition way

$$\mathbf{C} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^\top$$

- Relationship to eigendecomposition
  - In our case the input will be the $\mathbf{C}$ matrix
  - $\mathbf{U}$ and $\mathbf{S}$ will be like the eigenvectors/values of $\mathbf{C}$
- Why the SVD?
  - More stable, more robust, fancy extensions

PCA through the SVD





Dimensionality reduction

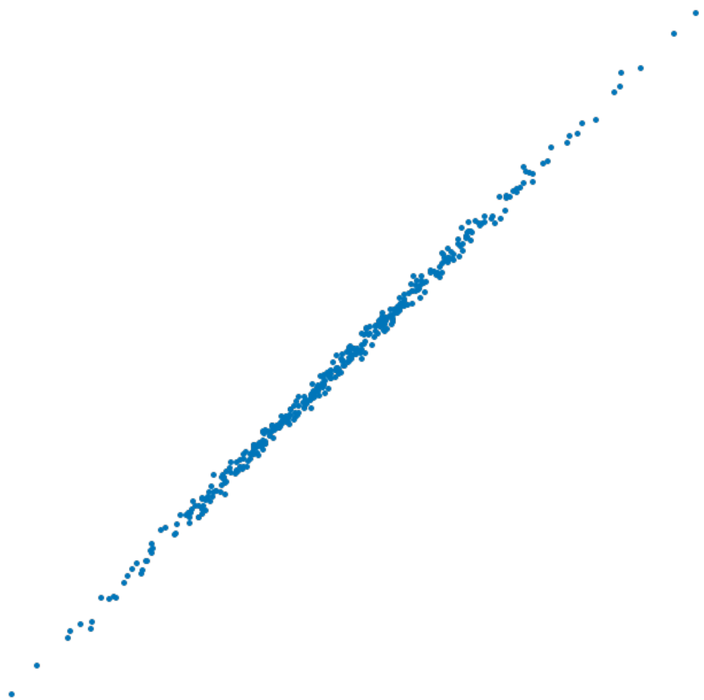
- PCA is great for high dimensional data
- Allows us to perform dimensionality reduction
  - Helps us find relevant structure in data
  - Helps us throw away things that won't matter

A simple example

- Two very correlated dimensions
  - e.g. size and weight of fruit
  - Effectively, one dimensional data
- Feature matrix is:

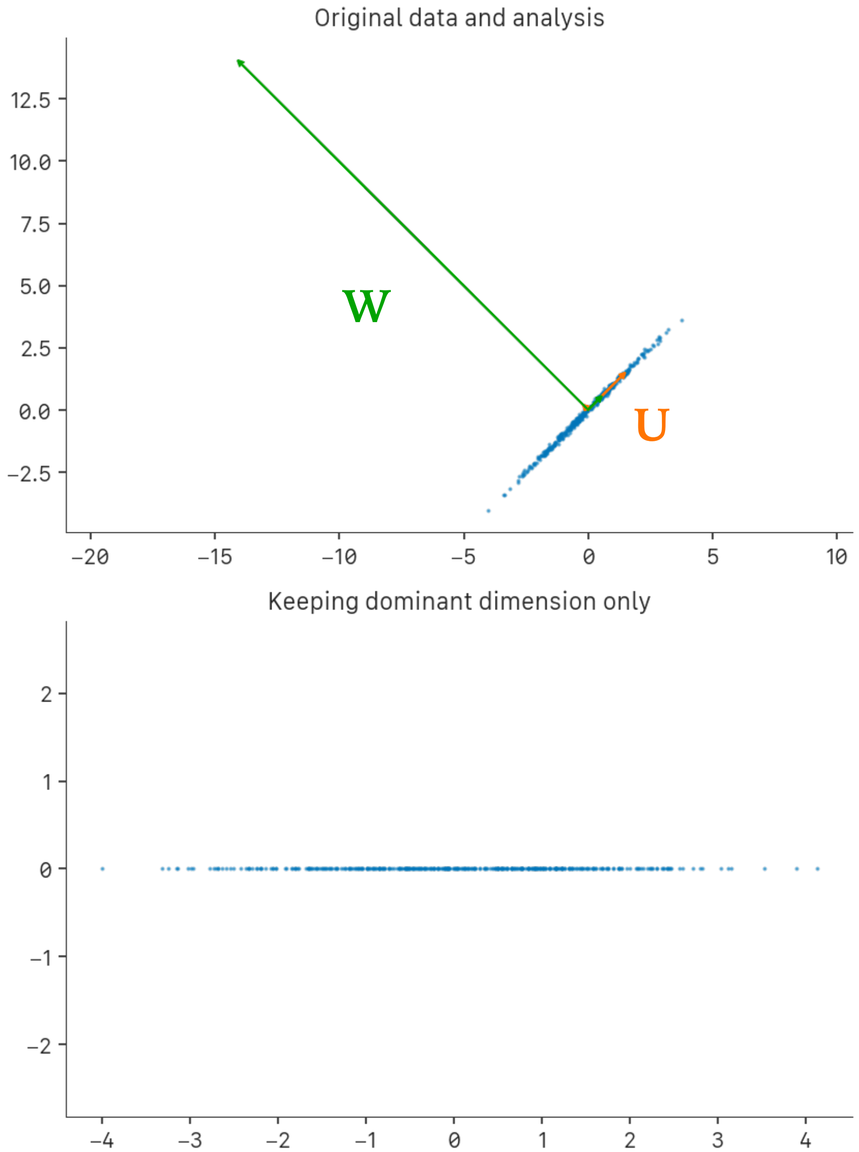
$$\mathbf{W} = \begin{bmatrix} -0.36 & -0.34 \\ -14.1 & 14.8 \end{bmatrix}$$

