CS9840
Learning and Computer Vision
Prof. Olga Veksler

Lecture 2

Some Concepts from Computer Vision

Curse of Dimensionality

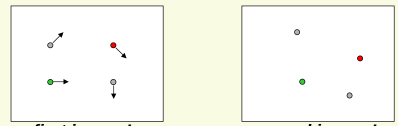
PCA

Some Slides are from Cornelia, Fermüller, Mubarak Shah,

Gary Bradski,

Sebastian Thrun

Optical flow



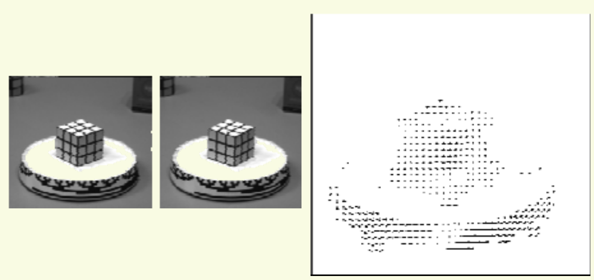
first image I_1 *second image I_2*

- How to estimate pixel motion from image I_1 to image I_2?
  - Solve pixel correspondence problem
    - given a pixel in I_1, look for **nearby** pixels of the **same** color in I_2
- Key assumptions
  - **color constancy:** a point in I_1 looks the same in I_2
    - For grayscale images, this is **brightness constancy**
  - **small motion**: points do not move very far
- This is called the **optical flow** problem

Outline

- Some Concepts in Image Processing/Vision
  - Optical Flow Field (related to motion field)
  - Correlation
- Curse of Dimensionality and Dimensionality reduction with PCA
- Next time:
  - *"Recognizing Action at a Distance" by A. Efros, A.Berg, G. Mori, Jitendra Malik*
  - *Also: "80 million tiny images: a large dataset for non-parametric object and scene recognition", A. Torralba, R. Fergus, W. Freeman*
  - there should be a link to PDF file on our web site

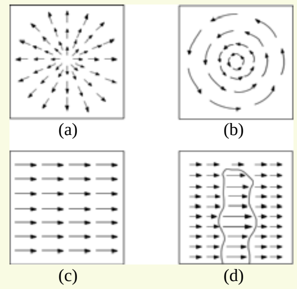
Optical Flow Field



Optical Flow and Motion Field

- Optical flow field is the apparent motion of brightness patterns between 2 (or several) frames in an image sequence
- Why does brightness change between frames?
- Assuming that illumination does not change:
  - changes are due to the RELATIVE MOTION between the scene and the camera
  - There are 3 possibilities:
    - Camera still, moving scene
    - Moving camera, still scene
    - Moving camera, moving scene

Examples of Motion Fields



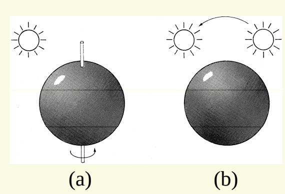
(a) Translation perpendicular to a surface. (b) Rotation about axis perpendicular to image plane. (c) Translation parallel to a surface at a constant distance. (d) Translation parallel to an obstacle in front of a more distant background.

Motion Field (MF)

- The MF assigns a velocity vector to each pixel in the image
- These velocities are INDUCED by the RELATIVE MOTION between the camera and the 3D scene
- The MF is the *projection* of the 3D velocities on the image plane

Optical Flow vs. Motion Field

- Recall that Optical Flow is the apparent motion of brightness patterns
- We equate Optical Flow Field with Motion Field
- Frequently works, but now always:

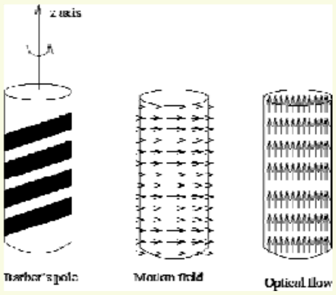


(a) A smooth sphere is rotating under constant illumination. Thus the optical flow field is zero, but the motion field is not

(b) A fixed sphere is illuminated by a moving source—the shading of the image changes. Thus the motion field is zero, but the optical flow field is not

