Linear Algebra: Part II

Math Tools: Week 2

Overview

1. Eigenvectors and eigenvalues
    1. Quick review of matrix-vector multiplication
    2. General applications

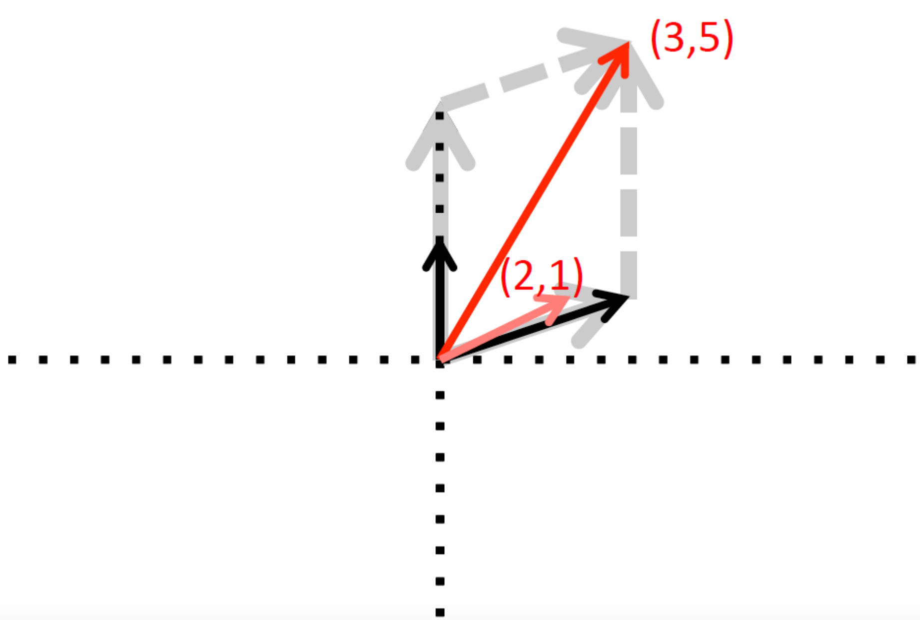
2. PCA
    1. Mathematical derivation
    2. Example

3. SVD
    1. Applications

Recall from last week: What do matrices do to vectors?

Outer product interpretation

$$\begin{bmatrix} 0 & 3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = 2 \begin{bmatrix} 0 \\ 2 \end{bmatrix} + 1 \begin{bmatrix} 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \end{bmatrix} + \begin{bmatrix} 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$



Columns of M 'span the plane'

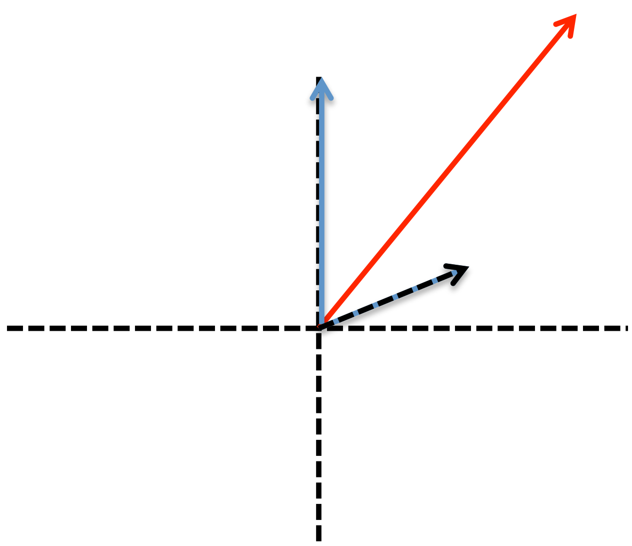
→ Different combinations of the columns of M can give you any vector in this plane

New vector is rotated and scaled

Recall from last week: What do matrices do to vectors?

Inner product interpretation

$$\begin{bmatrix} 0 & 3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 0 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} \\ \begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$



Rows of M also 'span the plane'

Inner product multiplication can be thought of as a shorthand way to do several inner products

New vector is rotated and scaled

Relating this to eigenvalues and eigenvectors

$$\overleftrightarrow{W}\,\overrightarrow{v} = \overrightarrow{u}$$

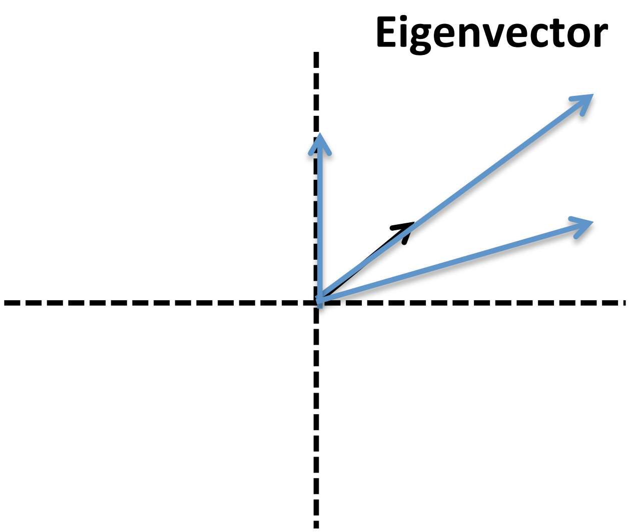
The matrix *W* transforms *v* into *u* – maps *v* onto *u.* Resulting vector can be a rotated and scaled version of *v*.