  - Large difference between the two rows
    - Implies that only one dimension is useful



Principal components reveal redundancy

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



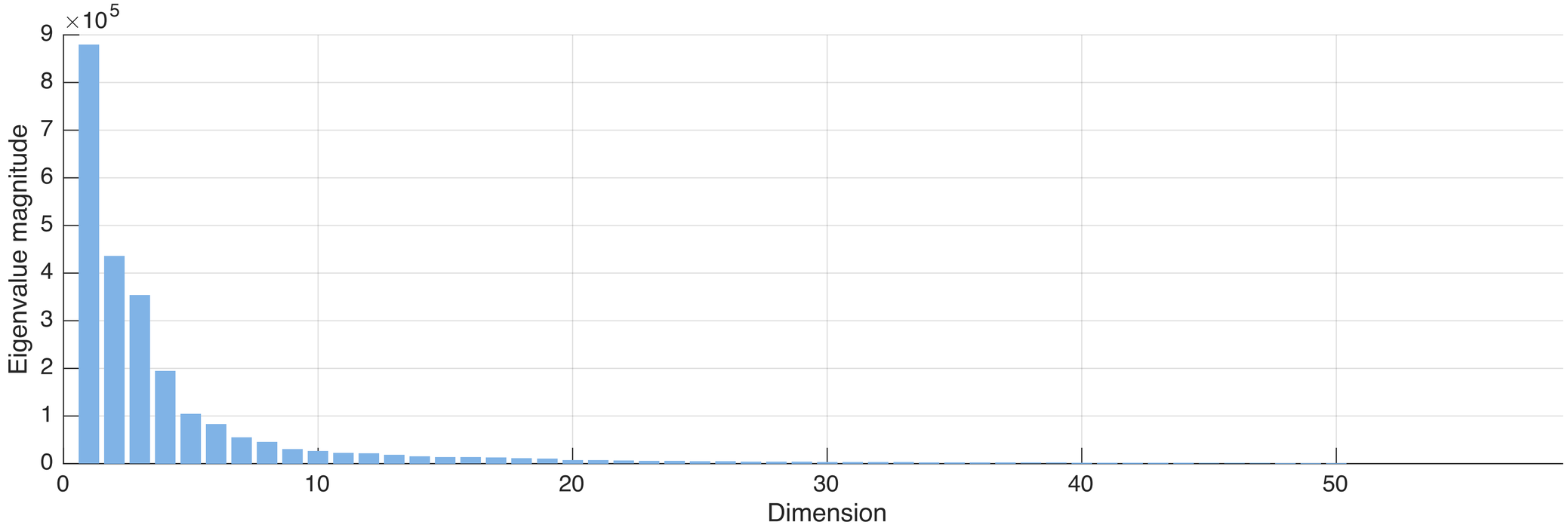
How many dimensions are enough?

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

- Look at the eigenvalues
  - They will show the variance of each component, at some point it will become negligible

Example

- Eigenvalues of 4800-dimensional video data
  - Insignificant variance after component 20
  - We don't need to keep dimensions after that



...

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?