Optical Flow vs. Motion Field

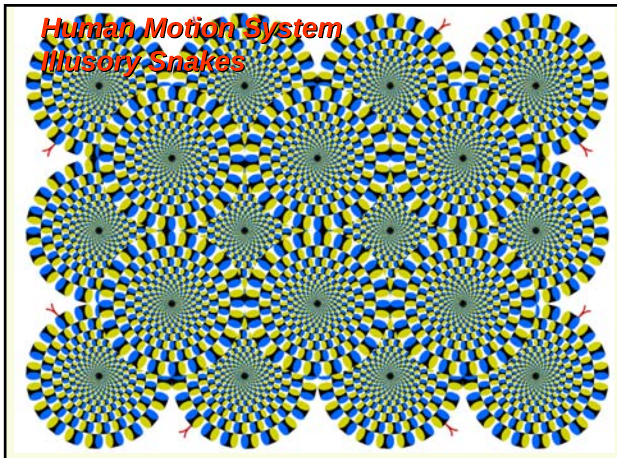
- Often (but not always) optical flow corresponds to the true motion of the scene



Computing Optical Flow: Brightness Constancy Equation

- Let ***P*** be a moving point in 3D:
  - At time ***t***, ***P*** has coordinates (***X***(***t***),***Y***(***t***),***Z***(***t***))
  - Let ***p***=(***x***(***t***),***y***(***t***)) be the coordinates of its image at time ***t***
  - Let ***E***(***x***(***t***),***y***(***t***),***t***) be the brightness at ***p*** at time ***t***.
- Brightness Constancy Assumption:
  - As ***P*** moves over time, ***E***(***x***(***t***),***y***(***t***),***t***) remains constant

Human Motion System Illusory Snakes



from Gary Bradski and Sebastian Thrun

Computing Optical Flow: Brightness Constancy Equation

$$E(x(t), y(t), t) = Constant$$

Taking derivative wrt time:

$$\frac{dE(x(t), y(t), t)}{dt} = 0$$

$$\frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t} = 0$$

Computing Optical Flow: Brightness Constancy Equation

1 equation with 2 unknowns

$$\frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t} = 0$$

Let

$$\nabla E = \begin{bmatrix} \frac{\partial E}{\partial x} \\ \frac{\partial E}{\partial y} \end{bmatrix}$$ (Frame spatial gradient)

$$v = \begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \end{bmatrix}$$ (optical flow)

and

$$E_t = \frac{\partial E}{\partial t}$$ (derivative across frames)

Video Sequence

*

* Picture from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Computing Optical Flow: Brightness Constancy Equation

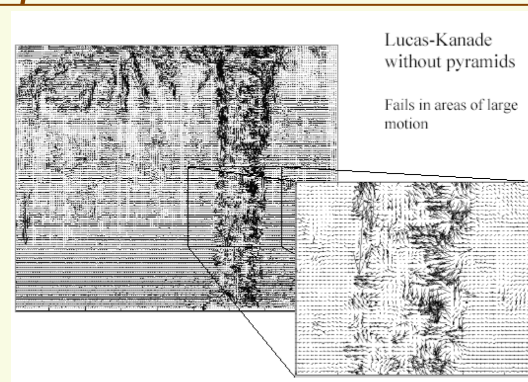
- How to get more equations for a pixel?
  - Basic idea: impose additional constraints
    - most common is to assume that the flow field is smooth locally
    - one method: pretend the pixel's neighbors have the same (u,v)
      - If we use a 5x5 window, that gives us 25 equations per pixel!

$$\boldsymbol{E_t(p_i) + \nabla E(p_i) \cdot [u \;\; v] = 0}$$

$$\begin{bmatrix} E_x(p_1) & E_y(p_1) \\ E_x(p_2) & E_y(p_2) \\ \vdots & \vdots \\ E_x(p_{25}) & E_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} E_t(p_1) \\ E_t(p_2) \\ \vdots \\ E_t(p_{25}) \end{bmatrix}$$

