Principal Components Analysis (PCA)
Principal Components

- Assume we have a population of vectors $x$
- Example: create a set of input data points (vectors in 2D)

\[
\mathbf{x}_{in} = \begin{bmatrix} x \\ y \end{bmatrix}
\]

```matlab
clear all
close all

randn('state',0);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Make sample data
N = 50;

xIn(:,1) = randn(N,1) + 1;
xIn(:,2) = 0.5*randn(N,1) + 0.5;

theta = -0.707;
R = [ cos(theta) sin(theta);
     -sin(theta) cos(theta)];

xIn = xIn*R;
figure, plot(xIn(:,1), xIn(:,2), '.');
title('Input vectors');
axis equal
axis([-3.0 3.0 -3.0 3.0]);
```
Example (continued)

- The mean is \( \mathbf{m}_x = E[\mathbf{x}] \)

\[
\mathbf{m}_x = E\{\mathbf{x}_{in}\} = \frac{1}{K} \sum_{k=1}^{K} \mathbf{x}_{in(k)}
\]

```matlab
ux = mean(xIn); % mean of input vectors
hold on
plot(ux(1), ux(2), '+r');
hold off
pause
```
Example (continued)

- The covariance matrix is \( C_x = E[ (x-m_x)(x-m_x)^T ] \)
  
  - To compute covariance we first subtract off the mean of data points

\[
x = x_{in} - m_x
\]

```matlab
x = xIn - repmat(ux,N,1);    % subtract off mean
figure, plot(x(:,1), x(:,2), '.');
title('Centered input vectors');
axis equal
axis([-3.0 3.0 -3.0 3.0]);
```
Example (continued)

• Find covariance matrix

\[ C_x = E[(x - m_x)(x - m_x)^T] \]
\[ = \frac{1}{K} \sum_{k=1}^{K} x_k x_k^T - m_x m_x^T \]

\[
C_x = \begin{bmatrix}
\sigma_{x}^2 & \sigma_{xy}^2 \\
\sigma_{xy}^2 & \sigma_{y}^2
\end{bmatrix}
\]

Covariance of input

\[
\begin{array}{cc}
0.5353 & -0.4118 \\
-0.4118 & 0.5600
\end{array}
\]

• The covariance matrix is real and symmetric

• If dimensions \( x_1, x_2 \) are uncorrelated, their covariance (the off diagonal terms) are zero

% Covariance of input vectors
Cx = cov(x);
disp('Covariance of input');
disp(Cx);
Principal Components

• Consider the eigenvectors and eigenvalues of $C_x$

$$C_x e_i = \lambda_i e_i$$

• $e_i$ are the eigenvectors
• $\lambda_i$ are the corresponding eigenvalues

• By definition, eigenvectors are orthonormal
  – The magnitude (or length) of each eigenvector equals 1
  – The dot product of any pair of vectors is 0 (i.e., they are perpendicular to each other)

$$|e_i| = 1 \quad e_i \cdot e_j = 0, \text{ if } i \neq j$$

• We sort the eigenvectors and eigenvalues in descending order

$$\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \ldots \geq \lambda_N$$

• The eigenvectors are called “principal components”
Principal Components

- Let $A$ be the matrix whose rows are the eigenvectors of $C_x$
- We can use $A$ as a transformation matrix that maps $x$ into $y$
  $$y = A(x - m_x)$$
  - This is called the Hotelling transform, or principal components transform

- The covariance matrix of $y$ is
  $$C_y = E[(y - m_y)(y - m_y)^T]$$

- But $m_y = 0$ because
  $$E[y] = E[A(x - m_x)] = A(E[x] - E[m_x]) = 0$$

- So
  $$C_y = E[yy^T] = E[A(x - m_x)(x - m_x)^T A^T]$$
  $$= AE[xx^T]A^T = AC_xA^T$$
Principal Components

