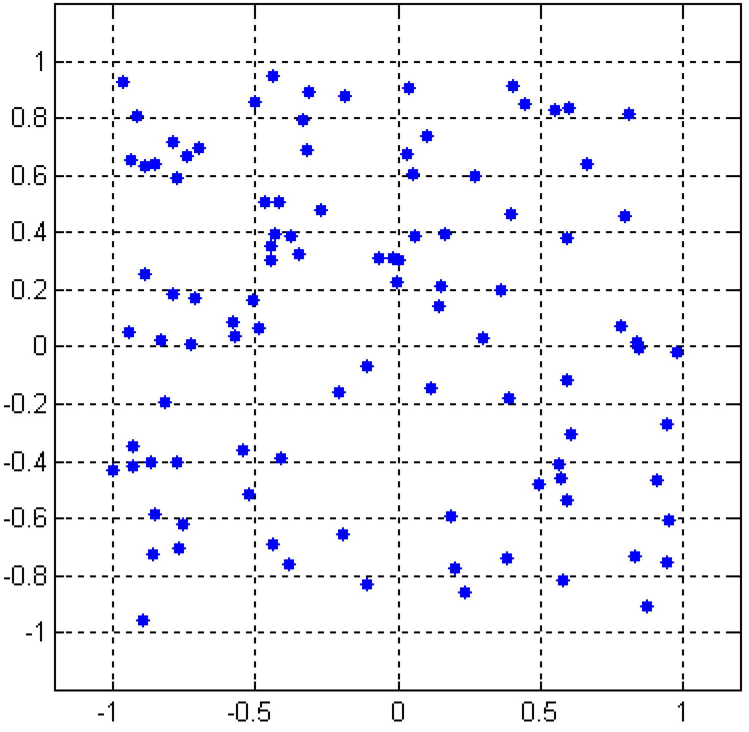
Principal Component Analysis

CSE 6367 – Computer Vision
Vassilis Athitsos
University of Texas at Arlington

Vector Spaces

- For our purposes, a vector is a tuple of d numbers: X = (x1, x2, ..., xd).
- An example vector space: the 2D plane.
  - Every point in the plot is a vector.
    - Specified by two numbers.



Images are Vectors

- An M-row x N-column image is a vector of dimensionality ...

Images are Vectors

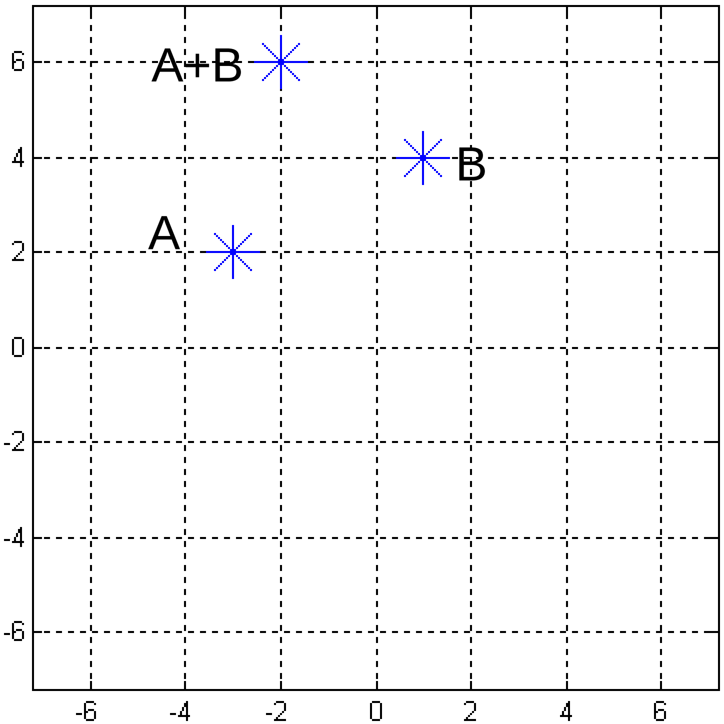
- An M-row x N-column image is a vector of dimensionality.
  - M*N if the image is grayscale.
  - M*N*3 if the image is in color.

Images are Vectors

- Consider a 4x3 grayscale image:
  - ```
    A = [A11 A12 A13
         A21 A22 A23
         A31 A32 A33
         A41 A42 A43];
    ```
- The (Matlab) vector representation Av of A is:
  ```
  Av = [A11 A21 A31 A41 A12 A22 A32 A42 A13 A23 A33 A43];
  ```
- Mathematically, order does not matter, IF IT IS THE SAME FOR ALL IMAGES.
- In Matlab, to vectorize an image:
  - Av = A(:);
  - NOTE: The above returns a COLUMN vector.
  ```

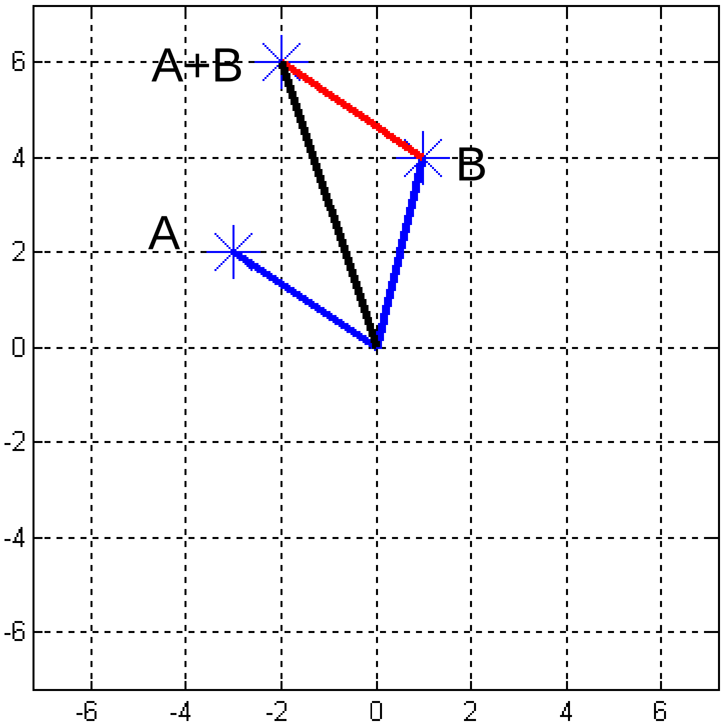
Vector Operations: Addition

- (x1, ..., xd) + (y1, ..., yd) = (x1+y1, ..., xd+yd)
- Example: in 2D:
  - A = (-3, 2)
  - B = (1, 4)
  - A+B = (-2, 6).

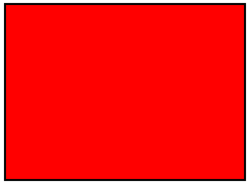


Vector Operations: Addition

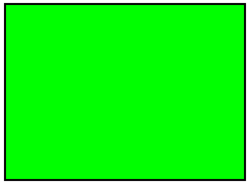
- (x1, ..., xd) + (y1, ..., yd) = (x1+y1, ..., xd+yd)
- Example: in 2D:
  - A = (-3, 2)
  - B = (1, 4)
  - A+B = (-2, 6).



Addition in Image Space



+



=

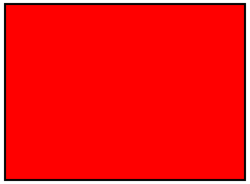
MxN image
all pixels (255, 0, 0)

MxN image
all pixels (0, 255, 0)

+

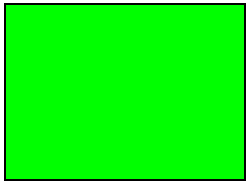
=

Addition in Image Space



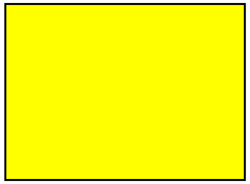
MxN image
all pixels (255, 0, 0)

+



MxN image
all pixels (0, 255, 0)

=



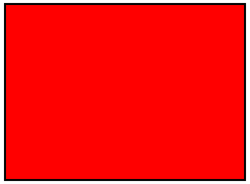
MxN image
all pixels (255, 255, 0)

+

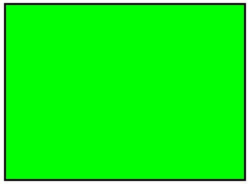
=

What is the range
of pixel values here?

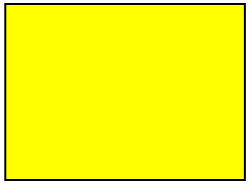
Addition in Image Space



+



=



MxN image
all pixels (255, 0, 0)

MxN image
all pixels (0, 255, 0)

MxN image
all pixels (255, 255, 0)

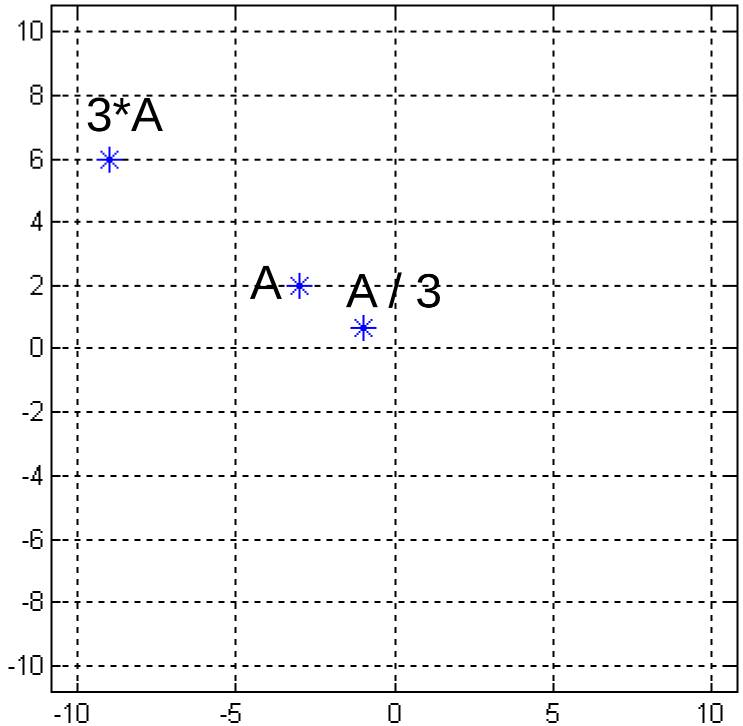
+

=

Range of pixel values:
from 0 to 2*255

Vector Operations: Scalar Multiplication

- c * (x1, ..., xd) = (c* x1, ..., c * xd)
- (x1, ..., xd) / c = 1/c * (x1, ..., xd)
  = (x1/c, ..., xd/c)
- Example: in 2D:
  - A = (-3, 2)
  - 3 * A = (-9, 6)
  - A / 3 = (-1, 0.67).



Multiplication in Image Space

image

image * 0.7

image * 0.5

Operations in Image Space

- Note: with addition and multiplication we can easily generate images with values outside the [0, 255] range.
  - These operations are perfectly legal.
  - However, we cannot visualize the results directly.
    - Must convert back to [0 255] range to visualize.

Linear Combinations

- Example: c1 * v1 + c2 * v2 + c3 * v3
  - c1, c2, c3: real numbers.
  - v1, v2, v3: vectors
  - result:

Linear Combinations

- Example: $c1 * v1 + c2 * v2 + c3 * v3$
  - c1, c2, c3: real numbers.
  - v1, v2, v3: vectors
  - result: vector.