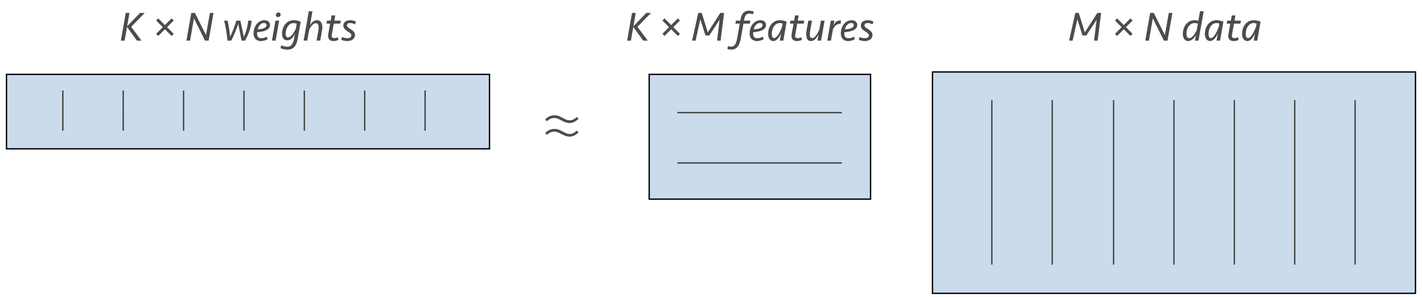
Face analysis

- 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 see some code in action

The Eigenfaces

Low-rank decompositions

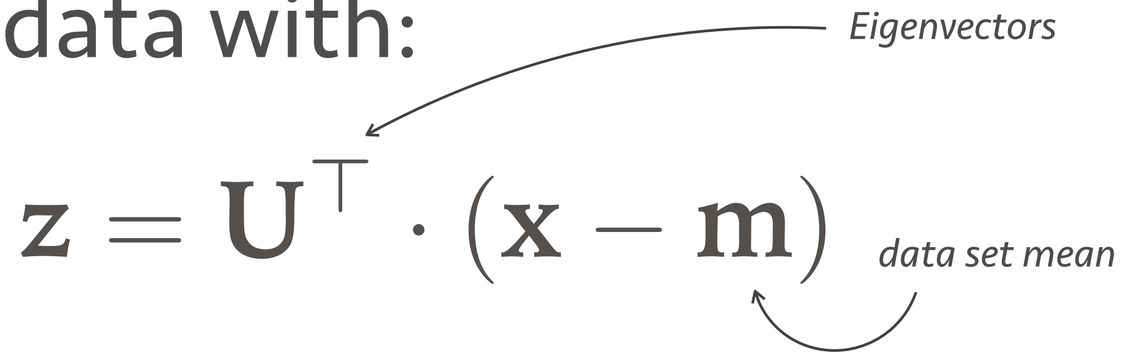
- Big data is painful to deal with
  - We want to reduce their quantity without losing information



- For many operations we will use the low-rank weights
  - A recurrent theme in machine learning

Low-rank projection with PCA

- Make low-rank data with:



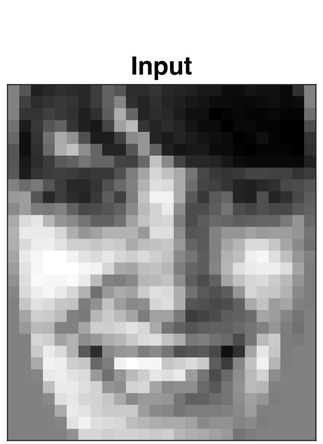
  - Why not use eigenvalues here? We do not want to amplify lesser info
    - Covariance stays diagonal, but the dimension variances are decreasing

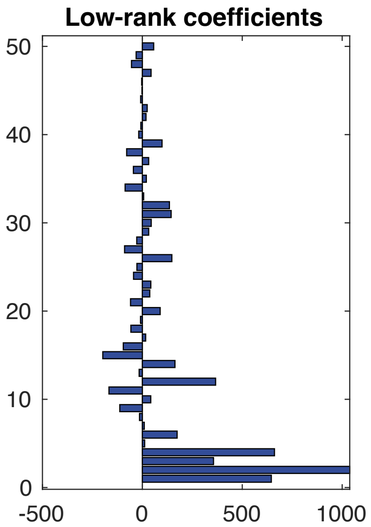
- Reconstruct data from low-rank representation with:

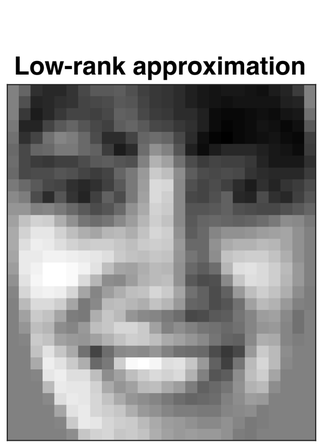
$$\hat{\mathbf{x}} = \mathbf{U} \cdot \mathbf{z} + \mathbf{m}$$

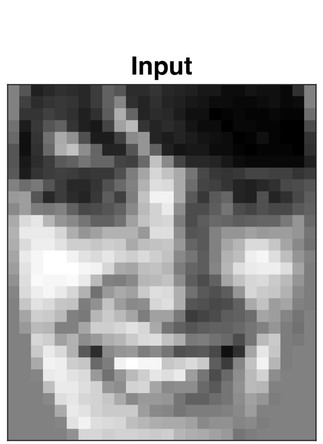
Low-rank advantage with face data

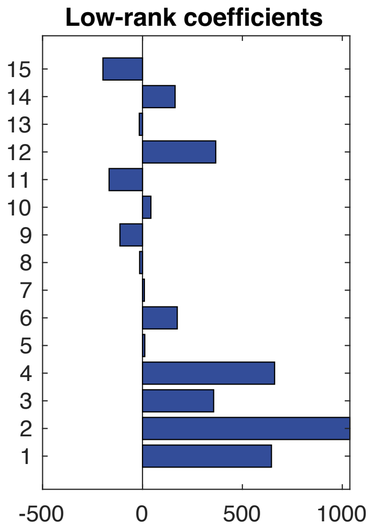
- Full data
  - 600 faces with 30 × 26 pixels = 600 × 780 = 468,000 values
- Low-rank version
  - K=50: 600 weights with 50 values + 50 × 780 bases = 69,000 values
  - K=15: 600 weights with 15 values + 15 × 780 bases = 11,700 values

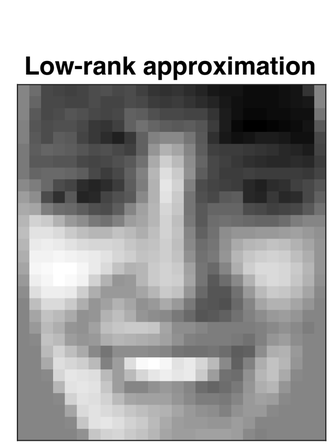












PCA for large dimensionalities

- Sometimes the data is very high dimensional
  - e.g. 720p videos: $1280 \times 720 \times T = 921{,}600\text{-D} \times T$ frames
- You will not do an SVD that big!
  - Complexity is $\mathcal{O}(m^2 n + n^3)$
- We can now use approximate methods
  - Faster, but sloppier