- We have
  \[ C_y = AC_xA^T \]
  where
  \[ A = \begin{bmatrix} e_1^T \\ e_2^T \\ \vdots \\ e_N^T \end{bmatrix}, \text{ and } A^T = [e_1 \ e_2 \ \ldots \ \ e_N] \]

- Now, since the columns of \( A^T \) are the eigenvectors of \( C_x \), then multiplying \( A^T \) by \( C_x \) just gives us the eigenvectors back (times the eigenvalues)
  \[ C_xA^T = C_x[e_1 \ e_2 \ \ldots \ e_N] = [\lambda_1 e_1 \ \lambda_2 e_2 \ \ldots \ \lambda_N e_N] \]

- So
  \[ C_y = AC_xA^T = \begin{bmatrix} e_1^T \\ e_2^T \\ \vdots \\ e_N^T \end{bmatrix} \begin{bmatrix} \lambda_1 e_1 & \lambda_2 e_2 & \ldots & \lambda_N e_N \end{bmatrix} = \begin{pmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & 0 & \ldots \\ & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \lambda_N \end{pmatrix} \]
Principal Components

• $C_y$ is a diagonal matrix

$$C_y = \begin{pmatrix} \lambda_1 & 0 \\ & \lambda_2 \\ & & \ddots \\ & & & \lambda_n \end{pmatrix}$$

– where the $\lambda_i$ are the eigenvalues of $C_x$
– $C_x$ and $C_y$ have the same eigenvalues

• Again, the eigenvectors of $C_x$ (the rows of the matrix $A$) are called “principal components”
  – They are orthogonal and orthonormal
  – As a result, $A^{-1} = A^T$

• We can express any vector $x$ as a linear combination of the principal components

$$y = A(x - m_x) \quad \rightarrow \quad A^T y = A^T A(x - m_x) \quad \rightarrow \quad x = A^T y + m_x$$

So the dimensions of the $y$’s are uncorrelated
Principal Components Example

• Find eigenvalues and eigenvectors

\[ C_x e = \lambda e \]

Eigenvector e1:
-0.7176
-0.6964
Eigenvector e2:
-0.6964
0.7176
Eigenvalue d1:
0.1357
Eigenvalue d2:
0.9597

% Get eigenvalues and eigenvectors of Cx
% Produces V,D such that Cx*V = V*D.
[V,D] = eig(Cx);
e1 = V(:,1);
disp('Eigenvector e1:'), disp(e1);
e2= V(:,2);
disp('Eigenvector e2:'), disp(e2);
d1 = D(1,1);
disp('Eigenvalue d1:'), disp(d1);
d2 = D(2,2);
disp('Eigenvalue d2:'), disp(d2);

• Verify

\[
\begin{align*}
C_x e_1 &= \left[\begin{array}{cc}
-0.0974 \\
-0.0945
\end{array}\right] \\
C_x e_2 &= \left[\begin{array}{cc}
-0.6684 \\
0.6887
\end{array}\right] \\
d_1 e_1 &= \left[\begin{array}{cc}
-0.0974 \\
-0.0945
\end{array}\right] \\
d_2 e_2 &= \left[\begin{array}{cc}
-0.6684 \\
0.6887
\end{array}\right]
\end{align*}
\]
Principal Components Example

- Draw eigenvectors (principal components)
Principal Components Example

• Project input data onto principal components
  \[ y = A \left( x - \bar{x} \right) \]
  
  – where \( A \) is the matrix whose rows are eigenvectors of \( Cx \)

```matlab
% Project input data onto principal components
y = [e2'; e1']*x';

figure, plot(y(1,:),y(2,:), '.');
title('Projections onto principal components');
axis equal
axis([-3.0 3.0 -3.0 3.0]);
```
Principal Components

• We can reconstruct $\mathbf{x}$ from the $\mathbf{y}$’s

\[ \mathbf{x} = \mathbf{A}^\top \mathbf{y} + \mathbf{m}_x \]

• Instead of using all the eigenvectors of $\mathbf{C}_x$, we can take only the eigenvectors corresponding to the $k$ largest eigenvalues $\Rightarrow \mathbf{A}_k$