matrix ***E*** 25x2 | vector ***d*** 2x1 | vector ***b*** 25x1

Optical Flow Results

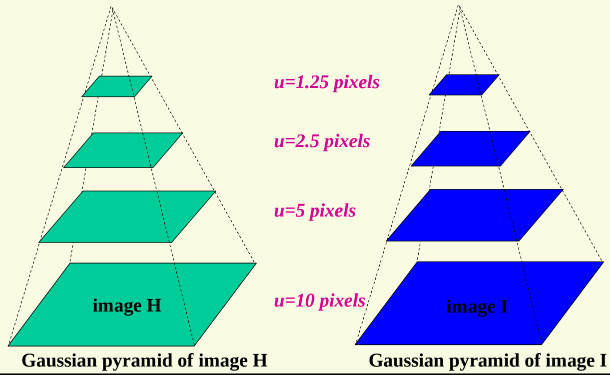


* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

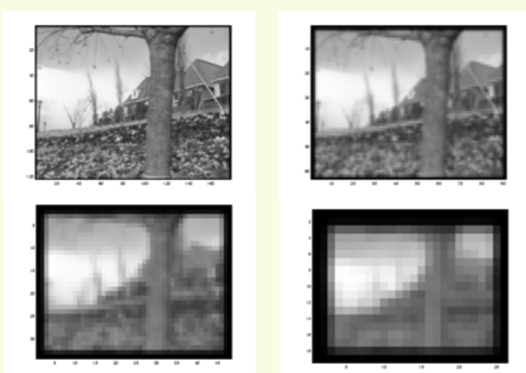
Revisiting the small motion assumption

- Is this motion small enough?
  - Probably not—it's much larger than one pixel (2^{nd} order terms dominate)
  - How might we solve this problem?

Coarse-to-fine optical flow estimation



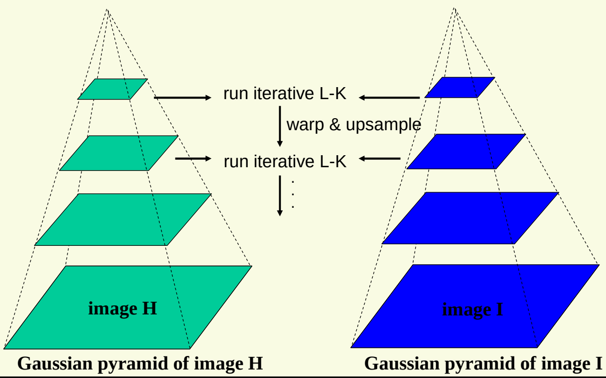
Reduce the resolution!



Iterative Refinement

- Iterative Lukas-Kanade Algorithm
  1. Estimate velocity at each pixel by solving Lucas-Kanade equations
  2. Warp H towards I using the estimated flow field
     - *- use image warping techniques*
  3. Repeat until convergence

Coarse-to-fine optical flow estimation



Other Concepts to Review

- Convolution is the operation of applying a “kernel” to each pixel of an image

image

I_{11}	I_{12}	I_{13}	I_{14}	I_{15}	I_{16}	I_{17}	I_{18}	I_{19}
I_{21}	I_{22}	I_{23}	I_{24}	I_{25}	I_{26}	I_{27}	I_{28}	I_{29}
I_{31}	I_{32}	I_{33}	I_{34}	I_{35}	I_{36}	I_{37}	I_{38}	I_{39}
I_{41}	I_{42}	I_{43}	I_{44}	I_{45}	I_{46}	I_{47}	I_{48}	I_{49}
I_{51}	I_{52}	I_{53}	I_{54}	I_{55}	I_{56}	I_{57}	I_{58}	I_{59}
I_{61}	I_{62}	I_{63}	I_{64}	I_{65}	I_{66}	I_{67}	I_{68}	I_{69}

kernel

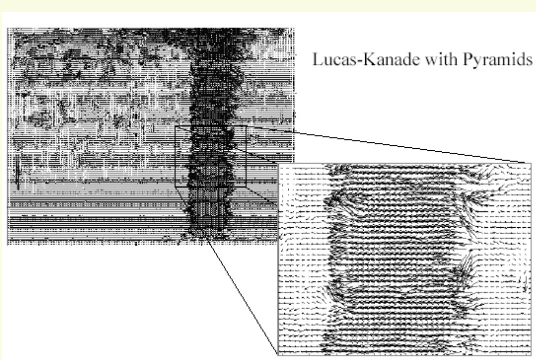
K_{11}	K_{12}	K_{13}
K_{21}	K_{22}	K_{23}