Vectors Must Be of Same Size

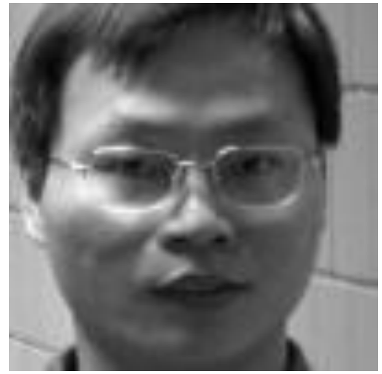
- We cannot add vectors that are not of the same size.
  - (1,2) + (0,1,0) is NOT DEFINED.
- To add images A and B:
  - IMPORTANT: Most of the time, it only makes sense to add A and B <u>ONLY</u> if they have the same number of rows and the same number of columns.
  - WARNING: Matlab will happily do the following:

```
a = rand(4,3);
b = rand(6,2);
c = a(:) + b(:);
```

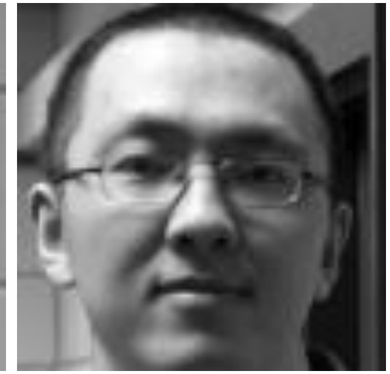
Example Linear Combination

a

b



c



d

e

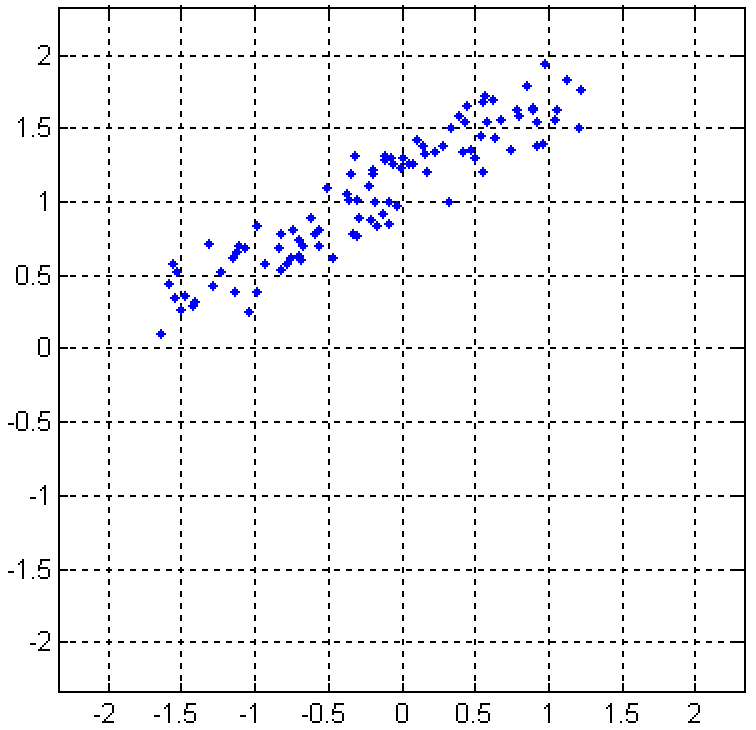
avg

```
a = read_gray('4309d111.bmp');
b = read_gray('4286d201.bmp');
c = read_gray('4846d101.bmp');
d = read_gray('4848d101.bmp');
e = read_gray('4853d101.bmp');
avg = 0.2*a + 0.2*b + 0.2*c + 0.2*d + 0.2*e;

% or, equivalently:
avg = (a+b+c+d+e) / 5;
```

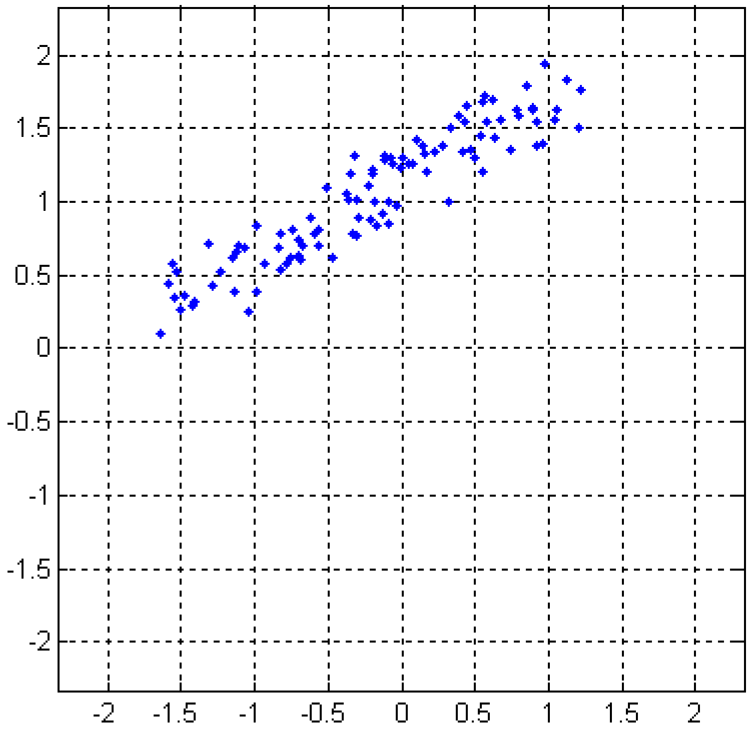
Dimensionality Reduction

- Consider a set of vectors in a d-dimensional space.
- How many numbers do we need to represent each vector?



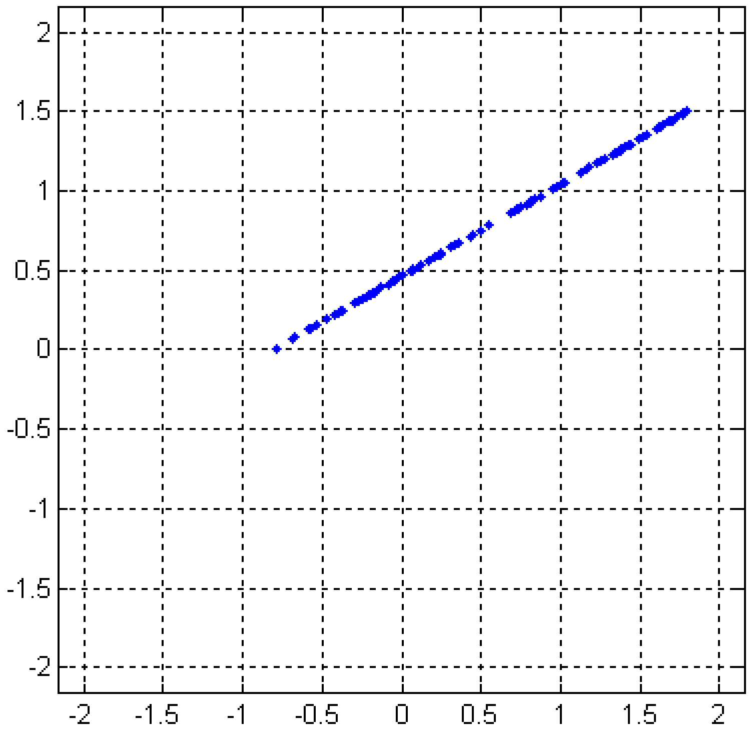
Dimensionality Reduction

- Consider a set of vectors in a d-dimensional space.
- How many numbers do we need to represent each vector?
  - At most d: the same as the number of dimensions.
- Can we use fewer?



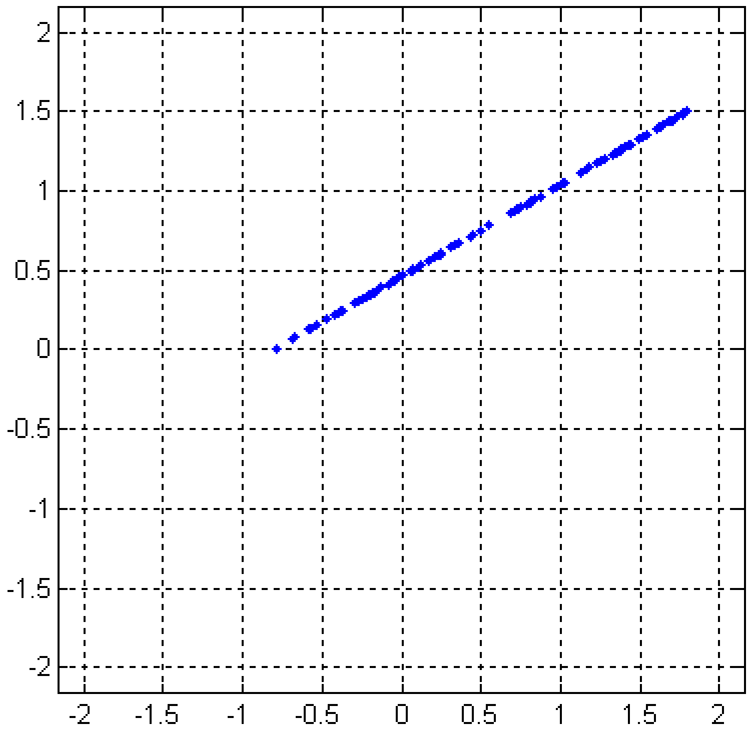
Dimensionality Reduction

- Consider a set of vectors in a d-dimensional space.
- How many numbers do we need to represent each vector?
  - At most d: the same as the number of dimensions.
- Can we use fewer?



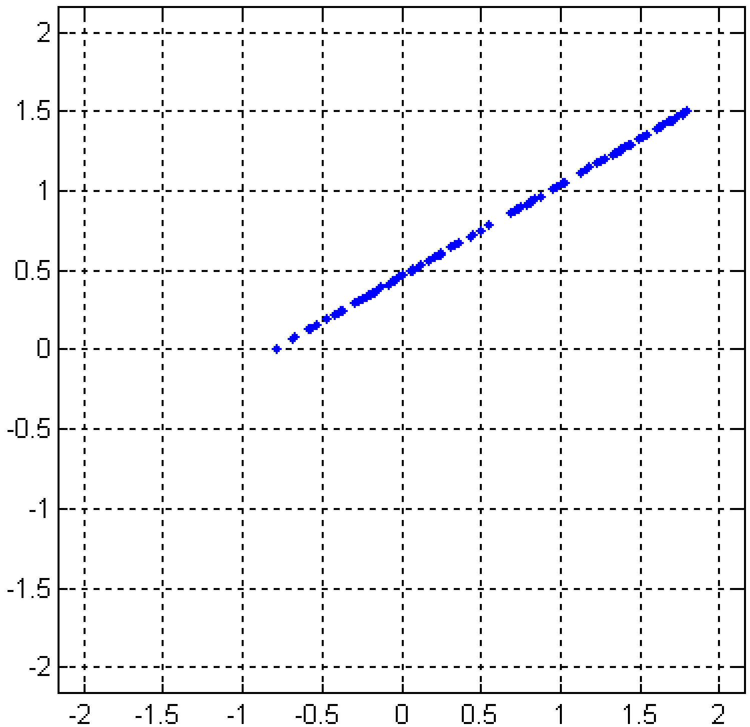
Dimensionality Reduction

- In this example, every point (x, y) is on a line
  - $y = ax + b$;
- If we have 100 points on this plot, how many numbers do we need to specify them?