• We can reconstruct an approximation of $\mathbf{x}$ from only a few principal components (PC’s):

\[ \mathbf{x}' = \mathbf{A}_k^\top \mathbf{y} + \mathbf{m}_x \]

• So to represent the input data approximately, we just need to use $k$ principal components and the value of the projections onto those components

• The mean squared error is

\[ e_{ms} = E\left\{ (\mathbf{x} - \hat{\mathbf{x}})^2 \right\} = \sum_{j=1}^{n} \lambda_j - \sum_{j=1}^{k} \lambda_j \]

\[ = \sum_{j=k+1}^{n} \lambda_j \]
Principal Components Example

- Project input data onto only k principal components
  \[ y = A_k (x - m_x) \]
  
  - where \( A_k \) is the matrix whose rows are the first \( k \) eigenvectors of \( C_x \)

```matlab
 PROJECT input data using only one principal component
 y = [e2']*x';
 figure, plot(y(1,:),zeros(1,length(y)), '.');
title('Projections onto one principal component');
axis equal
axis([-3.0 3.0 -3.0 3.0]);
```
Principal Components Example

• Reconstruction of input data using only $k$ principal components

$$\mathbf{x} = \mathbf{A}_k^T \mathbf{y} + \mathbf{m}_x$$

— where $\mathbf{A}_k$ is the matrix whose rows are the first $k$ eigenvectors of $\mathbf{C}_x$

% Reconstruct
xx = e2*y + repmat(ux',1,length(y));
figure, plot(xx(1,:),xx(2,:),
axis equal
axis([-3.0 3.0 -3.0 3.0]);
Use of PCA for description

• We can use principal components analysis (PCA) to
  – represent images more concisely
  – help with recognition and matching

• We’ll look at two ways to use PCA on images
  – If we have a multidimensional image (e.g., color RGB)
    • We have a collection of vectors, each represents a pixel (e.g., R,G,B values)
    • There is no notion of spatial position ... the set is a “bag of pixels”
    • We can potentially represent each pixel using fewer dimensions
    • We’ll call the principal components “eigenpixels”

  – If we have a set of images (monochrome)
    • We have a collection of vectors, each represents an entire image
    • We can potentially represent each image using a linear combination of “basis” images
    • The number of basis images can be much smaller than our original collection
    • We’ll call the principal components “eigenimages”
We have an image where each pixel is composed of 6 values $[x_1, x_2, x_3, x_4, x_5, x_6]$

But some of these values may be redundant (e.g., middle wavelength infrared may be very similar to long wavelength infrared)

We may be able to represent each pixel using fewer than 6 values

**FIGURE 11.38** Multispectral images in the (a) visible blue, (b) visible green, (c) visible red, (d) near infrared, (e) middle infrared, and (f) thermal infrared bands. (Images courtesy of NASA.)
Example

• Each pixel is a 6-element vector
• Image size is 564x564 = 318,096
• So we have this many vectors

• Compute mean, covariance, eigenvalues and eigenvectors

\[
x = \begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6 \\
\end{bmatrix}
\]

\[
\lambda_1 \quad \lambda_2 \quad \lambda_3 \quad \lambda_4 \quad \lambda_5 \quad \lambda_6
\]
\[
10344 \quad 2966 \quad 1401 \quad 203 \quad 94 \quad 31
\]

**FIGURE 11.39**
Formation of a vector from corresponding pixels in six images.

**TABLE 11.6**
Eigenvalues of the covariance matrices obtained from the images in Fig. 11.38.
Coefficients of the $\mathbf{y}$-vectors

- Note that $y1$ has the most variation
  - This is to be expected because $\lambda_1$ is the variance of $y1$

$$C_y = \begin{pmatrix}
\lambda_1 & 0 \\
0 & \lambda_2 \\
& \ddots \\
0 & 0 & \cdots & \lambda_n
\end{pmatrix}$$