But it doesn't have to be rotated…

The special vectors for which the direction is preserved but the length changes: ***EIGENVECTOR***

An example:

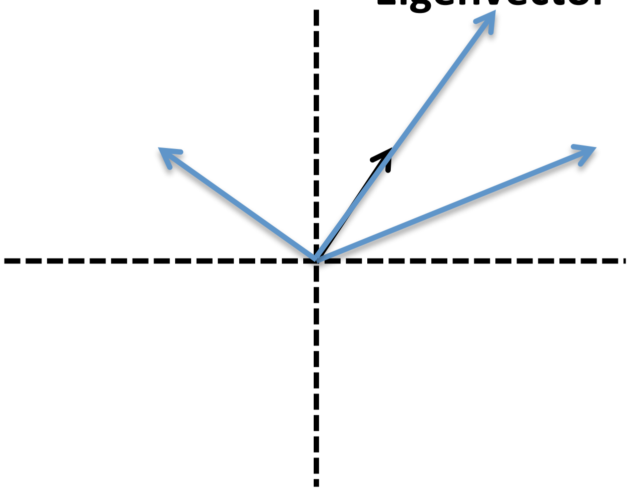
$$\begin{bmatrix} 0 & 3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1 \begin{bmatrix} 0 \\ 2 \end{bmatrix} + 1 \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$
$$= \begin{bmatrix} 3 \\ 3 \end{bmatrix} = 3 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$



Eigenvalues and eigenvectors

$$\begin{bmatrix} 0 & 3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 3 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\underset{\uparrow}{\overleftrightarrow{W}} \underset{\substack{\uparrow \\ \textbf{Eigenvector}}}{\overrightarrow{v}} = \underset{\substack{\uparrow \\ \textbf{Eigenvalue}}}{\lambda} \overrightarrow{v}$$



For an eigenvector, multiplication by the matrix is the same as multiplying the vector by a scalar. This scalar is called an eigenvalue.

How many eigenvectors can a matrix have???

→ An *n* x *n* matrix (the only type we will consider here) can have <u>up to</u> *n* distinct eigenvalues and *n* eigenvectors

→ The set of eigenvectors, with distinct eigenvalues, are linearly independent. In other words, they are orthogonal, or their dot product is 0.

Illustration of eigenvector and eigenvalues

Population dynamics of rabbits

$$x = \begin{bmatrix} \text{\# of rabbits b/w 0 and 1 year old} \\ \text{\# of rabbits b/w 1 and 2 years old} \\ \text{\# of rabbits b/w 2 and 3 years old} \end{bmatrix} \begin{bmatrix} 10 \\ 10 \\ 10 \end{bmatrix}$$

A = age transition matrix – probability that a member of the ith age class will become a member of the (i+1)th age class

$$\begin{bmatrix} 0 & 6 & 8 \\ 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \end{bmatrix}$$

Population growth/loss: $Ax_t = x_{t+1}$

$$\begin{bmatrix} 0 & 6 & 8 \\ 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \end{bmatrix} \begin{bmatrix} 10 \\ 10 \\ 10 \end{bmatrix} = \begin{bmatrix} 140 \\ 5 \\ 5 \end{bmatrix}$$

In this case, an eigenvector of matrix A represents a ‘stable’ population distribution

$$Ax_t = \lambda x_{t+1} \qquad x_t = x_{t+1}$$

If $\lambda = 1$, then there is a solution for which the population doesn’t change every year.

If $\lambda < 1$, then the population is shrinking

If $\lambda > 1$, then the population is growing

If the population starts on an eigenvector, it will stay on the eigenvector

This will be revisited in a few classes!

http://homepage.ntu.edu.tw/~jryanwang/course/Mathematics%20for%20Management%20(undergraduate%20level)/Applications%20in%20Ch7.pdf

Side-note on eigenvectors

Previous example

$$\overleftrightarrow{W}\,\overrightarrow{v} = \lambda \overrightarrow{v}$$

discusses RIGHT eigenvectors – the eigenvector is on the RIGHT side of the matrix.

Does this matter? Yes, maybe. If a matrix is symmetric, the right and left eigenvectors are the same.

Left eigenvectors can be found by solving this equation:

$$\overrightarrow{v}^T\overleftrightarrow{W} = \lambda \overrightarrow{v}$$

Note: This eigenvector is a row vector

Most of the time, left eigenvectors aren't that useful. But it's good to know that they exist.

How to find eigenvalues and eigenvectors

Real life and easiest way: Using MATLAB's eig command

[V,D] = eig(W) will return a matrix V, where each column is an eigenvector, and a diagonal matrix D with the corresponding eigenvalues along the diagonal

MATLAB output:

```
[V,D] = eig(W)

V =
   -0.8321   -0.7071
    0.5547   -0.7071

D =
    -2     0
     0     3
```

W = $\begin{bmatrix} 0 & 3 \\ 2 & 1 \end{bmatrix}$

Eigenvectors are defined only up to a scale factor, and so they are typically scaled such that the norm of the vector is 1

Eigenvalue is 3, like we found before!

The set of eigenvalues is called the *spectrum* of a matrix

How to find eigenvalues and eigenvectors

The math way:

Eigenvector-eigenvalue equation: $\overleftrightarrow{W}\,\overrightarrow{v} = \lambda \overrightarrow{v}$

$$\overleftrightarrow{W}\,\overrightarrow{v} - \lambda \overrightarrow{v} = 0$$

$$(\overleftrightarrow{W} - \lambda I)\overrightarrow{v} = 0$$

(WHITEBOARD) Assume that *v* is not 0, this equation only has a solution if:

$$det(\overleftrightarrow{W} - \lambda I) = 0$$

If $\overleftrightarrow{W} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ then $\overleftrightarrow{W} - \lambda I = \begin{bmatrix} a - \lambda & b \\ c & d - \lambda \end{bmatrix}$

$$\begin{aligned} det(\overleftrightarrow{W} - \lambda I) &= (a - \lambda)(d - \lambda) - bc \\ &= \lambda^2 - \lambda(a + d) + ad - bc \\ &= \lambda^2 - \lambda \mathrm{Trace} + \mathrm{Det} \end{aligned}$$

$$\lambda = \frac{\mathrm{Trace} \pm \sqrt{\mathrm{Trace}^2 - 4\mathrm{Det}}}{2}$$

Practice

Find eigenvectors and eigenvalues of this matrix:

$$W = \begin{bmatrix} 4 & -1 \\ 2 & 1 \end{bmatrix}$$

Some fun facts that actually come in handy

$$W = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Determinant of a matrix is equal to the product of eigenvalues

$$ad - bd = \lambda_1 \lambda_2$$

Trace of a matrix is equal to the sum of eigenvalues

$$a + d = \lambda_1 + \lambda_2$$

Eigenvalues of a triangular matrix can be read off the diagonal:

$$\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \qquad \lambda_1 = 3 \quad \lambda_2 = 2$$

When are eigenvectors/eigenvalues useful?