Approximate eig/SVD methods

- Rich literature on approximate methods
  - E.g. Lanczos, conjugate gradient, etc.
  - Much faster than regular eig/SVD
    - Especially for estimating a small number of components
- What to use:
  - Python: `scipy.sparse.linalg.eigs()`, `scipy.sparse.linalg.svds()`
    - For symmetric inputs use `scipy.sparse.linalg.eigsh()`
  - MATLAB: `eigs()` and/or `svds()`

EM-PCA

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

$$\mathbf{Z} = \mathbf{W}^{+} \cdot \mathbf{X}, \quad \mathbf{W} = \mathbf{X} \cdot \mathbf{Z}^{+}$$

  - After convergence $\mathbf{W}$ spans the PCA space
  - For low rank $\mathbf{W}$ the computations are much faster!
    - More later when we cover EM

PCA for online data

- Online estimation is also sometimes necessary
  - Irrespective of the dimensionality, we might have too much data
    - e.g. long video recordings
  - Or we might be analyzing a live stream

- Incremental SVD algorithms
  - Update the $\mathbf{U}$, $\mathbf{S}$, $\mathbf{V}$ matrices using only a small data subset or even a single sample point
  - Very efficient updates

PCA for online data II

- "Neural net" algorithms
  - Algorithms that update $\mathbf{W}$ one sample at a time
  - Gradient methods

- E.g. Sanger's rule:

$$\Delta\mathbf{W} = \mathbf{W} \cdot \mathbf{x}_i \cdot \left(\mathbf{x}_i - \mathbf{W}^\top \cdot \mathbf{W} \cdot \mathbf{x}_i\right)^\top$$

$$\mathbf{W} = \mathbf{W} + \mu\Delta\mathbf{W}$$

Back to signals and human perception

- We already talked about perceptual features
  - And how they correlate with classical DSP

- Is there a connection to make with PCA?

Finding the bases of signals

- Let's take a time series which is not "white"
  - Each sample is somehow correlated with the previous one (Markov process) $\left[x(t),\ \cdots,\ x(t+T)\right]$
- We'll make it a data set of N-long sound snippets

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

Which is something we've seen before

- In this context, features will be repeating temporal patterns at most N samples long

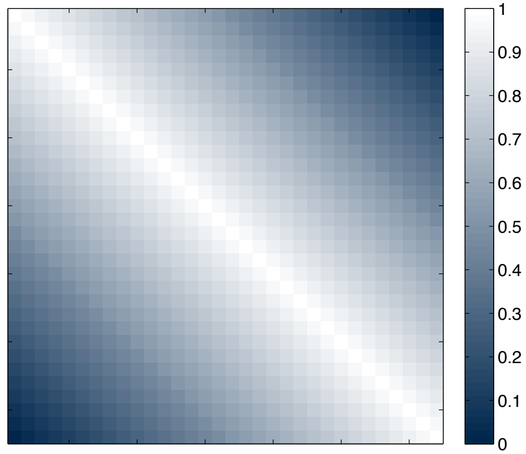
$$\mathbf{Z} = \mathbf{W} \cdot \begin{bmatrix} x(t) & x(t+1) & \\ \vdots & \vdots & \dots \\ x(t+N) & x(t+N+1) & \end{bmatrix}$$

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

PCA on the time series

- By our definition (Markov process) there is a correlation between successive samples:

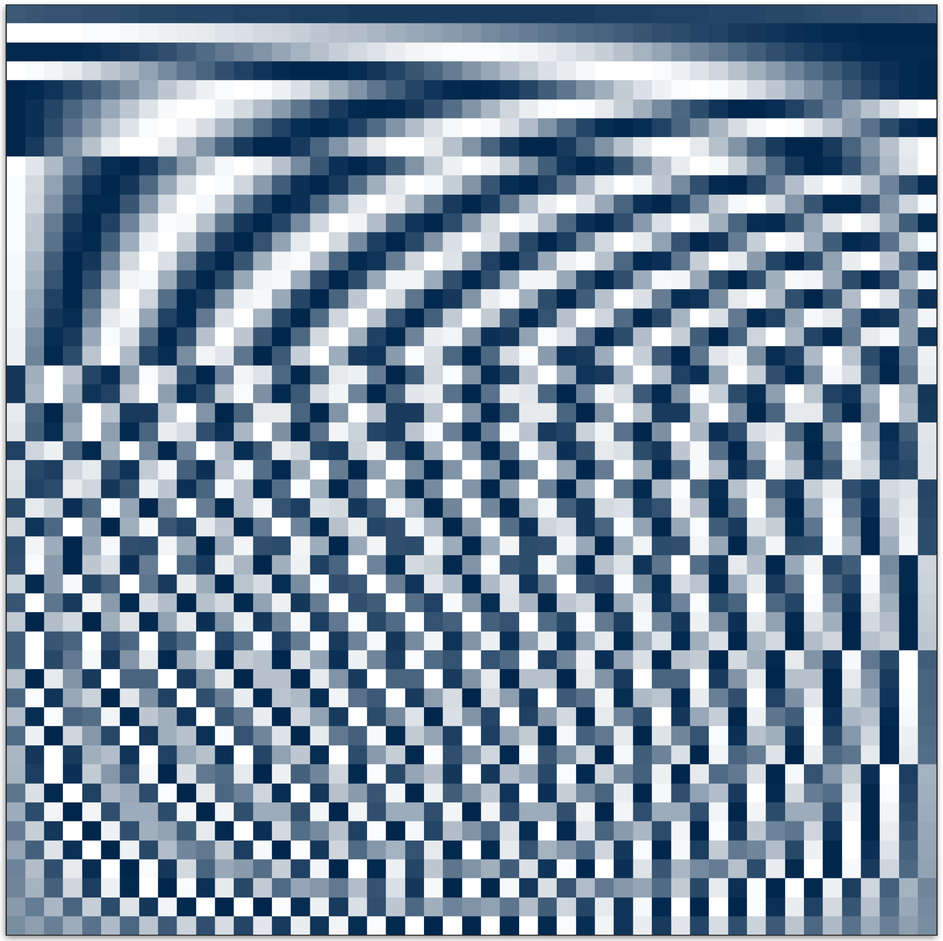
$$\mathrm{cov}(\mathbf{X}) \approx \begin{bmatrix} 1 & 1-e & \cdots & 0 \\ 1-e & 1 & 1-e & \vdots \\ \vdots & 1-e & 1 & 1-e \\ 0 & \cdots & 1-e & 1 \end{bmatrix} =$$



