



10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

PCA + Neural Networks

PCA Readings:

Murphy 12
Bishop 12
HTF 14.5
Mitchell --

Neural Net Readings:

Murphy --
Bishop 5
HTF 11
Mitchell 4

Matt Gormley
Lecture 18
March 27, 2017

Reminders

- **Homework 6: Unsupervised Learning**
 - **Release: Wed, Mar. 22**
 - **Due: Wed, Mar. 22 at 11:59pm**

DIMENSIONALITY REDUCTION

PCA Outline

- **Dimensionality Reduction**
 - High-dimensional data
 - Learning (low dimensional) representations
- **Principal Component Analysis (PCA)**
 - Examples: 2D and 3D
 - Data for PCA
 - PCA Definition
 - Objective functions for PCA
 - PCA, Eigenvectors, and Eigenvalues
 - Algorithms for finding Eigenvectors / Eigenvalues
- **PCA Examples**
 - Face Recognition
 - Image Compression



Last Lecture

This Lecture

Big & High-Dimensional Data

- High-Dimensions = Lot of Features

Document classification

Features per document =
thousands of words/unigrams
millions of bigrams, contextual
information



Surveys - Netflix

480189 users x 17770 movies

	movie 1	movie 2	movie 3	movie 4	movie 5	movie 6
Tom	5	?	?	1	3	?
George	?	?	3	1	2	5
Susan	4	3	1	?	5	1
Beth	4	3	?	2	4	2

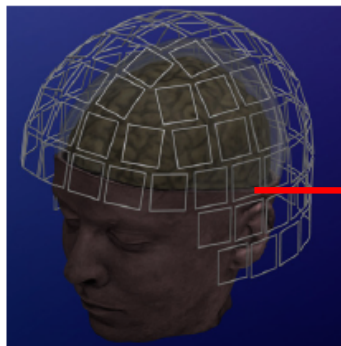
Big & High-Dimensional Data

- High-Dimensions = Lot of Features

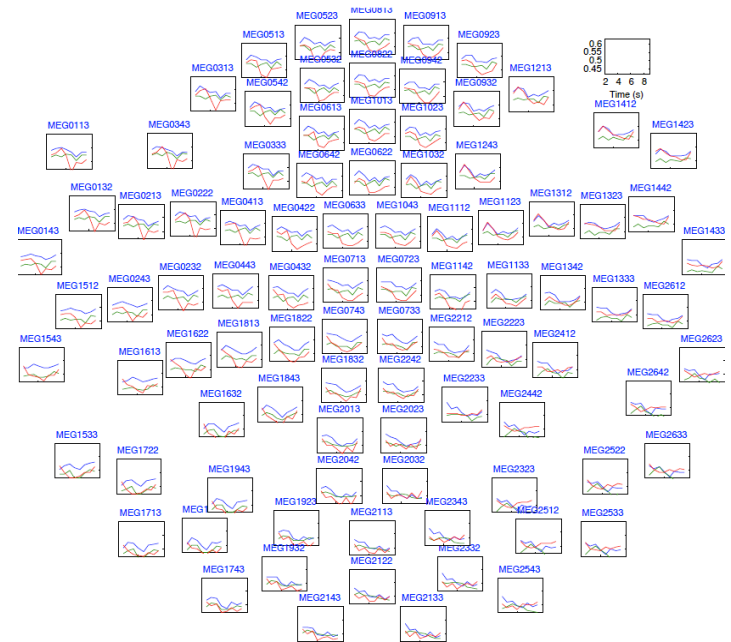
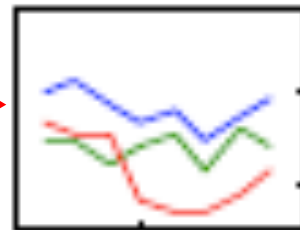
MEG Brain Imaging

120 locations x 500 time points

x 20 objects



MEG0633



Or any high-dimensional image data



- Big & High-Dimensional Data.
- Useful to learn lower dimensional representations of the data.

Learning Representations

PCA, Kernel PCA, ICA: Powerful unsupervised learning techniques for extracting hidden (potentially lower dimensional) structure from high dimensional datasets.

Useful for:

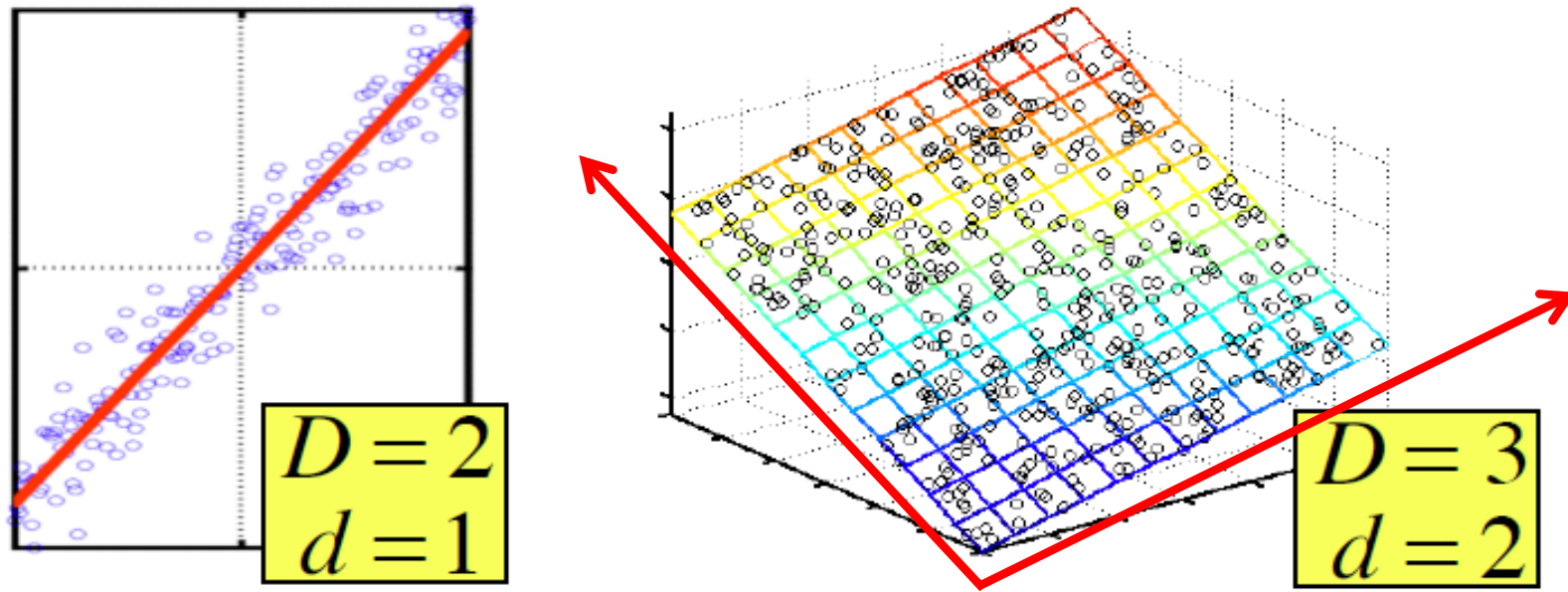
- Visualization
- More efficient use of resources (e.g., time, memory, communication)
- Statistical: fewer dimensions → better generalization
- Noise removal (improving data quality)
- Further processing by machine learning algorithms

PRINCIPAL COMPONENT ANALYSIS (PCA)

PCA Outline

- **Dimensionality Reduction**
 - High-dimensional data
 - Learning (low dimensional) representations
- **Principal Component Analysis (PCA)**
 - Examples: 2D and 3D
 - Data for PCA
 - PCA Definition
 - Objective functions for PCA
 - PCA, Eigenvectors, and Eigenvalues
 - Algorithms for finding Eigenvectors / Eigenvalues
- **PCA Examples**
 - Face Recognition
 - Image Compression

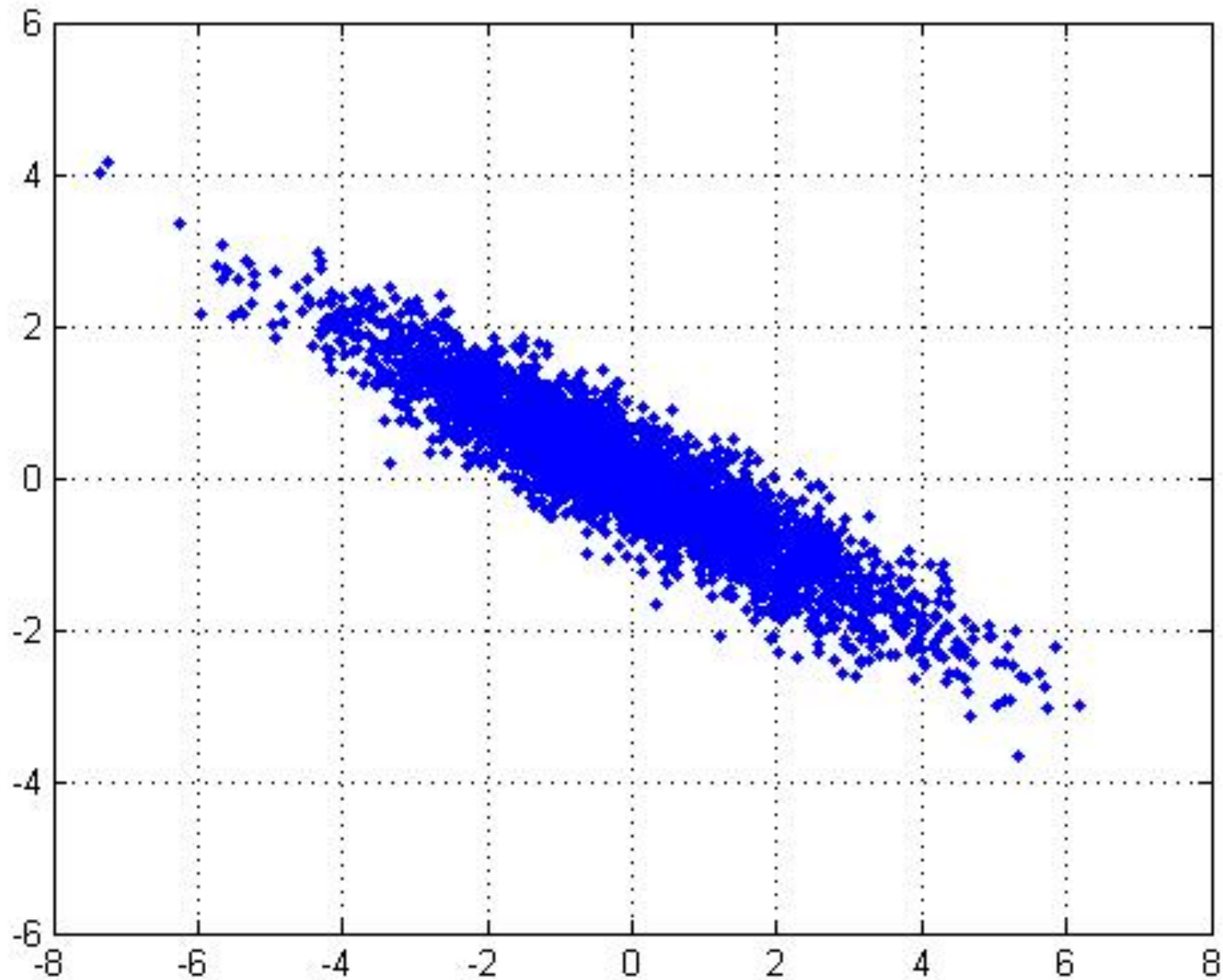
Principal Component Analysis (PCA)



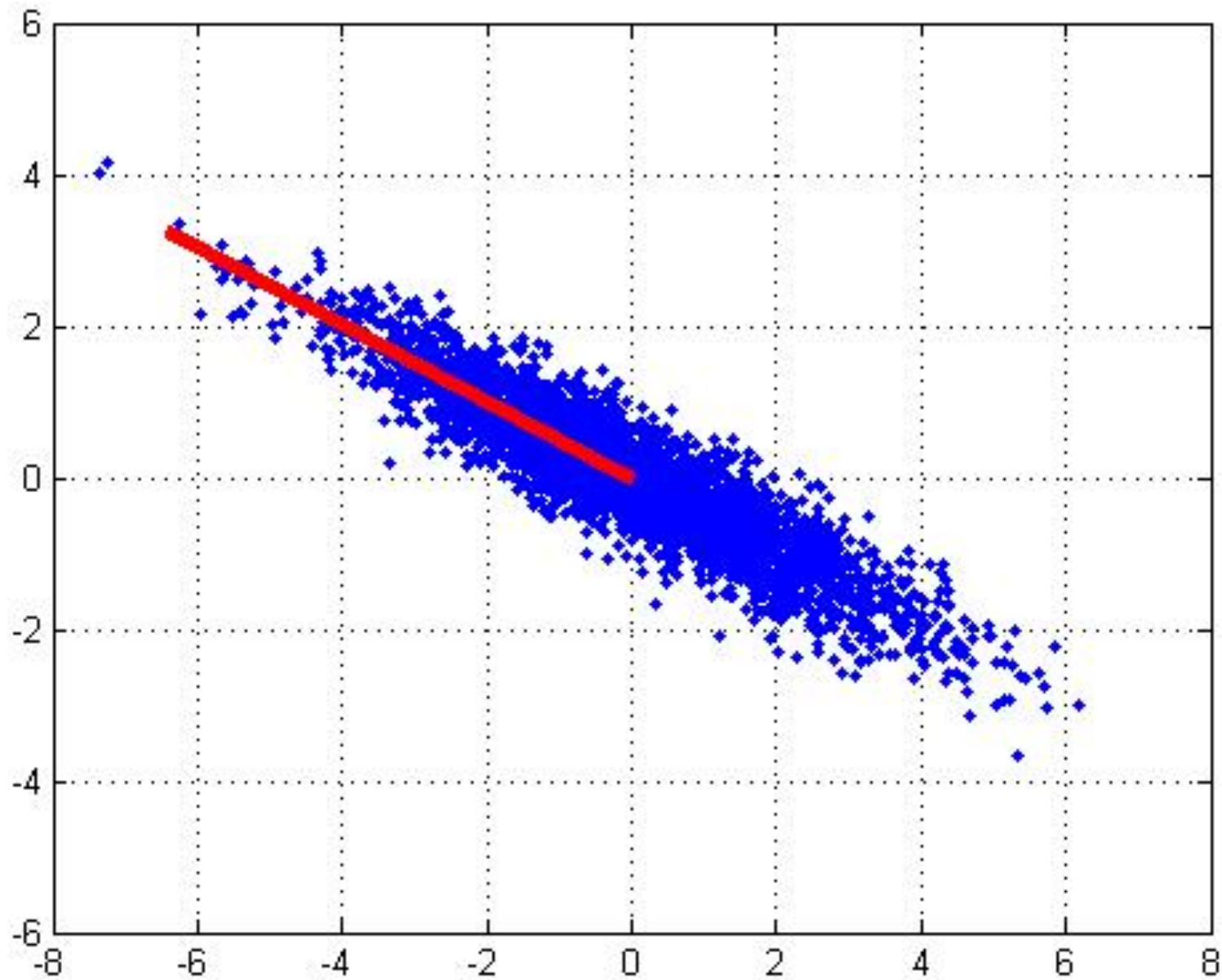
In case where data lies on or near a low d -dimensional linear subspace, axes of this subspace are an effective representation of the data.

Identifying the axes is known as [Principal Components Analysis](#), and can be obtained by using classic matrix computation tools (Eigen or Singular Value Decomposition).

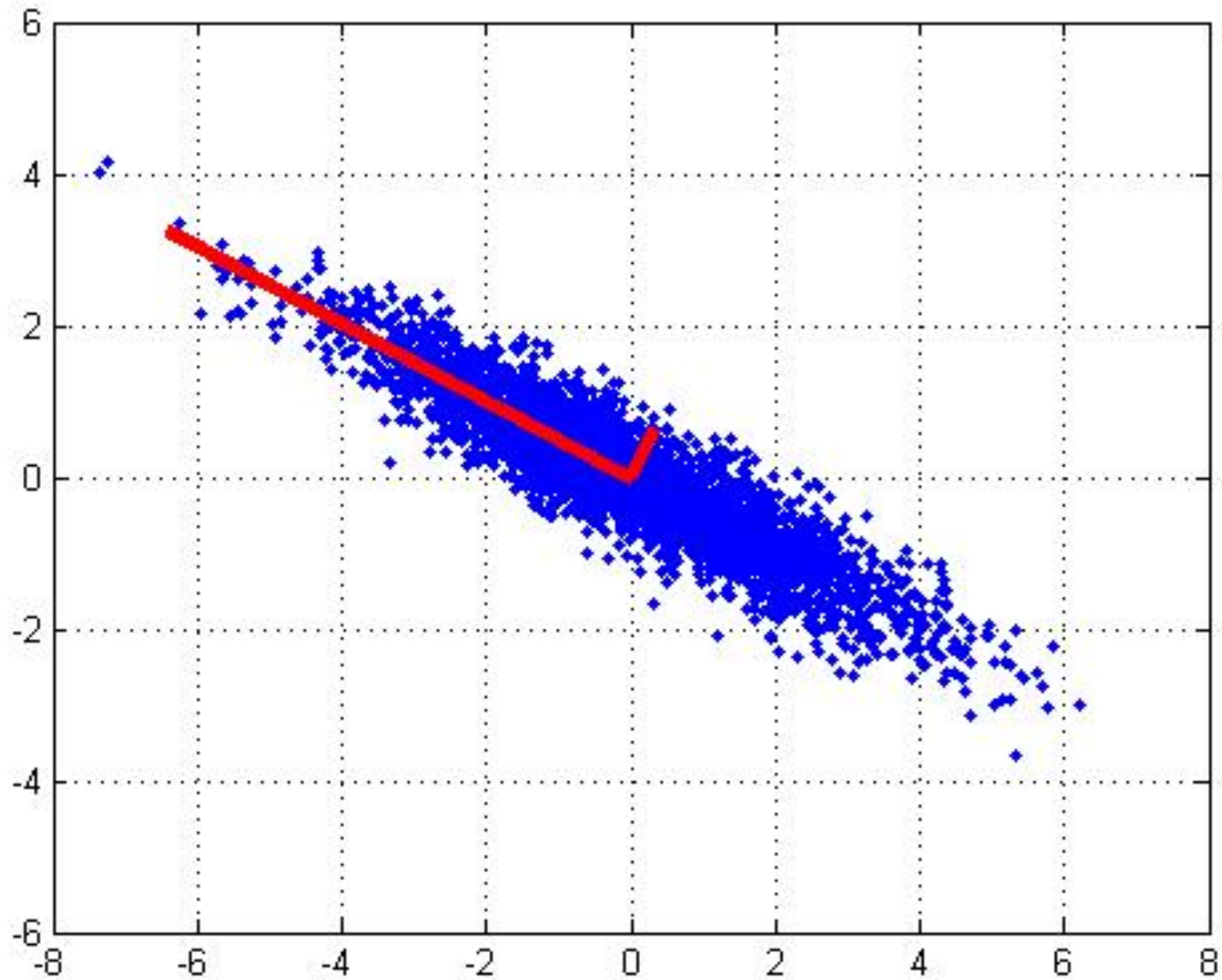
2D Gaussian dataset



1st PCA axis



2nd PCA axis



Principal Component Analysis (PCA)

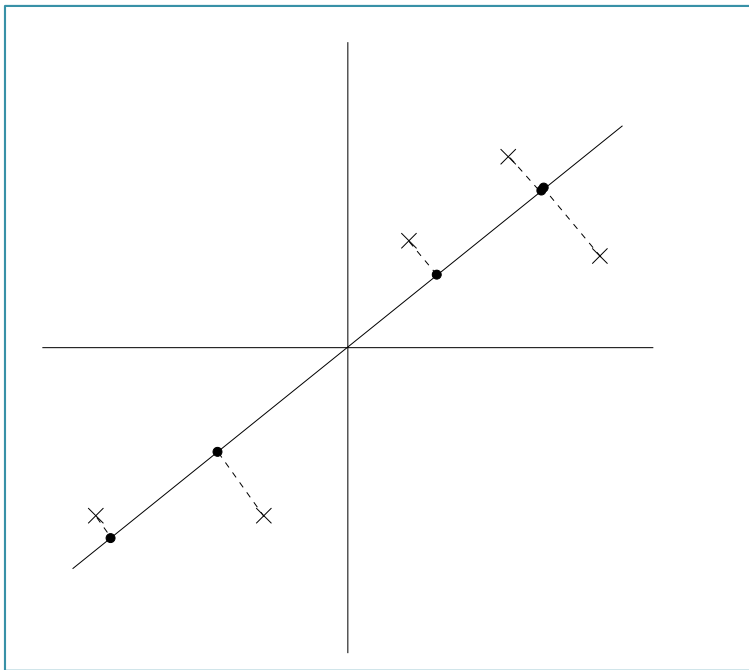
Whiteboard

- Data for PCA
- PCA Definition
- Objective functions for PCA

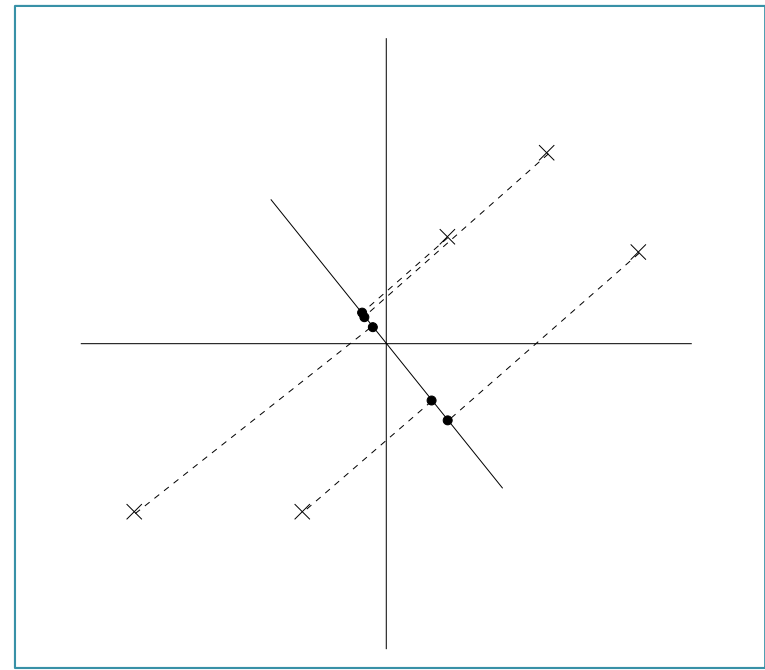
Maximizing the Variance

- Consider the two projections below
- Which maximizes the variance?

Option A



Option B



Principal Component Analysis (PCA)

Whiteboard

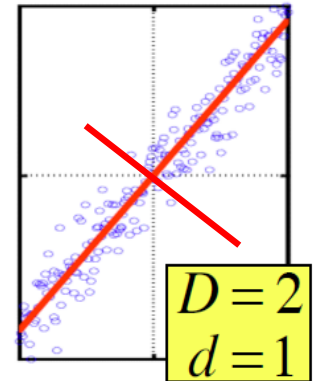
- PCA, Eigenvectors, and Eigenvalues
- Algorithms for finding Eigenvectors / Eigenvalues

Principal Component Analysis (PCA)

$(X X^T)v = \lambda v$, so v (the first PC) is the eigenvector of sample correlation/covariance matrix $X X^T$

Sample variance of projection $v^T X X^T v = \lambda v^T v = \lambda$

Thus, the eigenvalue λ denotes the amount of variability captured along that dimension (aka amount of energy along that dimension).

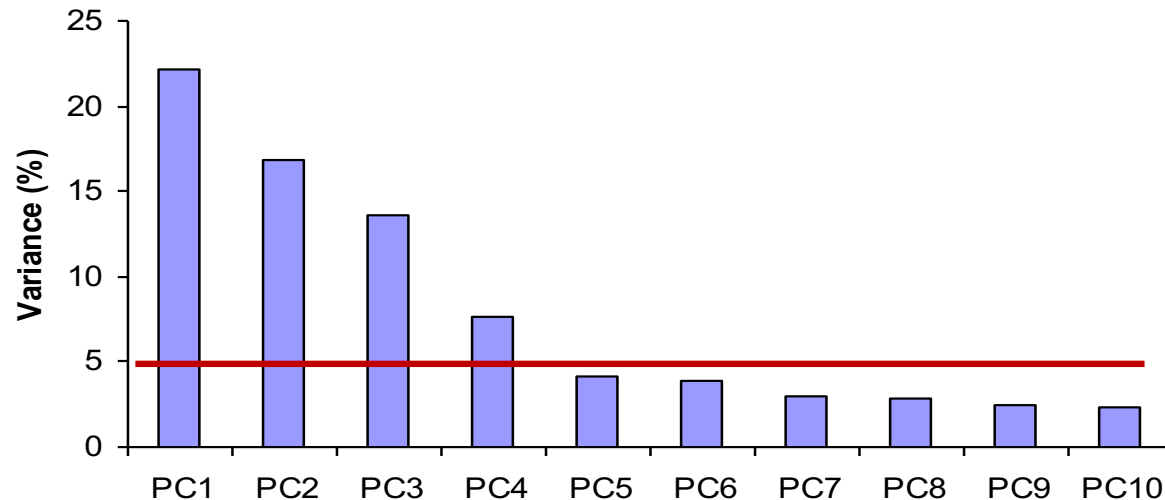


Eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \dots$

- The 1st PC v_1 is the the eigenvector of the sample covariance matrix $X X^T$ associated with the largest eigenvalue
- The 2nd PC v_2 is the the eigenvector of the sample covariance matrix $X X^T$ associated with the second largest eigenvalue
- And so on ...

How Many PCs?

- For M original dimensions, sample covariance matrix is $M \times M$, and has up to M eigenvectors. So M PCs.
- Where does dimensionality reduction come from?
Can *ignore the components of lesser significance*.



- You do *lose some information*, but if the eigenvalues are small, you don't lose much
 - M dimensions in original data
 - calculate M eigenvectors and eigenvalues
 - choose only the first D eigenvectors, based on their eigenvalues
 - final data set has only D dimensions

Slides from Barnabas Poczos

Original sources include:

- Karl Booksh Research group
- Tom Mitchell
- Ron Parr

PCA EXAMPLES

Face recognition

Challenge: Facial Recognition

- Want to identify specific person, based on facial image
 - Robust to glasses, lighting,...
- ⇒ Can't just use the given 256 x 256 pixels

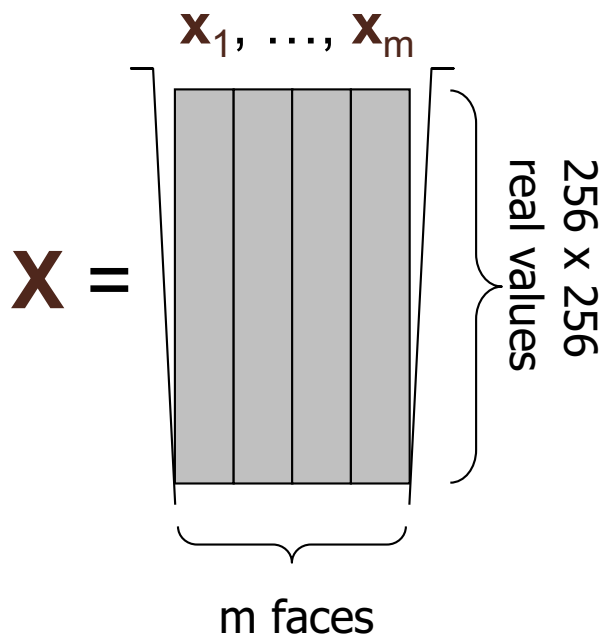


Applying PCA: Eigenfaces

Method: Build one PCA database for the whole dataset and then classify based on the weights.



- Example data set: Images of faces
 - Famous Eigenface approach
[Turk & Pentland], [Sirovich & Kirby]
- Each face $\mathbf{x}$ is ...
 - 256×256 values (luminance at location)
 - $\mathbf{x}$ in $\mathbb{R}^{256 \times 256}$ (view as 64K dim vector)



Principle Components



Reconstructing...



- ... faster if train with...
 - only people w/out glasses
 - same lighting conditions

Shortcomings

- Requires carefully controlled data:
 - All faces centered in frame
 - Same size
 - Some sensitivity to angle
- Alternative:
 - “Learn” one set of PCA vectors for each angle
 - Use the one with lowest error
- Method is completely knowledge free
 - (sometimes this is good!)
 - Doesn't know that faces are wrapped around 3D objects (heads)
 - Makes no effort to preserve class distinctions

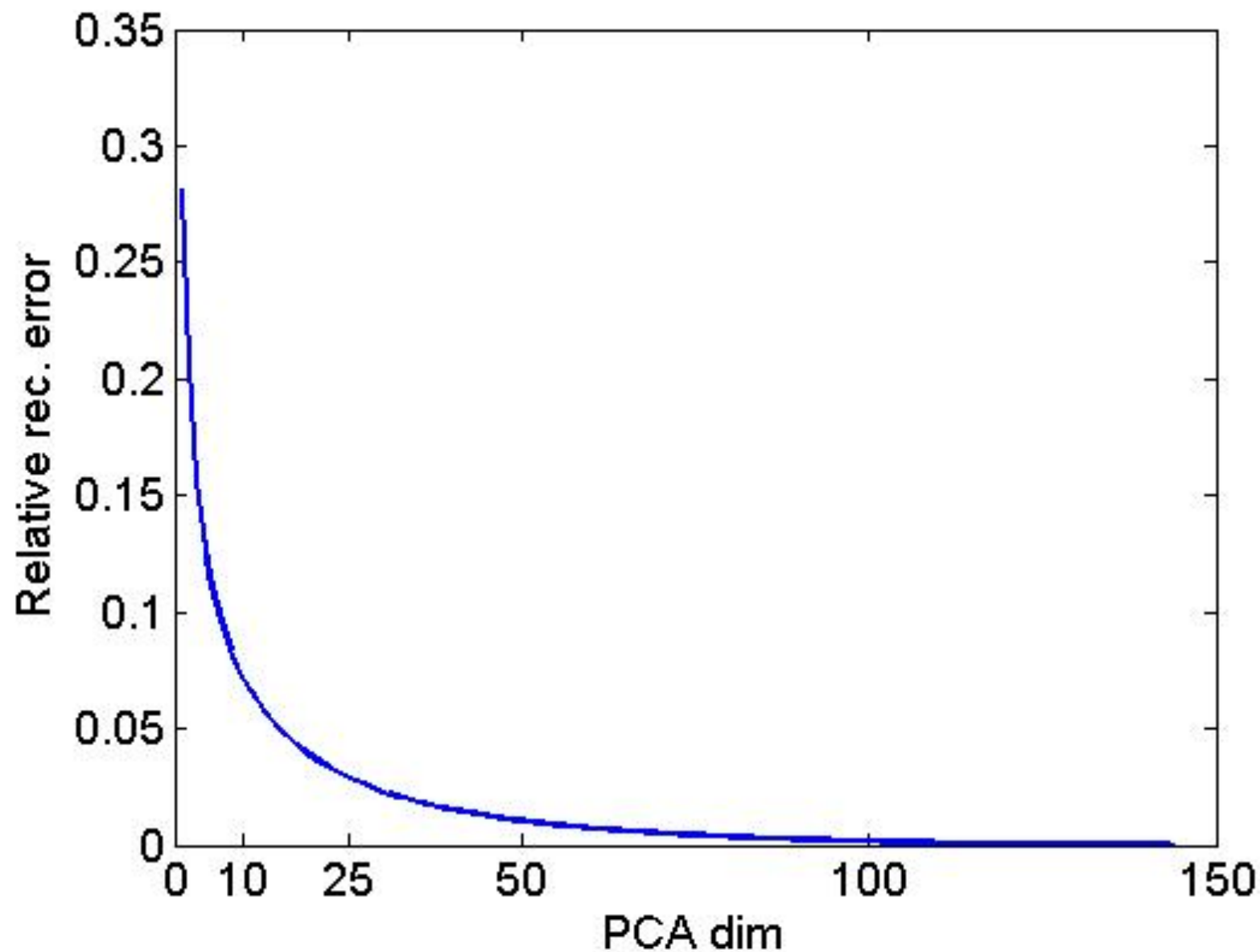
Image Compression

Original Image



- Divide the original 372x492 image into patches:
 - Each patch is an instance that contains 12x12 pixels on a grid
- View each as a 144-D vector

L_2 error and PCA dim



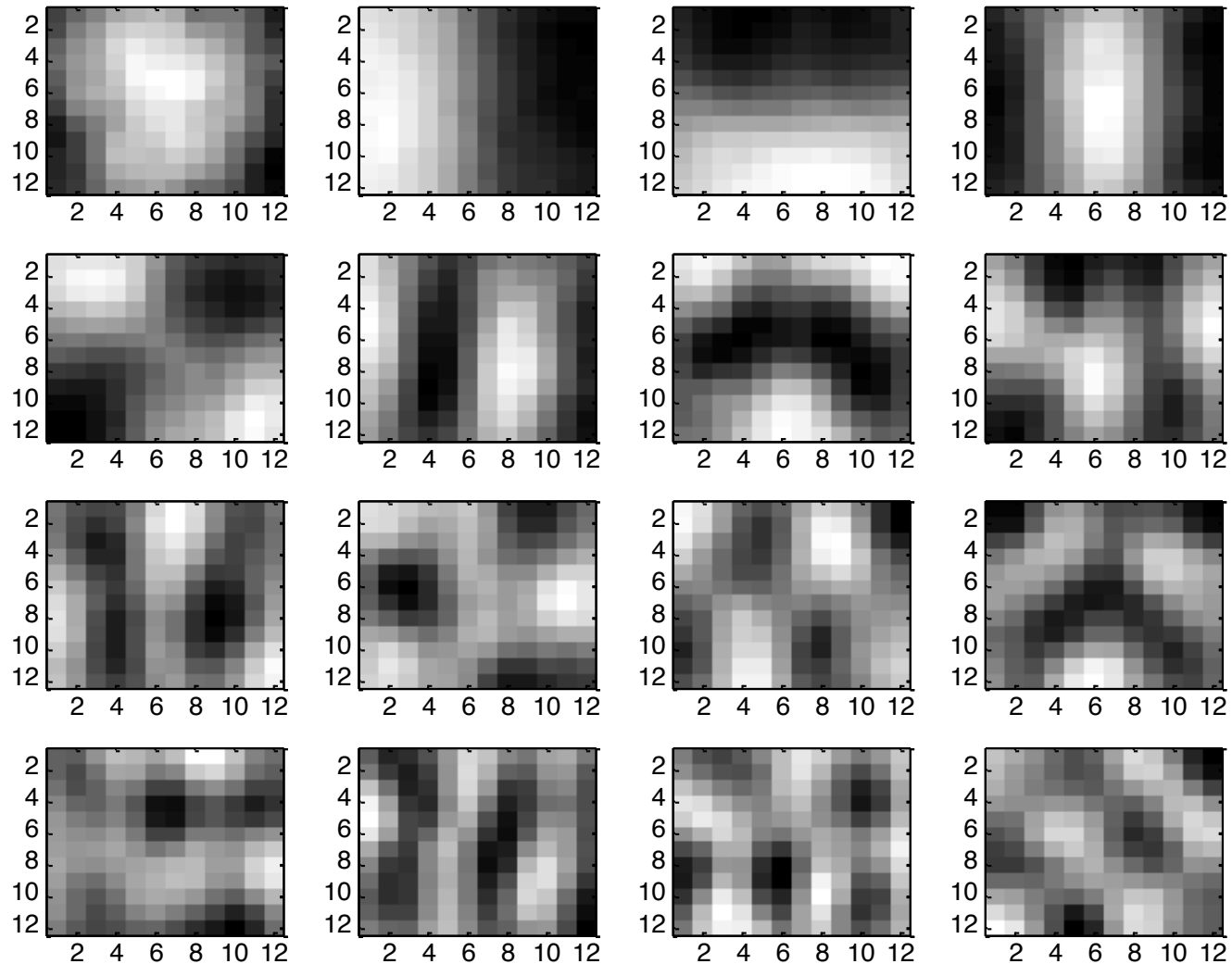
PCA compression: 144D $\rightarrow$ 60D



PCA compression: 144D $\rightarrow$ 16D



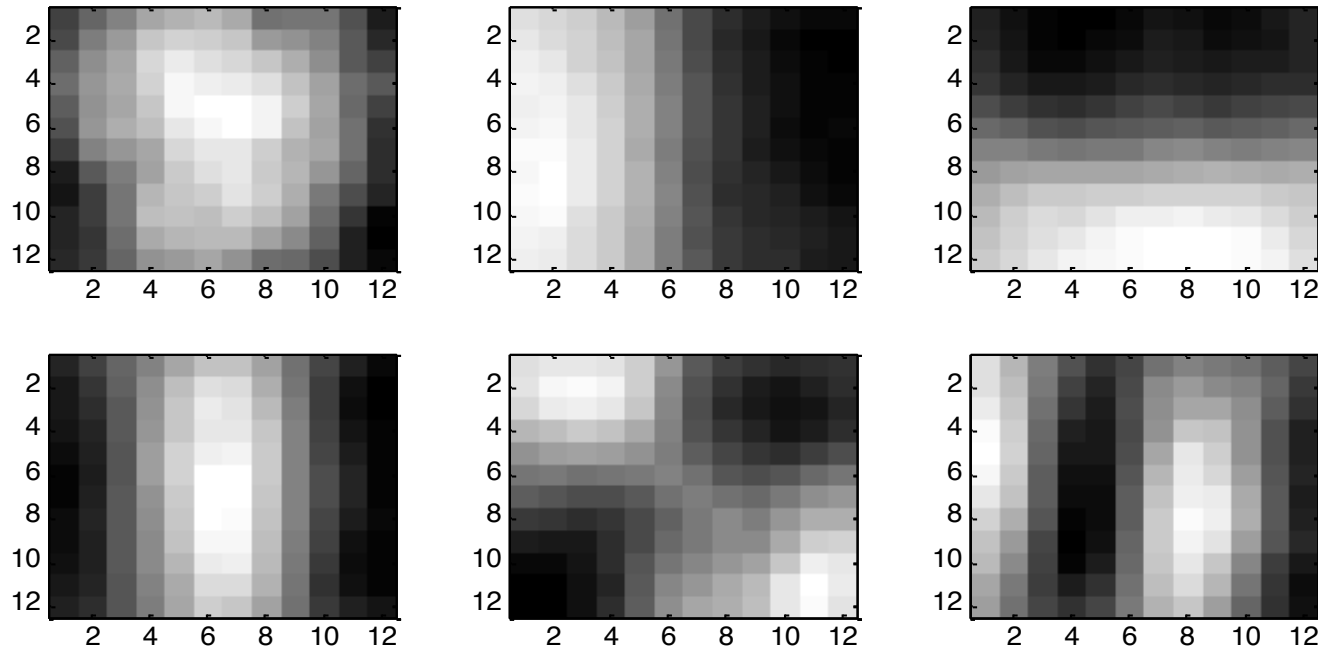
16 most important eigenvectors



PCA compression: 144D $\rightarrow$ 6D



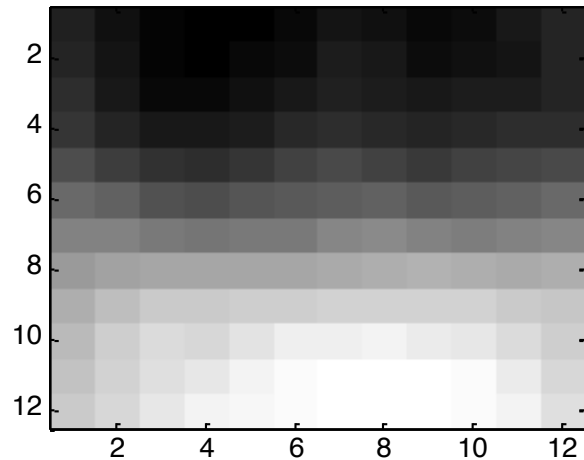
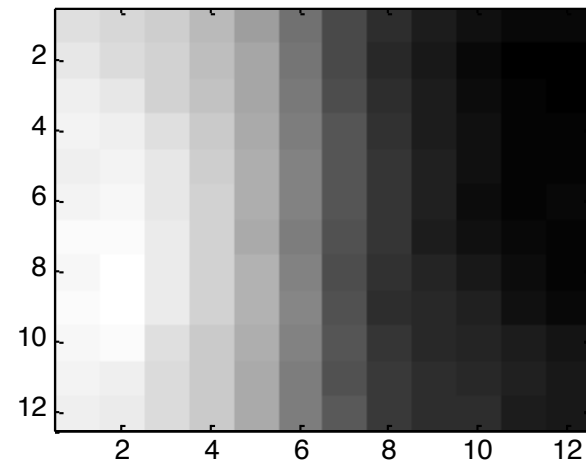
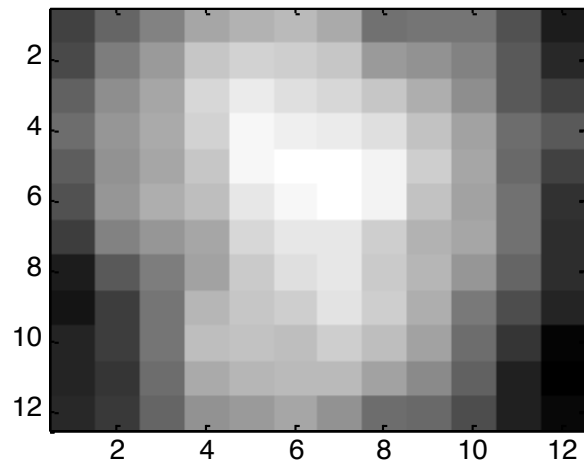
6 most important eigenvectors



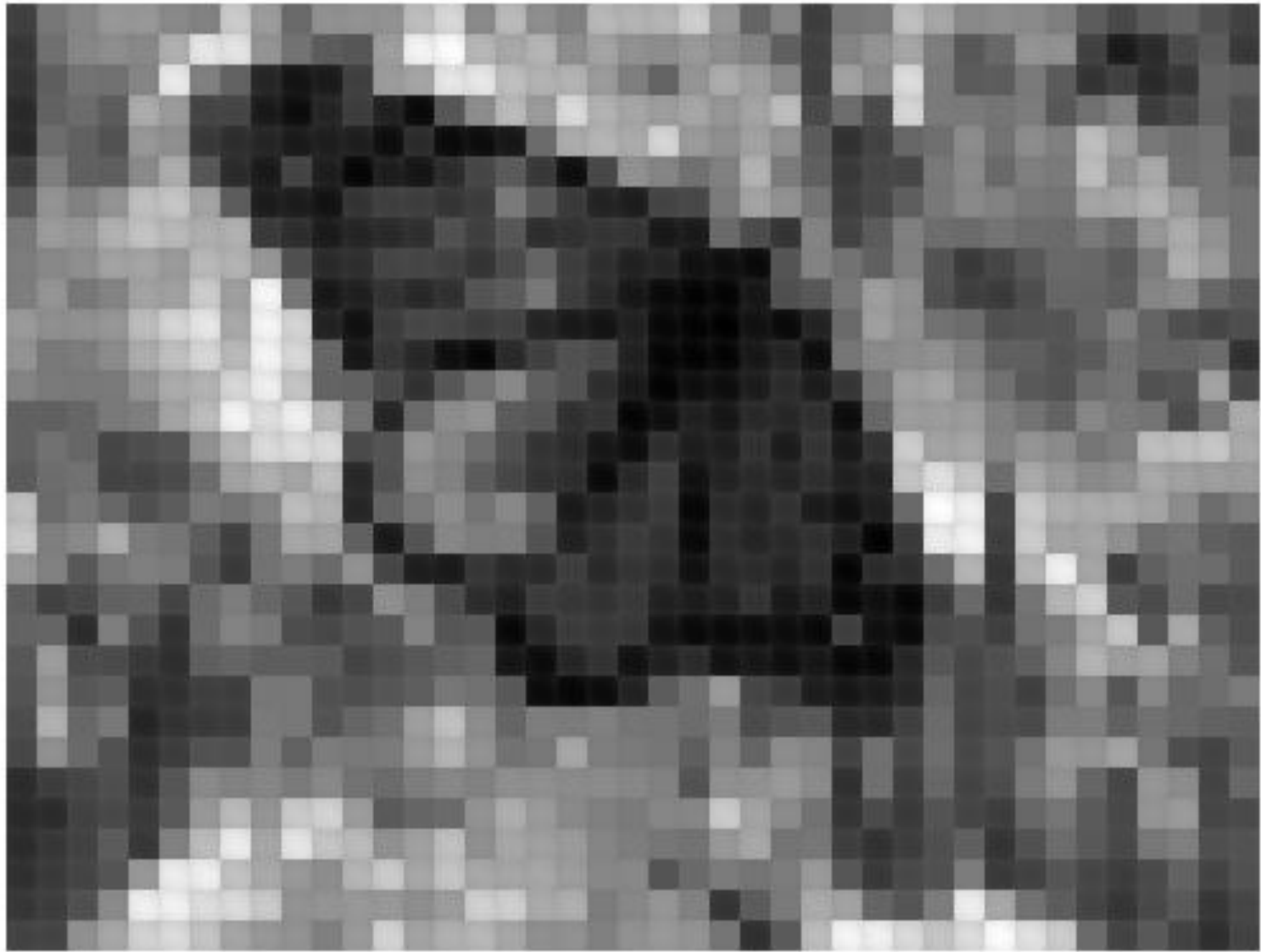
PCA compression: 144D $\rightarrow$ 3D



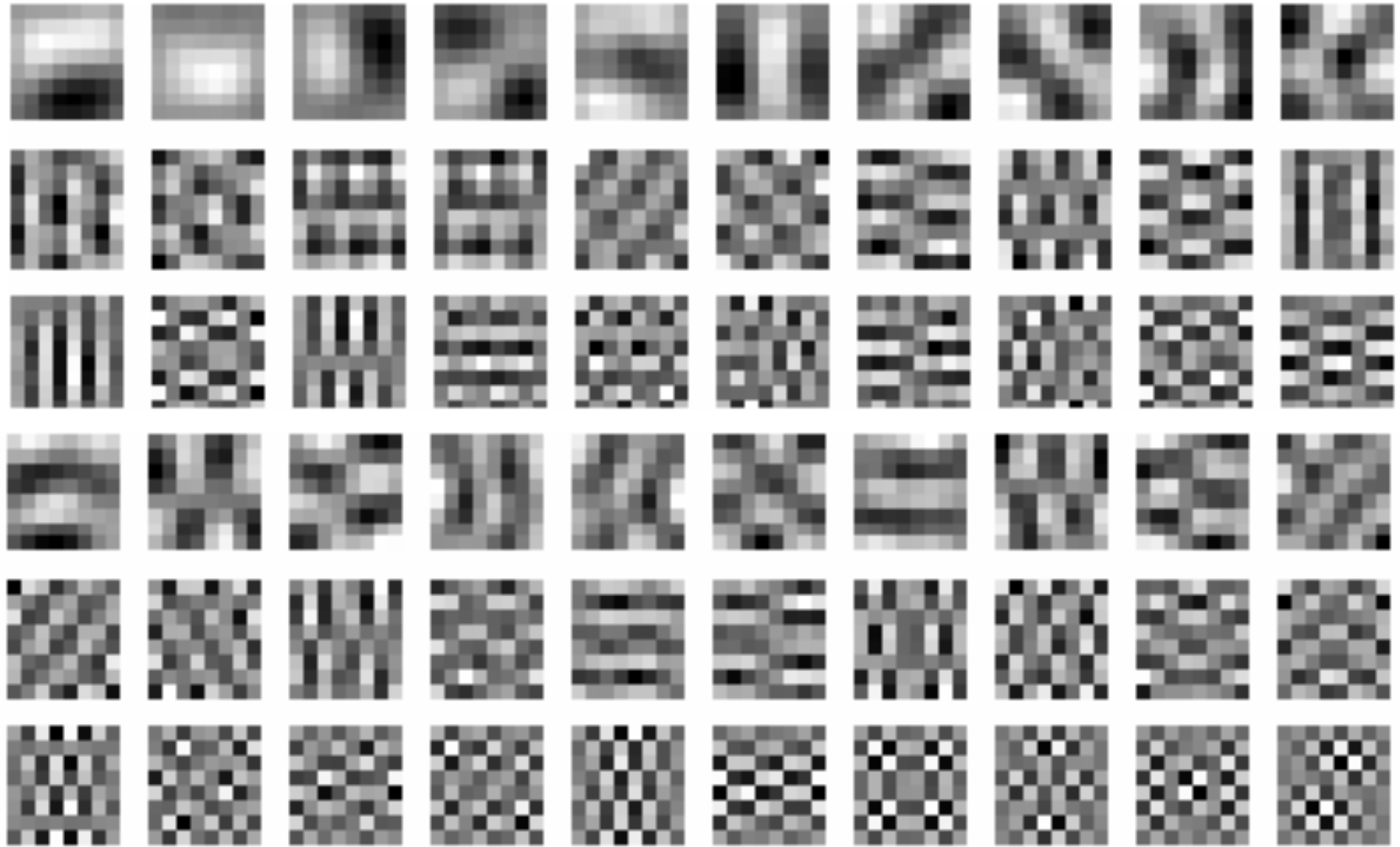
3 most important eigenvectors



PCA compression: 144D $\rightarrow$ 1D

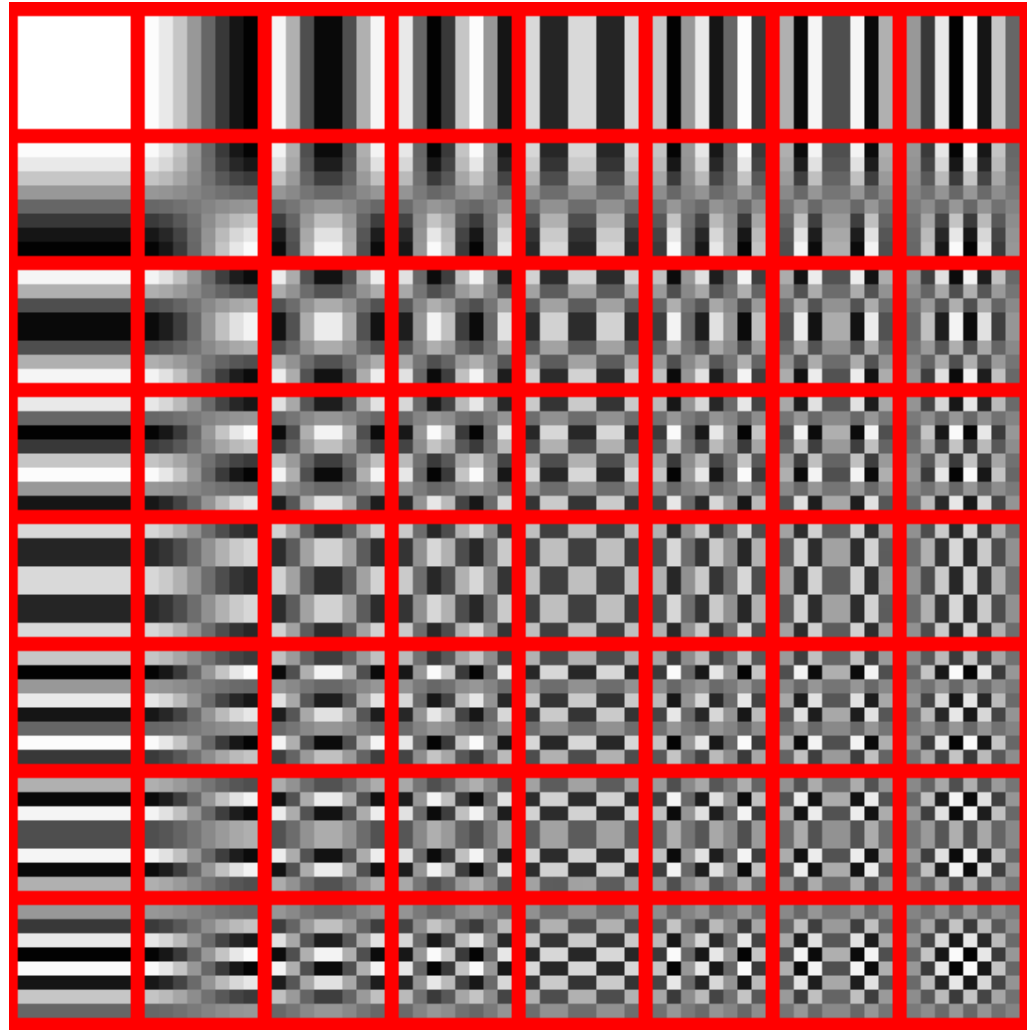


60 most important eigenvectors



Looks like the discrete cosine bases of JPG!...

2D Discrete Cosine Basis



http://en.wikipedia.org/wiki/Discrete_cosine_transform

Neural Networks Outline

- **Logistic Regression (Recap)**
 - Data, Model, Learning, Prediction
- **Neural Networks**
 - A Recipe for Machine Learning
 - Visual Notation for Neural Networks
 - Example: Logistic Regression Output Surface
 - 1-Layer Neural Network
 - 2-Layer Neural Network
- **Neural Net Architectures**
 - Objective Functions
 - Activation Functions
- **Backpropagation**
 - Basic Chain Rule (of calculus)
 - Chain Rule for Arbitrary Computation Graph
 - Backpropagation Algorithm
 - Module-based Automatic Differentiation (Autodiff)

RECALL: LOGISTIC REGRESSION

Using gradient ascent for linear classifiers

Recall...

Key idea behind today's lecture:

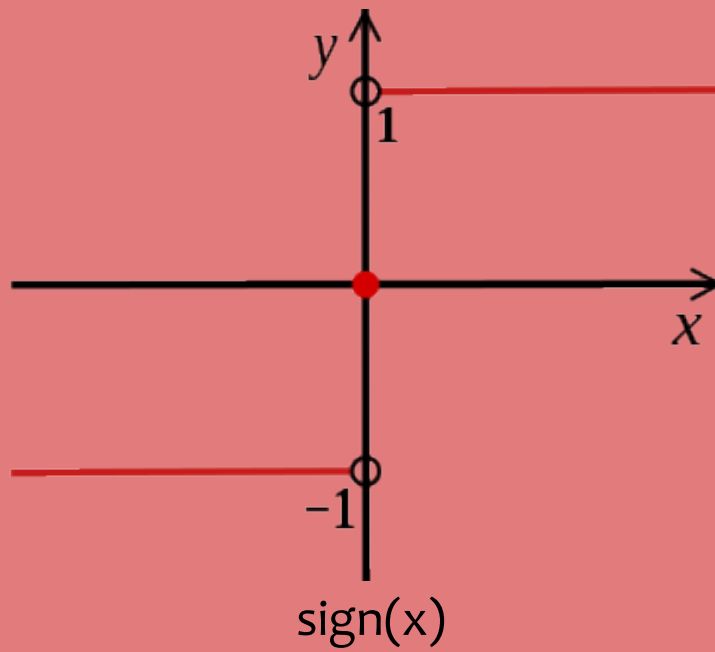
1. Define a linear classifier (logistic regression)
2. Define an objective function (likelihood)
3. Optimize it with gradient descent to learn parameters
4. Predict the class with highest probability under the model

Using gradient ascent for linear classifiers

Recall...

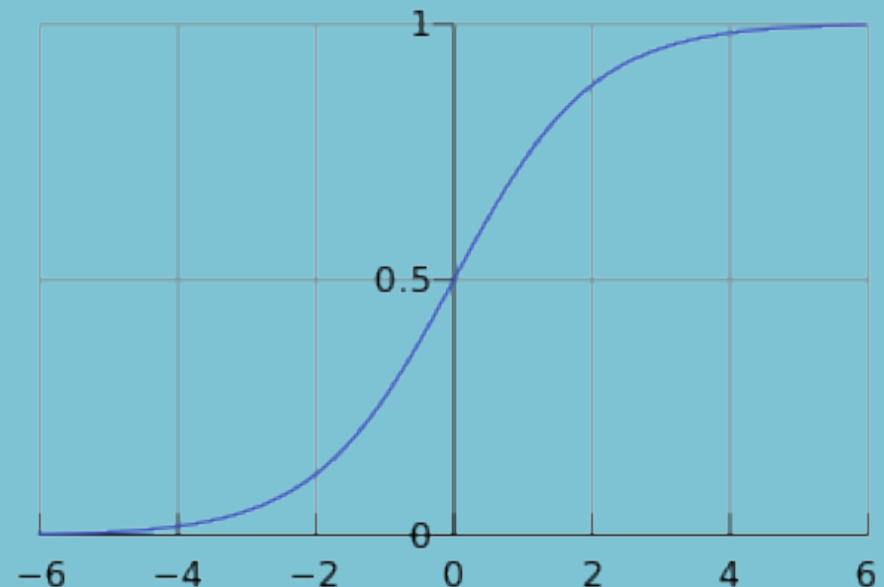
This decision function isn't differentiable:

$$h(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$



Use a differentiable function instead:

$$p_{\boldsymbol{\theta}}(y = 1 | \mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{x})}$$



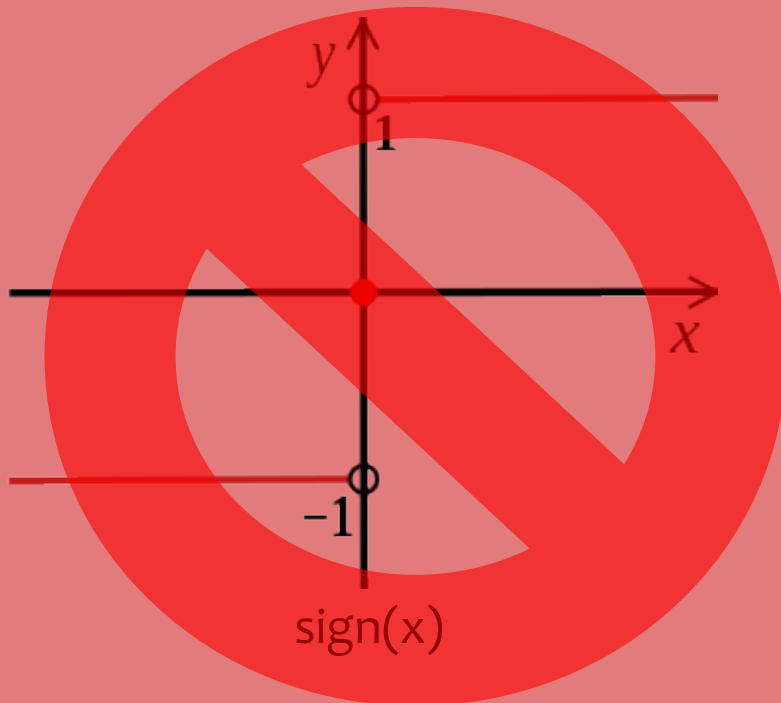
$$\text{logistic}(u) \equiv \frac{1}{1 + e^{-u}}$$

Using gradient ascent for linear classifiers

Recall...

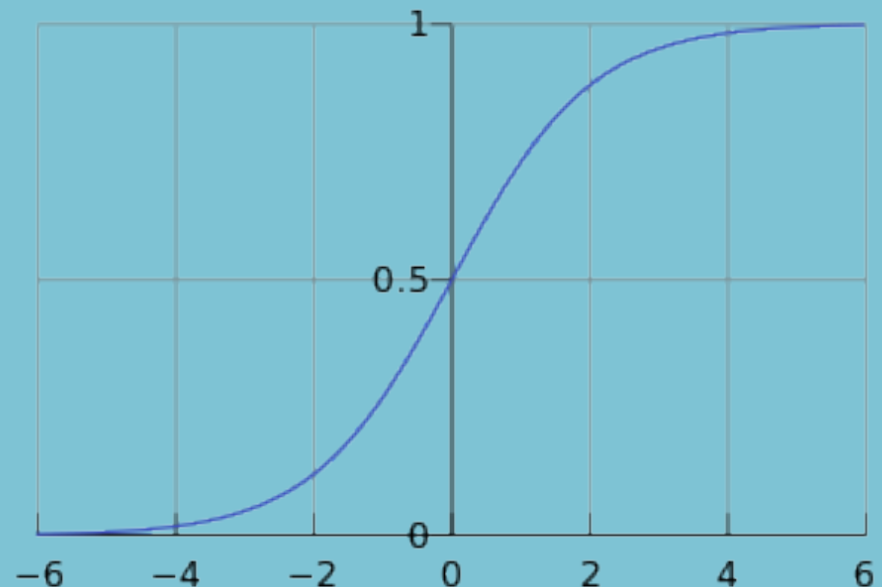
This decision function isn't differentiable:

$$h(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$



Use a differentiable function instead:

$$p_{\boldsymbol{\theta}}(y = 1 | \mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{x})}$$



$$\text{logistic}(u) \equiv \frac{1}{1 + e^{-u}}$$

Logistic Regression

Data: Inputs are continuous vectors of length K . Outputs are discrete.

$$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^N \text{ where } \mathbf{x} \in \mathbb{R}^K \text{ and } y \in \{0, 1\}$$

Model: Logistic function applied to dot product of parameters with input vector.

$$p_{\boldsymbol{\theta}}(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{x})}$$

Learning: finds the parameters that minimize some objective function. $\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} J(\boldsymbol{\theta})$

Prediction: Output is the most probable class.

$$\hat{y} = \underset{y \in \{0,1\}}{\operatorname{argmax}} p_{\boldsymbol{\theta}}(y|\mathbf{x})$$

NEURAL NETWORKS

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function

$$\hat{\mathbf{y}} = f_{\theta}(\mathbf{x}_i)$$

– Loss function

$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$$

Face



Face



Not a face



Examples: Linear regression,
Logistic regression, Neural Network

Examples: Mean-squared error,
Cross Entropy

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function

$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

– Loss function

$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

4. Train with SGD:

(take small steps
opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

Background

1. Given training data

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

2. Choose each of the

– Decision function

$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

– Loss function

$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$$

A Recipe for Gradients

Backpropagation can compute this gradient!

And it's a **special case of a more general algorithm** called reverse-mode automatic differentiation that can compute the gradient of any differentiable function efficiently!

opposite the gradient)


$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

Goals for Today's Lecture

1. Explore a **new class of decision functions** (Neural Networks)
2. Consider **variants of this recipe** for training

– Decision function

$$\hat{y} = f_{\theta}(x_i)$$

– Loss function

$$\ell(\hat{y}, y_i) \in \mathbb{R}$$

4. Train with SGD:

– Take small steps opposite the gradient)

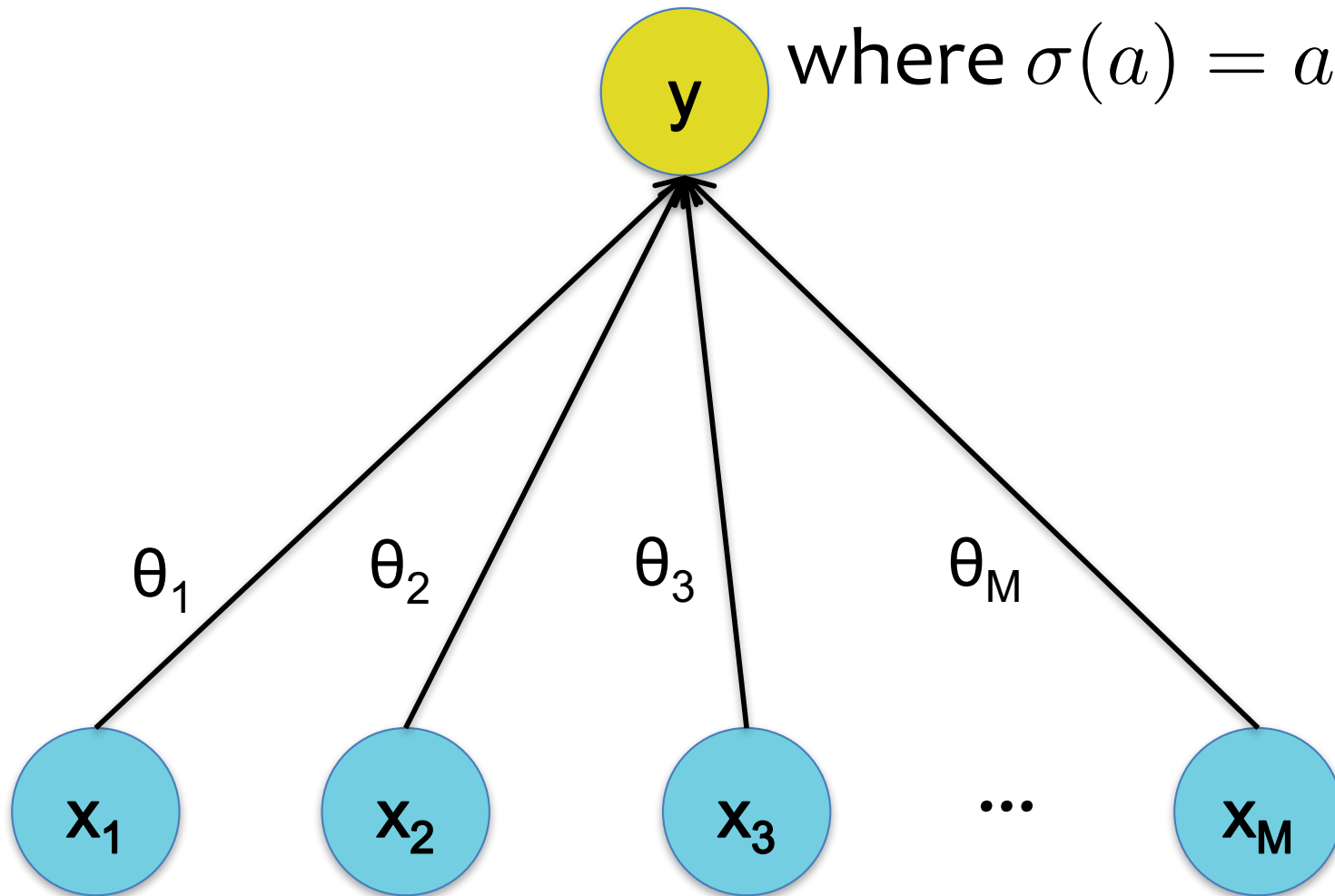
$$\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_{\theta}(x_i), y_i)$$

$$y = h_{\theta}(x) = \sigma(\theta^T x)$$

where $\sigma(a) = a$

Output

Input

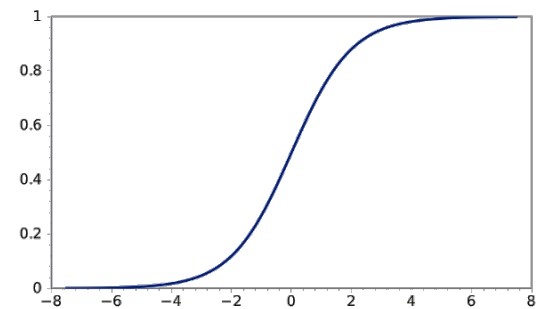
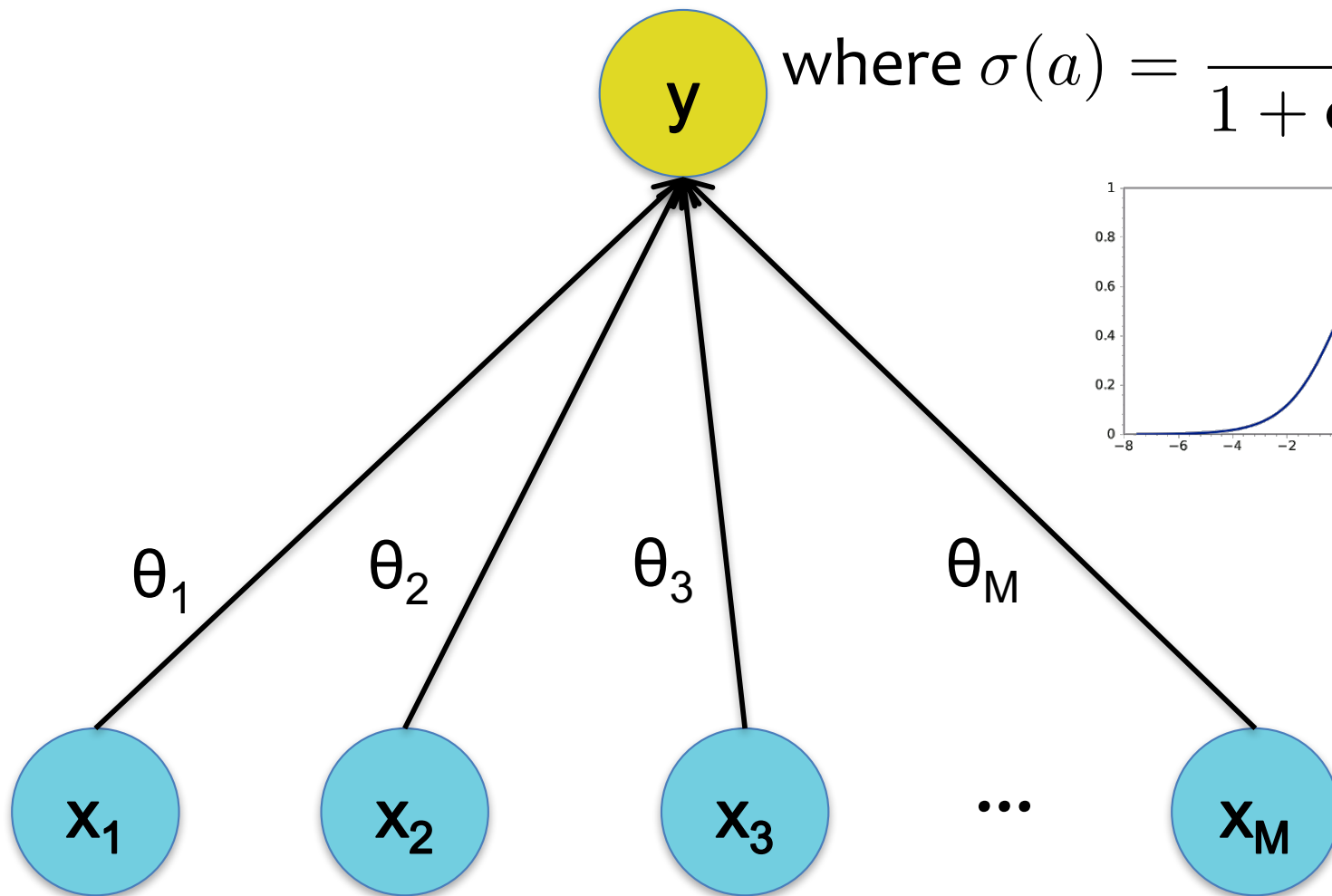


$$y = h_{\theta}(x) = \sigma(\theta^T x)$$

$$\text{where } \sigma(a) = \frac{1}{1 + \exp(-a)}$$

Output

Input



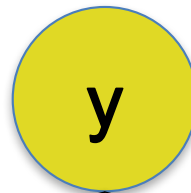
Decision Functions

Logistic Regression

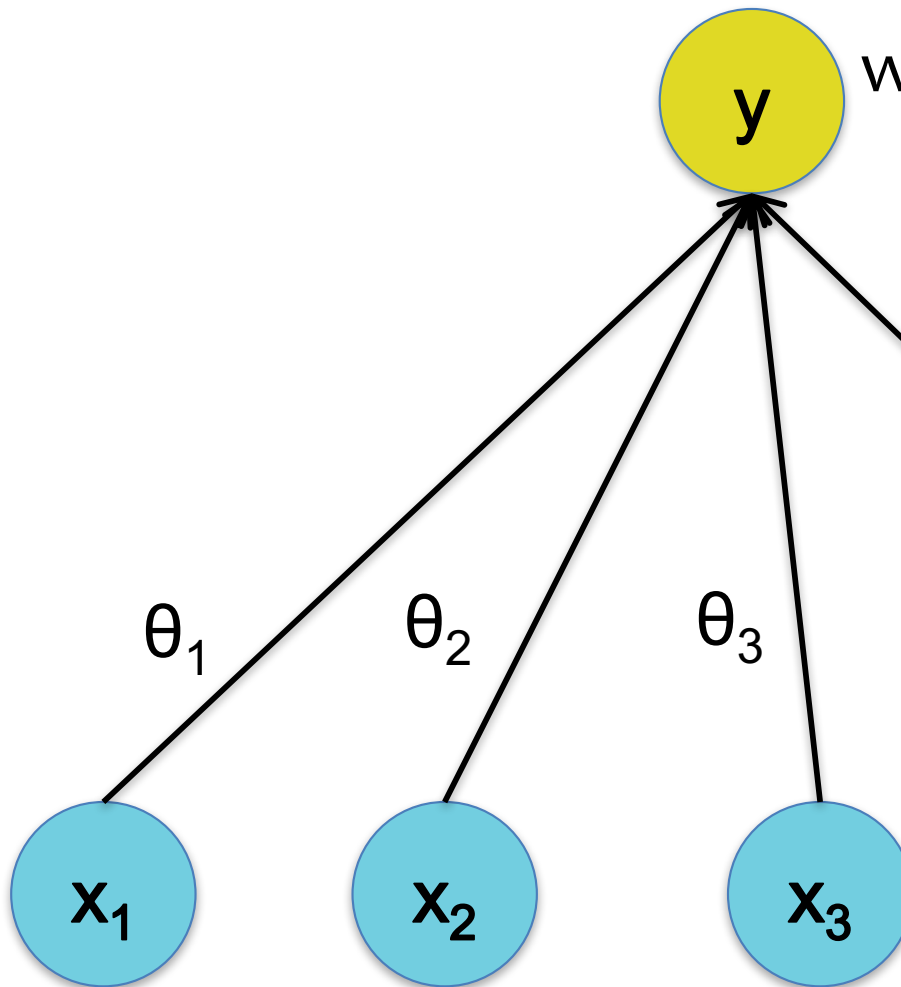
$$y = h_{\theta}(x) = \sigma(\theta^T x)$$

$$\text{where } \sigma(a) = \frac{1}{1 + \exp(-a)}$$

Output



Input



Face



Face



Not a face



$$y = h_{\theta}(x) = \sigma(\theta^T x)$$