- Result of convolution has the same dimension as the image
- For example:

$$O_{57} = I_{57}K_{11} + I_{58}K_{12} + I_{59}K_{13} + I_{67}K_{21} + I_{68}K_{22} - I_{69}K_{23}$$

- Convolution is frequently denoted by *, for example I*K

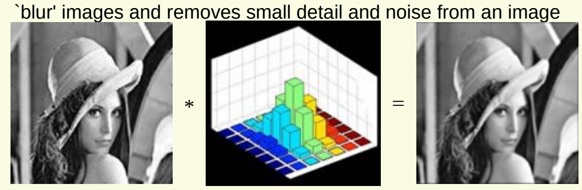
Optical Flow Results



* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Other Concepts to Review

- Gaussian smoothing (blurring): convolution operator that is used to `blur' images and removes small detail and noise from an image



$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Gaussian Smoothing vs. Averaging

Gaussian Smoothing

Smoothing by Averaging

$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

$\frac{1}{25}$

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

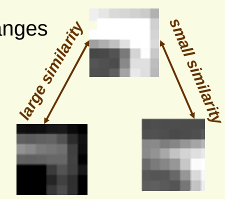
Other Concepts to Review

- Cross-correlation

$$c(f,g) = \sum_{i=1}^{d} f(i)g(i)$$

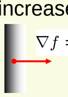
  - measures similarity between images (or image regions) f and g
  - works OK if there is no change in intensity
- Normalized cross correlation, more popular in image processing
  - Insensitive to linear intensity changes between image patches f and g

$$NCC(f,g) = \frac{\sum_{i=1}^{d}\left(f(i)-\bar{f}\right)\left(g(i)-\bar{g}\right)}{\left[\sum_{i=1}^{d}\left(f(i)-\bar{f}\right)^2 \sum_{k=1}^{d}{}^2\left(g(i)-\bar{g}\right)\right]^{1/2}}$$

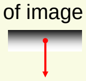


Other Concepts to Review

- Image gradient: points in the direction of the most rapid increase in intensity of image ***f***



$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$



$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$

$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

- Sobel operator to compute gradient:

$\frac{1}{8}$

-1	0	1
-2	0	2
-1	0	1

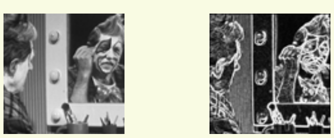
$\frac{\partial f}{\partial x}$

$\frac{1}{8}$

1	2	1
0	0	0
-1	-2	-1

$\frac{\partial f}{\partial y}$

- Results:



Curse of Dimensionality

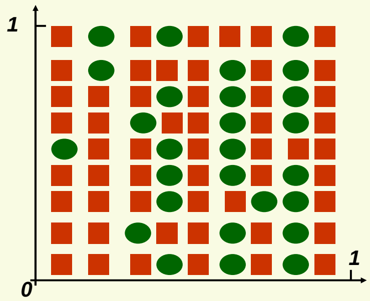
- Problems of high dimensional data, "the curse of dimensionality"
  - running time
  - overfitting
  - number of samples required
- Dimensionality Reduction Methods
  - Principle Component Analysis

Curse of Dimensionality: Complexity

- Complexity (running time) increases with dimension d
- A lot of methods have at least $O(nd^2)$ complexity, where n is the number of samples
  - For example if we need to estimate covariance matrix
- So as d becomes large, $O(nd^2)$ complexity may be too costly

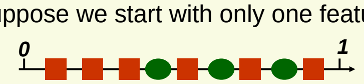
Curse of Dimensionality: Number of Samples

- We need 9^2 samples to maintain the same density as in ***1D***



Curse of Dimensionality: Number of Samples

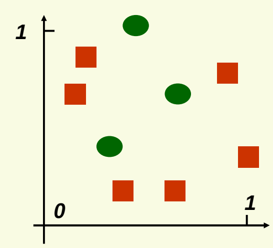
- Suppose we want to use the nearest neighbor approach with $k = 1$ (***1NN***)
- Suppose we start with only one feature



- This feature is not discriminative, i.e. it does not separate the classes well
- We decide to use 2 features. For the 1NN method to work well, need a lot of samples, i.e. samples have to be dense
- To maintain the same density as in 1D (9 samples per unit length), how many samples do we need?