- and $\lambda_1$ is the largest eigenvalue

$$\mathbf{y} = \mathbf{A}(\mathbf{x} - \mathbf{m}_x)$$

**Figure 11.40** The six principal component images obtained from vectors computed using Eq. (11.4-6). Vectors are converted to images by applying Fig. 11.39 in reverse.
Example

- Reconstruction of all 6 values, using the top 2 principal components
- We only use \([y_1, y_2]\)
- \(x' = A_k^T y + m_x\)

**FIGURE 11.41** Multispectral images reconstructed using only the two principal component images corresponding to the two principal component images with the largest eigenvalues (variance). Compare these images with the originals in Fig. 11.38.
Example

- Error between reconstructed images and original images

**FIGURE 11.42** Differences between the original and reconstructed images. All difference images were enhanced by scaling them to the full [0, 255] range to facilitate visual analysis.
Eigenimage example

- We have a collection of images, \(I_1..I_K\)
  - We subtract off the mean of the collection of images
- We transform each \(MxN\) image \(I_i\) into a column vector
  \[
x_i = [I_i(1,1), I_i(1,2), \ldots, I_i(1,N), I_i(2,1), \ldots, I_i(M,N)]^T
  \]
- We put the column vectors into a single matrix \(B\)
Eigenimage example

- Covariance of our vectors

\[
C_x = E[(x - m_x)(x - m_x)^T] = \frac{1}{K} \sum_{k=1}^{K} x_k x_k^T - m_x m_x^T
\]

- since we have subtracted off the mean from the input vectors, \( m_x = 0 \)
- So

\[
C_x = \frac{1}{K} \sum_{k=1}^{K} x_k x_k^T = \frac{1}{K} BB^T
\]

- As before, let \( A \) be the matrix whose rows are the eigenvectors of \( C_x \)
  - The eigenvectors (principal components) represent “basis” images
  - We’ll call them “eigenimages”
Application - Face images

Input images (Yale database)

from: http://www.cs.princeton.edu/~cdecoro/eigenfaces/
Eigenfaces

- 10 PC’s capture 36% of the variance
- 25 PC’s capture 56% of the variance
Reconstruction with a small number of Principal Components

Each new image from left to right corresponds to using 1 additional PC for reconstruction.

Each new image from left to right corresponds to using 8 additional PC for reconstruction.
Note on computational expense

- **Size of $B$**
  - Number of rows in $B$ = the number of pixels in the image
  - Number of columns in $B$ = the number of images in our collection
  - Typically #rows >> #columns

- **So $BB^T$ is a large matrix**
  - It is square (size = number of pixels in the image squared)

- **We want to compute the eigenvectors of $BB^T$; ie find the vectors $u_i$ such that $(BB^T)u_i = \lambda_i u_i$**
  - It can be expensive to compute eigenvectors of a large matrix
Note on computational expense (continued)

• Look at $B^T B$, a much smaller matrix
  – It is square (size = number of images squared)

• We compute the eigenvectors of $B^T B$; ie find the vectors $v_i$ such that
  \[(B^T B)v_i = \lambda_i v_i\]

• If we multiply each side by $B$
  \[B (B^T B)v_i = B \lambda_i v_i \rightarrow (BB^T) (B v_i) = \lambda_i (B v_i)\]

• So the eigenvectors of $BB^T$ are the vectors $B v_i$ (we still need to scale them so that they are unit vectors)
  • This is a cheaper way to get the eigenvectors of $BB^T$ (at least the first $K$ eigenvectors)
Finding Matches in Eigenspace

• Say we want to find a match for an image \( I_1 \)

• One way to find a match is to look for an image \( I_2 \) such that the sum-of-squared differences is small
  
  – This is the (squared) Euclidean distance between the two image vectors \( x_1 \) and \( x_2 \)

• For large images, computing this is expensive because \( x_1 \) and \( x_2 \) are large vectors

• However, it is equivalent to computing the (squared) Euclidean distance between the vectors in eigenspace
  