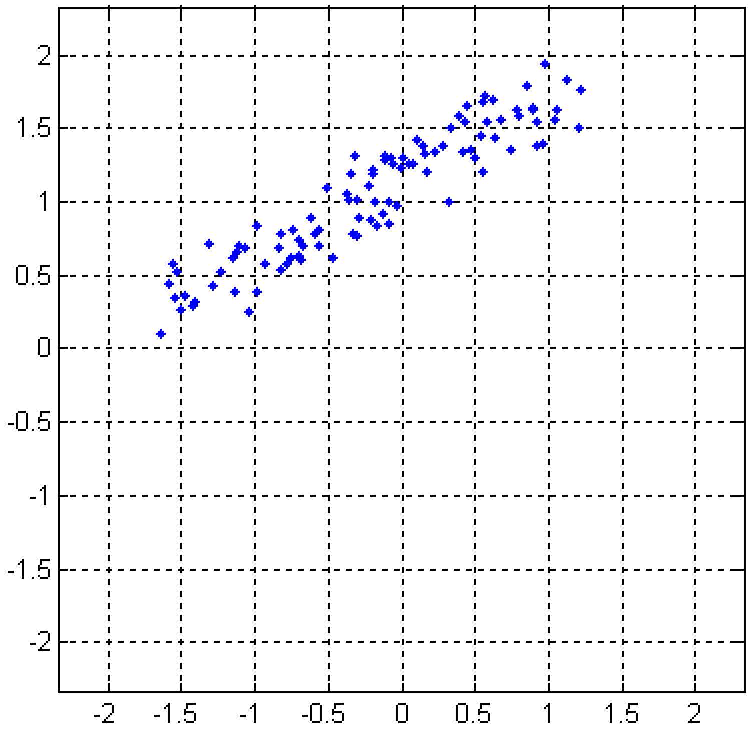
Dimensionality Reduction

- In this example, every point (x, y) is on a line
  - y = ax + b;
- If we have 100 points on this plot, how many numbers do we need to specify them?
  - 102: a, b, and the x coordinate of each point.
  - Asymptotically: one number per point.



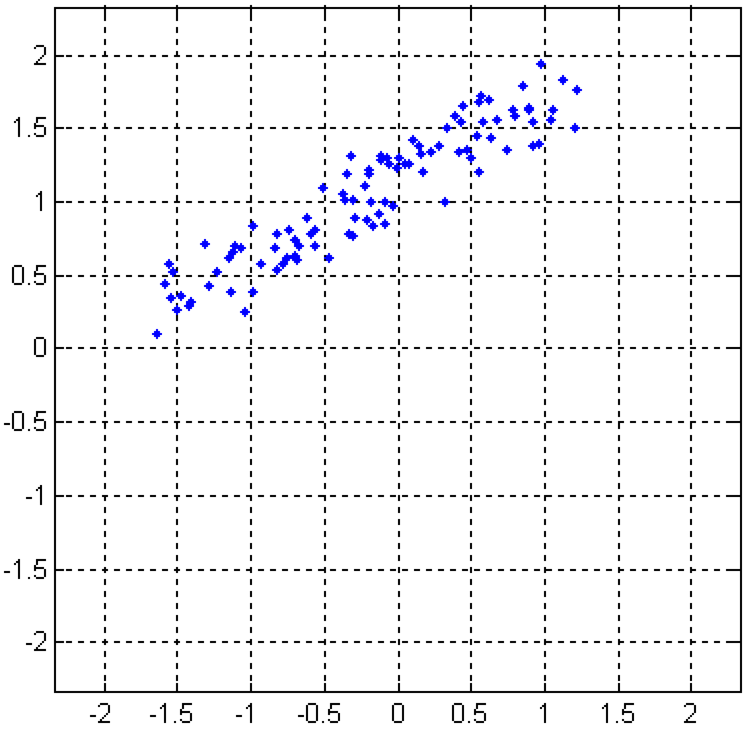
Lossy Dimensionality Reduction

- Suppose we want to project all points to a single line.
- This will be *lossy*.
- What would be the best line?



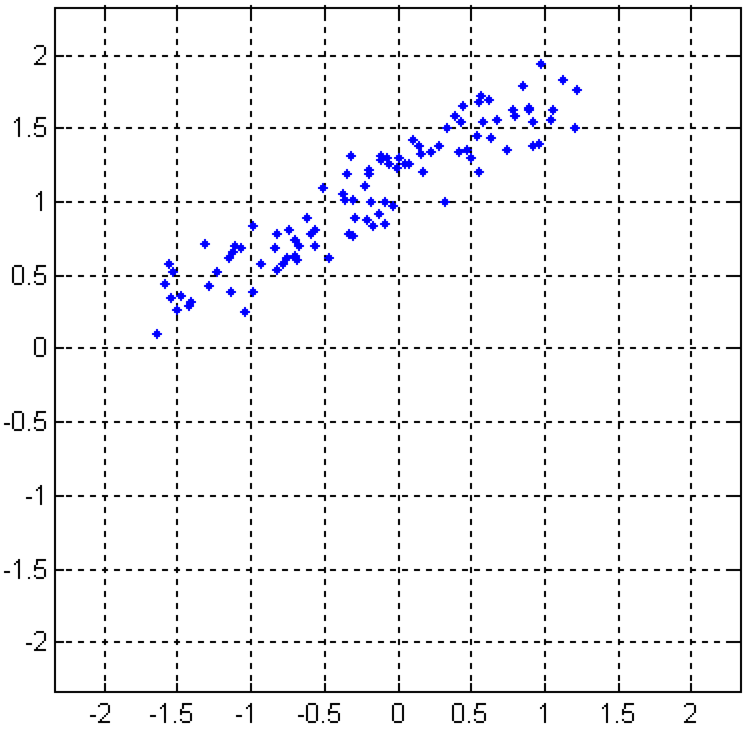
Lossy Dimensionality Reduction

- Suppose we want to project all points to a single line.
- This will be *lossy*.
- What would be the best line?
- Optimization problem.
  - Infinite answers.
  - We must define how to evaluate each answer.



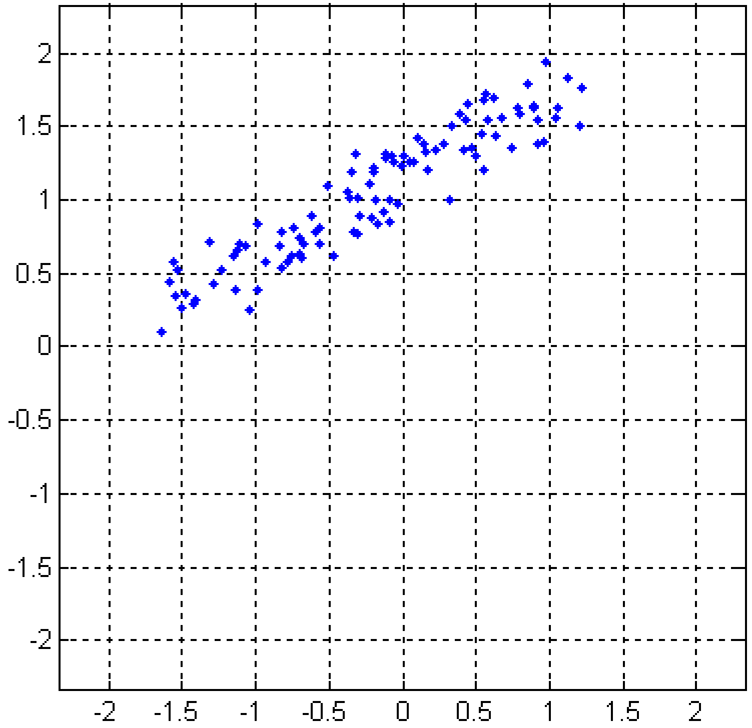
Optimization Criterion

- We want to measure how good a projection P is, GIVEN A SET OF POINTS.
  - If we don't have a specific set of points in mind, what would be the best projection?



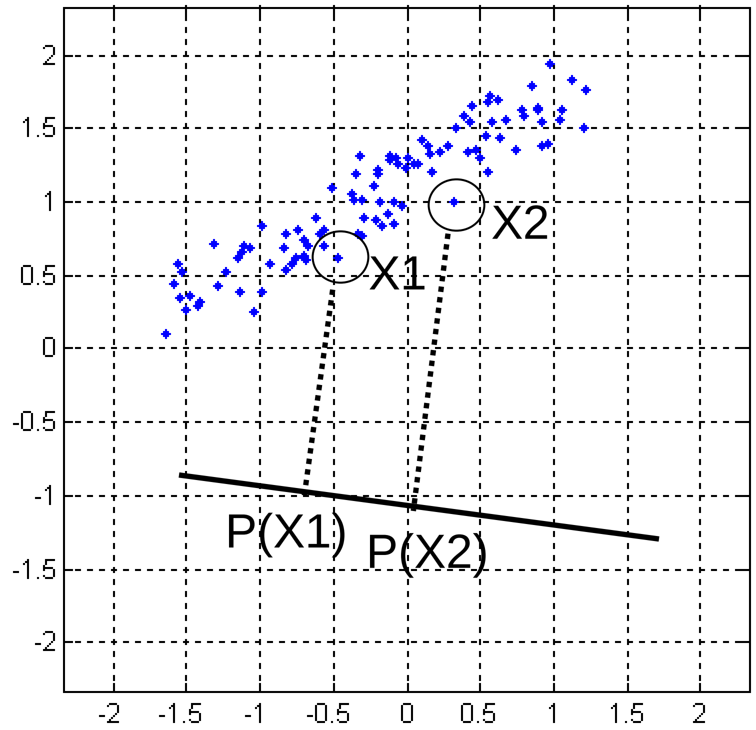
Optimization Criterion

- We want to measure how good a projection P is, GIVEN A SET OF POINTS.
  - If we don’t have a specific set of points in mind, what would be the best projection?
  - NONE: all are equally good/bad.



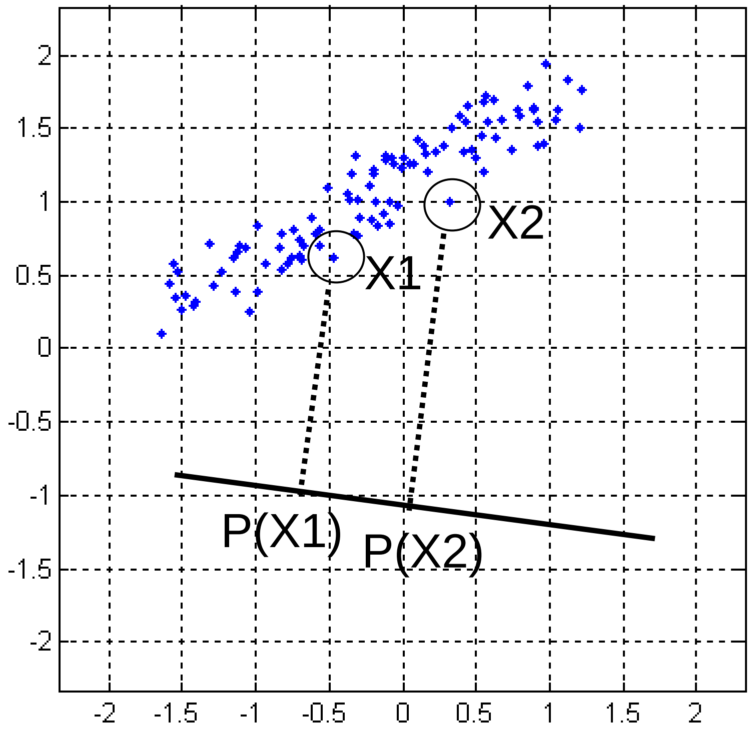
Optimization Criterion

- Consider a pair of points: X1, X2.
- D1 = squared distance from X1 to X2.
  - sum((X1–X2) .* (X1–X2))
- D2 = squared distance from P(X1) to P(X2).
- Error(X1, X2) = D1 – D2.
  - Will it ever be negative?