1. **Gives you a way to ‘decompose a matrix’**
2. Allows some easy shortcuts in computation
3. Give you a sense of what kind of ‘matrix’ or dynamics you are dealing with
4. Allows for a convenient change of basis
5. Frequently used in both modeling and data analysis

Eigen-decomposition (matrix diagonalization)

Eigen-decomposition, or the decomposition of a matrix into eigenvalues and eigenvectors, can be thought of as similar to prime factorization

Example with primes: 12 = 2*2*3

Example with matrices: $W = VDV^{-1}$

V is a matrix of eigenvectors, and D is a diagonal matrix of eigenvalues (WB)

Note: only works for NxN matrices with N linearly independent eigenvectors

Eigenvectors of a matrix form an orthogonal basis for the matrix – any other vector in the column space of the matrix can be re-written with eigenvectors:

General example:

$$\overrightarrow{v} = c_1\overrightarrow{v_1} + c_2\overrightarrow{v_2}$$

$$\overrightarrow{u} = W\overrightarrow{v}$$

$$\overrightarrow{u} = W(c_1\overrightarrow{v_1} + c_2\overrightarrow{v_2})$$

$$\overrightarrow{u} = c_1W\overrightarrow{v_1} + c_2W\overrightarrow{v_2} \qquad W\overrightarrow{v_1} = \lambda_1\overrightarrow{v_1}$$

Don't even need the matrix!

$$\overrightarrow{u} = c_1\lambda_1\overrightarrow{v_1} + c_2\lambda_2\overrightarrow{v_2}$$

When are eigenvectors/eigenvalues useful?

Eigenvalues can reveal the preferred inputs to a system

$$\overrightarrow{u} = W\overrightarrow{v} \qquad \overrightarrow{v} = c_1\overrightarrow{v_1} + c_2\overrightarrow{v_2} \qquad \overrightarrow{u} = c_1\lambda_1\overrightarrow{v_1} + c_2\lambda_2\overrightarrow{v_2}$$

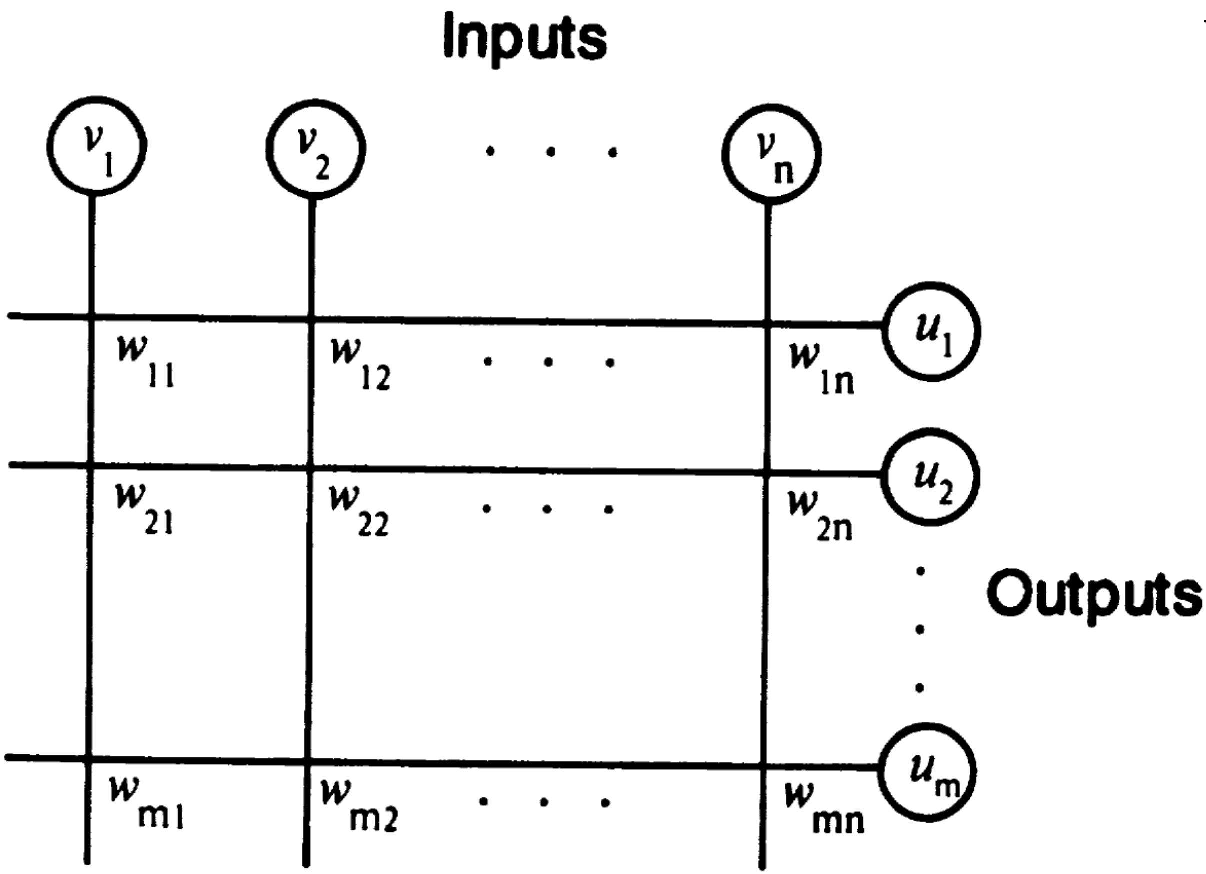
Assume that all *v*'s are of unit length, so the length of *u* depends on the *c*'s and λ's

If a vector v points in the direction of eigenvector v_1, then c_1 will be large (or at least positive)