Curse of Dimensionality: Number of Samples

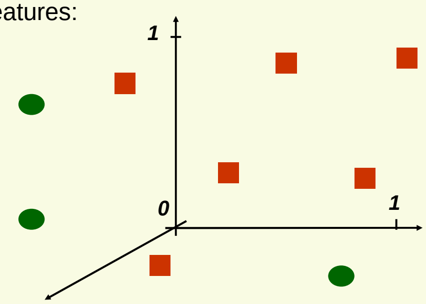
- Of course, when we go from 1 feature to 2, no one gives us more samples, we still have 9



- This is way too sparse for ***1NN*** to work well

Curse of Dimensionality: Number of Samples

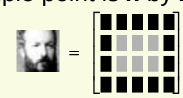
- Things go from bad to worse if we decide to use 3 features:



- If ***9*** was dense enough in 1D, in 3D we need $\boldsymbol{9^3=729}$ samples!

The Curse of Dimensionality

- We should try to avoid creating lot of features
- Often no choice, problem starts with many features
- Example: Face Detection
  - One sample point is ***k*** by ***m*** array of pixels



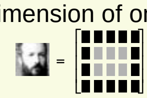
  - Feature extraction is not trivial, usually every pixel is taken as a feature
  - Typical dimension is 20 by 20 = 400
  - Suppose ***10*** samples are dense enough for 1 dimension. Need only $\boldsymbol{10^{400}}$ samples

Curse of Dimensionality: Number of Samples

- In general, if ***n*** samples is dense enough in ***1D***
- Then in ***d*** dimensions we need $\boldsymbol{n^d}$ samples!
- And $\boldsymbol{n^d}$ grows really really fast as a function of ***d***
- Common pitfall:
  - If we can't solve a problem with a few features, adding more features seems like a good idea
  - However the number of samples usually stays the same
  - The method with more features is likely to perform worse instead of expected better

The Curse of Dimensionality

- Face Detection, dimension of one sample point is ***km***



- The fact that we set up the problem with ***km*** dimensions (features) does not mean it is really a ***km***-dimensional problem
- Space of all ***k*** by ***m*** images has ***km*** dimensions
- Space of all ***k*** by ***m*** faces must be much smaller, since faces form a tiny fraction of all possible images
- Most likely we are not setting the problem up with the right features
- If we used better features, we are likely need much less than ***km***-dimensions

Dimensionality Reduction

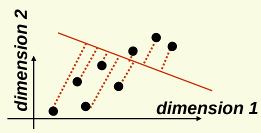
- High dimensionality is challenging and redundant
- It is natural to try to reduce dimensionality
- Reduce dimensionality by feature combination: combine old features ***x*** to create new features ***y***

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \rightarrow f\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}\right) = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix} = y \quad \text{with } k < d$$

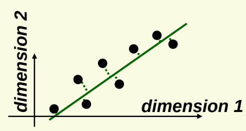
- For example, $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 + x_2 \\ x_3 + x_4 \end{bmatrix} = y$
- Ideally, the new vector ***y*** should retain from ***x*** all information important for classification

Principle Component Analysis (PCA)

- Main idea: seek most accurate data representation in a lower dimensional space
- Example in 2-D
  - Project data to 1-D subspace (a line) which minimize the projection error



large projection errors, bad line to project to



small projection errors, good line to project to

- Notice that the the good line to use for projection lies in the direction of largest variance

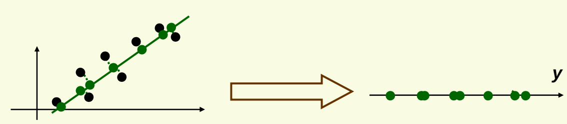
Dimensionality Reduction

- The best ***f***(***x***) is most likely a non-linear function
- Linear functions are easier to find though
- For now, assume that ***f***(***x***) is a linear mapping
- Thus it can be represented by a matrix ***W***:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \Rightarrow W\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} w_{11} & \cdots & w_{1d} \\ \vdots & & \vdots \\ w_{k1} & \cdots & w_{kd} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix} \quad \text{with } k < d$$

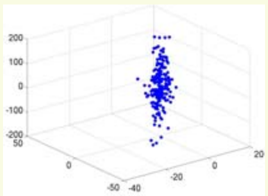
PCA

- After the data is projected on the best line, need to transform the coordinate system to get 1D representation for vector ***y***