- Resulting covariance matrix will be symmetric Toeplitz, tapering from one towards zero

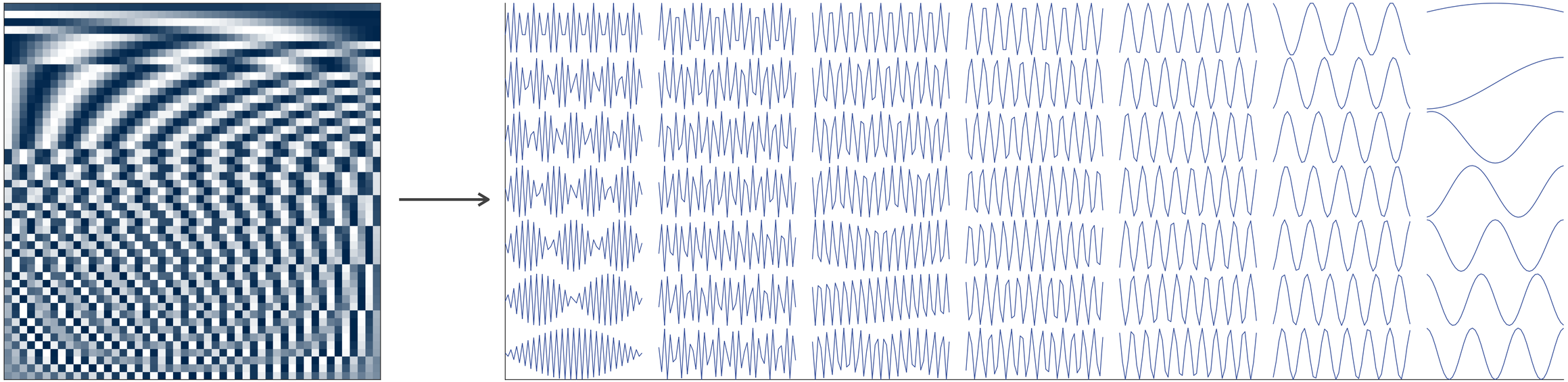
The resulting eigenvectors

- What does this look like? (approximately)