If λ_1 is also large, then multiplication by *W* effectively lengthens the vector along this direction

More generally – the eigenvectors with the large eigenvalues can tell you which input vectors (*v*) the matrix "prefers", or will give a large response to

Connect this back to Neuroscience



$$\vec{u} = W\vec{v}$$

$$\vec{u} = c_1\lambda_1\vec{v_1} + c_2\lambda_2\vec{v_2}$$

Each output value u is computed by an inner product with the input vector *v* and the weight matrix row, *w*

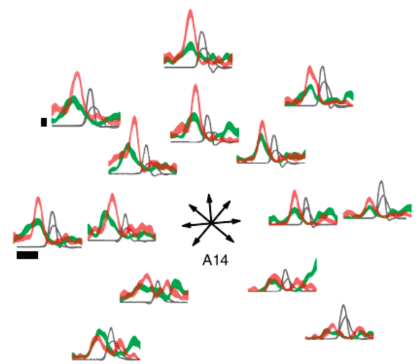
Large activation happens when the input vector closely matches eigenvectors with large eigenvalues

Also: Given *v* and *u,* you can compute the matrix *W* (well-defined if *u* is long enough)

Utility of eigenvectors/eigenvalues: PCA

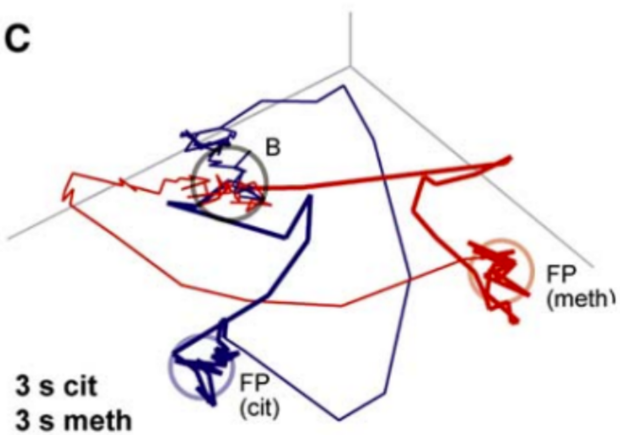
What is PCA?

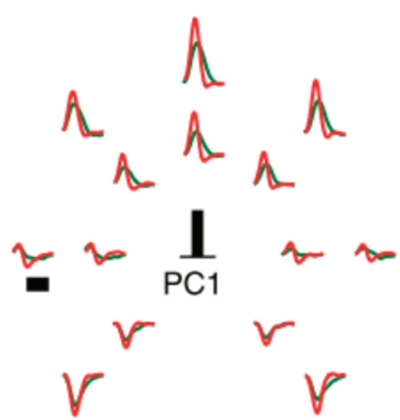
- Dimensionality reduction method (that keeps as much variation as possible)
- Pull structure out of seemingly complicated datasets
- De-noising technique
- Useful when # of variables that you record may be ‘excessive’



Neural data

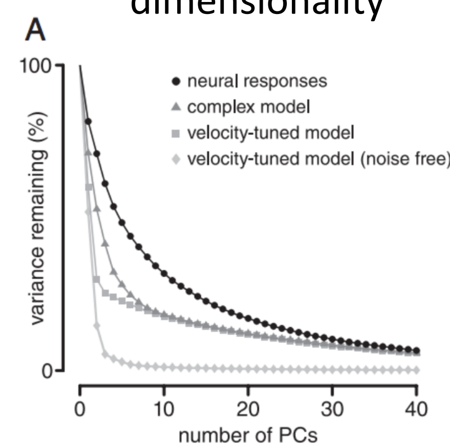
800 PN responding to 2 separate odors





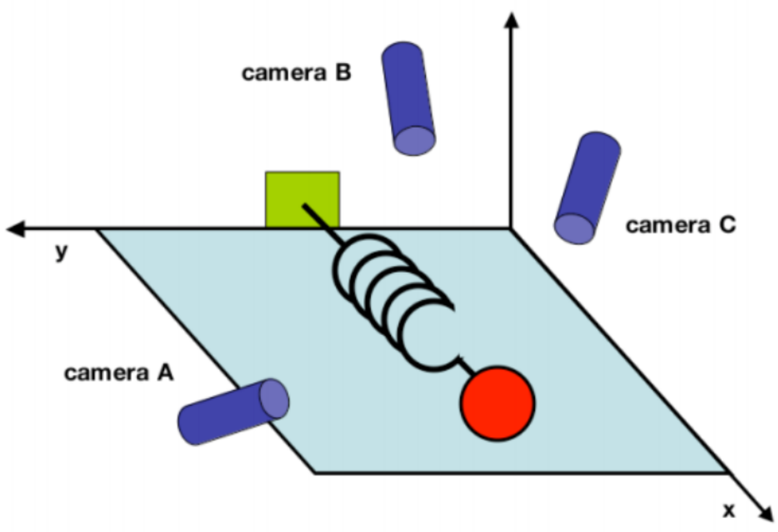
PC1

Analyze dimensionality



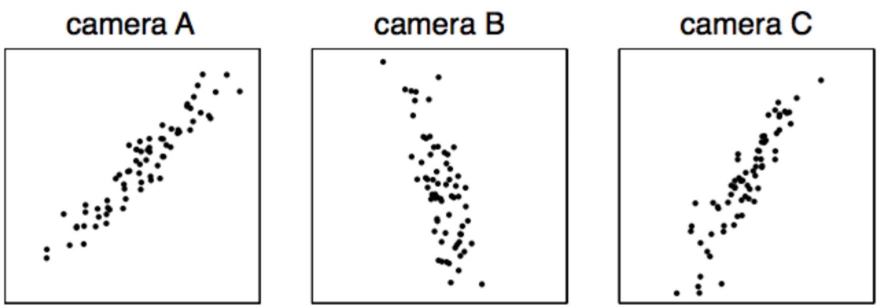
PCA: a motivating example

We do an experiment: we record a spring (with a mass attached at the end) oscillate in one dimension:



BUT, we don't know this – don't know which measurements best reflect the dynamics of our system

Record in 3 dimensions

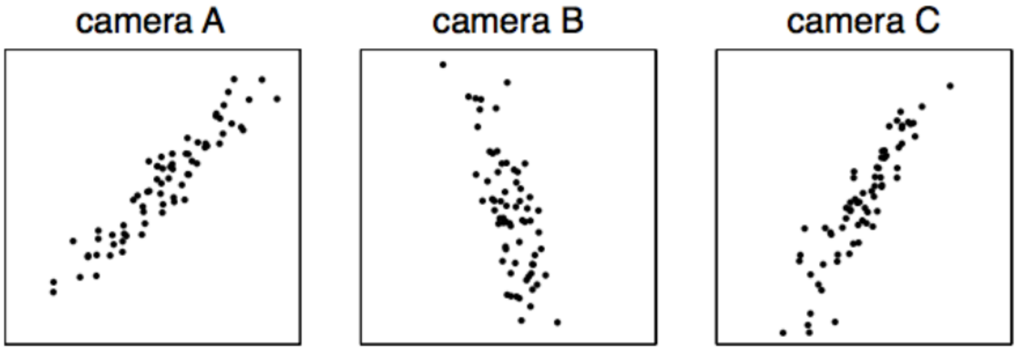


Each black dot – position of ball at each time frame

Can we determine the dimension along which the dynamics occur? Want to uncover that the dynamics are truly 1-dimensional

http://arxiv.org/pdf/1404.1100v1.pdf

PCA: a motivating example



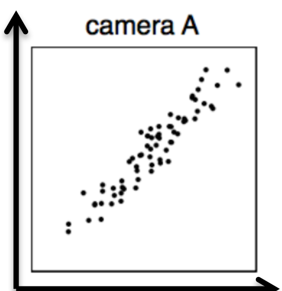
Data vector for 1 time point:

$$\vec{X} = \begin{bmatrix} x_A \\ y_A \\ x_B \\ y_B \\ x_C \\ y_C \end{bmatrix}$$

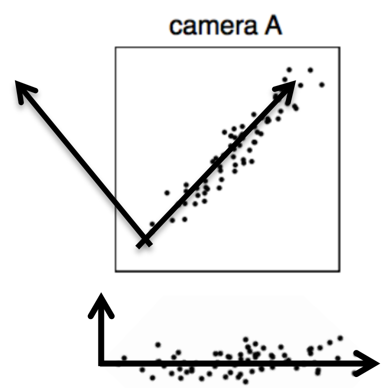
Together, data points form some cloud in a 6-dimensional space

6-d space has too many dimensions to visualize, and the data might not be truly 6-dimensional anyway

Is there a better way to represent this data?



Naïve basis for each camera: {(0,1), (1,0)}



Could choose a different basis that makes more sense for this data

Assumption/limit of PCA: new basis will be a linear combination of the old basis

PCA: into the details

Define the relevant matrices:

Data matrix: $\mathbf{X} = \begin{bmatrix} \overrightarrow{X_1} & \overrightarrow{X_2} & \cdots & \overrightarrow{X_T} \end{bmatrix}$

Each vector: $\overrightarrow{X} = \begin{bmatrix} x_A \\ y_A \\ x_B \\ y_B \\ x_C \\ y_C \end{bmatrix}$

If we record for 10 mins at 120 Hz, X has 6 rows and 10*60*120 = 72000 columns

To project the data into a more sensible basis: $\mathbf{PX} = \mathbf{Y}$

Projection matrix	Data matrix	New data matrix
6 x 6	6 x 72000	6 x 72000

P transforms **X** into **Y** through a rotation and scaling

The *rows* of **P** are the basis vectors for expressing the *columns* of **X** – they are the principal components!

$$\mathbf{PX} = \begin{bmatrix} \overrightarrow{p_1} \\ \overrightarrow{p_2} \\ \vdots \\ \overrightarrow{p_m} \end{bmatrix} \begin{bmatrix} \overrightarrow{X_1} & \overrightarrow{X_2} & \cdots & \overrightarrow{X_T} \end{bmatrix}$$

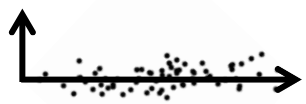
$$\mathbf{Y} = \begin{bmatrix} \overrightarrow{p_1} \cdot \overrightarrow{X_1} & \cdots & \overrightarrow{p_1} \cdot \overrightarrow{X_T} \\ \vdots & \ddots & \vdots \\ \overrightarrow{p_m} \cdot \overrightarrow{X_1} & \cdots & \overrightarrow{p_m} \cdot \overrightarrow{X_T} \end{bmatrix}$$

PCA: into the details

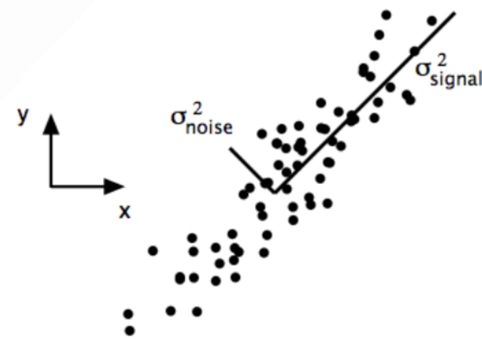
$$\mathbf{PX} = \mathbf{Y}$$

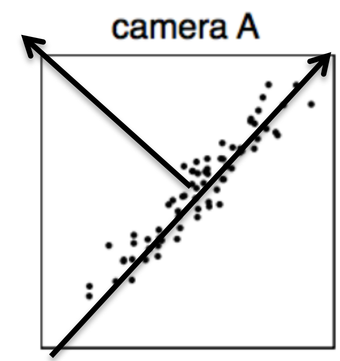
$$\mathbf{Y} = \begin{bmatrix} \overrightarrow{p_1} \cdot \overrightarrow{X_1} & \cdots & \overrightarrow{p_1} \cdot \overrightarrow{X_T} \\ \vdots & \ddots & \vdots \\ \overrightarrow{p_m} \cdot \overrightarrow{X_1} & \cdots & \overrightarrow{p_m} \cdot \overrightarrow{X_T} \end{bmatrix}$$

Okay… so how do we choose P?
What do we want?