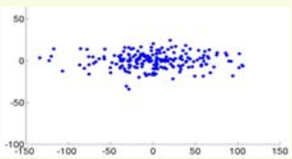


- Note that new data ***y*** has the same variance as old data ***x*** in the direction of the green line
- PCA preserves largest variances in the data

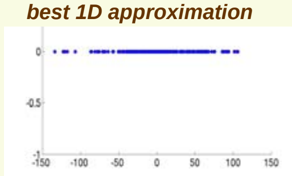
PCA: Approximation of Elliptical Cloud in 3D



best 2D approximation



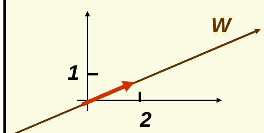
best 1D approximation



PCA: Linear Algebra Review

- Let ***V*** be a ***d*** dimensional linear space, and ***W*** be a ***k*** dimensional linear subspace of ***V***
- We can always find a set of ***d*** dimensional vectors $\{\mathbf{e}_1,\mathbf{e}_2,\dots,\mathbf{e}_k\}$ which forms an orthonormal basis for ***W***
  - $\langle \mathbf{e}_i,\mathbf{e}_j\rangle = 0$ if ***i*** is not equal to ***j*** and $\langle \mathbf{e}_i,\mathbf{e}_i\rangle = 1$
- Thus any vector in ***W*** can be written as

$$\alpha_1\mathbf{e}_1+\alpha_2\mathbf{e}_2+\dots+\alpha_k\mathbf{e}_k=\sum_{i=1}^{k}\alpha_i\mathbf{e}_i \quad \textbf{for scalars} \ \alpha_1,\dots,\alpha_k$$

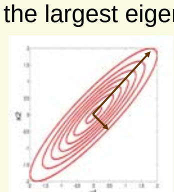


Let $V = R^2$ and W be the line x-2y=0. Then the orthonormal basis for W is

$$\left\{\begin{bmatrix}2/\sqrt{5}\\1/\sqrt{5}\end{bmatrix}\right\}$$

PCA

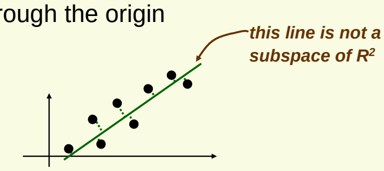
- What is the direction of largest variance in data?
- Recall that if ***x*** has multivariate distribution $N(\mu,\Sigma)$, direction of largest variance is given by eigenvector corresponding to the largest eigenvalue of Σ



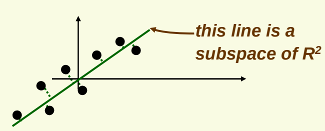
- This is a hint that we should be looking at the covariance matrix of the data (note that PCA can be applied to distributions other than Gaussian)

PCA: Linear Algebra

- Recall that subspace ***W*** contains the zero vector, i.e. it goes through the origin



- It is convenient to project to subspace ***W***: thus we need to shift everything

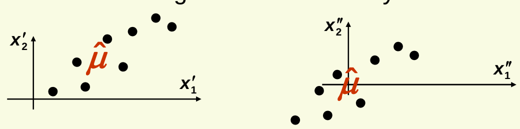


PCA Derivation: Shift by the Mean Vector

- Before PCA, subtract sample mean from the data

$$x - \frac{1}{n}\sum_{i=1}^{n} x_i = x - \hat{\mu}$$

- The new data has zero mean: E(***X***-E(***X***)) = E(***X***)-E(***X***) = 0
- All we did is change the coordinate system



- Another way to look at it:
  - first step of getting ***y*** is to subtract the mean of ***x***

$$x \rightarrow y = f(x) = g(x - \hat{\mu})$$

PCA: Derivation

- To find the total error, we need to sum over all x_j's
- Any x_j can be written as $\sum_{i=1}^{k} \alpha_{ji} e_i$
- Thus the total error for representation of all data ***D*** is:

$$\underbrace{J(e_1,\ldots,e_k,\alpha_{11},\ldots\alpha_{nk})}_{\text{unknowns}} = \overbrace{\sum_{j=1}^{n}}^{\text{sum over all data points}} \underbrace{\left\| x_j - \sum_{i=1}^{k} \alpha_{ji} e_i \right\|^2}_{\text{error at one point}}$$