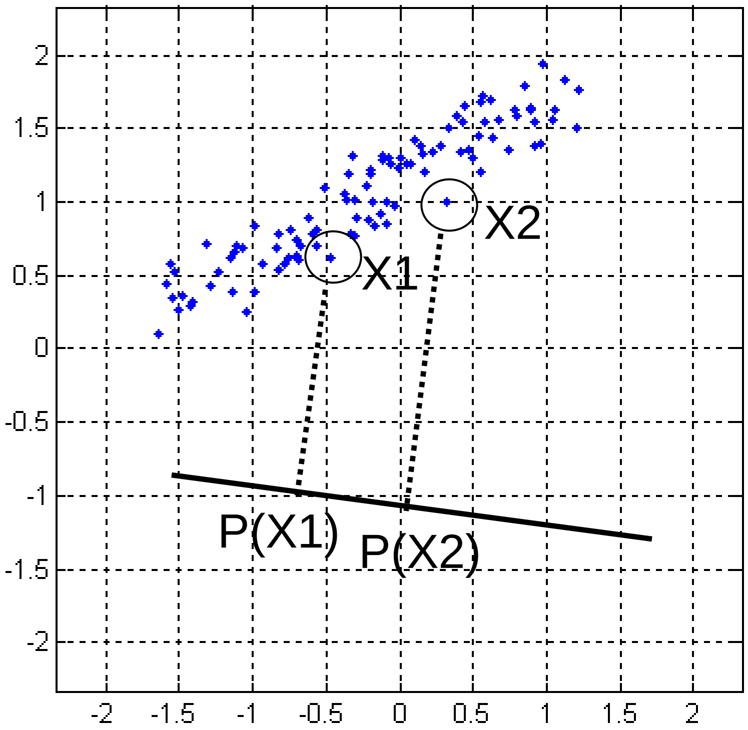
Optimization Criterion

- Consider a pair of points: X1, X2.
- D1 = squared distance from X1 to X2.
  - sum((X1–X2) .* (X1–X2))
- D2 = squared distance from P(X1) to P(X2).
- Error(X1, X2) = D1 – D2.
  - Will it ever be negative?
  - NO: D1 >= D2 always.



Optimization Criterion

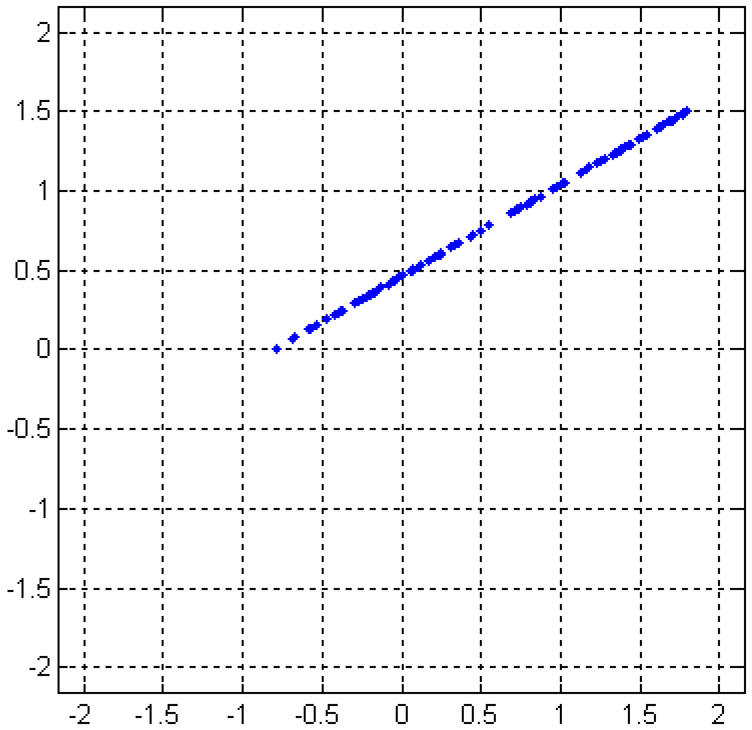
- Now, consider the entire set of points:
  - X1, X2, ..., Xn.
- Error(P) = sum(Error(Xi, Xj) | i, j = 1, ..., n, i != j).



- Interpretation:
  - We measure how well P preserves distances.
  - If P preserves distances, Error(P) = 0.

Example: Perfect Projection Exists

- In this case, projecting to a line oriented at 30 degrees would give zero error.



Finding the Best Projection: PCA

- First step: center the data.

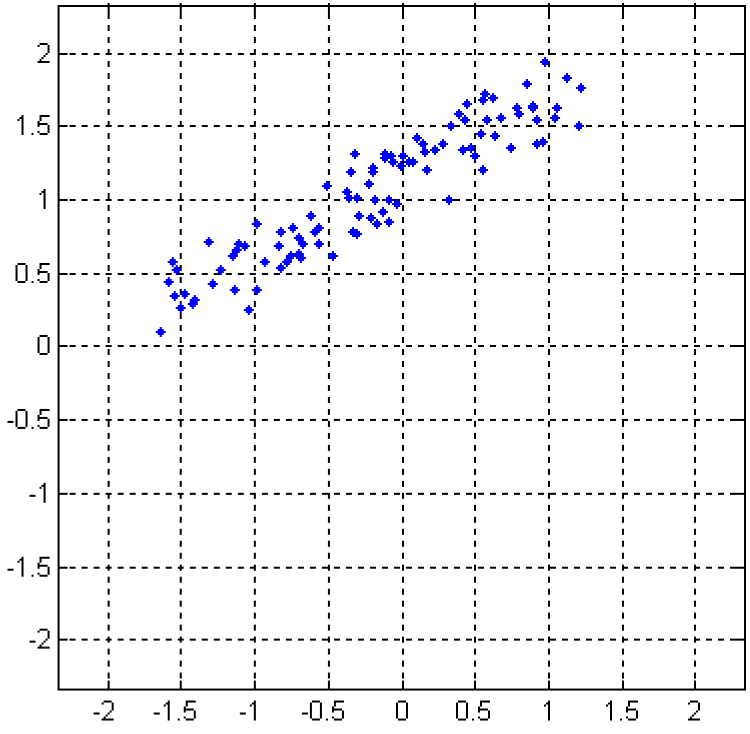
```
number = size(points, 2);
% note that we are transposing twice
average = [mean(points')]';
centered_points = zeros(size(points));

for index = 1:number
    centered_points(:, index) = points(:, index) - average;
end

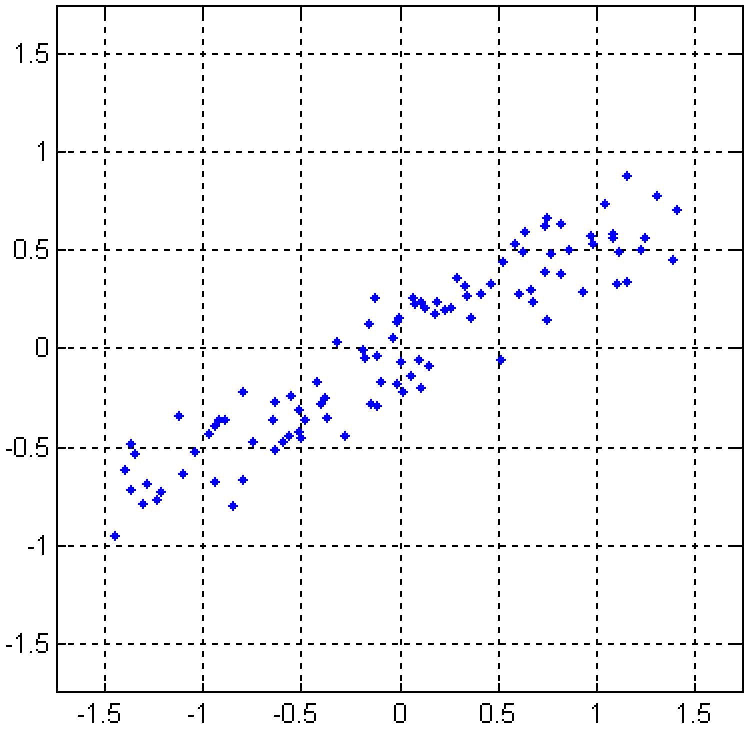
plot_points(centered_points, 2);
```

Finding the Best Projection: PCA

- First step: center the data.



points



centered_points

Finding the Best Projection: PCA

- Second step: compute the covariance matrix.

```
covariance_matrix = centered_points * centered_points';
```

- In the above line we assume that each column is a vector.

Finding the Best Projection: PCA

- Second step: compute the covariance matrix.

```
covariance_matrix = centered_points * centered_points';
```

- In the above line we assume that each column is a vector.
- Third step: compute the eigenvectors and eigenvalues of the covariance matrix.

```
[eigenvectors eigenvalues] = eig(covariance_matrix);
```

Eigenvectors and Eigenvalues

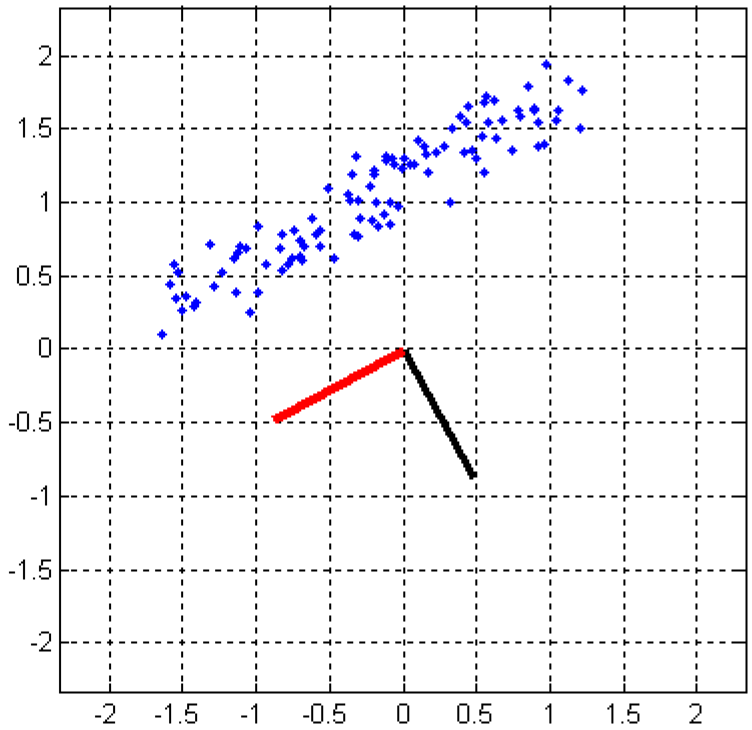
```
eigenvectors =
    0.4837   -0.8753
   -0.8753   -0.4837

eigenvalues =
    2.0217         0
         0   77.2183
```

- Each eigenvector v is a column, that specifies a line going through the origin.
- The importance of the i-th eigenvector is reflected by the i-th eigenvalue.
  - second eigenvalue = 77, first eigenvalue = 2, => second eigenvector is far more important.