Assume that the direction with the largest variance comes from real dynamics





Look for directions that capture redundancy – might allow us to reduce the dimensions

Want **Y** to have low redundancy

PCA: into the details

Can quantify these goals through the covariance matrix!

For two vectors:

Variance: $\sigma_A^2 = \frac{1}{n}\sum_i a_i^2$ $\qquad \sigma_B^2 = \frac{1}{n}\sum_i b_i^2$ (assume the data is z-scored)

Covariance: $\sigma_{AB}^2 = \frac{1}{n}\sum_i a_i b_i$ Rewriting using vector notation:

$$a = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix}$$

$$b = \begin{bmatrix} b_1 & b_2 & \cdots & b_n \end{bmatrix}$$

$$\sigma_{AB}^2 = \frac{1}{n} a b^T$$

For lotsa vectors:

$$Z = \begin{bmatrix} \overrightarrow{z_1} \\ \overrightarrow{z_2} \\ \vdots \\ \overrightarrow{z_n} \end{bmatrix}$$

Each row corresponds to measurements of a certain type (from one camera)

$$C_Z = \frac{1}{n} Z Z^T$$

Diagonal elements correspond to variance and off-diagonal elements correspond to covariance between measurement types

PCA: into the details

$$C_Z = \frac{1}{n} Z Z^T$$

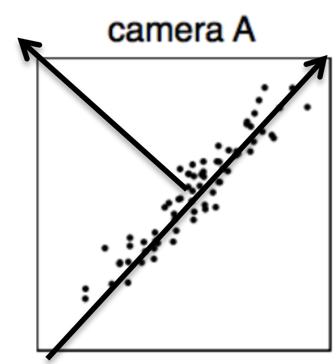
Diagonal elements correspond to variance within measurement types and off-diagonal elements correspond to covariance between measurement types

Want the covariance matrix for our projected data (***Y***) to have large values along the diagonal (large variance for the signal) and small values everywhere else (low covariance – low redundancy)

Technical term for this: diagonalize $\boldsymbol{C_Y}$

One algorithm that accomplishes this:

1. Select a normalized direction in *m*-dimensional space along which the variance is maximized. This is p_1.
2. Find an orthogonal direction along which remaining variance is maximized. This is p_2.
3. Repeat until *m* vectors are chosen.



PCA: eigenvector-based solution

Goal: Find some orthonormal matrix ***P*** for ***Y*** = ***PX*** such that $\boldsymbol{C_Y}$ = (1\n)$\boldsymbol{YY^T}$ is a diagonal matrix

$$
\begin{aligned}
C_Y &= \frac{1}{n} Y Y^T \\
&= \frac{1}{n}(PX)(PX)^T \\
&= \frac{1}{n} P X X^T P^T \qquad C_x = \frac{1}{n} X X^T \\
&= P C_X P^T
\end{aligned}
$$

$\boldsymbol{C_X}$ is symmetric, so we can decompose it into the following: $C_X = EDE^T$

E is a matrix of eigenvectors and *D* is a diagonal matrix with eigenvalues along the diagonal

$$C_Y = P(EDE^T)P^T$$

What if we choose P = E^T?

$$C_Y = P(P^T D P)P^T$$

Because P is a matrix of orthogonal eigenvectors, $P^T = P^{-1}$

$$C_Y = (PP^{-1})D(PP^{-1})$$

$$C_Y = D$$

Our choice of P diagonalized C_Y!

PCA: summary and utility

What just happened:

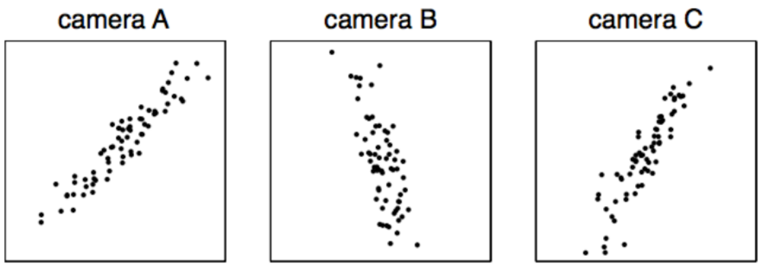
If we choose P to be the matrix of eigenvectors of the covariance matrix of X, then Y = PX returns a Y matrix whose covariance matrix is diagonalized.

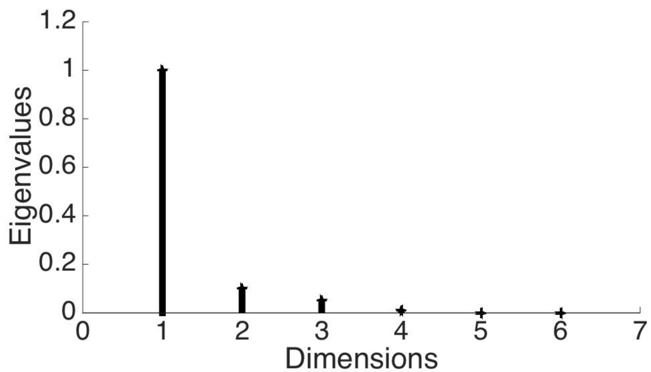
→ The principal components (or 'axes') are the eigenvectors of the covariance matrix of the data matrix X

Now what???