  – This is much faster because the \( y \) vectors are small

\[
\|x_1 - x_2\|^2 = \left\| \sum_{i=1}^{n} y_{1i} e_i - \sum_{i=1}^{n} y_{2i} e_i \right\|^2 \\
\approx \left\| \sum_{i=1}^{k} y_{1i} e_i - \sum_{i=1}^{k} y_{2i} e_i \right\|^2 \\
= \left\| \sum_{i=1}^{k} (y_{1i} - y_{2i}) e_i \right\|^2 \\
= \sum_{i=1}^{k} (y_{1i} - y_{2i})^2 \\
= \|y_1 - y_2\|^2
\]
Approach

• Training phase
  – We read in a set of training images $x_1,...,x_n$ and put them in vector form
  – We compute the mean of the vectors, $m$, and subtract off the mean from each vector
  – We put the vectors in the columns of the matrix $B$
  – We compute the eigenvectors $v$ of $B^TB$
  – We compute the principal components $u$, using $U = BV$

• Testing phase
  – We read in a new image $x$ and subtract off $m$
  – We project that onto the space of PCs, using $y = U^Tx$
  – We find the closest match of $y$ to the other images in the database
AT&T Face Database

• Located at
  – Download att_faces.zip

• Extract to a directory called “images”

• Code on next slides
  – Loads images of a subset of people
  – Displays the images
  – (This code was developed by Chris Ostrum, CSM student)
function [faces, irow, icol, resize, pidx, fidx] = load_faces(directory, person_count, person_max, face_count, face_max, ext)
% This function loads the face images
% Chris Ostrum
resize = false;
fidx = 1;

% test image size
img = imread(strcat(directory,'1/1',ext));
img = img(:,:,1);
irow,icol = size(img);
if( size(img,1) > 400 )
    img = imresize(img,[200, 200]);
    resize = true;
end

% setup variables
[irow,icol] = size(img);
img_size = irow*icol;
v=zeros(img_size,person_count*face_count);
w=zeros(img_size,person_count*face_count);

% create a random permutation of the people to pick from
people = randperm(person_max);
% choose a random person as our recognition person
rand_idx = mod(round(person_count*rand),person_count)+1;
% pidx is now the randomly selected person index
pidx = people(rand_idx);

for i=1:person_count
    % similarly, create a random permutation of possible faces
    faces = randperm(face_max);

    for j=1:face_count
        file = strcat(directory,num2str(people(i)),'/',num2str(faces(j)),ext);

        % pre-processing
        img = imread(file);
        if( resize ), img=imresize(img,[irow icol]); end
        img = img(:,:,1);

        w(:,(i-1)*face_count+j)=reshape(img,img_size,1);
    end

% select our random face that is not in the training set
if( i == rand_idx )
    fidx = faces(face_count+1);
end
end

faces = uint8(w);
end

Function to load face images
Function to display faces

```matlab
function display_faces(face_space,irow,icol,sub_row,sub_col)
% Face Display
%   Chris Ostrum
[H W] = size(face_space);

%figure('Name','Face Display', 'NumberTitle','off', 'MenuBar', 'none')
figure('Name','Face Display')
for i = 1:W
    subplot(sub_row,sub_col,i);
    face = reshape(face_space(:,i),irow,icol);
    imshow(face,[]);
end
end
```
%% Chris Ostrum Eigenface recognition
% This algorithm uses the eigenface system (based on principal component analysis - PCA) to recognize faces.
clear all; close all;

directory = 'att_faces/s'; ext = '.pgm'; person_max = 40; face_max = 10;

person_count = 10; % up to person_max
face_count = 9; % up to face_max-1; leaving at least one unknown for recognizing

% Principal components to keep
N = ceil(.25 * (person_count*face_count));
fprintf('Keep only the top %d principal components out of %d\n', N, person_count*face_count);

if ( person_count > person_max || face_count > face_max-1 )
    fprintf('Count values cannot exceed maximums\n');
    return;
end