Visualizing the Eigenvectors

black: v1 (eigenvalue = 2.02)
red: v2 (eigenvalue = 77.2)




```
plot_points(points, 1);
p1 = eigenvectors(:, 1);
p2 = eigenvectors(:, 2);
plot([0, p1(1)], [0, p1(2)], 'k-', 'linewidth', 3);
hold on;
plot([0, p2(1)], [0, p2(2)], 'r-', 'linewidth', 3);
```

PCA Code

```
function [average, eigenvectors, eigenvalues] = ...
                                     compute_pca(vectors)

number = size(vectors, 2);
% note that we are transposing twice
average = [mean(vectors')]';
centered_vectors = zeros(size(vectors));

for index = 1:number
    centered_vectors(:, index) = vectors(:, index) - average;
end

covariance_matrix = centered_vectors * centered_vectors';
[eigenvectors eigenvalues] = eig( covariance_matrix);

% eigenvalues is a matrix, but only the diagonal
% matters, so we throw away the rest
eigenvalues = diag(eigenvalues);
[eigenvalues, indices] = sort(eigenvalues, 'descend');
eigenvectors = eigenvectors(:, indices);
```

Reducing Dimensions From 2 to 1

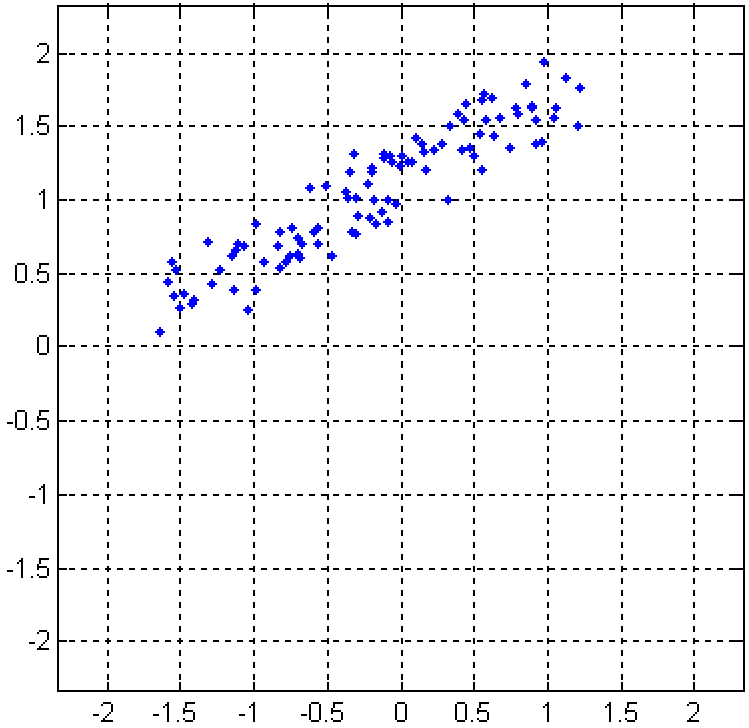
- Precompute:
  - the eigenvector P1 with the largest eigenvalue.
  - the mean **`avg`** of all the data.

Reducing Dimensions From 2 to 1

- Two key operations:
  - projection to lower-dimensional space.
  - backprojection to the original space.
- Projection:
  - P(V) = <V-avg, P1> = P1’ * (V – avg)
    - Dot product between (V-avg) and P1.
    - NOTE: The eigenvectors that Matlab returns have unit norm.
  - Backprojection:
    - B(P(V)) = P1 * P(V) + avg.

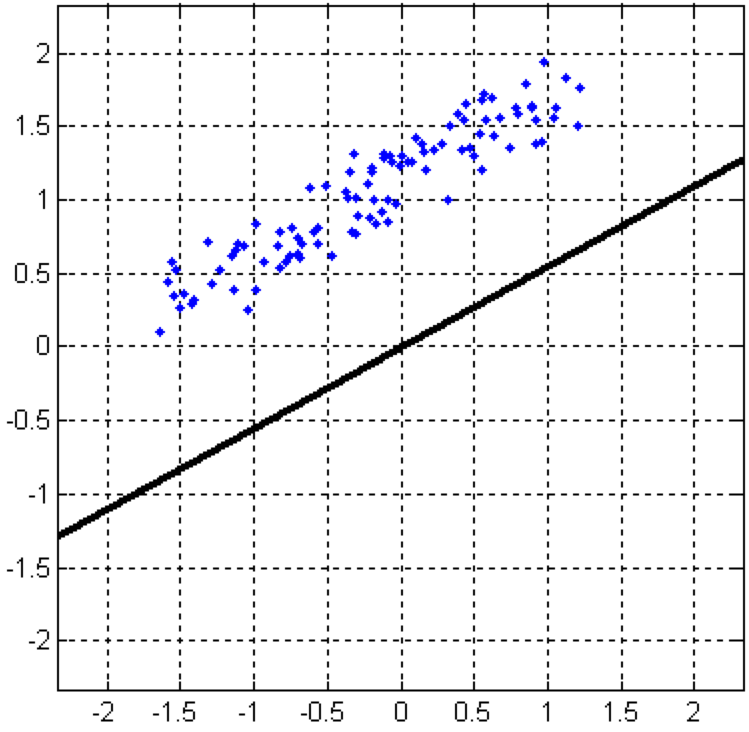
Example: From 2 Dimensions to 1

- A set of points.



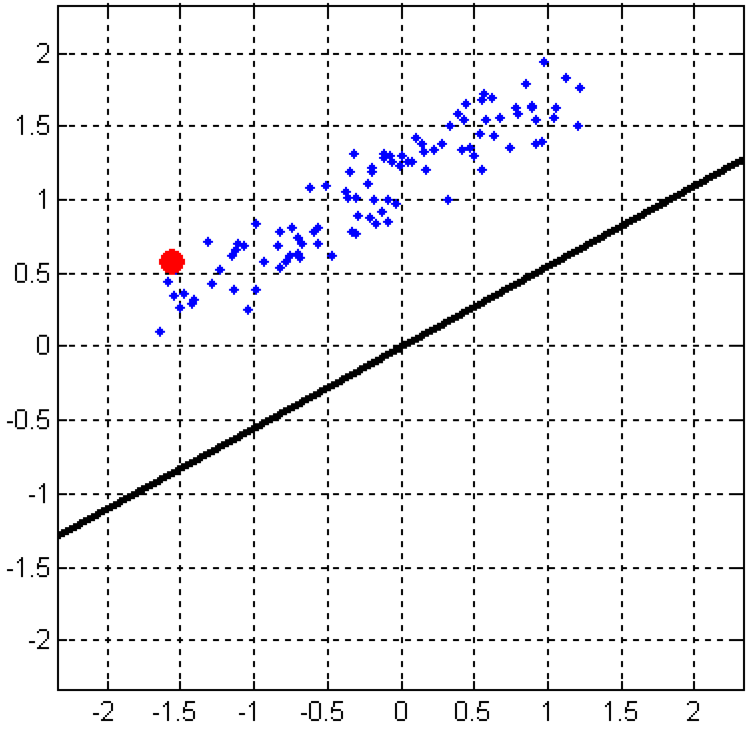
Example: From 2 Dimensions to 1

- Do PCA, identify p1 = the first eigenvector.
- We plot the line of direction of p1.



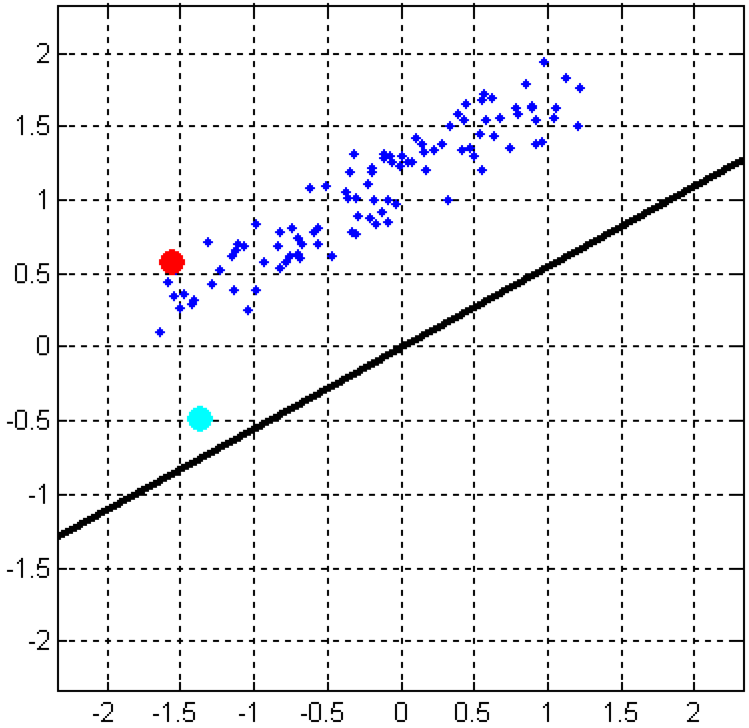
Example: From 2 Dimensions to 1

- Choose a point v4 = [-1.556, 0.576]’.
  - Shown in red.



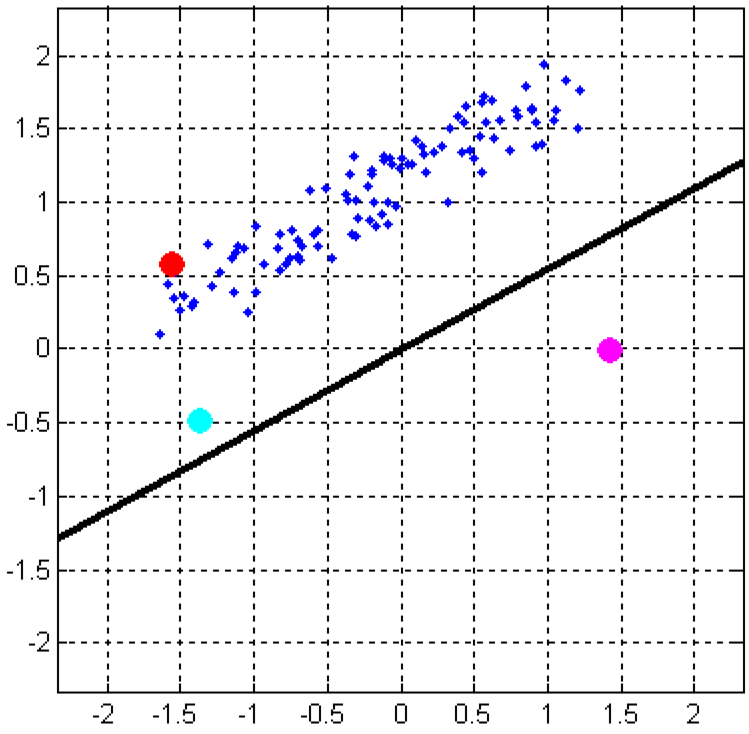
Example: From 2 Dimensions to 1

- centered_v4 = v4 – average.
  - Shown in cyan.



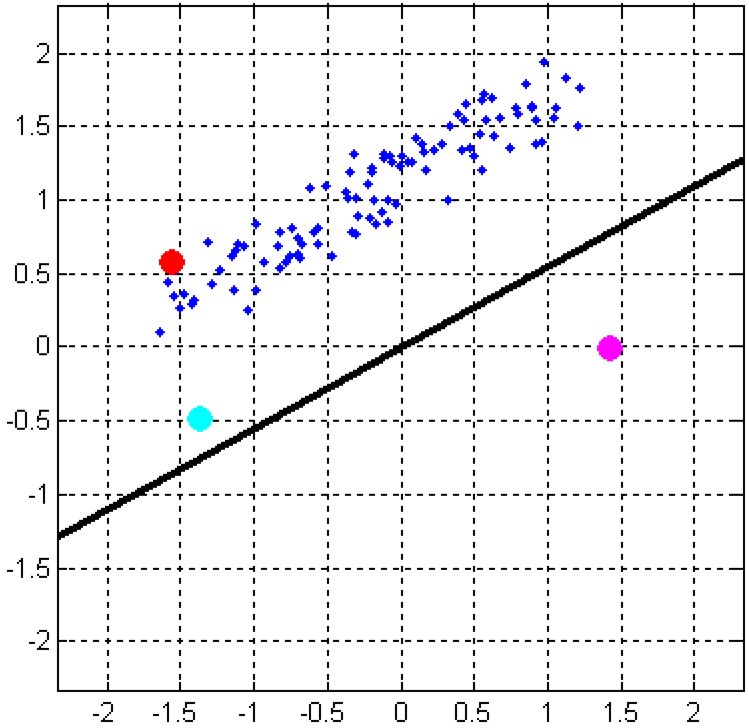
Example: From 2 Dimensions to 1

- projection = p1' * (centered_v4);
- result: projection = 1.43 (shown in magenta)



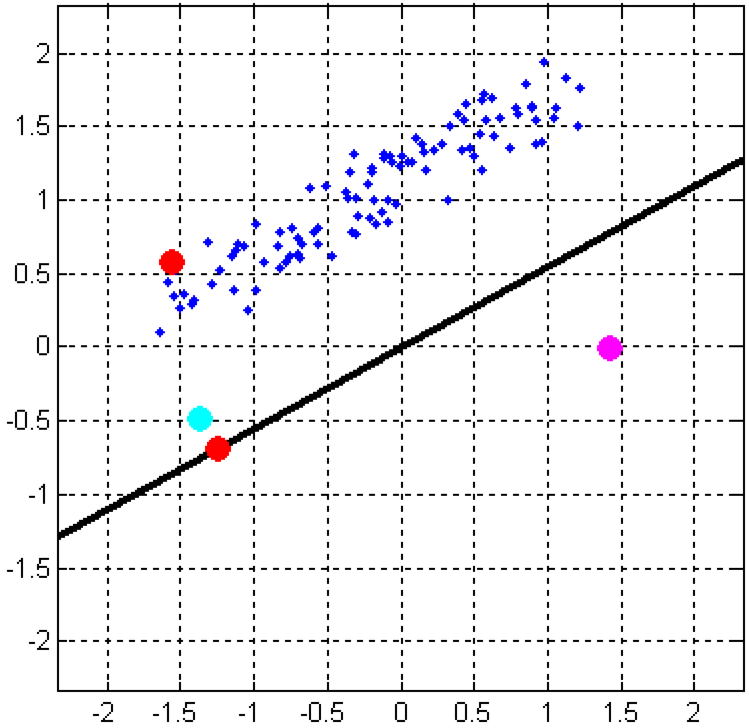
Example: From 2 Dimensions to 1

- Note: projection is a single number. To plot it, we actually treat that number as the x value, and use a y value of 0.



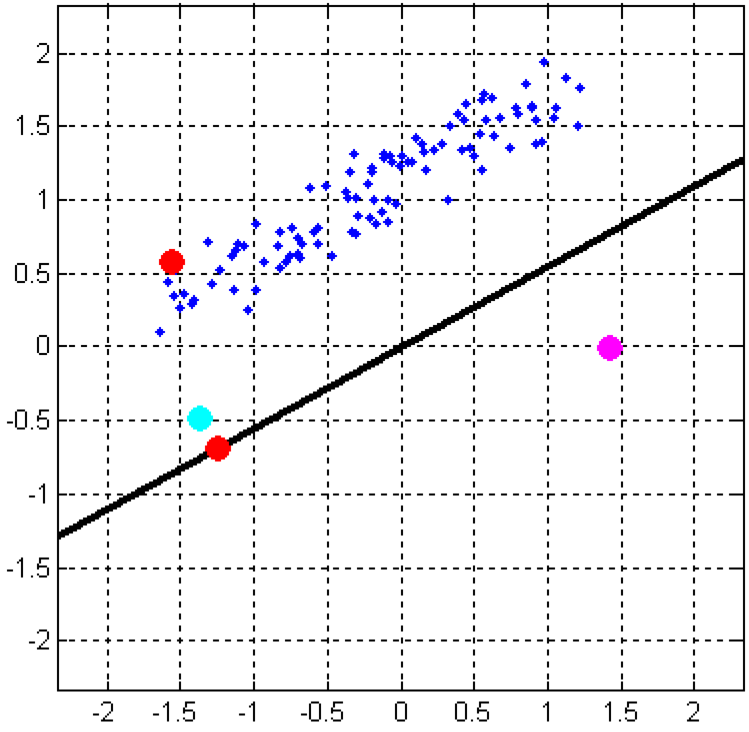
Example: From 1 Dimension to 2

- b1 = p1 * projection;
  - shown in red, on top of black line.
- How are b1 and projection related?



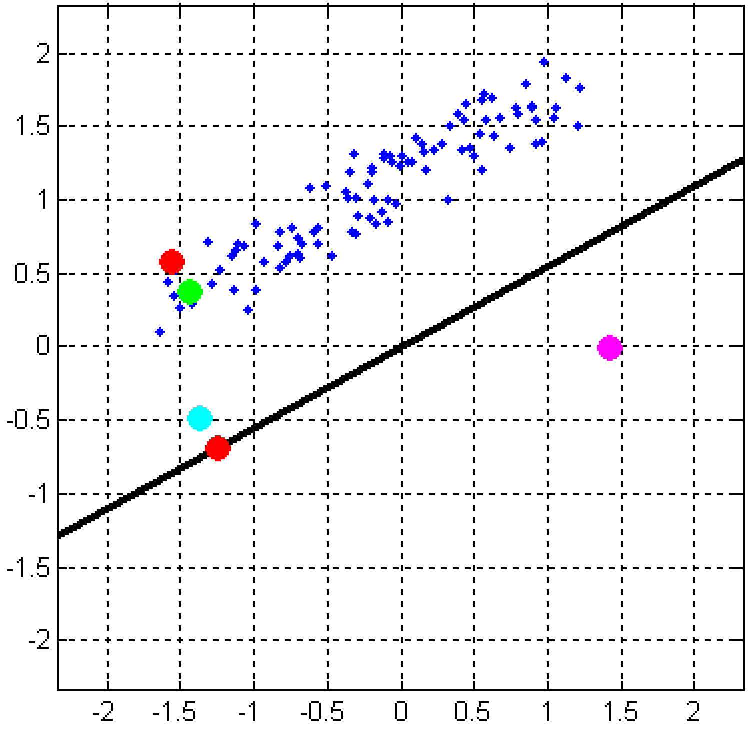
Example: From 1 Dimension to 2

- b1 = p1 * projection;
  - shown in red, on top of black line.
- projection = +-distance of b1 from the origin.



Example: From 1 Dimension to 2

- reconstructed v4 = b1 + average;
  - shown in green.



PCA on Faces

- Motivation: If a face is a 31x25 window, we need 775 numbers to describe the face.
- With PCA, we can approximate face images with much fewer numbers.
- Two benefits:
  - Faster computations.
  - The amount of loss can be used for face detection.

PCA vs Template Matching

- If we use template matching to detect faces, what is the perfect face (easiest to be detected)?
- How about PCA?

PCA vs Template Matching

- Template matching:
  - The average face is the only perfect face (after we normalize for brightness and contrast).
    - As a model, it is not rich enough.
- PCA:
  - There is a space of perfect faces.

Preparing a Dataset

- Get a set of face images.
  - They must be of same size, and aligned, so that eye locations match each other, nose locations match each other, and so on.
  - Make the mean and std of all faces equal.
    - Discards variations in intensity and contrast.

Code for Preprocessing Faces

```
clear;
load faces1032;

number = size(faces, 2);
dimensions = size(faces, 1);

% set mean of all faces to 0, and std to 1.
for index = 1: number
    face = faces(:, index);
    face = (face - mean(face)) / std(face);
    faces(:, index) = face;
end

% do pca
[mean_face, eigenvectors, eigenvalues] = compute_pca(faces);
```

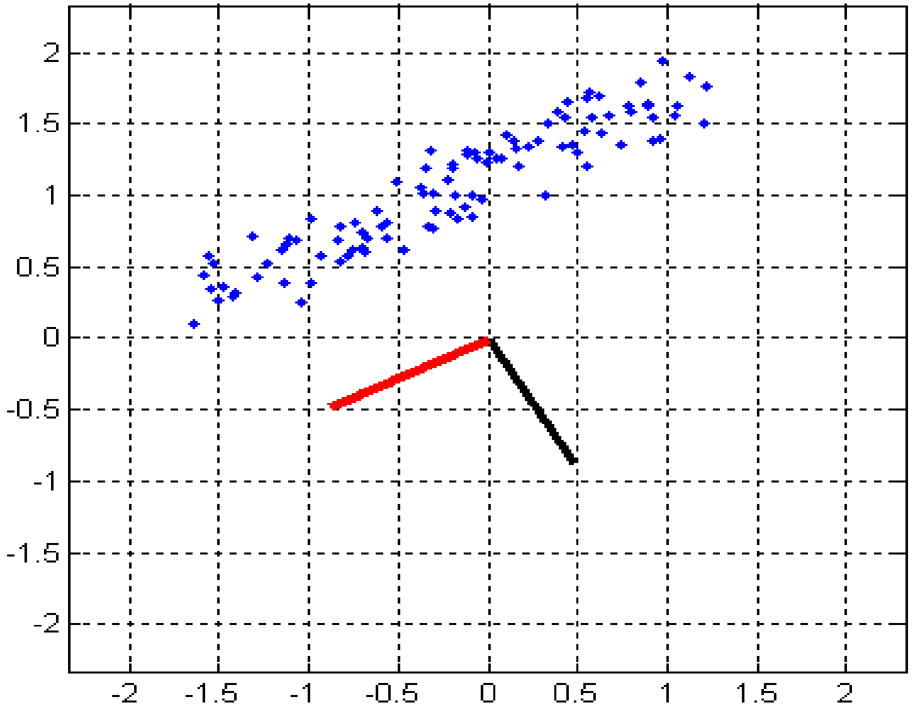
Visualizing Eigenfaces

Approximating a Face With 0 Numbers

- What is the best approximation we can get for a face image, if we know nothing about the face image (except that it is a face)?

Equivalent Question in 2D

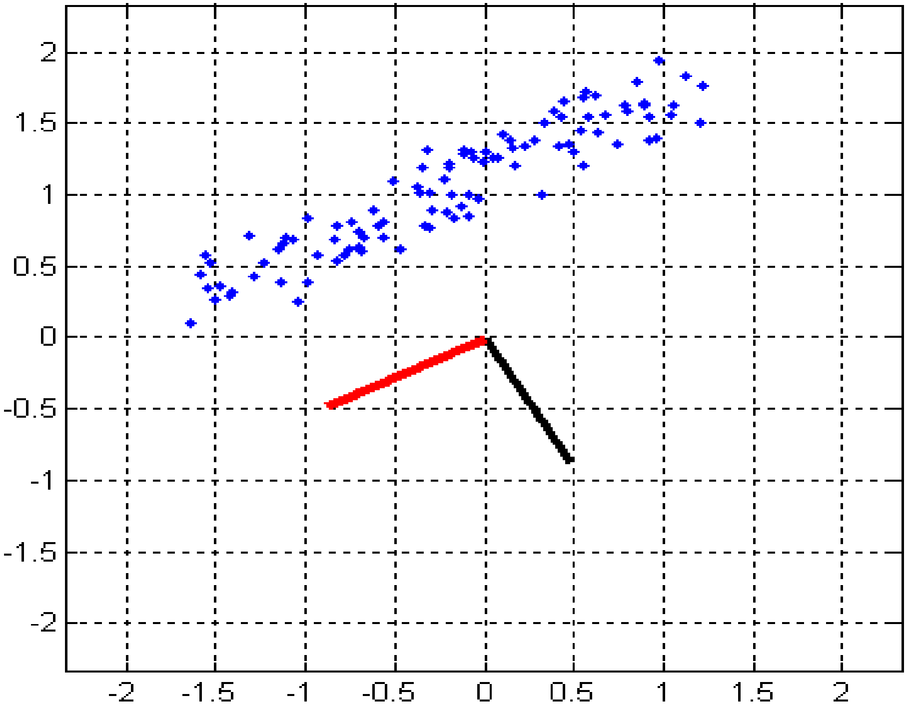
- What is our best guess for a 2D point from this point cloud, if we know nothing about that 2D point (except that it belongs to the cloud)?



Equivalent Question in 2D

- What is our best guess for a 2D point from this point cloud, if we know nothing about that 2D point (except that it belongs to the cloud)?

Answer: the average of all points in the cloud.



Approximating a Face With 0 Numbers

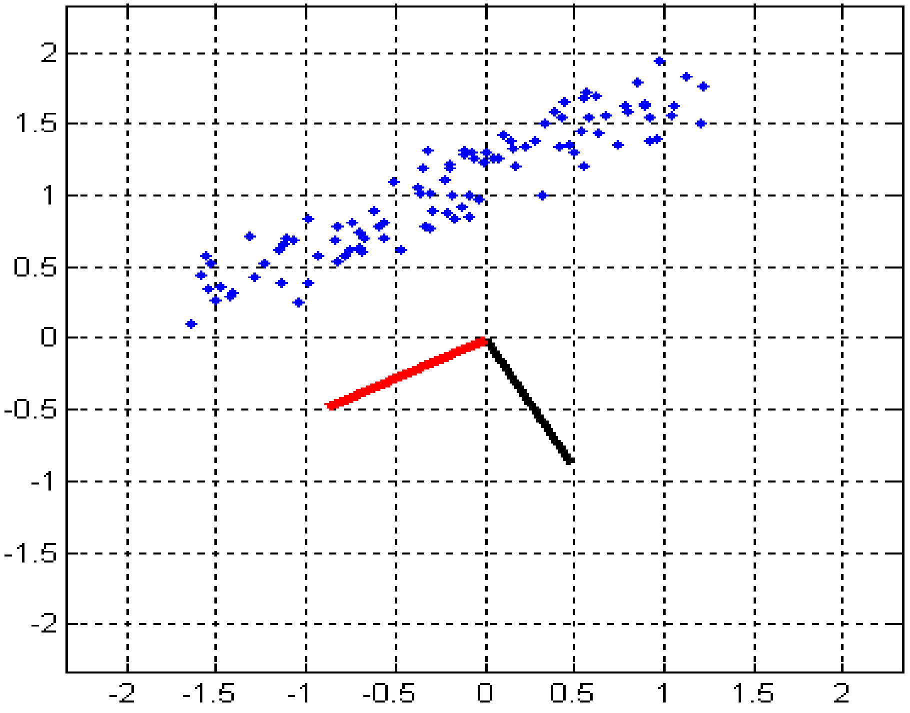
- What is the best approximation we can get for a face image, if we know nothing about the face image (except that it is a face)?

Approximating a Face With 0 Numbers

- What is the best approximation we can get for a face image, if we know nothing about the face image (except that it is a face)?
  - The average face.

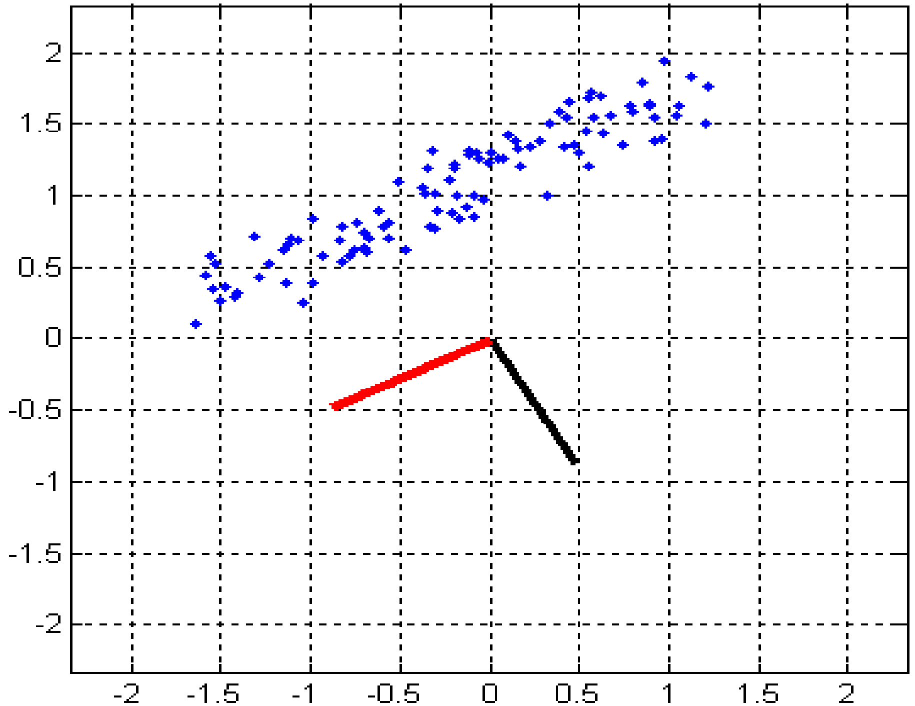
Guessing a 2D Point Given 1 Number

- What is our best guess for a 2D point from this point cloud, if we know nothing about that 2D point, except a single number?
  - What should that number be?



Guessing a 2D Point Given 1 Number

- What is our best guess for a 2D point from this point cloud, if we know nothing about that 2D point, except a single number?
  - What should that number be?
  - Answer: the projection on the first eigenvector.



Approximating a Face With 1 Numbers

- What is the best approximation we can get for a face image, if we can represent the face with a single number?

Approximating a Face With 1 Numbers

- With 0 numbers, we get the average face.
- 1 number: PCA projection to 1D.
- Reconstruction:
  - `average face + <face, eigenvector1> * eigenvector1`

Approximating a Face With 1 Number

```
v1 = faces(:, 13);
f1 = reshape(v1, size(mean_face));
k = 0;
%%
x1 = v1' * eigenvectors(:, 1);
term1 = x1 * eigenvectors(:, 1);
term1 = reshape(term1, size(mean_face));

reconstructed1 = mean_face + term1;
```

mean_face

Approximating a Face With 1 Number

```
v1 = faces(:, 13);
f1 = reshape(v1, size(mean_face));
k = 0;
%%
x1 = v1' * eigenvectors(:, 1);
term1 = x1 * eigenvectors(:, 1);
term1 = reshape(term1, size(mean_face));

reconstructed1 = mean_face + term1;
```

reconstructed1

Approximating a Face With 2 Numbers

```
x2 = v1' * eigenvectors(:, 2);
term2 = x2 * eigenvectors(:, 2);
term2 = reshape(term1, size(mean_face));

reconstructed2 = reconstructed1 + term2;
```

reconstructed2

Approximating a Face With 3 Numbers

```
x = v1' * eigenvectors(:, 3);
term = x * eigenvectors(:, 3);
term = reshape(term1, size(mean_face));

reconstructed3 = reconstructed2 + term;
```

reconstructed3

Approximating a Face With 4 Numbers

reconstructed4

Approximating a Face With 5 Numbers

reconstructed5

Approximating a Face With 6 Numbers

reconstructed6

Approximating a Face With 7 Numbers

reconstructed7

Approximating a Face With 10 Numbers

```
f1 = faces(:, 13);
f1 = reshape(f1, size(mean_face));

ev = eigenvectors(:, 1:10);
p = pca_projection(f1, mean_face, ev);
b = pca_backprojection(p, mean_face, ev);
reconstructed10 = reshape(b, size(mean_face));
```

reconstructed10

PCA Projection Code

```
function result = normalize_face(image_window, mean_face)

% function result = normalize_face(vector)
%
% normalizes the vector so that the size matches that of \
% mean face, the mean is 0 and the std is 1.

result = imresize(image_window, size(mean_face), 'bilinear');
result = result(:);
result = result - mean(result(:));
result = result / std(result(:));
result(isnan(result)) = 0;
```

PCA Projection Code

```
function result = eigenface_projection(image_window, ...
                                       mean_face, eigenvectors)

% function result = eigenface_projection(image_window, ...
%                                           average, eigenvectors)
%
% the vector is converted to a column vector. Each column in
% eigenvectors should be an eigenvector. The mean is converted
% to a column vector

normalized = normalize_face(image_window, mean_face);
% subtract mean from vector
centered = normalized(:) - mean_face(:);

% convert vector to a column vector.
result = eigenvectors' * centered;
```

PCA Backprojection Code

```
function result = pca_backprojection(projection, ...
                                     average, eigenvectors)

projection = projection(:);
centered_result = eigenvectors * projection;
result = centered_result + average(:);
```

- The PCA projection gives us a few numbers to represent a face.
- The backprojection uses those few numbers to generate a face image.

Projection/Backprojection Results


original


10 eigenfaces


40 eigenfaces


100 eigenfaces

Projection/Backprojection Results

original

10 eigenfaces

40 eigenfaces

100 eigenfaces

Projection/Backprojection Results

original

10 eigenfaces

40 eigenfaces

100 eigenfaces

Note: teeth not visible using 10 eigenfaces

Projection/Backprojection Results

original | 10 eigenfaces | 40 eigenfaces | 100 eigenfaces

Note: using 10 eigenfaces, gaze direction is towards camera

Projection/Backprojection Results

Note: using 10 eigenfaces, glasses are removed

Projecting a Non-Face

original

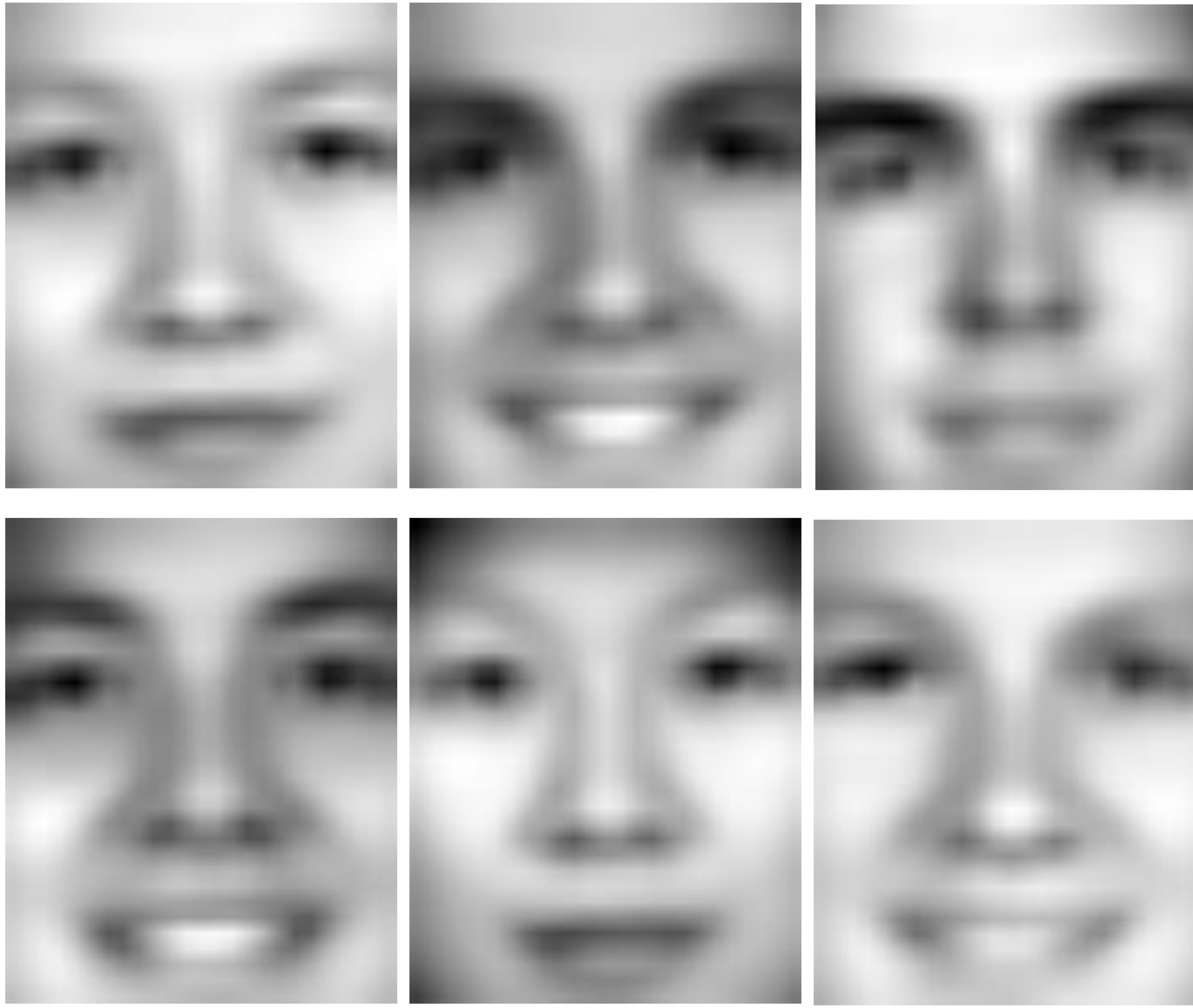
10 eigenfaces

40 eigenfaces

100 eigenfaces

How Much Can 10 Numbers Tell?

original



eigenfaces

Using Eigenfaces for Detection

- Intuition: Why are eigenfaces good for representing faces?
  - Because projection OF FACES to the eigenspace loses little information.
  - However, projection of NON-FACES to the eigenspace of faces loses more information.
- Criterion for face detection: how much is the reconstruction error?
- Quantitative:
  - Define f = face, b = backprojection(projection(f)).
  - Error: norm of (f – b).

Evaluating Error for a Window

```
function result = pca_score(window, mean_face, ...
                            eigenvectors, eigenface_number)

% normalize the window
window = imresize(window, size(mean_face), 'bilinear');
window = window(:);
window = window - mean(window);
window = window / std(window);
window(isnan(window)) = 0;

top_eigenvectors = eigenvectors(:, 1:eigenface_number);
projection = pca_projection(window, mean_face, ..
                            top_eigenvectors);
reconstructed = pca_backprojection(projection, mean_face, ...
                                   top_eigenvectors);

diff = reconstructed - window(:);
result = sum(diff .* diff);
```

Review: Correlation-based Detection

```
function [max_responses, max_scales, max_rotations] = ...
    template_search(image, template, scales, rotations, result_number)

% function [result, max_scales, max_rotations] = ...
%     template_search(image, template, scales, rotations, result_number)
%
% for each pixel, search over the specified scales and rotations,
% and record:
% - in result, the max normalized correlation score for that pixel
%   over all scales
% - in max_scales, the scale that gave the max score
% - in max_rotations, the rotation that gave the max score
%
% clockwise rotations are positive, counterclockwise rotations are
% negative.
% rotations are specified in degrees
```

Review: Correlation-based Detection

```
function [max_responses, max_scales, max_rotations] = ...
    template_search(image, template, scales, rotations, result_number)


max_responses = ones(size(image)) * -10;
max_scales = zeros(size(image));
max_rotations = zeros(size(image));

for rotation = rotations
    rotated = imrotate(image, -rotation, 'bilinear', 'crop');
    [responses, temp_max_scales] = ...
        multiscale_correlation(rotated, template, scales);
    responses = imrotate(responses, rotation, 'nearest', 'crop');
    temp_max_scales = imrotate(temp_max_scales, rotation, ...
                               'nearest', 'crop');
    higher_maxes = (responses > max_responses);
    max_responses(higher_maxes) = responses(higher_maxes);
    max_scales(higher_maxes) = temp_max_scales(higher_maxes);
    max_rotations(higher_maxes) = rotation;
end
```

Review: Correlation-based Detection

```
function result = find_template(image, template, scales, ...
                                rotations, result_number)

% function result = find_template(image, template, scales)
%
% returns the bounding boxes of the best matches for the template in
% the image, after searching all specified scales and rotations.
% result_number specifies the number of results (bounding boxes).

[max_responses, max_scales, max_rotations] = ...
    template_search(image, template, scales, ...
                    rotations, result_number);

result = detection_boxes(image, template, max_responses, ...
                         max_scales, result_number);
```

Review: Correlation-based Detection

```
function [result, boxes] = template_detector_demo(image, template, ...
                                       scales, rotations, result_number)

% function [result, boxes] =
%            template_detector_demo(image, template, ...
%                                       scales, rotations, result_number)
%
% returns an image that is a copy of the input image, with
% the bounding boxes drawn for each of the best matches for
% the template in the image, after searching all specified
% scales and rotations.

boxes = find_template(image, template, scales, ...
                      rotations, result_number);
result = image;

for number = 1:result_number
    result = draw_rectangle1(result, boxes(number, 1), ...
                             boxes(number, 2), ...
                             boxes(number, 3), boxes(number, 4));
end
```

Including Information From PCA

- One approach: call pca_score on every possible window.
  - Drawback: slow:
- How can we combine a slow but more accurate approach and a fast but less accurate approach?

Filter-and-refine Approach

- How can we combine a slow but more accurate approach and a fast but less accurate approach?
- **Filter step:** use the fast approach to get a shortlist of candidates.
- **Refine step:** use the slow approach to rerank the shortlist and choose the final results.

Filter-and-refine Face Detection

- **Filter step:** use correlation to identify a number of candidates.
  - number of candidates is 50 in the code.
- **Refine step:** use pca_score to evaluate those candidates.

```
function result = find_faces(image, scales, result_number, ...
                              eigenface_number)

load face_filter;
load final_eigens;

% start filter step
preliminary_number = 50;
preliminary_results = find_template(image, face_filter, ...
                                    scales, 0, preliminary_number);

% start refine step
for number = 1:preliminary_number
    top = preliminary_results(number, 1);
    bottom = preliminary_results(number, 2);
    left = preliminary_results(number, 3);
    right = preliminary_results(number, 4);
    window = image(top:bottom, left:right);
    preliminary_results(number, 6) = ...
        pca_score(window, mean_face, eigenvectors, eigenface_number);
end

[values, indices] = sort(preliminary_results(:, 6), 'ascend');
top_indices = indices(1: result_number);
result = preliminary_results(top_indices, :);
```

Recap on PCA for Face Detection

- What model for faces are we using?

Recap on PCA for Face Detection

- What model for faces are we using?
  - Faces are reconstructed well using the top K eigenvectors.
  - NOTE: each K defines a different model.
- What is a perfect face under this model?

Recap on PCA for Face Detection

- What model for faces are we using?
  - Faces are reconstructed well using the top *K* eigenvectors.
  - NOTE: each *K* defines a different model.
- What is a perfect face under this model?
  - A face spanned using the top *K* eigenvectors.
  - A face *F* such that:
    - **`pca_backprojection(pca_projection(F)) = F.`**
  - A face on which pca_score returns 0.

A Generalized View of Classifiers

- We have studied two face detection methods:
  - Normalized correlation.
  - PCA.
- Each approach exhaustively evaluates all image suwbwindows.
- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.

Features and Classifiers

- Our goal, in the next slides, is to get a better understanding of:
  - What is a feature?
  - What is a classifier?

Normalized Correlation Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- What is a feature here?

Normalized Correlation Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- What is a feature here?
- Two possible answers:
  - Each pixel value is a feature.
  - The feature is the result of normalized correlation with the template.

Normalized Correlation Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- What is the score of each subwindow?

Normalized Correlation Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- What is the score of each subwindow?
  - The result of normalized correlation with the template.
  - Arguably, the score is the feature itself.

Normalized Correlation Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- How does the decision depend on the score?

Normalized Correlation Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- How does the decision depend on the score?
  - In **`find_template`**, faces are the top N scores.
    - N is an argument to the **`find_template`** function.
  - An alternative, is to check if score > threshold.
    - Then we must choose a threshold instead of N.

PCA-based Detection Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- What is a feature here?

PCA-based Detection Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- What is a feature here?
  - The result of the pca_projection function.
  - In other words, the K numbers that describe the projection of the subwindow on the space defined by the top K eigenfaces.

PCA-based Detection Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- What is the score of each subwindow?

PCA-based Detection Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- What is the score of each subwindow?
  - The result of the pca_score function.
  - The sum of squared differences between:
    - the original subwindow W, and
    - pca_backprojection(pca_projection(W)).

PCA-based Detection Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- How does the decision depend on the score?

PCA-based Detection Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- How does the decision depend on the score?
  - In **`find_faces`**, faces are the top N scores.
    - N is an argument to the **`find_faces`** function.
  - An alternative, is to check if score > threshold.
    - Then we must choose a threshold instead of N.

Defining a Classifier

- Choose features.
- Choose a scoring function.
- Choose a decision process.
  - The last part is usually straightforward.
  - We pick the top N scores, or we apply a threshold.
- Therefore, the two crucial components are:
  - Choosing features.
  - Choosing a scoring function.

What is a Feature?

- Any information extracted from an image.
- The number of possible features we can define is enormous.
- Any function F we can define that takes in an image as an argument and produces one or more numbers as output defines a feature.
  – Correlation with a template defines a feature.
  – Projecting to PCA space defines features.
  – More examples?

What is a Feature?

- Any information extracted from an image.
- The number of possible features we can define is enormous.
- Any function F we can define that takes in an image as an argument and produces one or more numbers as output defines a feature.
  - Correlation with a template defines a feature.
  - Projecting to PCA space defines features.
  - Average intensity.
  - Std of values in window.