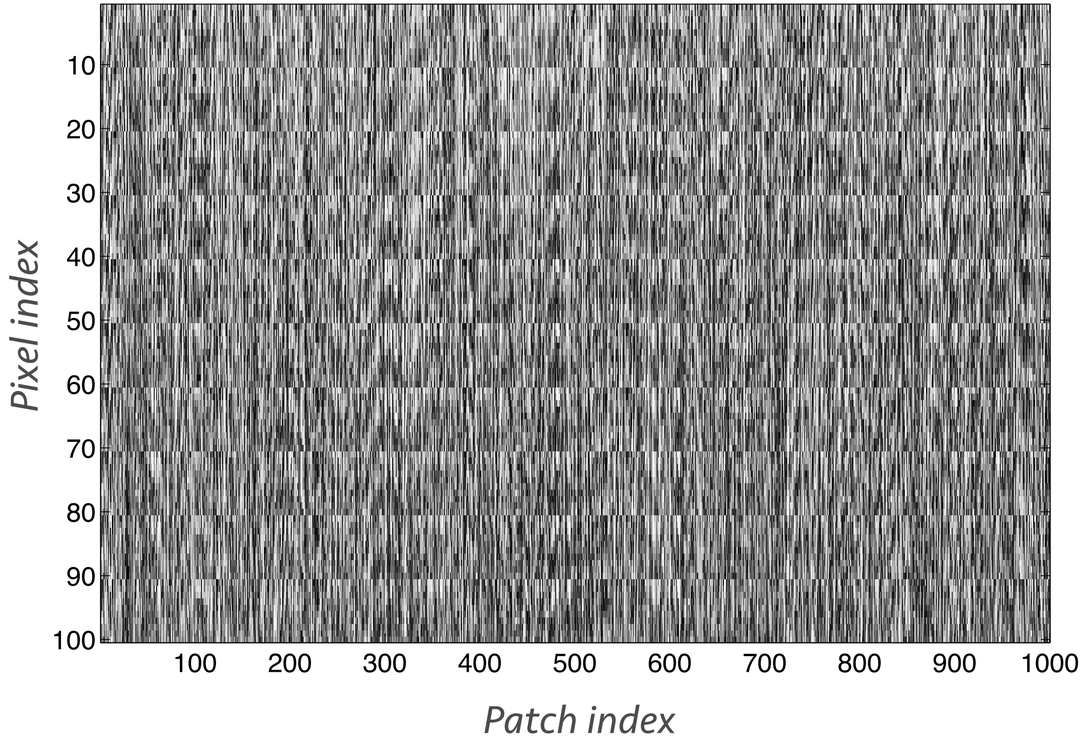
From PCA to a frequency transform

- The eigenvectors of Toeplitz matrices are (approximately) sinusoids of varying frequencies



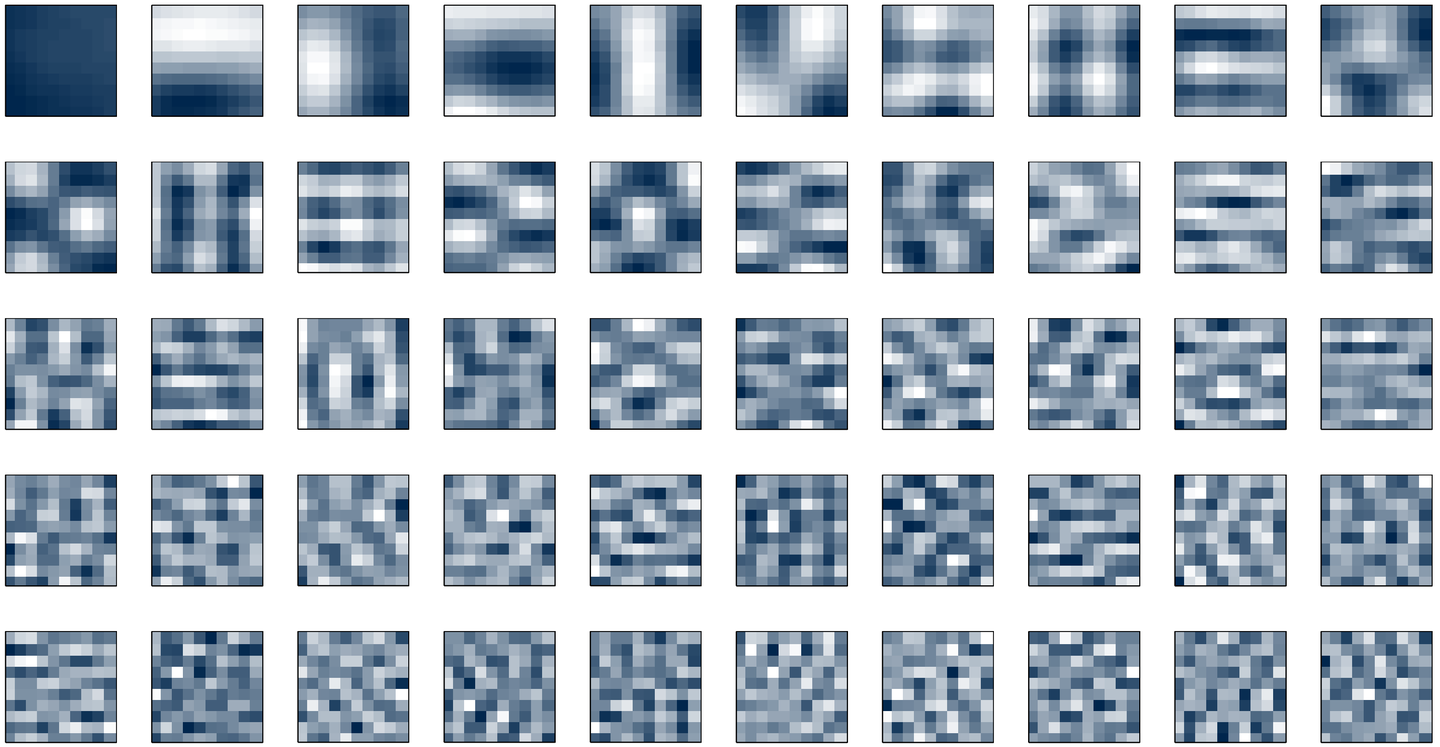
What about images?

- Take lots of pictures, pick random local patches
  - What are the features of this data set?



The components of image patches

- Do PCA on patch matrix and rewrap the components
  - What is this?