[faces,irow,icol,resize,pidx,fidx] = ...% load_faces(directory,person_count,person_max,face_count,face_max,ext);
if person_count <= 10
    display_faces(faces,irow,icol,person_count,face_count);
end
Pick a testing image

%% Choose recognition image
% We choose an image not in the training set as our image to identify
% Face to recognize
file = strcat(directory,num2str(pidx),'/',num2str(fidx),ext);

recognize = imread(file);
if( resize ), recognize = imresize(recognize,[irow icol]); end
recognize = recognize(:,:,1);

figure, imshow(recognize,:), title('Face to recognize');
Calculate the mean image

• Mean

```matlab
m = uint8(mean(faces, 2));
figure, imshow(reshape(m, irow, icol), []), title('Mean face');
```

• Subtract off mean from all faces

```matlab
faces_mean = faces - uint8(single(m) * single(uint8(ones(1, size(faces, 2)))));
```
Calculate Eigenfaces

%% Calculating eigenvectors
% L = A'A
L=single(faces_mean)'*single(faces_mean);
[V,D]=eig(L);

% Plot the eigenvalues. Matlab's "eig" function sorts them from low to
% high, so let's reverse the order for display purposes.
eigenvals = diag(D);
figure, plot(eigenvals(end:-1:1)), title('Eigenvalues');

% Look at the mean squared error, as we increase the number of PCs to
% keep.
figure, plot(sum(eigenvals) - cumsum(eigenvals(end:-1:1)));
title('Mean squared error vs number of PCs');

% Here are the principal components.
PC=single(faces_mean)*V;

% Pick the top N eigenfaces
PC=PC(:,end:-1:end-(N-1));

display_faces(PC(:,1:10),irow,icol,1,10);   % Display first 10 eigenfaces
Calculate Eigenspace Coefficients

- Project each of the input database images $\mathbf{x}$ onto the space of eigenimages, and get the coefficients $\mathbf{y} = \mathbf{U}^T \mathbf{x}$
- Save the $\mathbf{y}$’s for each database image

```matlab
%%% Calculate image signature
signatures=zeros(size(faces,2),N);
for i=1:size(faces,2);
    signatures(i,:)=single(faces_mean(:,i))'*PC; % Each row is an image signature
end
figure, imshow(signatures, [], 'InitialMagnification', 300);
title('Signatures of images in database');
```
Recognition

%% Recognition
% Now run the algorithm to see if we are able to match the new face
figure('Name','Result', 'NumberTitle','off', 'MenuBar','none')
subplot(231),imshow(recognize,[],),title('Face to recognize');

% Prepare the recognition face
rec=reshape(recognize,irow*icol,1)-m;
rec_weighted=single(rec)'*PC;
fprintf('Here is the signature of the new (input) face.
');
fprintf('These are the coefficients of the face projected onto the %d PCs:
', N);
disp(rec_weighted);

scores=zeros(1,size(signatures,1));
for i=1:size(faces,2)
    % calculate Euclidean distance as score
    scores(i)=norm(signatures(i,:)-rec_weighted,2);
end

% display results
[C,idx] = sort(scores,'ascend');
fprintf('Top 3 scores: %f, %f, %f
', C(1), C(2), C(3));

subplot(234);
imshow(reshape(faces(:,idx(1)),irow,icol),[]),title('Best match');
subplot(235);
imshow(reshape(faces(:,idx(2)),irow,icol),[]),title('2nd best');
subplot(236);
imshow(reshape(faces(:,idx(3)),irow,icol),[]),title('3rd best');

- Put the test image into vector form, find its eigenspace coefficients
- Find the distance between its vector of coefficients, against each image in the database
Summary / Questions

• In principal components analysis (PCA), we map a set of vectors into another coordinate system.

• The axes of that coordinate system are called “principal components” (PCs).
  – The PCs are sorted so that the first PC corresponds to the direction of the most variation in the input data, the second PC corresponds to the direction of the second most variation, and so on.
  – If we use only the top $k$ PCs for reconstruction, we can still get a good approximation to the input data.

• What two ways to use PCA on images were demonstrated?