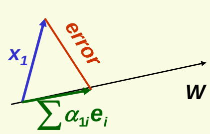
PCA: Derivation

- We want to find the most accurate representation of data ***D***={$x_1, x_2, \ldots, x_n$} in some subspace ***W*** which has dimension ***k*** < ***d***
- Let {$e_1, e_2, \ldots, e_k$} be the orthonormal basis for ***W***. Any vector in ***W*** can be written as $\sum_{i=1}^{k} \alpha_i e_i$
- Thus x_1 will be represented by some vector in ***W***

$$\sum_{i=1}^{k} \alpha_{1i} e_i$$

- Error this representation:

$$error = \left\| x_1 - \sum_{i=1}^{k} \alpha_{1i} e_i \right\|^2$$



PCA: Derivation

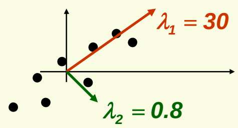
- A lot of math…….to finally get:
- Let ***S*** be the scatter matrix, it is just n-1 times the sample covariance matrix

$$\hat{\Sigma} = \frac{1}{n-1}\sum_{j=1}^{n}(x_j - \hat{\mu})(x_j - \hat{\mu})^t$$

- To minimize ***J*** take for the basis of ***W*** the ***k*** eigenvectors of ***S*** corresponding to the ***k*** largest eigenvalues

PCA

- The larger the eigenvalue of S, the larger is the variance in the direction of corresponding eigenvector



- This result is exactly what we expected: project x into subspace of dimension k which has the largest variance
- This is very intuitive: restrict attention to directions where the scatter is the greatest

PCA as Data Approximation

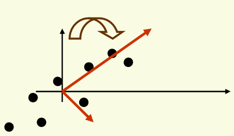
- Let $\{e_1, e_2, \ldots, e_d\}$ be all d eigenvectors of the scatter matrix S, sorted in order of decreasing corresponding eigenvalue
- Without any approximation, for any sample x_i:

$$x_i = \sum_{j=1}^{d} \alpha_j e_j = \underbrace{\alpha_1 e_1 + \ldots + \alpha_k e_k}_{\text{approximation of } x_i} + \overbrace{\alpha_{k+1} e_{k+1} \ldots + \alpha_d e_d}^{\text{error of approximation}}$$

- coefficients $\alpha_m = x^t_i e_m$ are called *principle components*
  - The larger k, the better is the approximation
  - Components are arranged in order of importance, more important components come first
- Thus PCA takes the first k most important components of x_i as an approximation to x_i

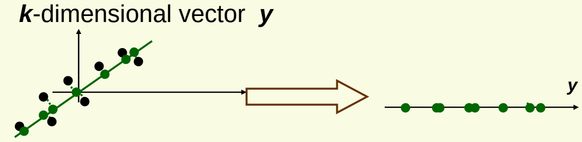
PCA

- Thus PCA can be thought of as finding new orthogonal basis by rotating the old axis until the directions of maximum variance are found



PCA: Last Step

- Now we know how to project the data
- Last step is to change the coordinates to get final k-dimensional vector y



- Let matrix $E = [e_1 \cdots e_k]$
- Then the coordinate transformation is $y = E^t x$
- Under E^t, the eigenvectors become the standard basis: $E^t e_i = \begin{bmatrix} e_1 \\ \vdots \\ e_i \\ \vdots \\ e_k \end{bmatrix} e_i = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$

Recipe for Dimension Reduction with PCA

Data $\boldsymbol{D}=\{\boldsymbol{x_1},\boldsymbol{x_2},\ldots,\boldsymbol{x_n}\}$. Each $\boldsymbol{x_i}$ is a $\boldsymbol{d}$-dimensional vector. Wish to use PCA to reduce dimension to $\boldsymbol{k}$