2D bases to 1D

- Remember the 2D Fourier transform?
  - We left/right multiply with the Fourier matrix

$$\mathbf{Y} = \mathbf{F} \cdot \mathbf{X} \cdot \mathbf{F}$$

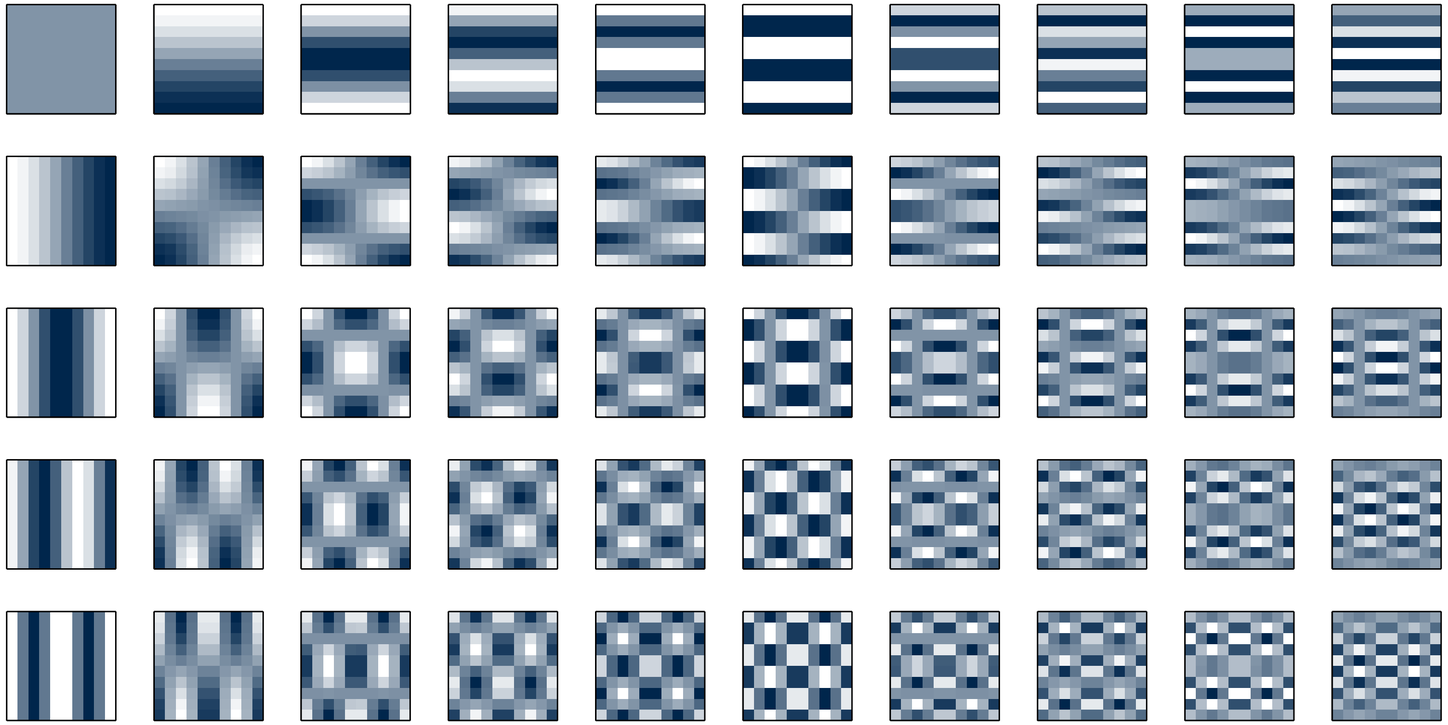
- Which in 1D translates to

$$\text{vec}(\mathbf{Y}) = \text{vec}(\mathbf{F} \cdot \mathbf{X} \cdot \mathbf{F}) = (\mathbf{F} \otimes \mathbf{F}) \cdot \text{vec}(\mathbf{X})$$

- So what do the features in that matrix look like?

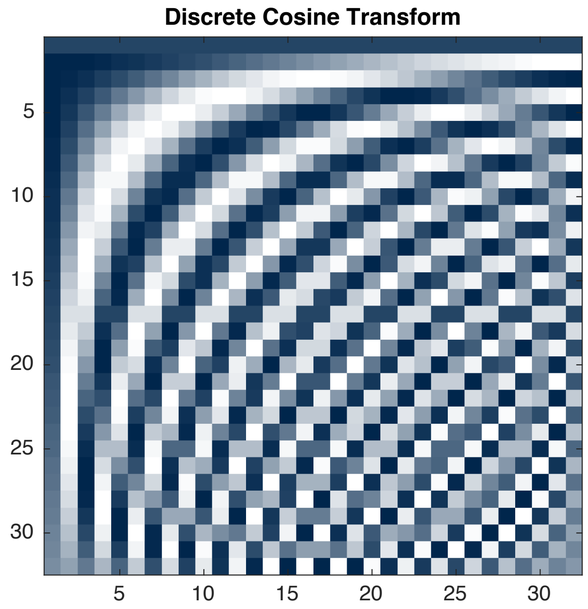
2D sinusoidal bases on 1D

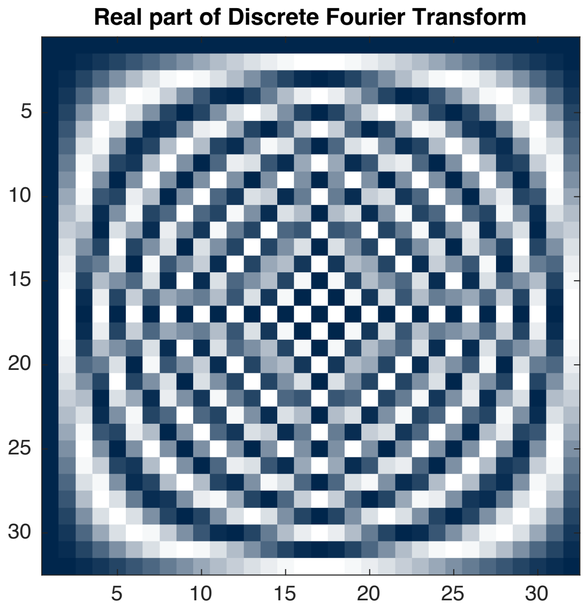
- Looks like the PCA components from images!