People usually use PCA to do 2 things:

1. Replot the data using Y





2. Look at the spectrum (the eigenvalues)

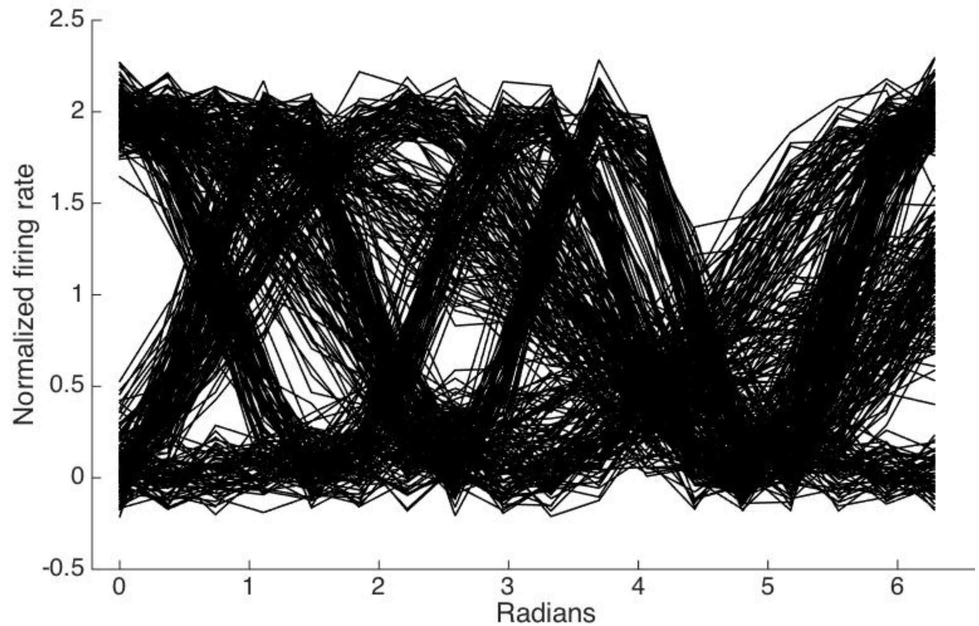
Can tell you the 'dimensionality of your data'

PCA: how to do this in the real world

Step 0. Explicitly define what the data matrix X is.

Let's say we have a bunch (360) of head direction tuning curves.

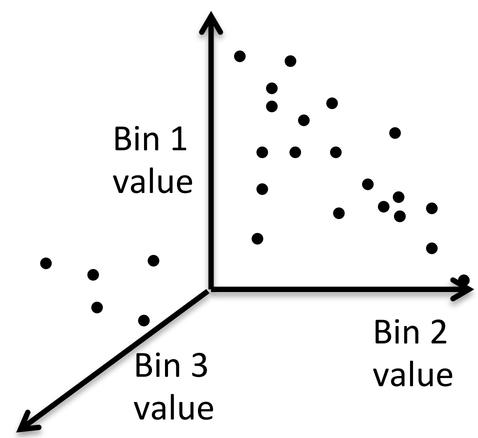
We can tell there are a couple different types… but can we say this quantitatively?



Bin each tuning curve into 18 bins

Each tuning curve is then a point in an 18-dimensional space

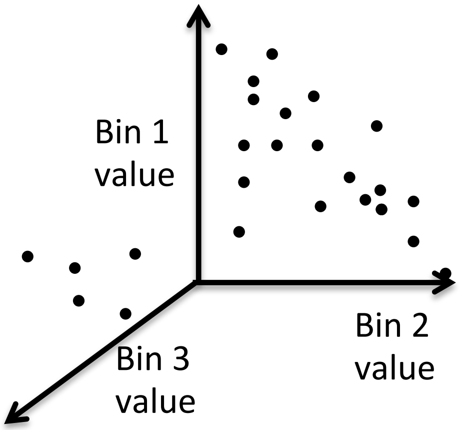
X = [TC1 TC2 … TC360]
X is a 18 x 360 matrix



PCA: how to do this in the real world

Step 1. Center the data

- Compute and subtract off the mean
- Compute and divide by standard deviation




```
MATLAB:
[~,N] = size(X);
mn = mean(X,2);
X= X- X(mn,1,N);
stdX = std(X,[],2);
X= X./repmat(stdX,1,N);
```

Otherwise, the first component will capture how far away from the origin the data is

Step 2. Compute covariance matrix

```
MATLAB:
covariance = 1/(N-1)*X*X';
```

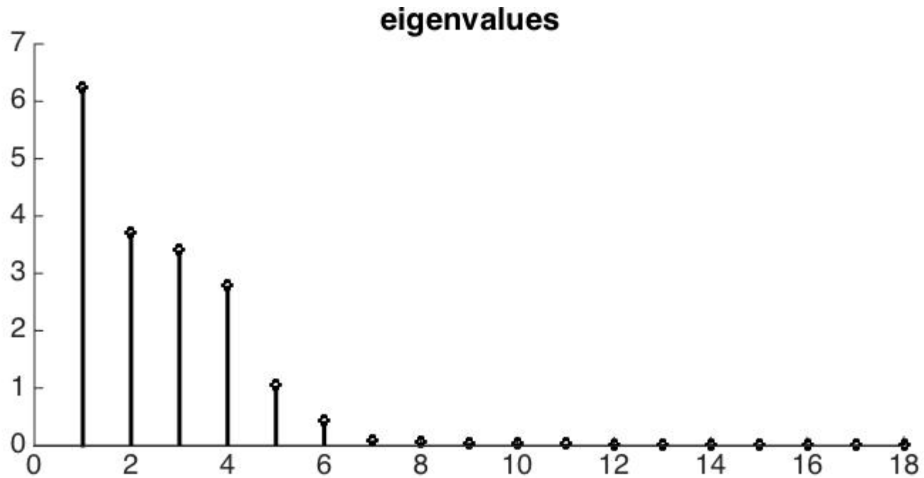
Step 3. Compute and sort eigenvectors and eigenvalues of covariance matrix