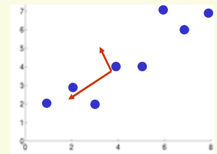
1. Find the sample mean $\hat{\mu}=\frac{1}{n}\sum_{i=1}^{n}x_i$
2. Subtract sample mean from the data $z_i=x_i-\hat{\mu}$
3. Compute the scatter matrix $S=\sum_{i=1}^{n}z_i z_i^t$
4. Compute eigenvectors $\boldsymbol{e_1},\boldsymbol{e_2},\ldots,\boldsymbol{e_k}$ corresponding to the $\boldsymbol{k}$ largest eigenvalues of $\boldsymbol{S}$
5. Let $\boldsymbol{e_1},\boldsymbol{e_2},\ldots,\boldsymbol{e_k}$ be the columns of matrix $E=[e_1 \cdots e_k]$
6. The desired $\boldsymbol{y}$ which is the closest approximation to $\boldsymbol{x}$ is $y=E^t z$

PCA Example Using Matlab

- Use [V,D] =eig($\boldsymbol{S}$) to get eigenvalues and eigenvectors of $\boldsymbol{S}$

$$\lambda_1=87 \text{ and } e_1=\begin{bmatrix}-0.8\\-0.6\end{bmatrix}$$

$$\lambda_2=3.8 \text{ and } e_2=\begin{bmatrix}0.6\\-0.8\end{bmatrix}$$



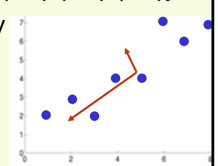
- Projection to 1D space in the direction of $\boldsymbol{e_1}$

$$Y=e_1^t Z^t=\left(\begin{bmatrix}-0.8 & -0.6\end{bmatrix}\begin{bmatrix}-3.6 & \cdots & 4.4\\-4.4 & \cdots & 2.6\end{bmatrix}\right)=\begin{bmatrix}4.3 & \cdots & -5.1\end{bmatrix}$$
$$=\begin{bmatrix}y_1 & \cdots & y_8\end{bmatrix}$$

PCA Example Using Matlab

- Let $\boldsymbol{D}$ = {(1,2),(2,3),(3,2),(4,4),(5,4),(6,7),(7,6),(9,7)}
- Convenient to arrange data in array

$$X=\begin{bmatrix}1 & 2\\ \vdots & \vdots\\ 9 & 7\end{bmatrix}=\begin{bmatrix}x_1\\ \vdots\\ x_8\end{bmatrix}$$



- Mean $\mu=mean(X)=\begin{bmatrix}4.6 & 4.4\end{bmatrix}$
- Subtract mean from data to get new data array Z

$$Z=X-\begin{bmatrix}\mu\\ \vdots\\ \mu\end{bmatrix}=X-repmat(\mu,8,1)=\begin{bmatrix}-3.6 & -4.4\\ \vdots & \vdots\\ 4.4 & 2.6\end{bmatrix}$$

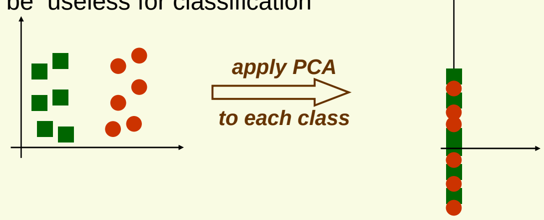
- Compute the scatter matrix $\boldsymbol{S}$

$$S=7*cov(Z)=\begin{bmatrix}-3.6 & -4.4\end{bmatrix}\begin{bmatrix}-3.6\\-4.4\end{bmatrix}+\ldots+\begin{bmatrix}4.4 & 2.6\end{bmatrix}\begin{bmatrix}4.4\\2.6\end{bmatrix}=\begin{bmatrix}57 & 40\\40 & 34\end{bmatrix}$$

*matlab uses unbiased estimate for covariance, so S=(n-1)*cov(Z)*

Drawbacks of PCA

- PCA was designed for accurate *data representation*, not for data classification
  - Preserves as much variance in data as possible
  - If directions of maximum variance is important for classification, will work
- However the directions of maximum variance may be useless for classification



Next Time

- Paper:*"Recognizing Action at a Distance" by A. Efros, A.Berg, G. Mori, Jitendra Malik*
  - will watch the conference presentation
- *Also: "80 million tiny images: a large dataset for non-parametric object and scene recognition", A. Torralba, R. Fergus, W. Freeman*
- When reading papers, think about following:
    - What is the problem paper tries to solve
    - What makes this problem difficult?
    - What is the method used in the paper to solve the problem
    - What is the contribution of the paper (what new does it do)?
    - Do the experimental results look "good" to you?