Not quite a Fourier transform, but close

- Real-valued bases are sinusoids of various frequencies
  - This is known as the Discrete Cosine Transform (DCT)
  - A real-valued variation of the DFT





So now you know

- A sinusoidal transform is a statistically “optimal” decomposition for time-correlated time series
  - In fact, you will often not do PCA and do a DCT
- There is also a connection with our perceptual system
  - E.g. employ similar filters in our ears
    - But it gets better later!

Comparing the DCT with PCA

- How to compress a picture
  - Chop into small pieces
  - Perform either:
    - DCT transform
    - PCA transform
  - Keep only large weights
  - Resynthesize using approximation

Comparing the DCT with PCA

Original image

DCT reconstruction, 0.016sec

PCA reconstruction, 0.9sec

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

Next lecture

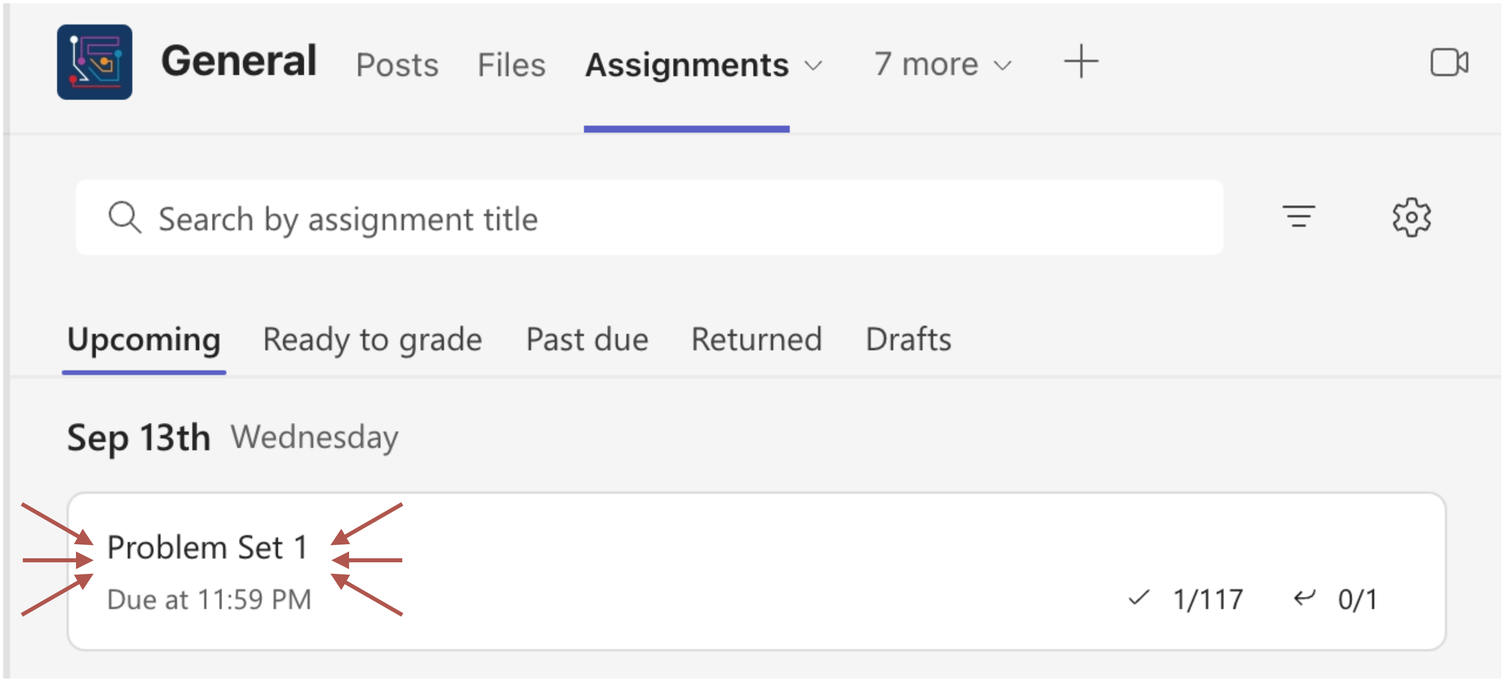
- Continuing with the features theme
- Independent Component Analysis
  - “PCA on steroids”
  - Strong statistical tool, very interesting properties and connections
- Other useful feature transforms

Reading

- Textbook sections 6.1-6.4
- Eigenfaces (optional)
  - 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 (optional)
  - http://www.merl.com/publications/TR2002-024/
- EM-PCA (optional)
  - http://cs.nyu.edu/~roweis/papers/empca.ps.gz

Reminder

- There is homework sent out already



- Talk to our TA for questions early!
  - Also if it feels like too much, consider giving up your spot for others