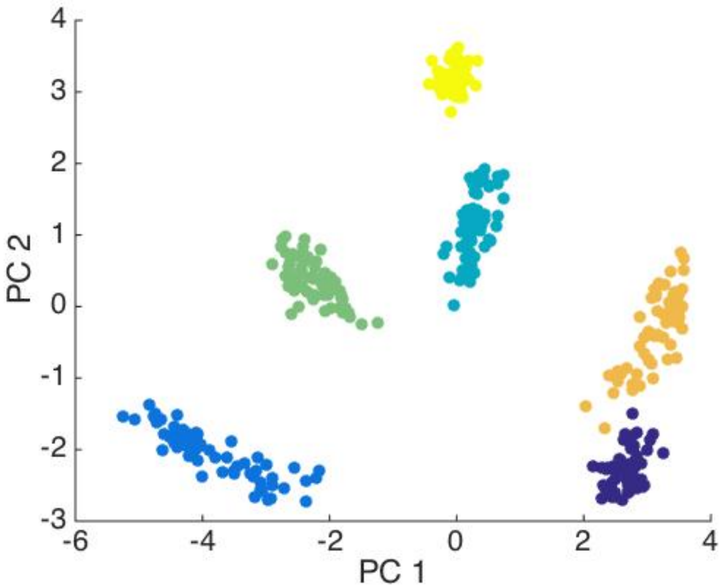
```
MATLAB:
[PC,V]= eig(X);
V = diag(V);
[~,ind] = sort(V,'descend');
PC = PC(:,ind); V = V(ind);
```

PCA: how to do this in the real world

Step 4. Transform the data and see what you can see!

```
MATLAB:
    Y = PC'*X;
```





7 types of cells

When PCA fails

The linearity assumption is not a good one

- One fix: kernel PCA, where a nonlinear transformation is applied first

Finding the most decorrelated components isn't what you want

- ICA: independent component analysis: finds the statistically independent components

Interpreting dimensionality in the face of noise can be difficult

- Noise will increase the number of 'dimensions'
- Can be difficult to draw the line between signal and noise

SVD

Recall that for some matrices (symmetric ones), we can decompose them into the following product:

$$X = VDV^{-1}$$

Where V is a matrix of eigenvectors and D is a matrix of eigenvalues along the diagonal

The SVD (singular value decomposition) is similar, but more general – you can do it for all matrices!

$$X = U\Sigma V$$

U is a matrix of eigenvectors for XX', V is a matrix of eigenvectors for X'X (the principal components!), and Σ is a matrix with singular values along the diagonal. Singular values are the square roots of X'X

→ Columns of U are 'left singular vectors', row of V are 'right singular vectors'

SVD and applications

One great thing about the SVD: it decomposes a matrix into a sum of outer products

$$X = U\Sigma V$$

$$= \begin{pmatrix} | & | & & | \\ \vec{u}^{(1)} & \vec{u}^{(2)} & \cdots & \vec{u}^{(N)} \\ | & | & & | \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & 0 & \cdots & 0 \\ 0 & 0 & \cdots & \lambda_N & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} - & \vec{v}^{(1)} & - \\ - & \vec{v}^{(2)} & - \\ & \vdots & \\ - & \vec{v}^{(N)} & - \\ & other & \end{pmatrix}$$

$$= \begin{pmatrix} | & | & & | \\ \vec{u}^{(1)} & \vec{u}^{(2)} & \cdots & \vec{u}^{(N)} \\ | & | & & | \end{pmatrix} \begin{pmatrix} - & \lambda_1\vec{v}^{(1)} & - \\ - & \lambda_2\vec{v}^{(2)} & - \\ & \vdots & \\ - & \lambda_N\vec{v}^{(N)} & - \end{pmatrix}$$

$$= \lambda_1 \begin{pmatrix} | \\ \vec{u}^{(1)} \\ | \end{pmatrix} \begin{pmatrix} - & \vec{v}^{(1)} & - \end{pmatrix} + \lambda_2 \begin{pmatrix} | \\ \vec{u}^{(2)} \\ | \end{pmatrix} \begin{pmatrix} - & \vec{v}^{(2)} & - \end{pmatrix} + \cdots + \lambda_N \begin{pmatrix} | \\ \vec{u}^{(N)} \\ | \end{pmatrix} \begin{pmatrix} - & \vec{v}^{(N)} & - \end{pmatrix}$$

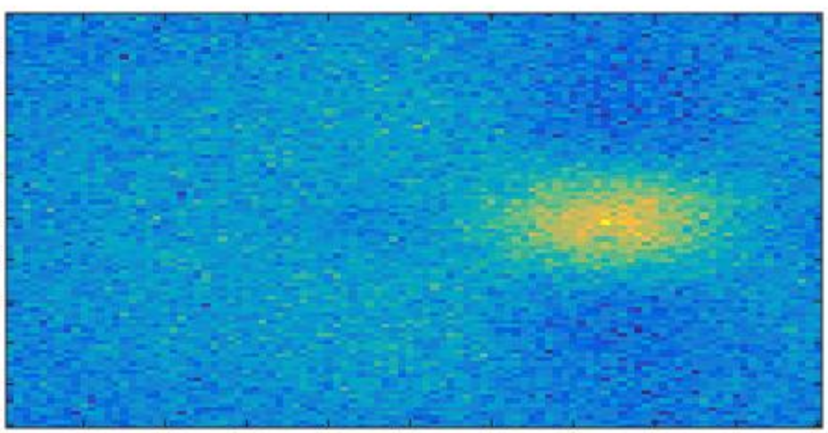
SVD and applications

$$= \underbrace{\lambda_1 \begin{pmatrix} | \\ \vec{u}^{(1)} \\ | \end{pmatrix} \begin{pmatrix} - & \vec{v}^{(1)} & - \end{pmatrix}}_{\text{Each term is a matrix}} + \lambda_2 \begin{pmatrix} | \\ \vec{u}^{(2)} \\ | \end{pmatrix} \begin{pmatrix} - & \vec{v}^{(2)} & - \end{pmatrix} + \cdots + \lambda_N \begin{pmatrix} | \\ \vec{u}^{(N)} \\ | \end{pmatrix} \begin{pmatrix} - & \vec{v}^{(N)} & - \end{pmatrix}$$

Each matrix is ordered in terms of 'importance'
One application: compute low-rank approximations of matrices

```
MATLAB:
[U,S,V] = svd(X,0); % take an svd
X_0 = U(:,1:p)*S(1:p,1:p)*V(:,1:p)';
```

Noisy receptive field:



Low-rank approximation: