

Machine Learning

DSECL ZG565

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in



BITS Pilani

Pilani Campus



Lecture No. – 9 | Neural Network

Date – 29/11/2020

Time – 1:55 PM – 04:05 PM

These slides are prepared by the instructor, with grateful acknowledgement of Tom Mitchell, Andrew Ng and many others who made their course materials freely available online.

Session Content

- Perceptron (Chapter 4 Tom Mitchell)
- Neural Network Architecture (Andrew Ng Notes and Chapter 4 Tom Mitchell)
- Back propagation Algorithm (Andrew Ng Notes)

Artificial Neural Network

- Artificial neural networks (ANNs) provide a general, practical method for learning real-valued, discrete-valued, and vector-valued functions from examples.
- Successfully applied to problems
 - Interpreting visual scenes
 - Speech recognition
 - Recognize handwritten character
 - Face recognition

Rumelhart, D., Widrow, B., & Lehr, M. (1994). The basic ideas in neural networks. Communications of the ACM, 37(3), 87-92.

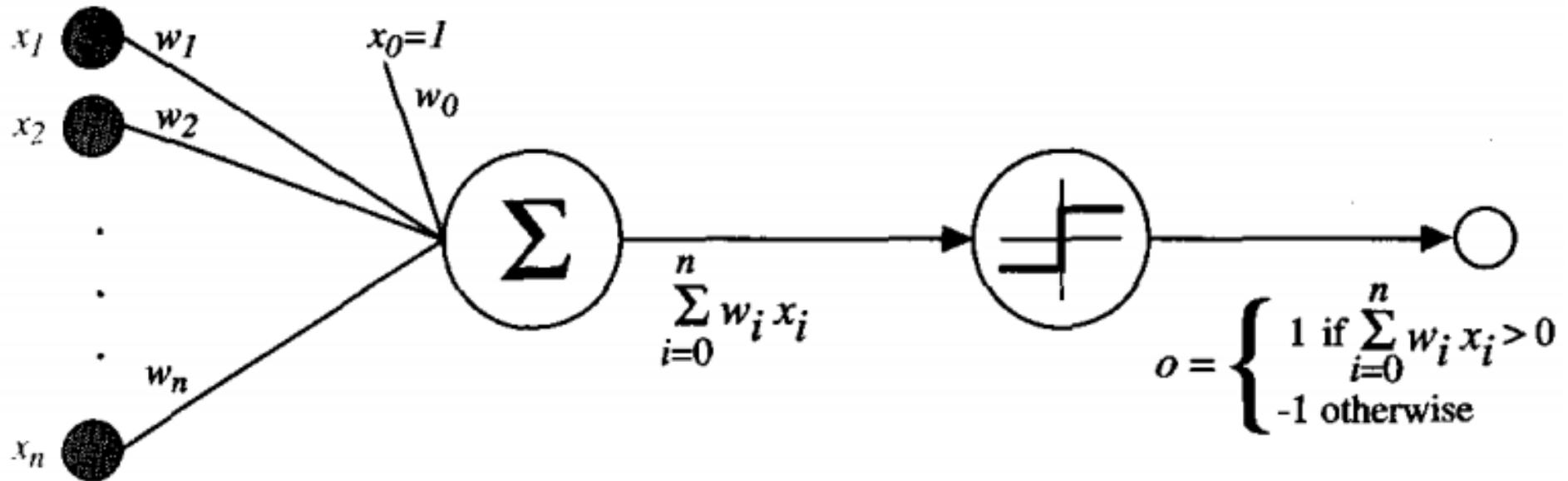
Neural Networks

- Origins: Algorithms that try to mimic the brain.
- Very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications
- Artificial neural networks are not nearly as complex or intricate as the actual brain structure

When to use Neural Network

- Input is high-dimensional discrete or real-valued (e.g. raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is unimportant

Perceptron



$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1x_1 + \dots + w_nx_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Perceptron Training rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Where:

- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., .1) called *learning rate*

Gradient Descent

To understand, consider simpler *linear unit*, where

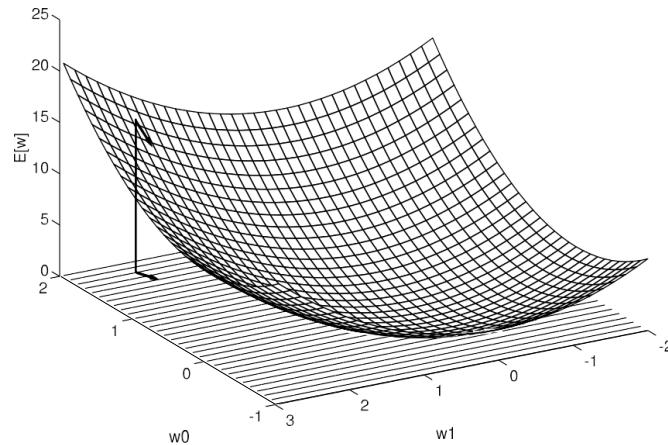
$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

Let's learn w_i 's that minimize the squared error

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Where D is set of training examples

Gradient Descent



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Gradient Descent

GRADIENT-DESCENT(*training_examples*, η)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where \vec{x} is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
 - Initialize each Δw_i to zero.
 - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
 - * Input the instance \vec{x} to the unit and compute the output o
 - * For each linear unit weight w_i , Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

- For each linear unit weight w_i , Do

$$w_i \leftarrow w_i + \Delta w_i$$

Perceptron Training

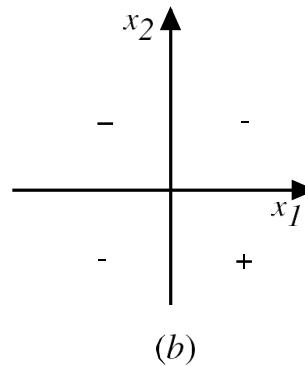
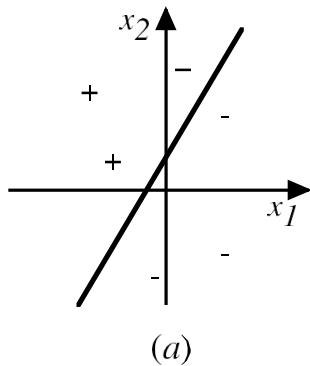
Perceptron training rule guaranteed to succeed if

- Training examples are linearly separable
- Sufficiently small learning rate η

Linear unit training rule uses gradient descent

- Guaranteed to converge to hypothesis with minimum squared error
- Given sufficiently small learning rate η
- Even when training data contains noise
- Even when training data not separable by H

Decision Surface of Perceptron



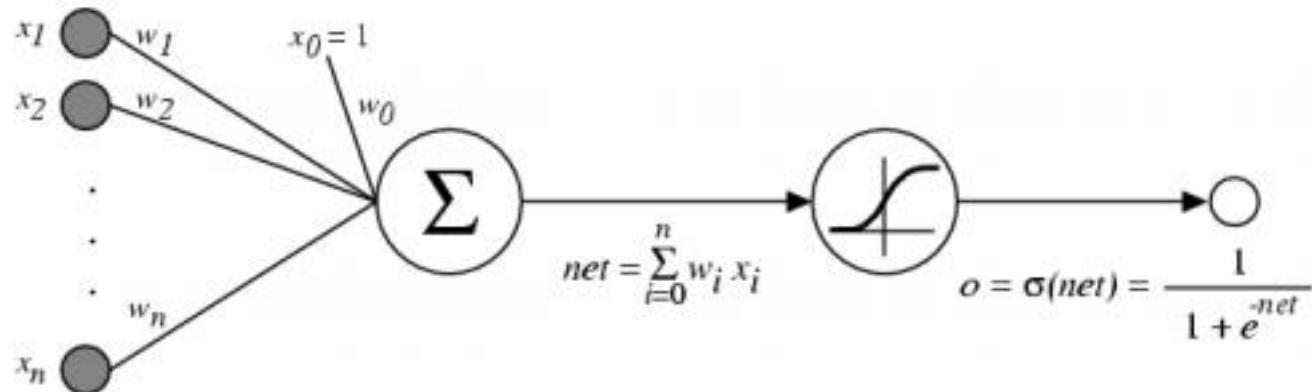
Represents some useful functions

- What weights represent
 $g(x_1, x_2) = AND(x_1, x_2)$?

But some functions not representable

- e.g., not linearly separable
- ..

Perceptron: Sigmoid Function



$\sigma(x)$ is the sigmoid function

$$\frac{1}{1 + e^{-x}}$$

$$\begin{aligned} g'(z) &= \frac{d}{dz} \frac{1}{1 + e^{-z}} \\ &= \frac{1}{(1 + e^{-z})^2} (e^{-z}) \\ &= \frac{1}{(1 + e^{-z})} \cdot \left(1 - \frac{1}{(1 + e^{-z})}\right) \\ &= g(z)(1 - g(z)). \end{aligned}$$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

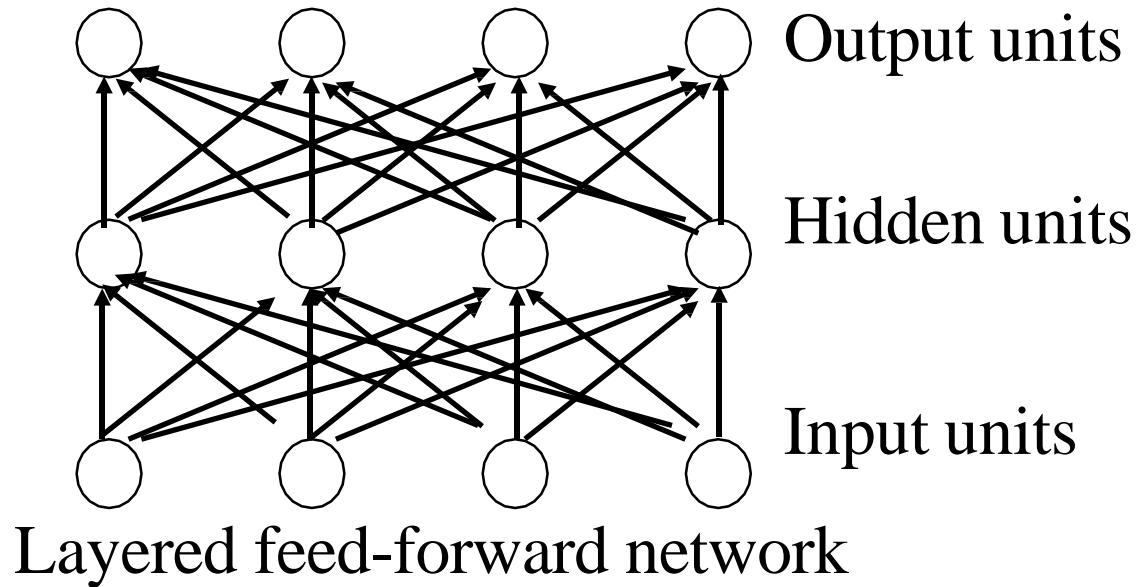
We can derive gradient decent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units → Backpropagation

Multilayer network

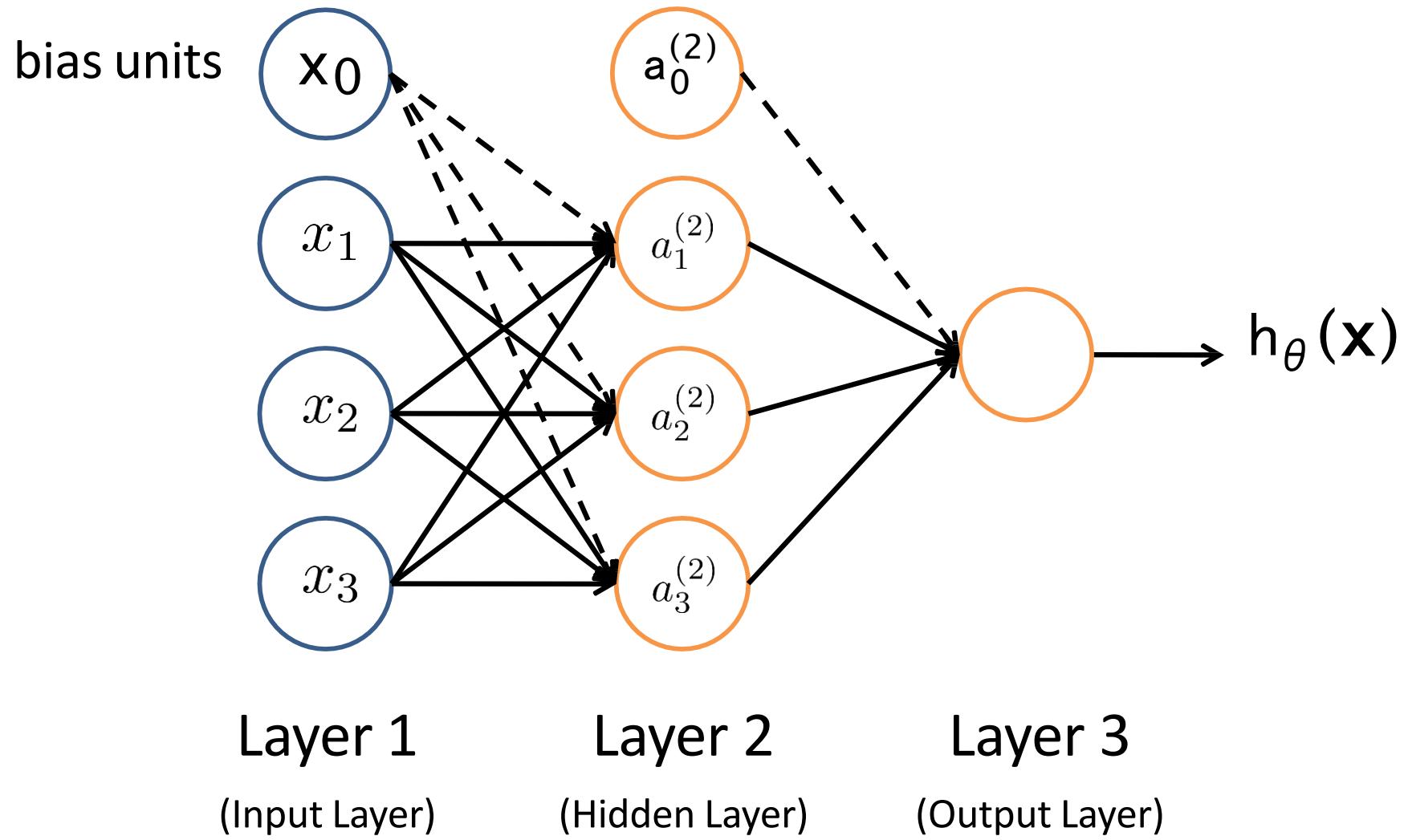
- Single perceptrons can only express linear decision surfaces.
- In contrast, the kind of multilayer networks learned by the FORWARD and BACKPROPAGATION algorithm are capable of expressing a rich variety of nonlinear decision surfaces

Neural networks



- Neural networks are made up of **nodes** or **units**, connected by **links**
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

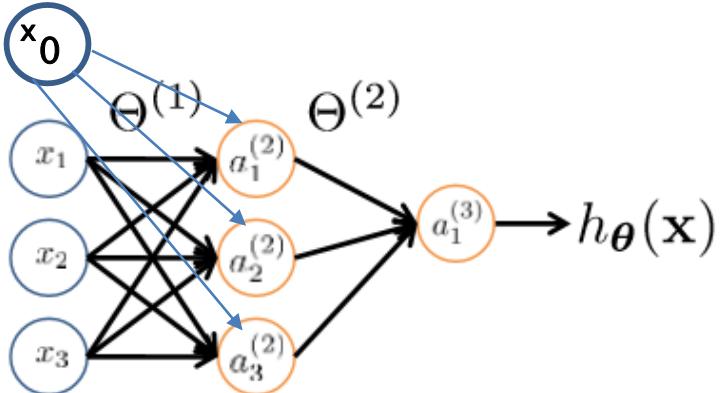
Neural Network



Feed-Forward Process

- Input layer units are set by some exterior function (think of these as **sensors**), which causes their output links to be **activated** at the specified level
- Working forward through the network, the **input function** of each unit is applied to compute the input value
 - Usually this is just the weighted sum of the activation on the links feeding into this node
- The **activation function** transforms this input function into a final value
 - Typically this is a **nonlinear** function, often a **sigmoid** function corresponding to the “threshold” of that node

Neural Network



$a_i^{(j)}$ = “activation” of unit i in layer j
 $\Theta^{(j)}$ = weight matrix controlling function mapping from layer j to layer $j + 1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

$$h_{\Theta}(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

If network has s_j units in layer j and s_{j+1} units in layer $j+1$,
then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j + 1)$.

$$\Theta^{(1)} \in \mathbb{R}^{3 \times 4} \quad \Theta^{(2)} \in \mathbb{R}^{1 \times 4}$$

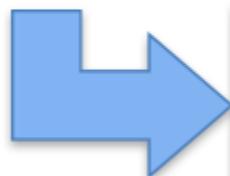
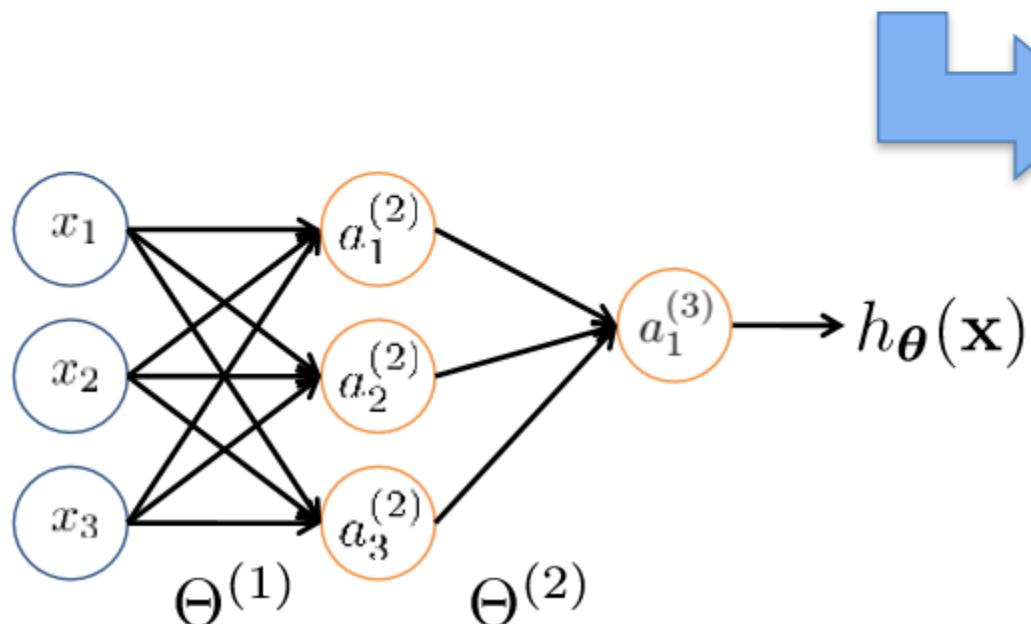
Vectorization

$$a_1^{(2)} = g \left(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3 \right) = g \left(z_1^{(2)} \right)$$

$$a_2^{(2)} = g \left(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3 \right) = g \left(z_2^{(2)} \right)$$

$$a_3^{(2)} = g \left(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3 \right) = g \left(z_3^{(2)} \right)$$

$$h_{\Theta}(\mathbf{x}) = g \left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)} \right) = g \left(z_1^{(3)} \right)$$



Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)} \mathbf{x}$$

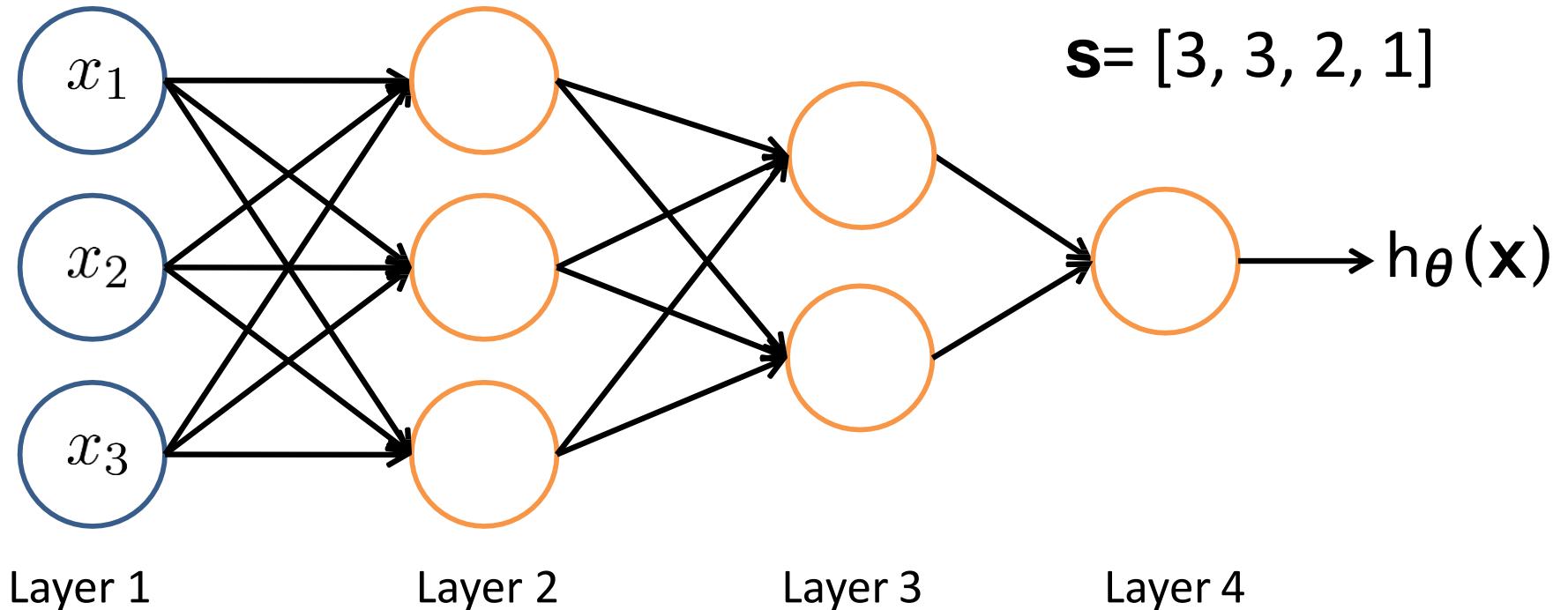
$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)} \mathbf{a}^{(2)}$$

$$h_{\Theta}(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

Other Network Architectures



L denotes the number of layers

$s \in \mathbb{N}^{+^L}$ contains the numbers of nodes at each layer

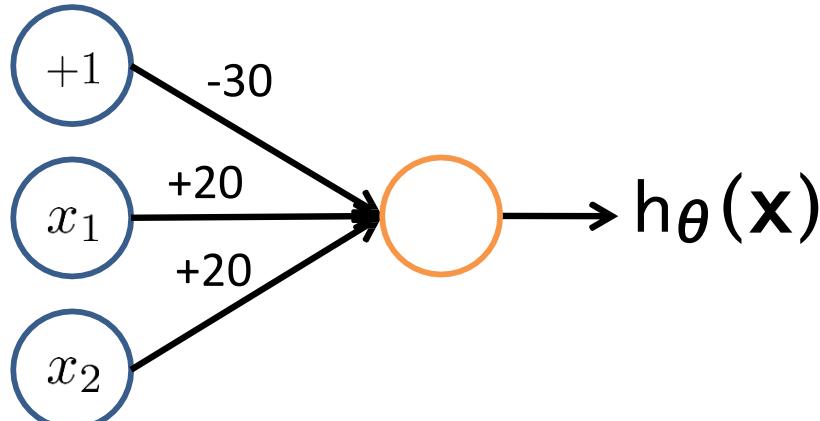
- Not counting bias units
- Typically, $s_0 = d$ (# input features) and $s_{L-1} = K$ (# classes)

Representing Boolean Functions

Simple example: AND

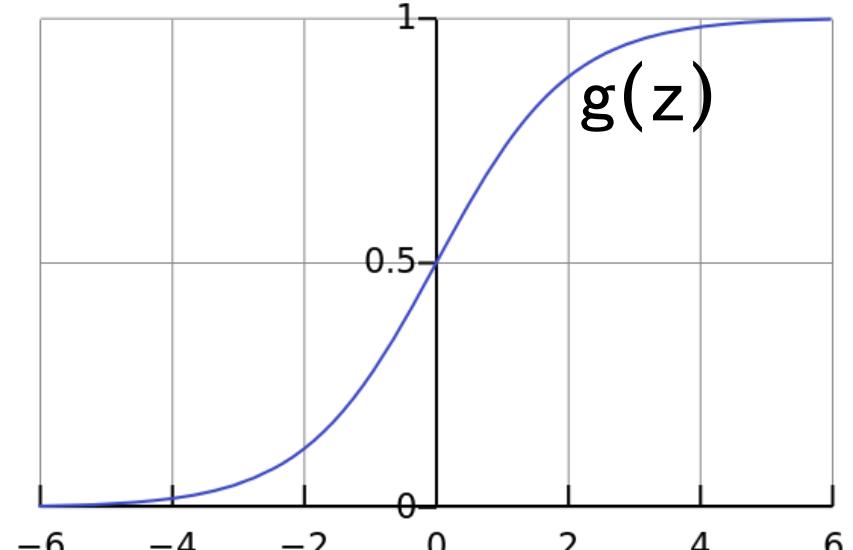
$$x_1, x_2 \in \{0, 1\}$$

$$y = x_1 \text{ AND } x_2$$



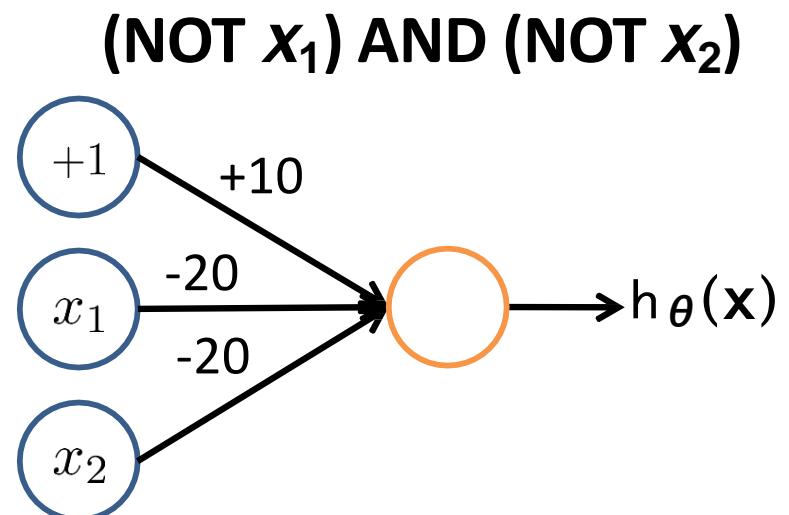
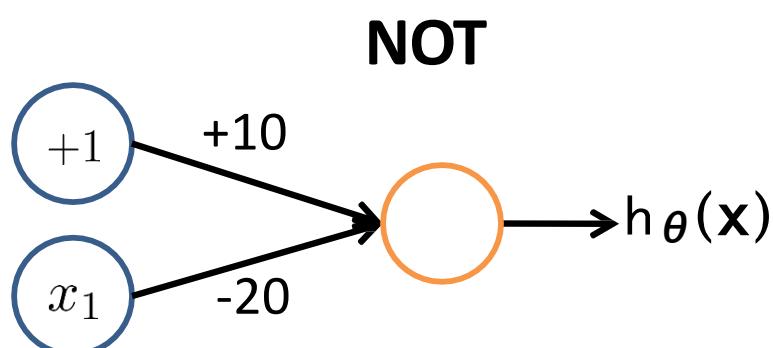
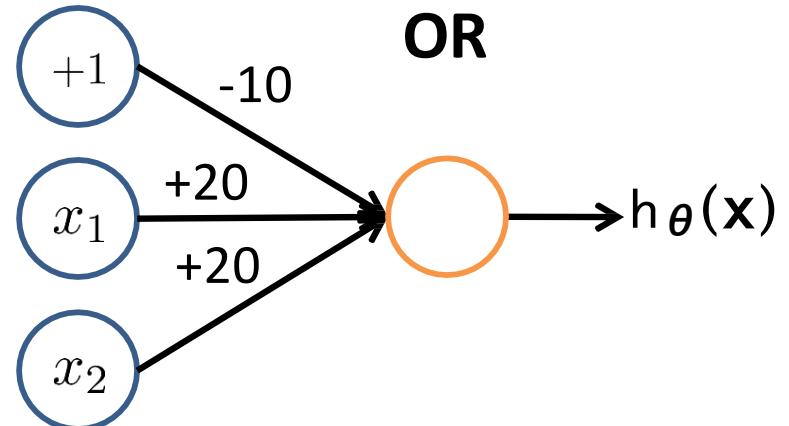
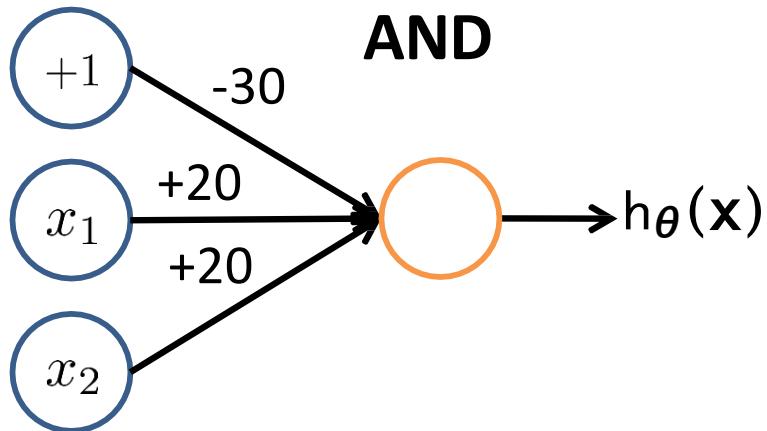
$$h_{\Theta}(\mathbf{x}) = g(-30 + 20x_1 + 20x_2)$$

Logistic / Sigmoid Function

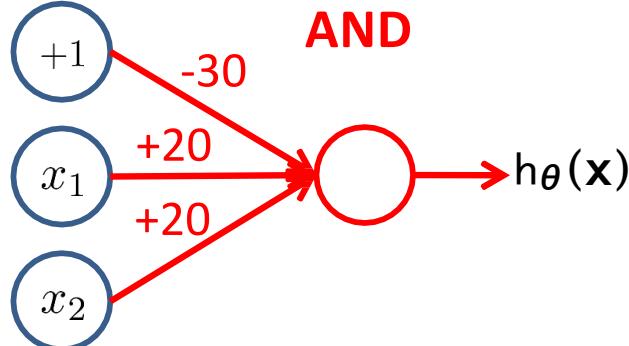


x_1	x_2	$h_{\theta}(\mathbf{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

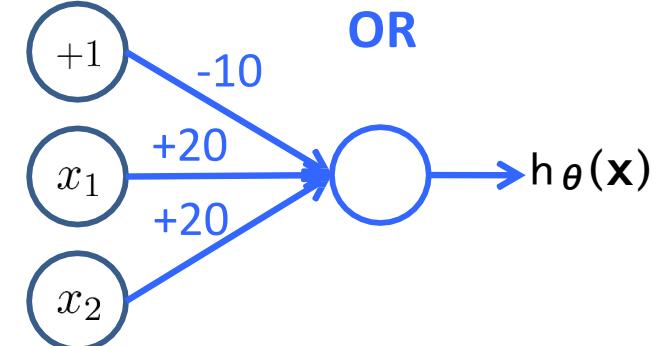
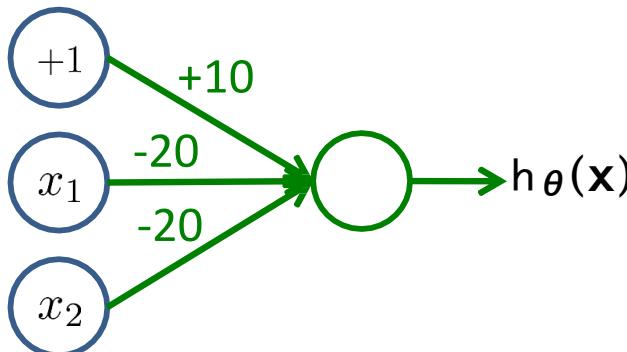
Representing Boolean Functions



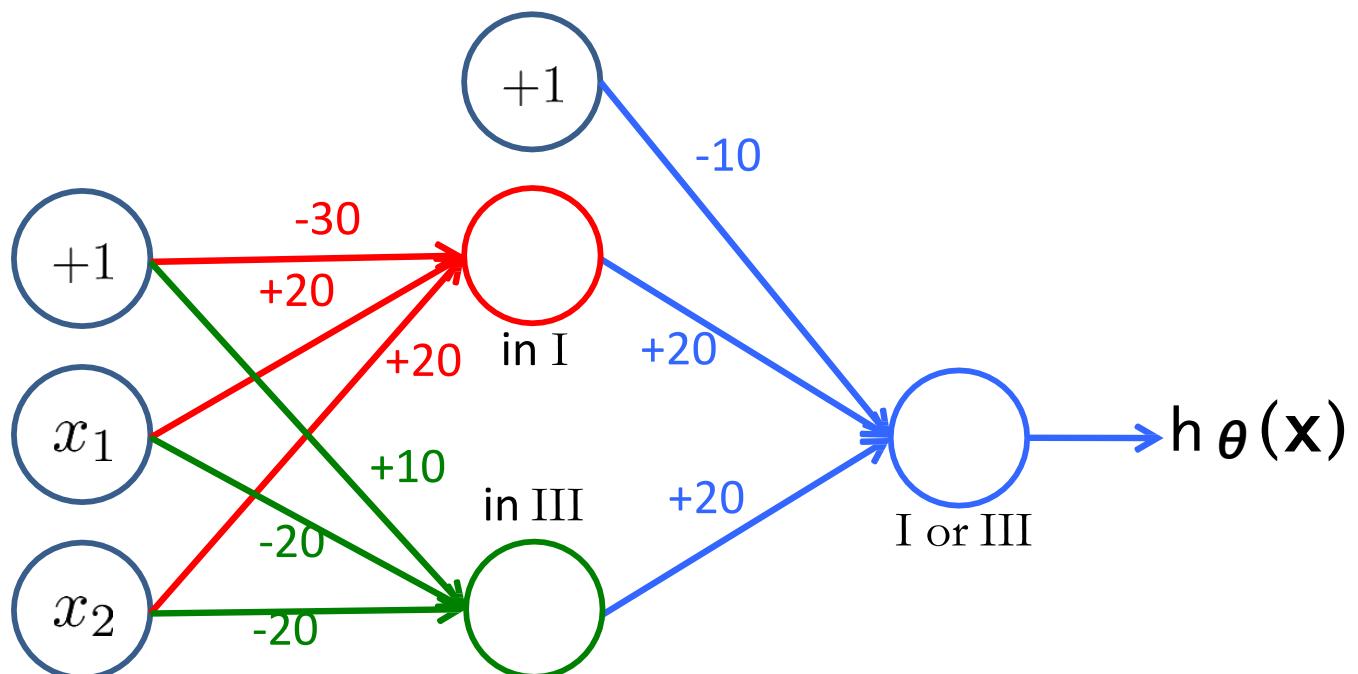
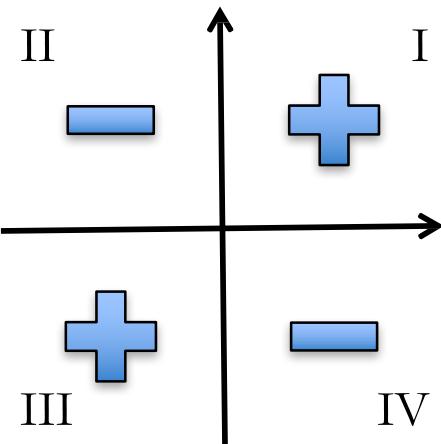
Combining Representations to Create Non-Linear Functions



(NOT x_1) AND (NOT x_2)



not(XOR)



Multiple Output Units: One-vs-Rest



Pedestrian



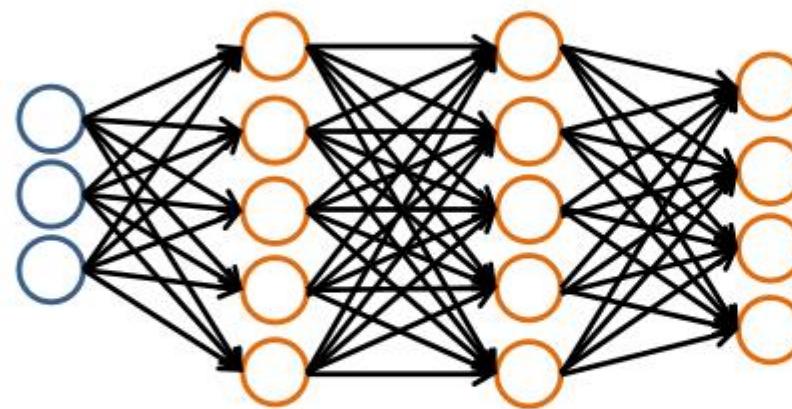
Car



Motorcycle



Truck



$$h_{\Theta}(\mathbf{x}) \in \mathbb{R}^K$$

We want:

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

when pedestrian

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

when car

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

when motorcycle

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

when truck

Multiple Output Units: One-vs-Rest



We want:

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{when pedestrian}$$

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{when car}$$

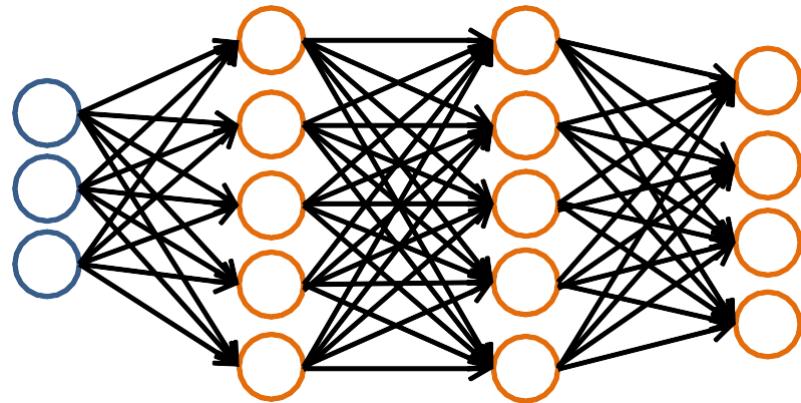
$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \text{when motorcycle}$$

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \text{when truck}$$

- Given $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$
- Must convert labels to 1-of- K representation

– e.g., $y_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ when motorcycle, $y_i = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ when car, etc.

Neural Network Classification



Given:

$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$$

$\mathbf{S} \in \mathbb{N}^{+L}$ contains # nodes at each layer
— $s_0 = d$ (#features)

Binary classification

$$y = 0 \text{ or } 1$$

1 output unit ($s_{L-1} = 1$)

Multi-class classification (K classes)

$$\mathbf{y} \in \mathbb{R}^K \quad \text{e.g. } \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

pedestrian car motorcycle truck

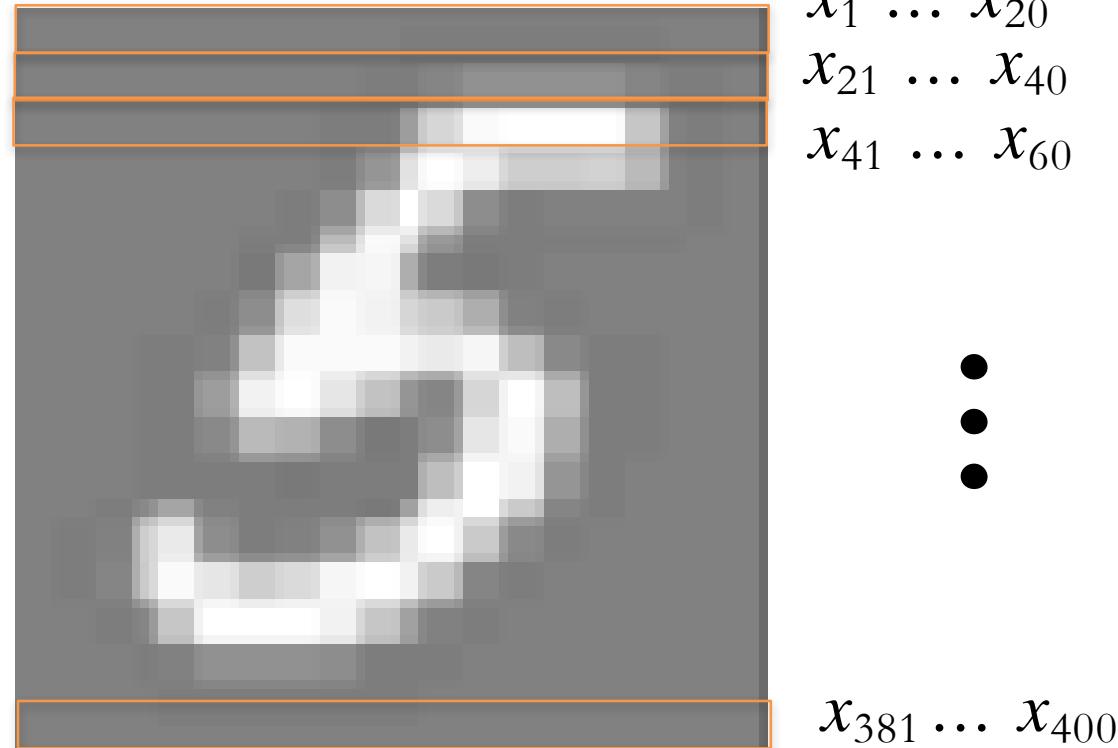
K output units ($s_{L-1} = K$)

Layering Representations



20 × 20 pixel images

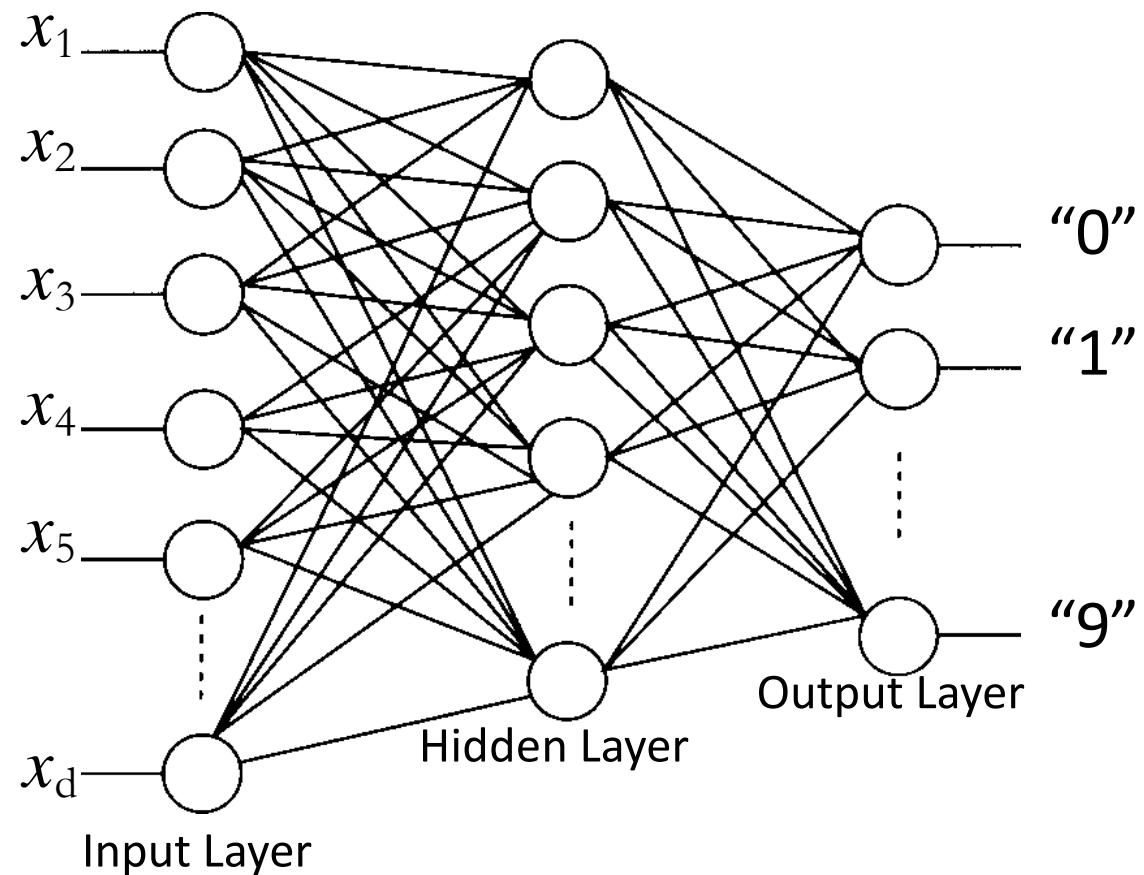
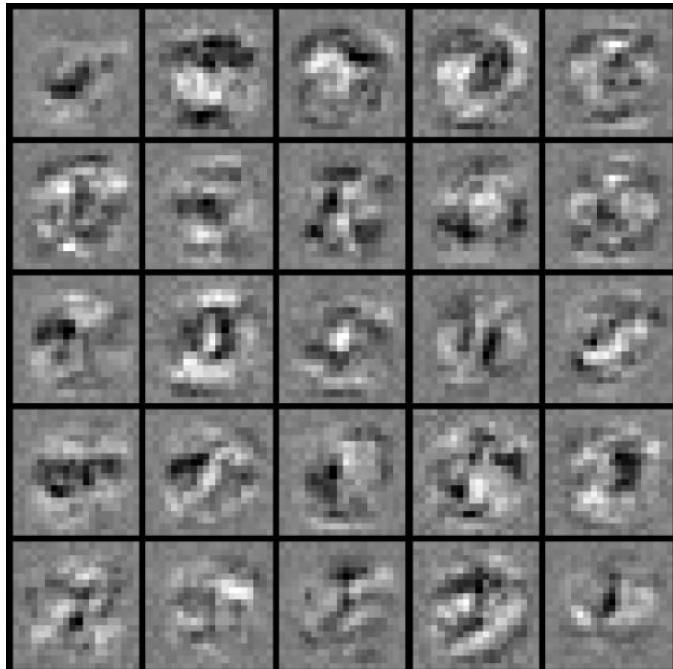
$d = 400$ 10 classes



Each image is “unrolled” into a vector \mathbf{x} of pixel intensities

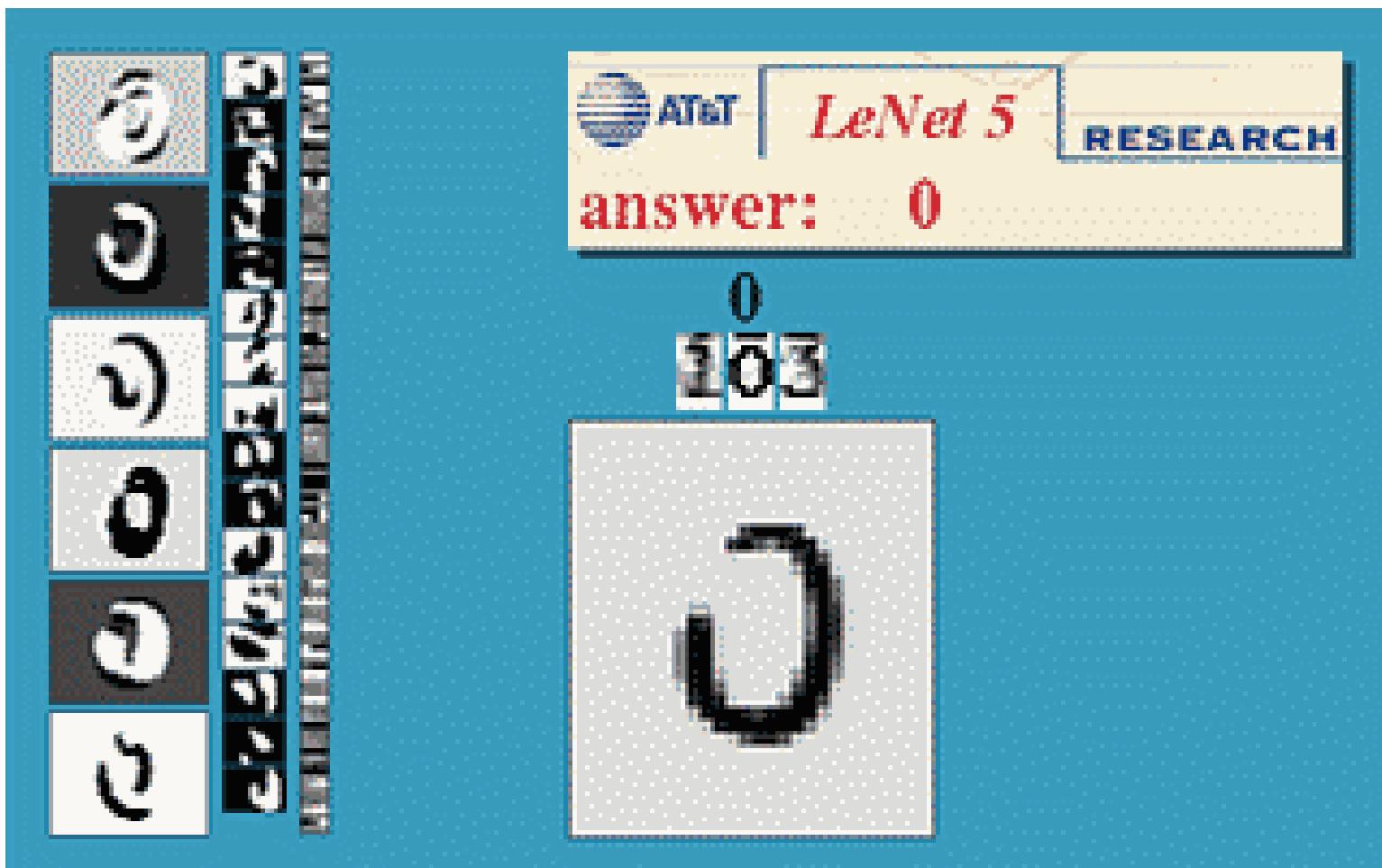
Layering Representations

7	9	6	5	8	7	4	4	1	8
0	7	3	3	2	4	8	4	5	1
6	6	3	2	9	1	3	3	2	6
1	3	7	1	5	6	5	2	4	4
7	0	9	2	7	5	8	9	5	4
4	6	6	5	0	2	1	3	6	9
8	5	1	8	9	3	8	7	3	6
1	0	2	8	2	3	0	5	1	5
6	7	8	2	5	3	9	7	0	0
7	9	3	9	8	5	7	2	9	8



Visualization of
Hidden Layer

Digit Recognition



Handwriting Recognition

LeNet 5 Demonstration:

<http://yann.lecun.com/exdb/lenet/>

<http://yann.lecun.com/exdb/lenet/weirdos.html>

Cost Function

(9.1 NN video of Andrew Ng)

Logistic Regression:

$$J(\theta) = -\frac{1}{n} \sum_{i=1}^n [y_i \log h_{\theta}(\mathbf{x}_i) + (1 - y_i) \log (1 - h_{\theta}(\mathbf{x}_i))] + \frac{\lambda}{2n} \sum_{j=1}^d \theta_j^2$$

Neural Network:

$$h_{\Theta} \in \mathbb{R}^K \quad (h_{\Theta}(\mathbf{x}))_i = i^{th} \text{output}$$

$$J(\Theta) = -\frac{1}{n} \left[\sum_{i=1}^n \sum_{k=1}^K y_{ik} \log (h_{\Theta}(\mathbf{x}_i))_k + (1 - y_{ik}) \log (1 - (h_{\Theta}(\mathbf{x}_i))_k) \right] + \frac{\lambda}{2n} \sum_{l=1}^{L-1} \sum_{i=1}^{s_{l-1}} \sum_{j=1}^{s_l} \left(\Theta_{ji}^{(l)} \right)^2$$

k^{th} class: true, predicted
not k^{th} class: true, predicted

Optimizing the Neural Network

$$J(\Theta) = -\frac{1}{n} \left[\sum_{i=1}^n \sum_{k=1}^K y_{ik} \log(h_\Theta(\mathbf{x}_i))_k + (1 - y_{ik}) \log(1 - (h_\Theta(\mathbf{x}_i))_k) \right] \\ + \frac{\lambda}{2n} \sum_{l=1}^{L-1} \sum_{i=1}^{s_{l-1}} \sum_{j=1}^{s_l} \left(\Theta_{ji}^{(l)} \right)^2$$

Solve via: $\min_{\Theta} J(\Theta)$

$J(\Theta)$ is not convex, so GD on a neural net yields a local optimum

- But, tends to work well in practice

Need code to compute:

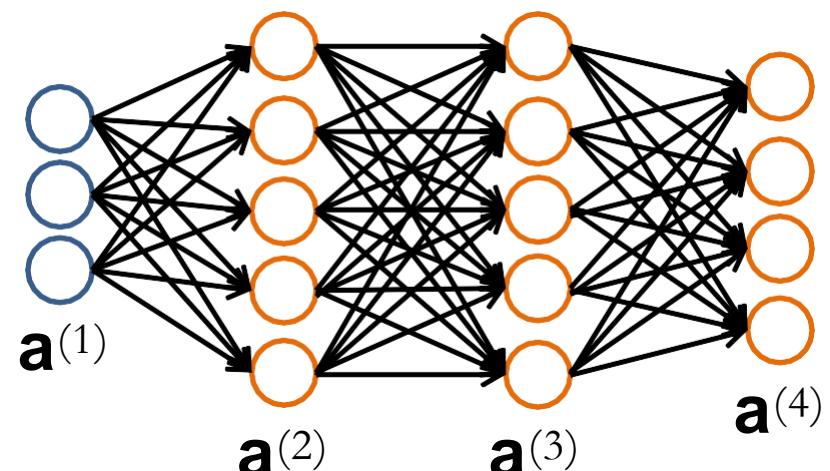
- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

Forward Propagation

- Given one labeled training instance (\mathbf{x}, y) :

Forward Propagation

- $\mathbf{a}^{(1)} = \mathbf{x}$
- $\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{a}^{(1)}$
- $\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$ [add $a_0^{(2)}$]
- $\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$
- $\mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$ [add $a_0^{(3)}$]
- $\mathbf{z}^{(4)} = \Theta^{(3)}\mathbf{a}^{(3)}$
- $\mathbf{a}^{(4)} = h_{\Theta}(\mathbf{x}) = g(\mathbf{z}^{(4)})$



Learning in NN: Backpropagation

- Similar to the perceptron learning algorithm, we cycle through our examples
 - If the output of the network is correct, no changes are made
 - If there is an error, weights are adjusted to reduce the error
- The trick is to assess the blame for the error and divide it among the contributing weights

Backpropagation Intuition

- Each hidden node j is “responsible” for some fraction of the error $\delta_j^{(l)}$ in each of the output nodes to which it connects
- $\delta_j^{(l)}$ is divided according to the strength of the connection between hidden node and the output node
- Then, the “blame” is propagated back to provide the error values for the hidden layer

Chain rule

Chain Rule of Differentiation (reminder)

- The rate of change of a function of a function is the multiple of the derivatives of those functions.
- You have probably learned this in school (nothing new here)

$$\frac{\partial}{\partial x} f(g(x)) = g'(x) \cdot f'(g)$$

$$\frac{\partial}{\partial x} f(g(h(i(j(k(x)))))) = \frac{\partial k}{\partial x} \frac{\partial j}{\partial k} \frac{\partial i}{\partial j} \frac{\partial h}{\partial i} \frac{\partial g}{\partial h} \frac{\partial f}{\partial g}$$

Backpropagation

Backpropagation Generalized to several layers

$$\begin{array}{ccccccc}
 a_4 & & a_3 = w_3 \cdot a_4 & a_2 = w_2 \cdot a_3 & a_1 = w_1 \cdot a_2 & a_0 = w_0 \cdot a_1 \\
 \hline
 \text{---} & & \text{---} & \text{---} & \text{---} & \text{---} \\
 \bullet & w_3 & \bullet & w_2 & \bullet & w_1 & \bullet & w_0 & \bullet
 \end{array}$$

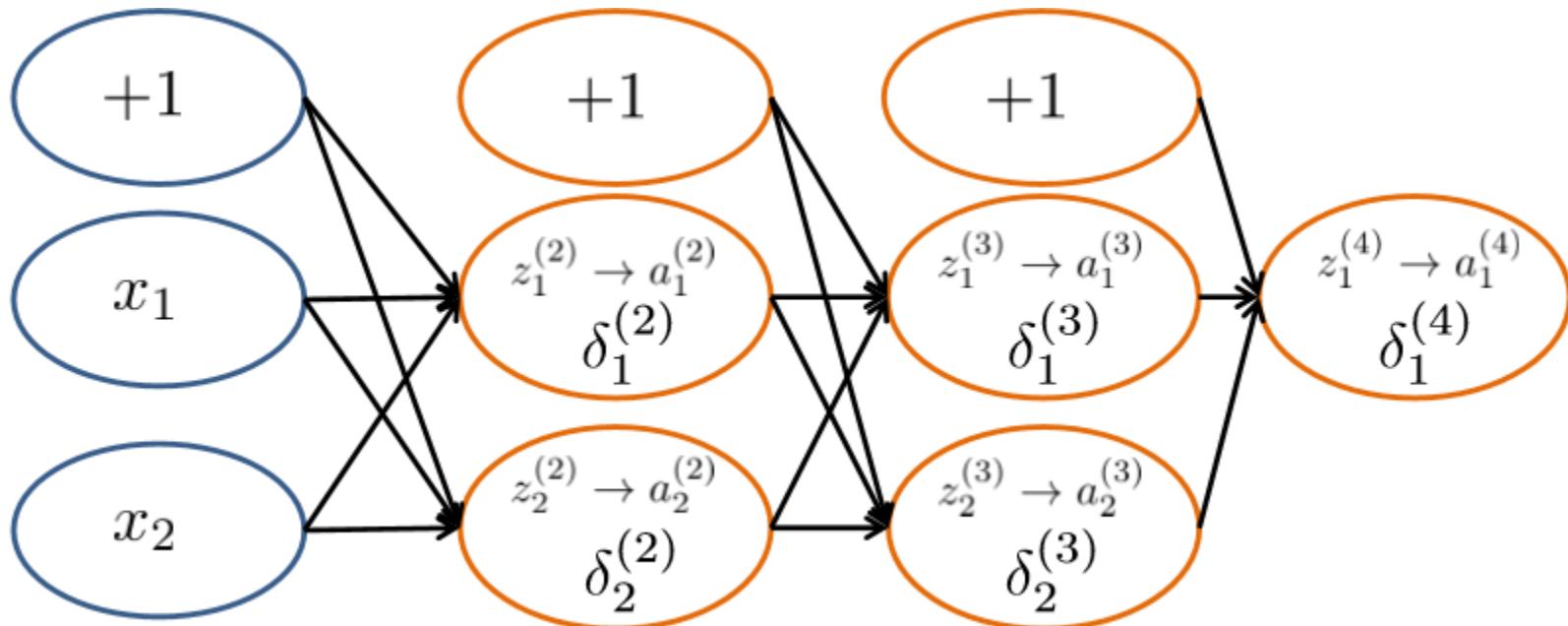
$$\begin{aligned}
 & -r \frac{\partial a_1}{\partial w_1} \frac{\partial a_0}{\partial a_1} \frac{\partial C}{\partial a_0} & -r \frac{\partial a_0}{\partial w_0} \frac{\partial C}{\partial a_0} \\
 & -r \frac{\partial a_2}{\partial w_2} \frac{\partial a_1}{\partial a_2} \frac{\partial a_0}{\partial a_1} \frac{\partial C}{\partial a_0} & C = (a_0 - y)^2 \\
 & -r \frac{\partial a_3}{\partial w_3} \frac{\partial a_2}{\partial a_3} \frac{\partial a_1}{\partial a_2} \frac{\partial a_0}{\partial a_1} \frac{\partial C}{\partial a_0}
 \end{aligned}$$

For example, we adjust w_3 as follows:

$$w'_3 = w_3 - r \cdot a_4 \cdot w_2 \cdot w_1 \cdot w_0 \cdot 2(a_0 - y)$$

$$-r \frac{\partial a_3}{\partial w_3} \frac{\partial a_2}{\partial a_3} \frac{\partial a_1}{\partial a_2} \frac{\partial a_0}{\partial a_1} \frac{\partial C}{\partial a_0}$$

Backpropagation Intuition

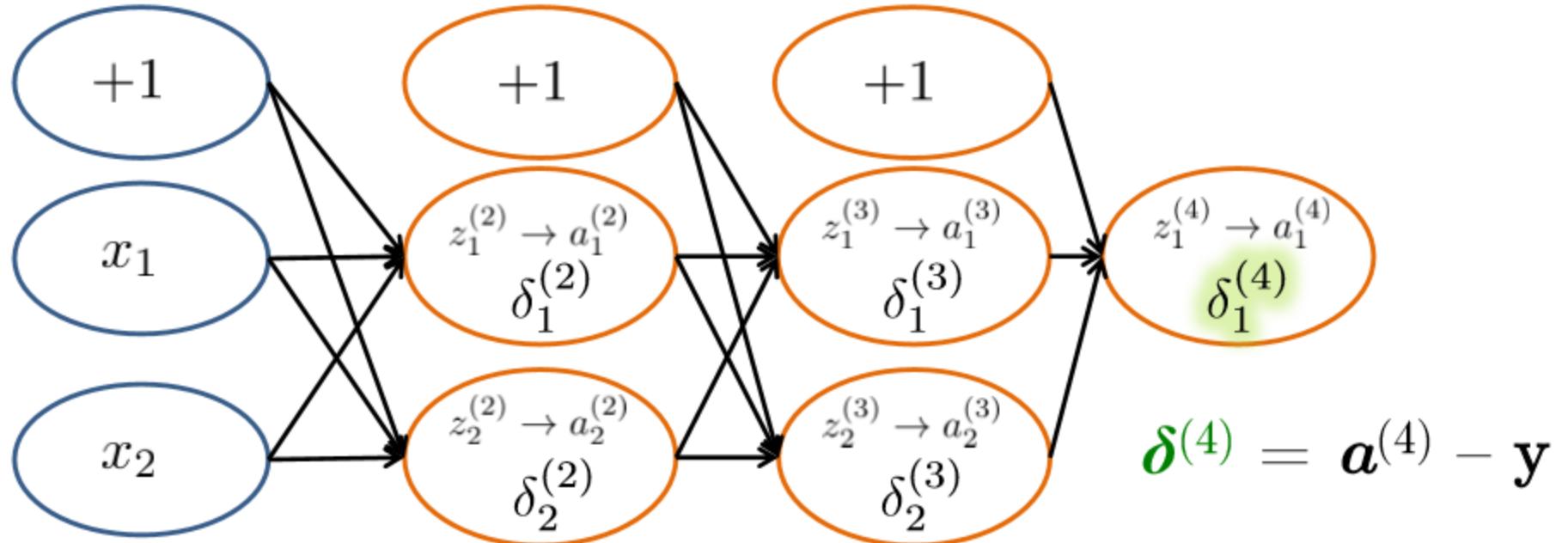


$\delta_j^{(l)}$ = “error” of node j in layer l

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

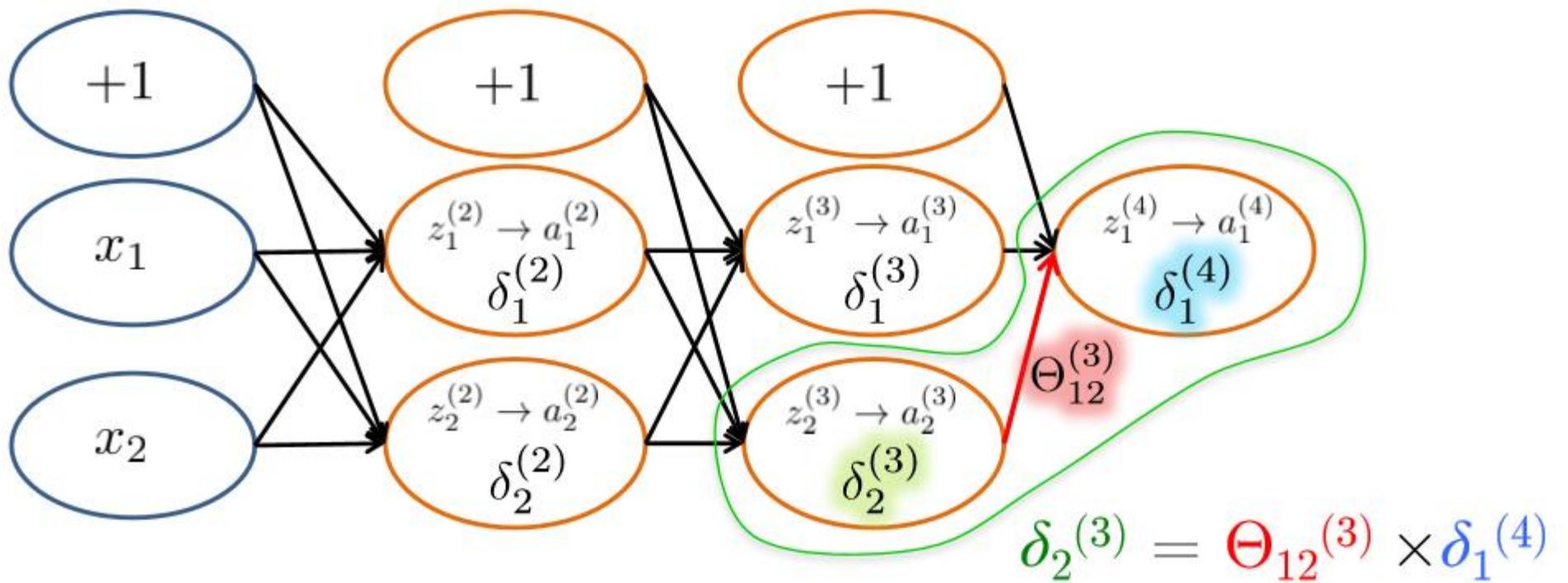
Backpropagation Intuition



$\delta_j^{(l)}$ = “error” of node j in layer l

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

Backpropagation Intuition

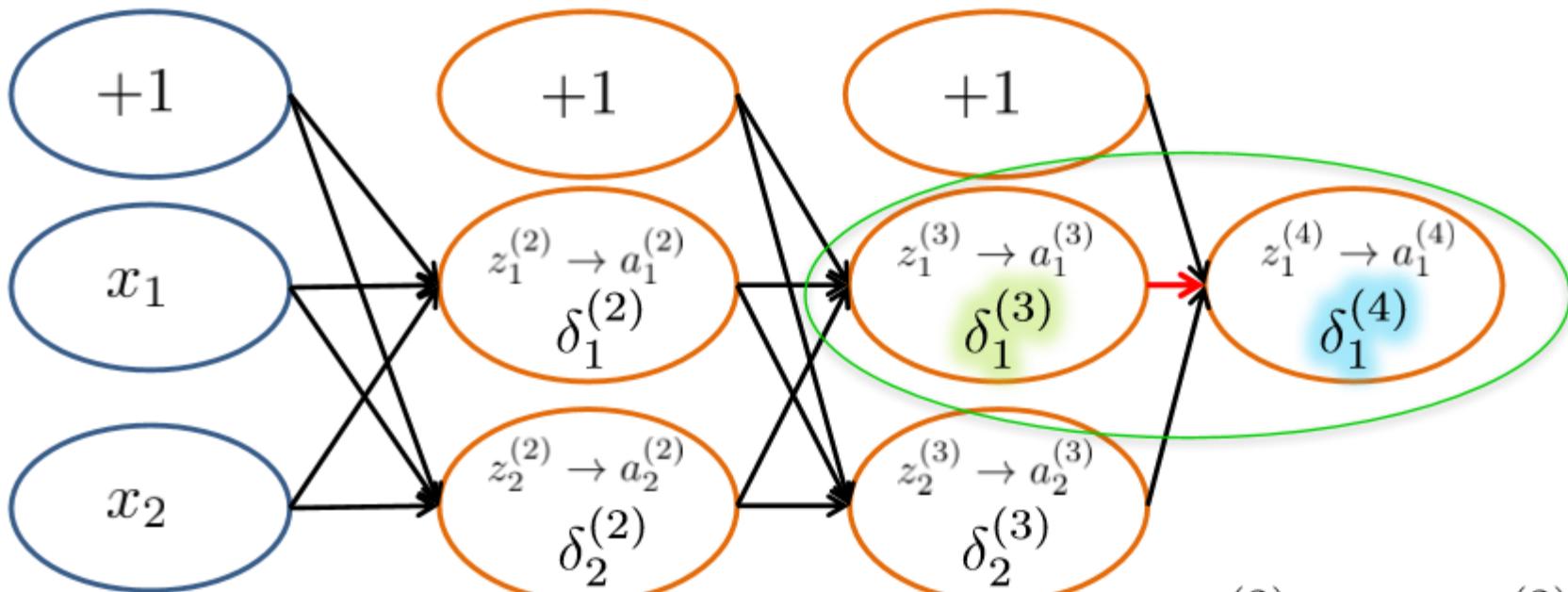


$\delta_j^{(l)}$ = “error” of node j in layer l

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Backpropagation Intuition



$$\delta_2^{(3)} = \Theta_{12}^{(3)} \times \delta_1^{(4)}$$

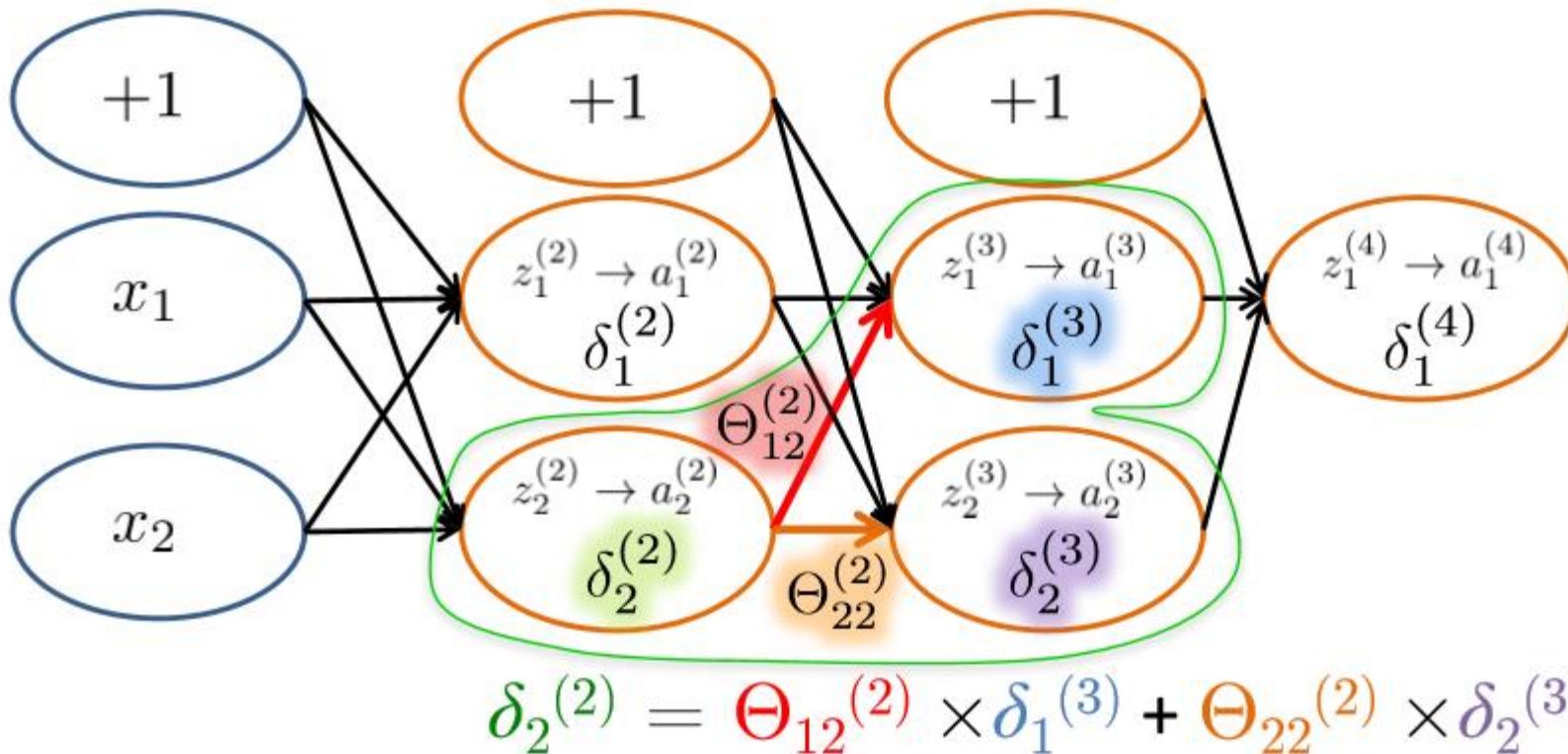
$$\delta_1^{(3)} = \Theta_{11}^{(3)} \times \delta_1^{(4)}$$

$\delta_j^{(l)}$ = “error” of node j in layer l

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Backpropagation Intuition



$\delta_j^{(l)}$ = “error” of node j in layer l

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

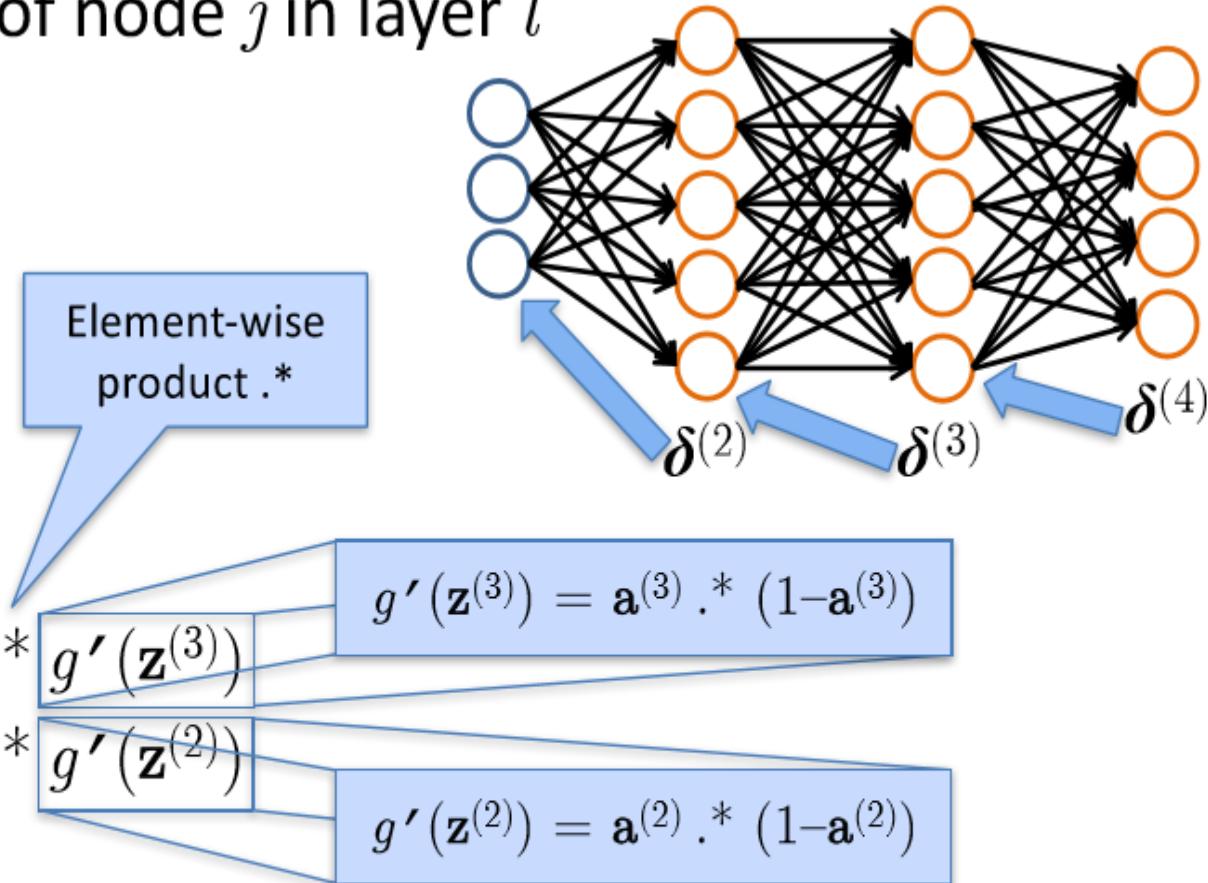
Backpropagation: Gradient Computation

Let $\delta_j^{(l)}$ = “error” of node j in layer l

(#layers $L = 4$)

Backpropagation

- $\delta^{(4)} = a^{(4)} - y$
- $\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .*$
- $\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .*$
- (No $\delta^{(1)}$)



$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)} \quad (\text{ignoring } \lambda; \text{ if } \lambda = 0)$$

Backpropagation

Set $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$

(Used to accumulate gradient)

For each training instance (\mathbf{x}_i, y_i) :

Set $\mathbf{a}^{(1)} = \mathbf{x}_i$

Compute $\{\mathbf{a}^{(2)}, \dots, \mathbf{a}^{(L)}\}$ via forward propagation

Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_i$

Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \dots, \boldsymbol{\delta}^{(2)}\}$

Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Compute avg regularized gradient $D_{ij}^{(l)} = \begin{cases} \frac{1}{n} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n} \Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$

$D^{(l)}$ is the matrix of partial derivatives of $J(\Theta)$

Note: Can vectorize $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ as $\Delta^{(l)} = \Delta^{(l)} + \boldsymbol{\delta}^{(l+1)} \mathbf{a}^{(l)\top}$

Backpropagation

Backpropagation

Given: training set $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$

Initialize all $\Theta^{(l)}$ randomly (NOT to 0!)

Loop // each iteration is called an epoch

Set $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$ (Used to accumulate gradient)

For each training instance (\mathbf{x}_i, y_i) :

Set $\mathbf{a}^{(1)} = \mathbf{x}_i$

Compute $\{\mathbf{a}^{(2)}, \dots, \mathbf{a}^{(L)}\}$ via forward propagation

Compute $\delta^{(L)} = \mathbf{a}^{(L)} - y_i$

Compute errors $\{\delta^{(L-1)}, \dots, \delta^{(2)}\}$

Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

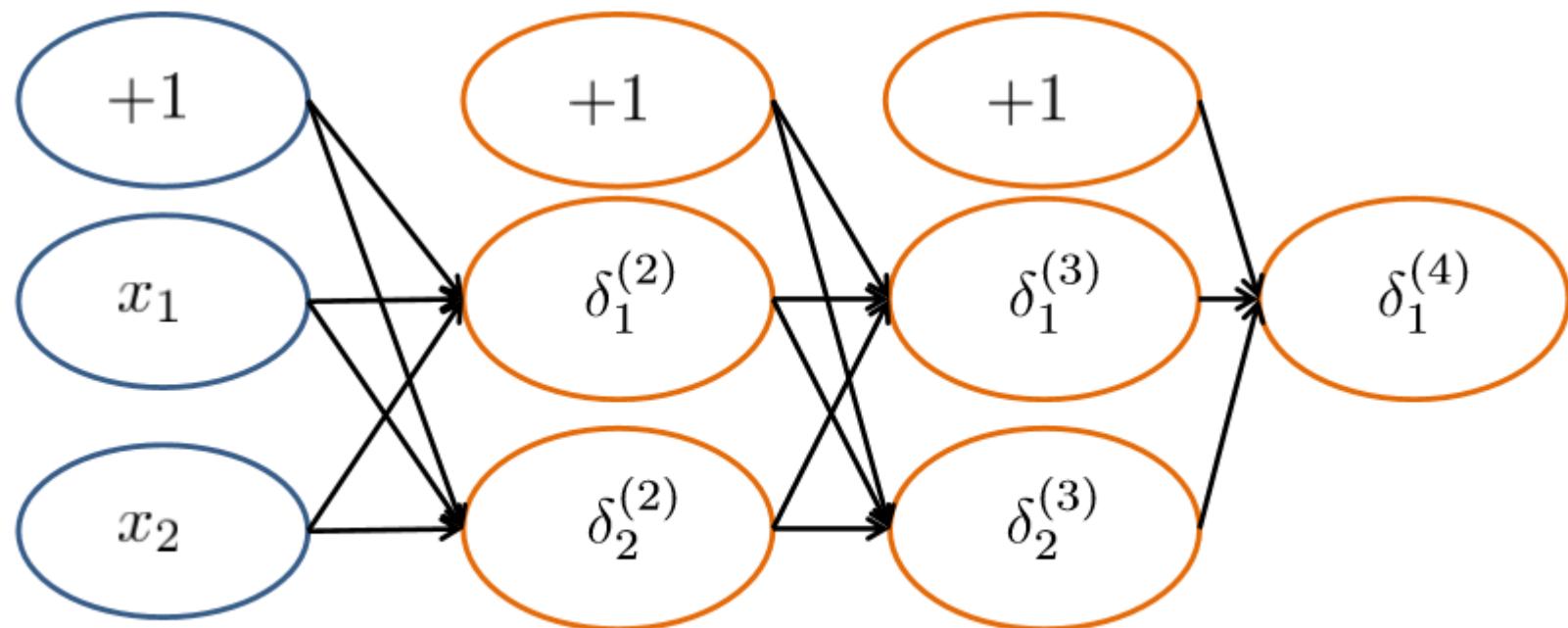
Compute avg regularized gradient $D_{ij}^{(l)} = \begin{cases} \frac{1}{n} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n} \Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$

Update weights via gradient step $\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha D_{ij}^{(l)}$

Until weights converge or max #epochs is reached

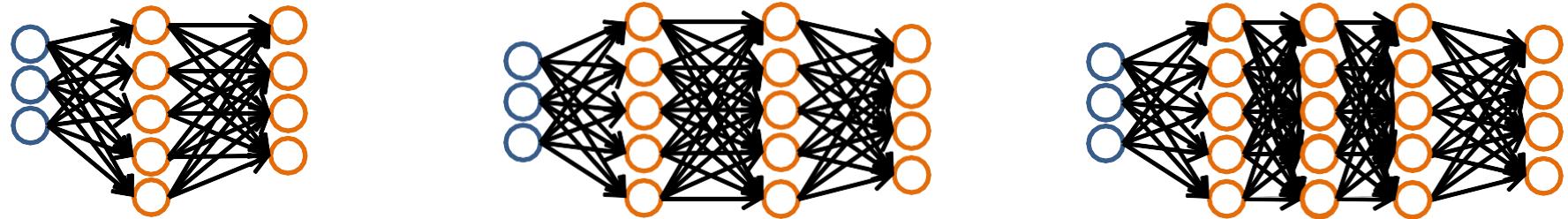
Random Initialization

- Important to randomize initial weight matrices
- Can't have uniform initial weights, as in logistic regression
 - Otherwise, all updates will be identical & the net won't learn



Training a Neural Network

Pick a network architecture (connectivity pattern between nodes)



- # input units = # of features in dataset
- # output units = # classes

Reasonable default: 1 hidden layer

- or if >1 hidden layer, have same # hidden units in every layer (usually the more the better)

Training a Neural Network

1. Randomly initialize weights
2. Implement forward propagation to get $h_{\Theta}(\mathbf{x}_i)$ for any instance \mathbf{x}_i
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivatives
$$\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$$
5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. the numerical gradient estimate.
 - Then, disable gradient checking code
6. Use gradient descent with backprop to fit the network

Good References for understanding Neural Network

Andrew Ng videos on neural network

https://www.youtube.com/watch?v=EVegrPGfuCY&list=PLLssT5z_DsK-h9vYZkQkYNWcItqhlRJLN&index=45

Autonomous driving using neural network

https://www.youtube.com/watch?v=ppFyPUx9RIU&list=PLLssT5z_DsK-h9vYZkQkYNWcItqhlRJLN&index=57

Gradient Descent

Initialise w, b

Iterate over data:

compute \hat{y}

compute $\mathcal{L}(w, b)$

$$w_{t+1} = w_t - \eta \Delta w_t$$

$$b_{t+1} = b_t - \eta \Delta b_t$$

till satisfied

Please note that w and θ are the two notations used to represent weights of the model. Some books refer as w and some books refer as θ .

Derivative of cross entropy

$$\mathcal{L}(\theta) = -[(1-y) \log(1-\hat{y}) + y \log \hat{y}]$$

Using Chain Rule:

$$\Delta w = \frac{\partial \mathcal{L}(\theta)}{\partial w} = \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w}$$

$$\frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}} \{ -(1-y) \log(1-\hat{y}) - y \log \hat{y} \}$$

$$= (-)(-1) \frac{(1-y)}{(1-\hat{y})} - \frac{y}{\hat{y}}$$

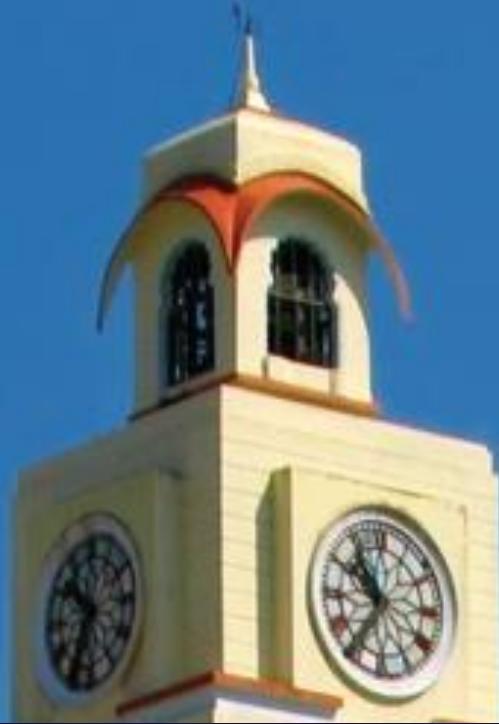
$$= \frac{\hat{y}(1-y) - y(1-\hat{y})}{(1-\hat{y})\hat{y}}$$

$$= \frac{\hat{y}-y}{(1-\hat{y})\hat{y}}$$

Derivative of cross entropy

$$\begin{aligned}\frac{\partial \hat{y}}{\partial w} &= \frac{\partial}{\partial w} \left(\frac{1}{1+e^{-(wx+b)}} \right) \\&= \frac{-1}{(1+e^{-(wx+b)})^2} \frac{\partial}{\partial w} (e^{-(wx+b)}) \\&= \frac{-1}{(1+e^{-(wx+b)})^2} * (e^{-(wx+b)}) \frac{\partial}{\partial w}(-(wx+b)) \\&= \frac{-1}{(1+e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1+e^{-(wx+b)})} * (-x) \\&= \frac{1}{(1+e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1+e^{-(wx+b)})} * (x) \\&= \hat{y} * (1 - \hat{y}) * x\end{aligned}$$

$$\Delta w = (\hat{y} - y) * x$$



Machine Learning

DSECL ZG565

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in



BITS Pilani

Pilani Campus



Lecture No. – 10 | Convolutional Neural Network

Date – 06/12/2020

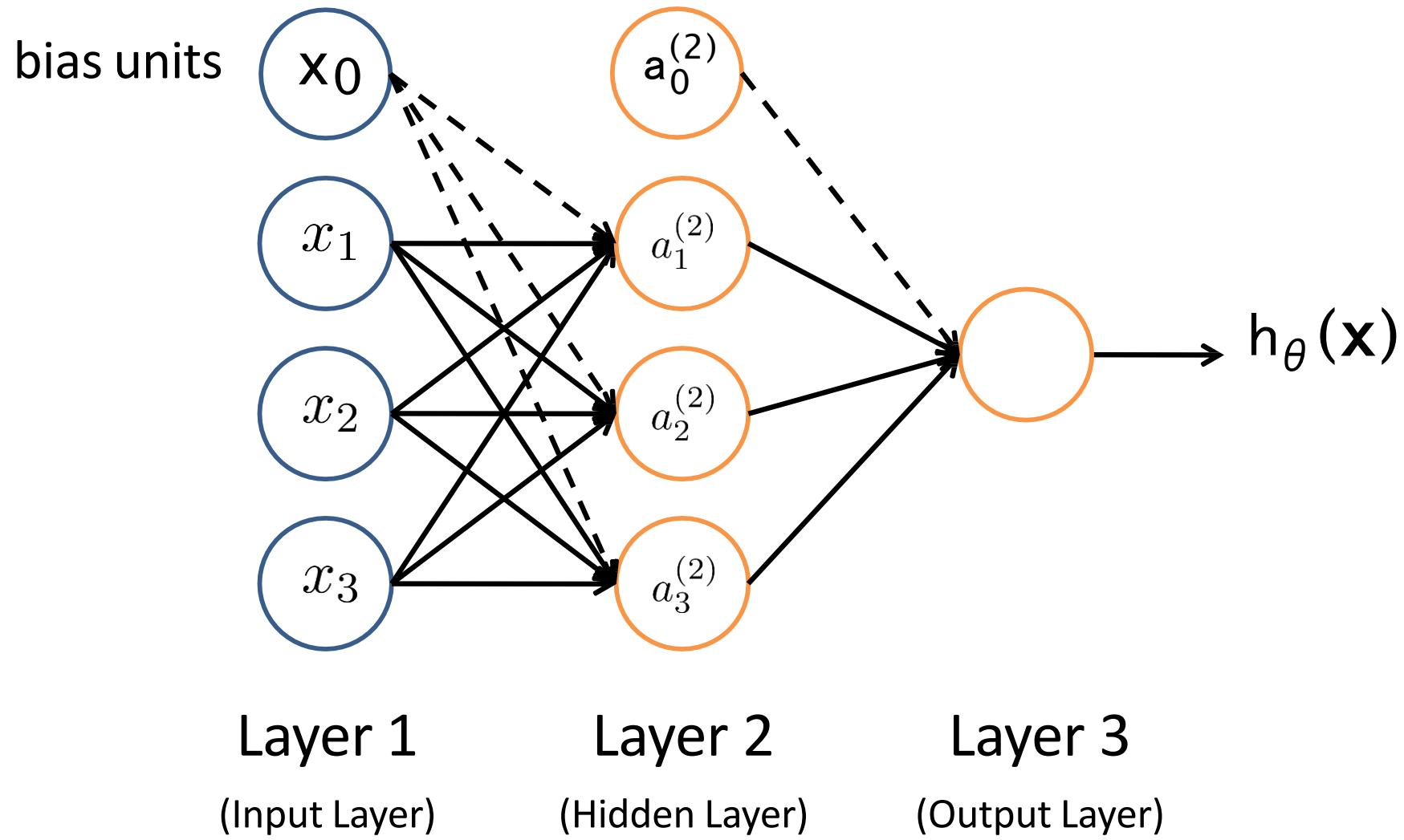
Time – 1:55 PM – 4:05 PM

These slides are prepared by the instructor, with grateful acknowledgement of Andrew Ng, Tom Mitchell, Mitesh Khapra and many others who made their course materials freely available online.

Session Content

- Back propagation Algorithm (Andrew Ng Notes and Tom Mitchell chapter 4)
 - Training Neural Network
 - Hidden Layer Representations
 - Overfitting in ANN
 - Early Stopping
 - Drop out
 - Parameter sharing
- Convolutional Neural Network

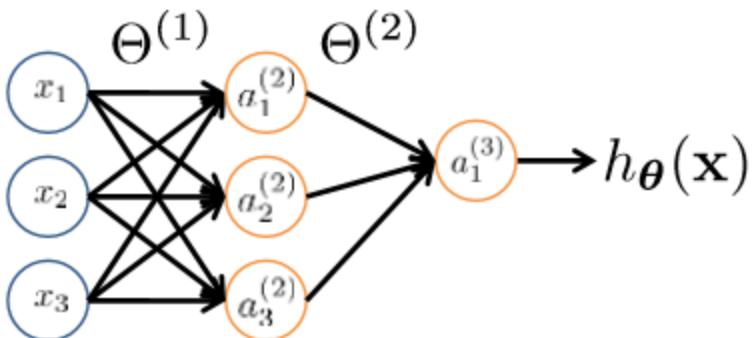
Neural Network



Feed-Forward Process

- Input layer units are set by some exterior function (think of these as **sensors**), which causes their output links to be **activated** at the specified level
- Working forward through the network, the **input function** of each unit is applied to compute the input value
 - Usually this is just the weighted sum of the activation on the links feeding into this node
- The **activation function** transforms this input function into a final value
 - Typically this is a **nonlinear** function, often a **sigmoid** function corresponding to the “threshold” of that node

Neural Network



$a_i^{(j)}$ = “activation” of unit i in layer j
 $\Theta^{(j)}$ = weight matrix controlling function mapping from layer j to layer $j + 1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

$$h_{\Theta}(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

If network has s_j units in layer j and s_{j+1} units in layer $j+1$,
then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j + 1)$.

$$\Theta^{(1)} \in \mathbb{R}^{3 \times 4} \quad \Theta^{(2)} \in \mathbb{R}^{1 \times 4}$$

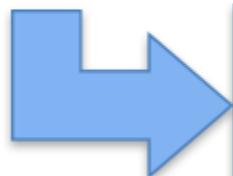
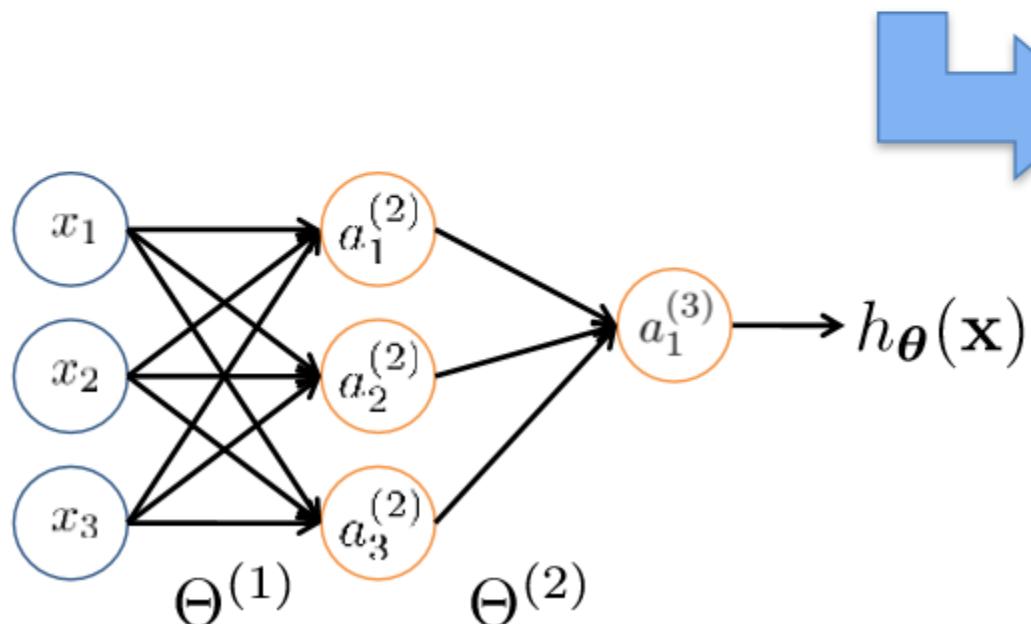
Vectorization

$$a_1^{(2)} = g \left(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3 \right) = g \left(z_1^{(2)} \right)$$

$$a_2^{(2)} = g \left(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3 \right) = g \left(z_2^{(2)} \right)$$

$$a_3^{(2)} = g \left(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3 \right) = g \left(z_3^{(2)} \right)$$

$$h_{\Theta}(\mathbf{x}) = g \left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)} \right) = g \left(z_1^{(3)} \right)$$



Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)} \mathbf{x}$$

$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)} \mathbf{a}^{(2)}$$

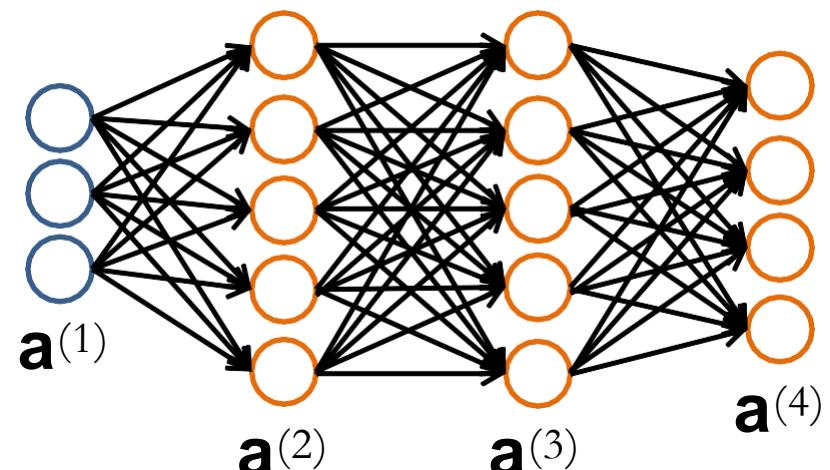
$$h_{\Theta}(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

Forward Propagation

- Given one labeled training instance (\mathbf{x}, y) :

Forward Propagation

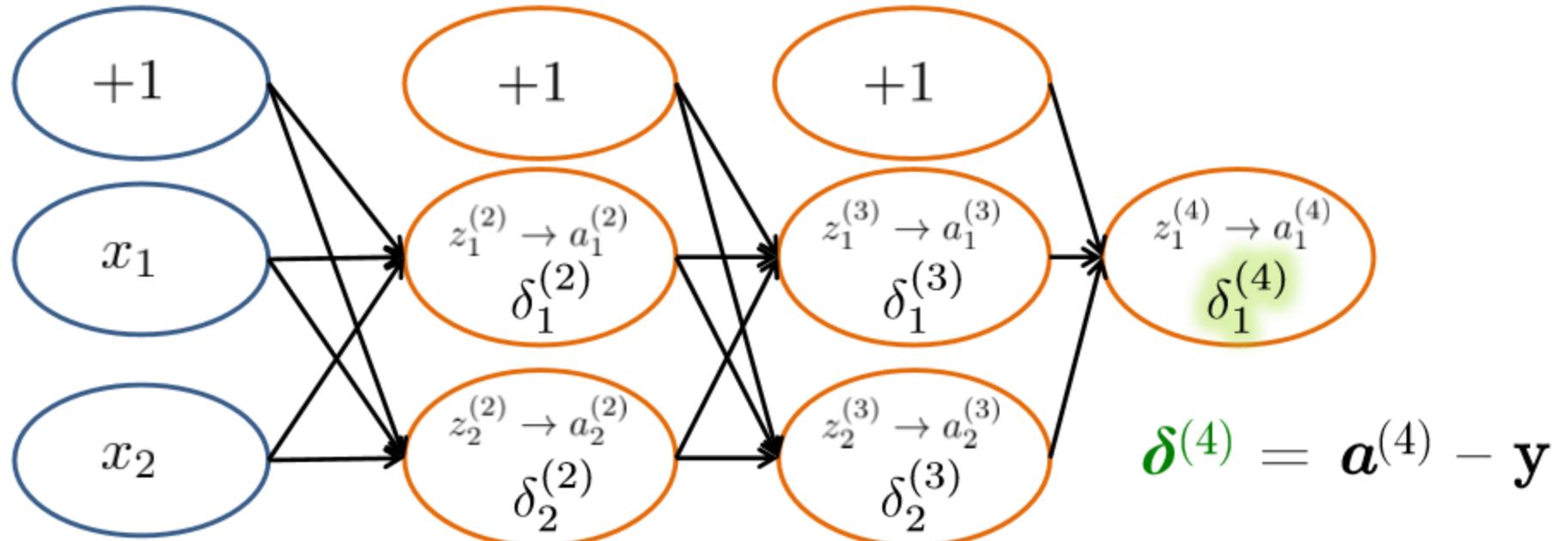
- $\mathbf{a}^{(1)} = \mathbf{x}$
- $\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{a}^{(1)}$
- $\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$ [add $a_0^{(2)}$]
- $\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$
- $\mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$ [add $a_0^{(3)}$]
- $\mathbf{z}^{(4)} = \Theta^{(3)}\mathbf{a}^{(3)}$
- $\mathbf{a}^{(4)} = h_{\Theta}(\mathbf{x}) = g(\mathbf{z}^{(4)})$



Backpropagation Intuition

- Each hidden node j is “responsible” for some fraction of the error $\delta_j^{(l)}$ in each of the output nodes to which it connects
- $\delta_j^{(l)}$ is divided according to the strength of the connection between hidden node and the output node
- Then, the “blame” is propagated back to provide the error values for the hidden layer

Backpropagation Intuition

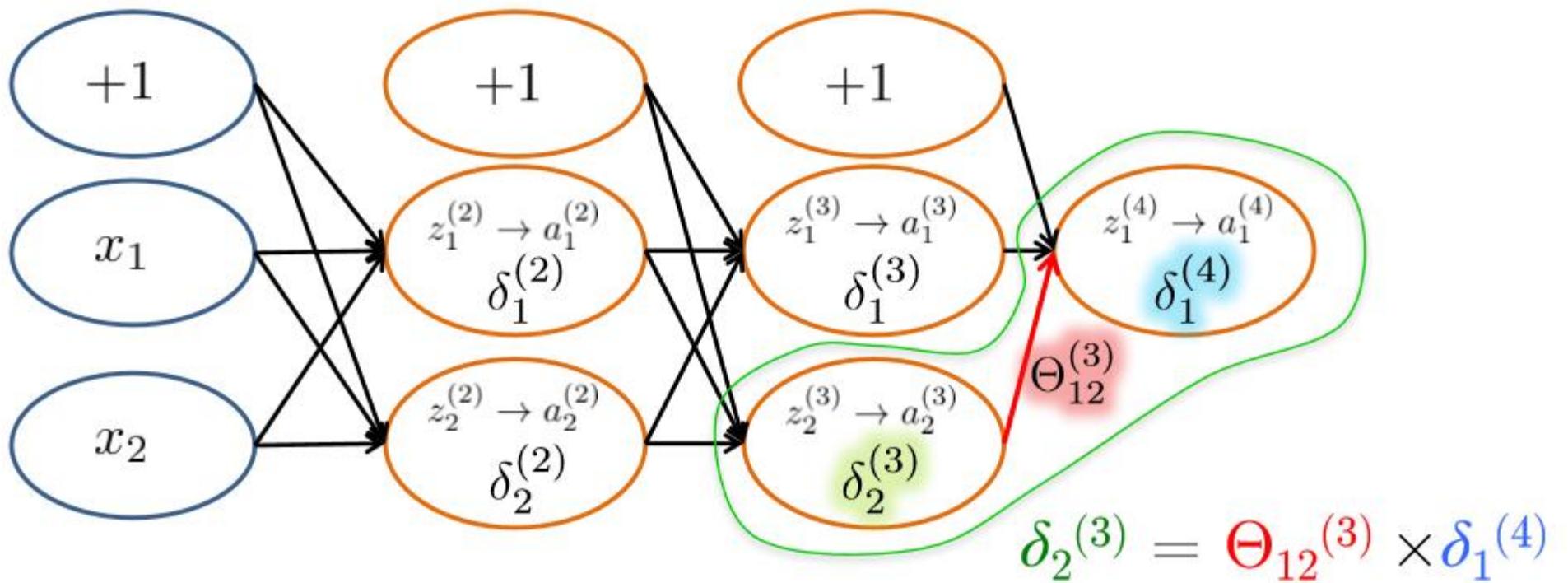


$\delta_j^{(l)}$ = “error” of node j in layer l

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Backpropagation Intuition

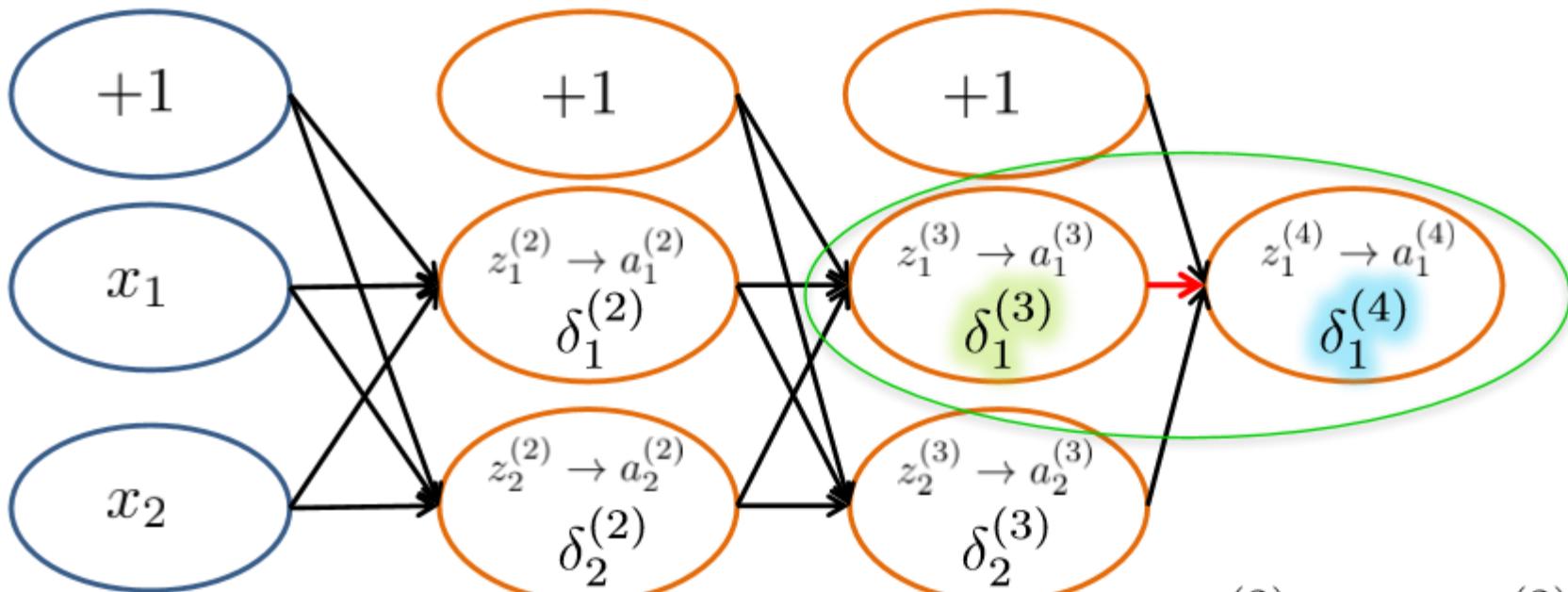


$\delta_j^{(l)}$ = “error” of node j in layer l

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Backpropagation Intuition



$$\delta_2^{(3)} = \Theta_{12}^{(3)} \times \delta_1^{(4)}$$

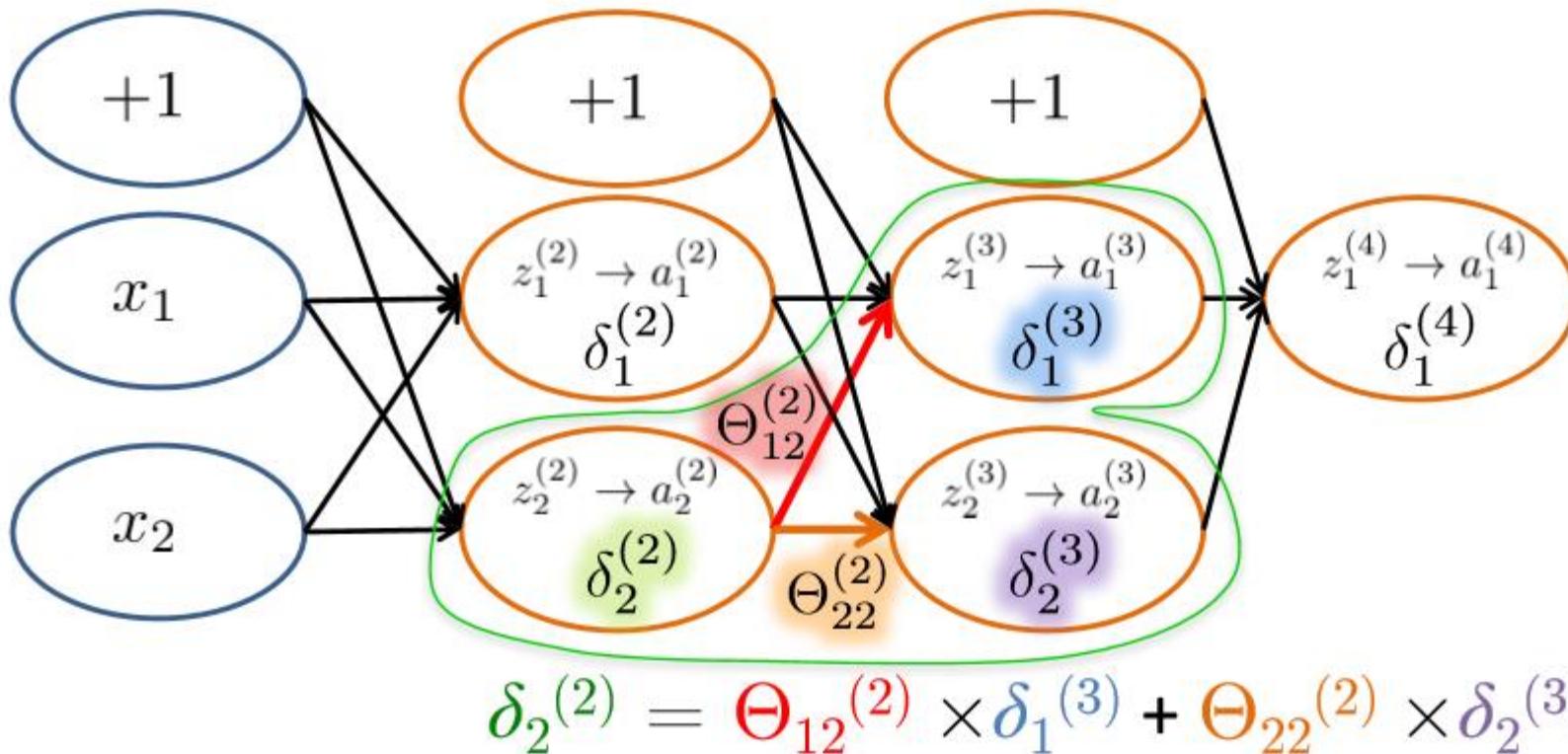
$$\delta_1^{(3)} = \Theta_{11}^{(3)} \times \delta_1^{(4)}$$

$\delta_j^{(l)}$ = “error” of node j in layer l

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Backpropagation Intuition



$\delta_j^{(l)}$ = “error” of node j in layer l

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

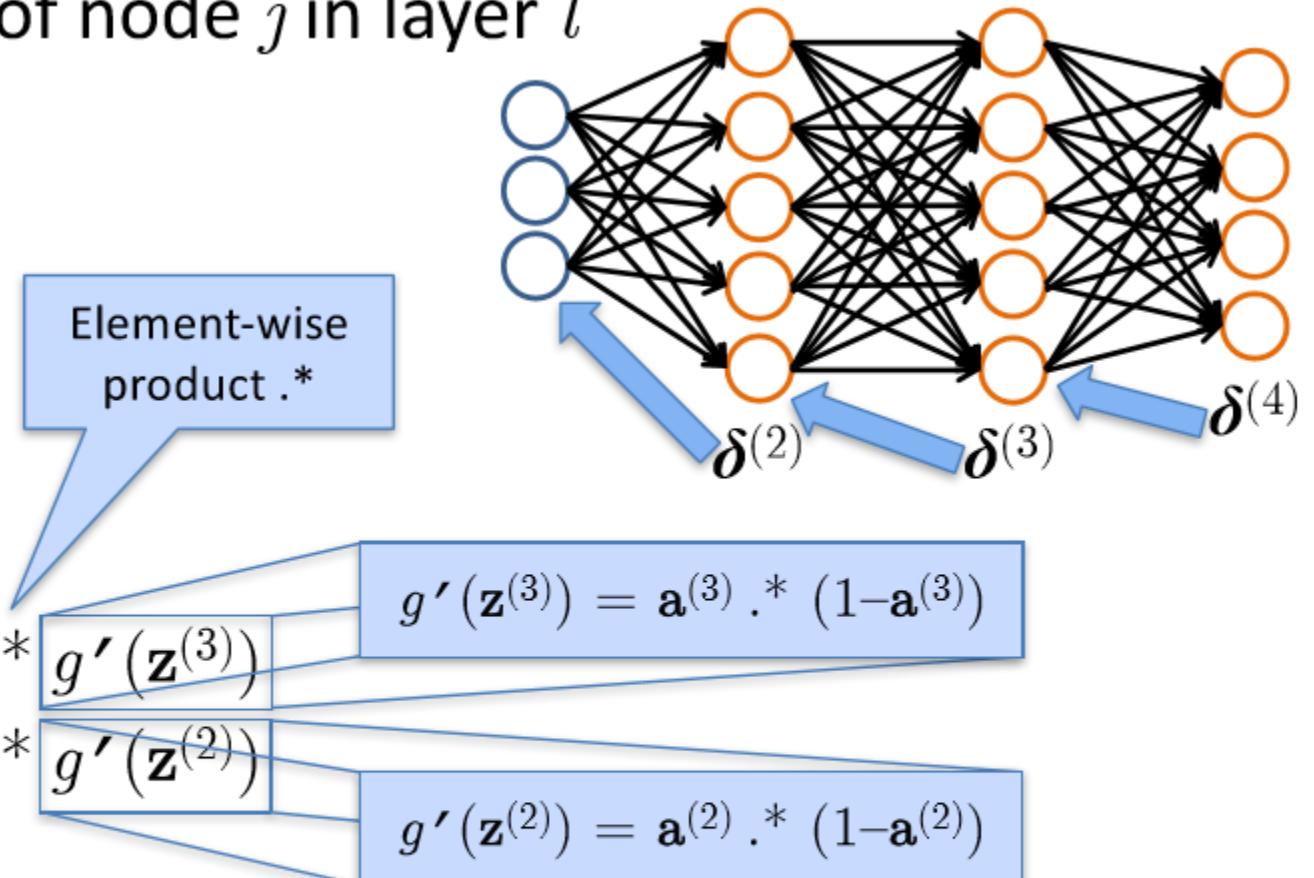
Backpropagation: Gradient Computation

Let $\delta_j^{(l)}$ = “error” of node j in layer l

(#layers $L = 4$)

Backpropagation

- $\delta^{(4)} = a^{(4)} - y$
- $\delta^{(3)} = (\Theta^{(3)})^\top \delta^{(4)} .*$
- $\delta^{(2)} = (\Theta^{(2)})^\top \delta^{(3)} .*$
- (No $\delta^{(1)}$)



$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)} \quad (\text{ignoring } \lambda; \text{ if } \lambda = 0)$$

Backpropagation

Given: training set $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$

Initialize all $\Theta^{(l)}$ randomly (NOT to 0!)

Loop // each iteration is called an epoch

Set $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$ (Used to accumulate gradient)

For each training instance (\mathbf{x}_i, y_i) :

Set $\mathbf{a}^{(1)} = \mathbf{x}_i$

Compute $\{\mathbf{a}^{(2)}, \dots, \mathbf{a}^{(L)}\}$ via forward propagation

Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_i$

Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \dots, \boldsymbol{\delta}^{(2)}\}$

Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

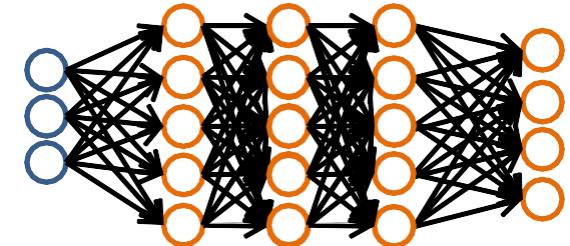
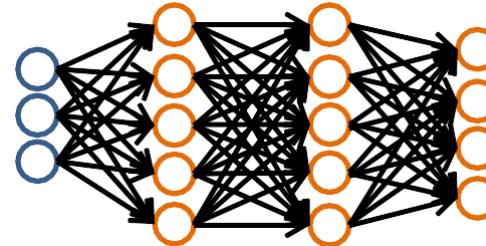
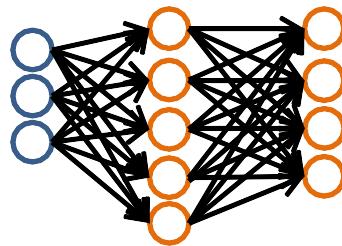
Compute avg regularized gradient $D_{ij}^{(l)} = \begin{cases} \frac{1}{n} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n} \Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$

Update weights via gradient step $\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha D_{ij}^{(l)}$

Until weights converge or max #epochs is reached

Training a Neural Network

Pick a network architecture (connectivity pattern between nodes)



- # input units = # of features in dataset
- # output units = # classes

Reasonable default: 1 hidden layer

- or if >1 hidden layer, have same # hidden units in every layer (usually the more the better)

Training a Neural Network

1. Randomly initialize weights
2. Implement forward propagation to get $h_{\Theta}(\mathbf{x}_i)$ for any instance \mathbf{x}_i
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivatives
$$\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$$
5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. the numerical gradient estimate.
 - Then, disable gradient checking code
6. Use gradient descent with backprop to fit the network

More on Backpropagation

- Gradient descent over entire *network* weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
 - In practice, often works well (can run multiple times)
- Often include weight *momentum* α
$$\Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n - 1)$$
- Minimizes error over *training* examples
 - Will it generalize well to subsequent examples?
- Training can take thousands of iterations → slow!
- Using network after training is very fast

Expressive Capabilities of ANNs

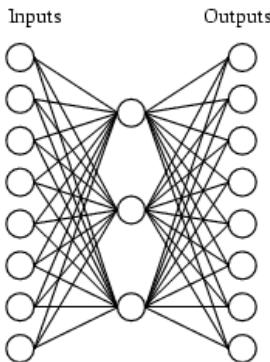
Boolean functions:

- Every boolean function can be represented by network with single hidden layer
- but might require exponential (in number of inputs) hidden units

Continuous functions:

- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].

Learning Hidden Layer Representations



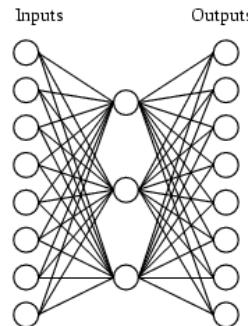
A target function:

Input	Output
10000000	\rightarrow 10000000
01000000	\rightarrow 01000000
00100000	\rightarrow 00100000
00010000	\rightarrow 00010000
00001000	\rightarrow 00001000
00000100	\rightarrow 00000100
00000010	\rightarrow 00000010
00000001	\rightarrow 00000001

Can this be learned??

Learning Hidden Layer Representations

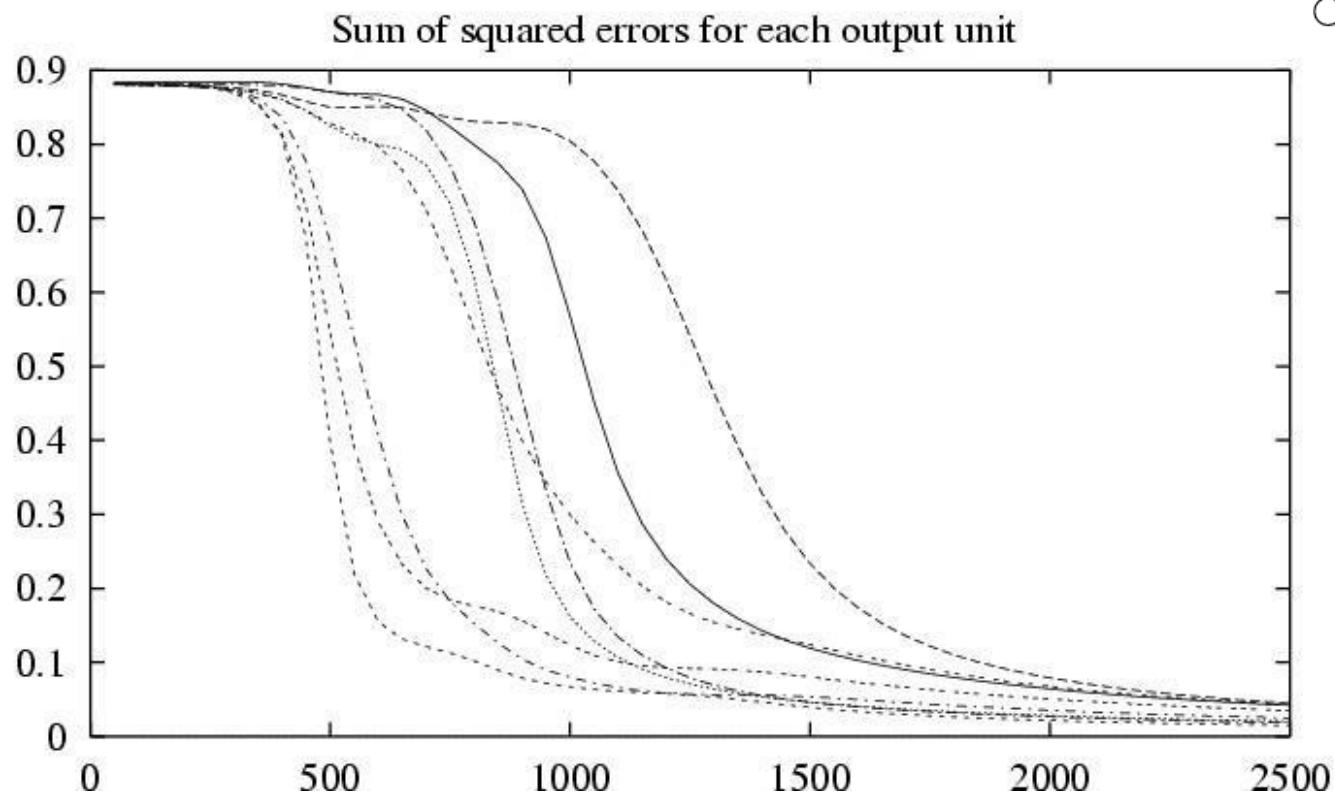
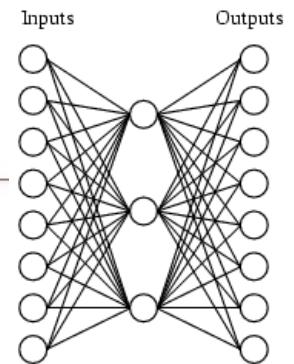
A network:



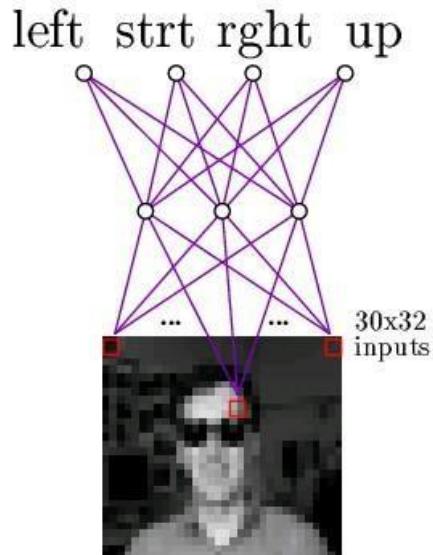
Learned hidden layer representation:

Input	Hidden Values	Output
10000000	→ .89 .04 .08	→ 10000000
01000000	→ .01 .11 .88	→ 01000000
00100000	→ .01 .97 .27	→ 00100000
00010000	→ .99 .97 .71	→ 00010000
00001000	→ .03 .05 .02	→ 00001000
00000100	→ .22 .99 .99	→ 00000100
00000010	→ .80 .01 .98	→ 00000010
00000001	→ .60 .94 .01	→ 00000001

Training



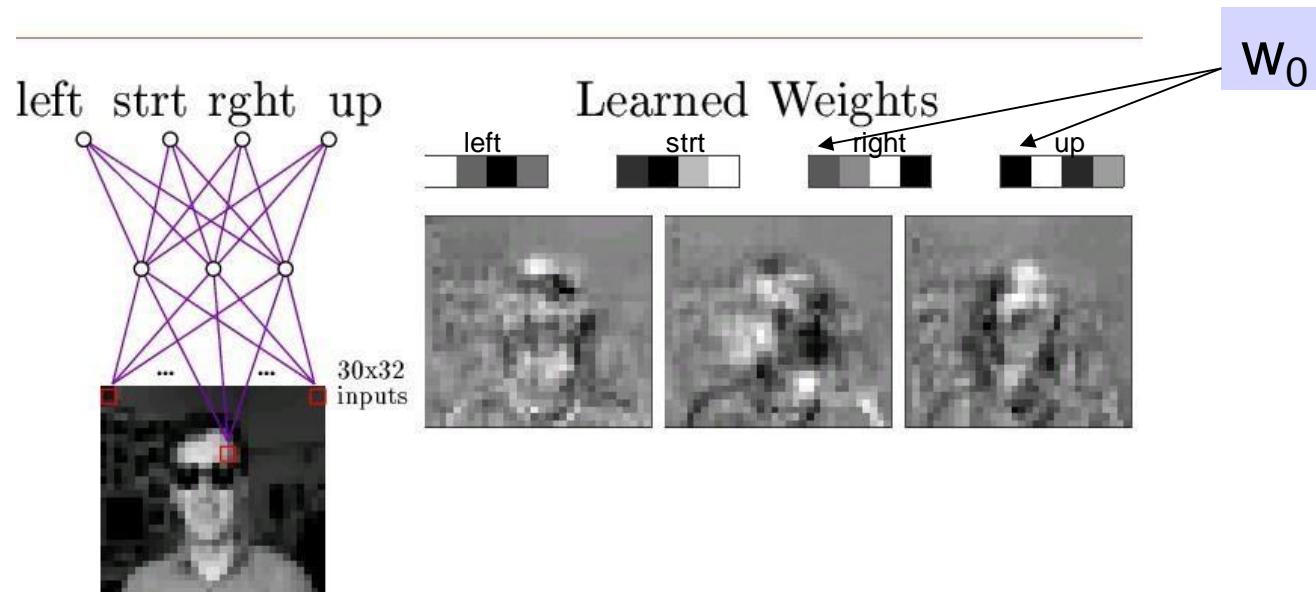
Neural Nets for Face Recognition



Typical input images

90% accurate learning head pose, and recognizing 1-of-20 faces

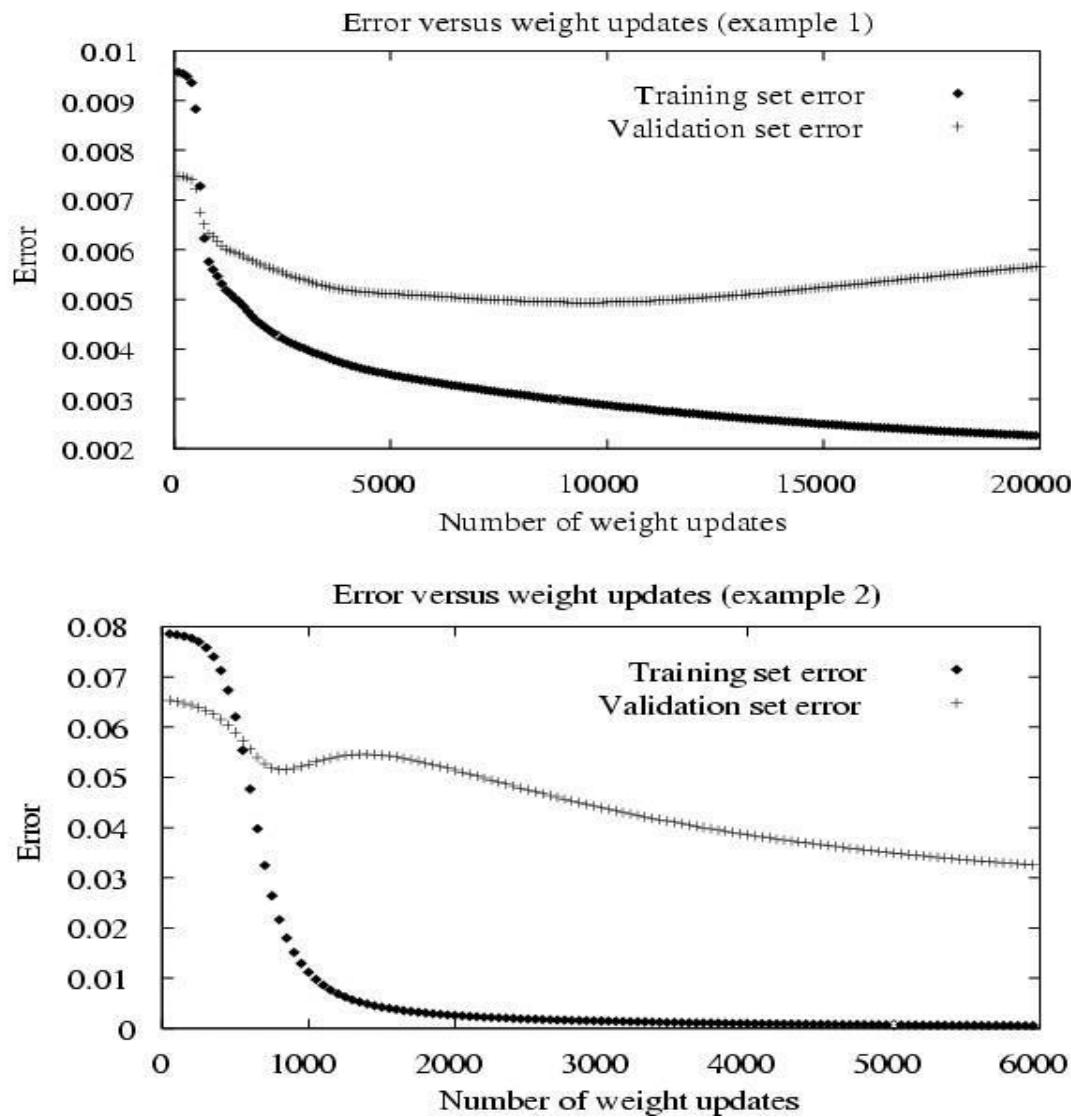
Learned Hidden Unit Weights



Typical input images

<http://www.cs.cmu.edu/~tom/faces.html>

Overfitting in ANNs



Early Stopping

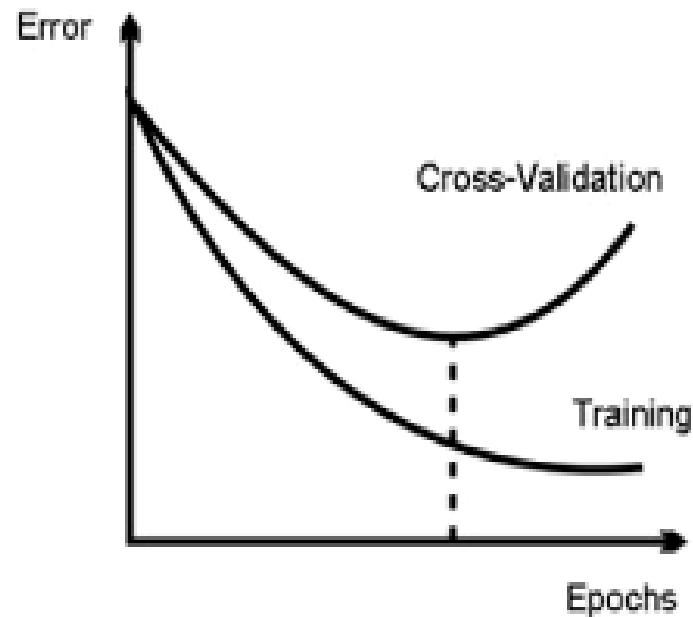


Figure 2: Profiles for training and cross-validation errors.

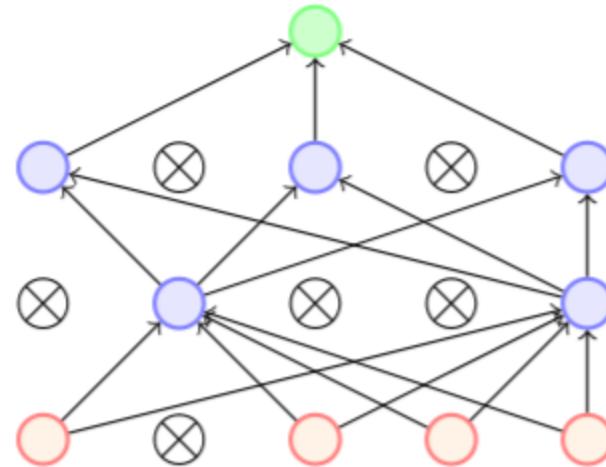
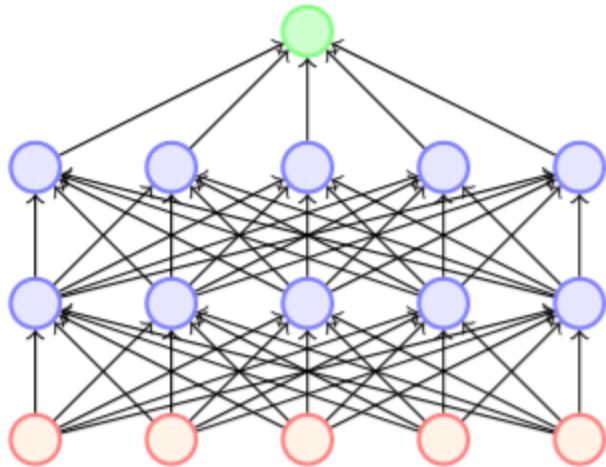
If we let a complex model train long enough on a given data set it can eventually learn the data exactly.

Given data that isn't represented in the training set, the model will perform poorly when analyzing the data (overfitting).

How is the sweet spot for training located?

When the error on the training set begins to deviate from the error on the validation set, a threshold can be set to determine the early stopping condition and the ideal number of epochs to train.

Dropout



- Dropout refers to dropping out units
- Temporarily remove a node and all its incoming/outgoing connections resulting in a thinned network
- Each node is retained with a fixed probability (typically $p = 0.5$) for hidden nodes and $p = 0.8$ for visible nodes

Dropouts

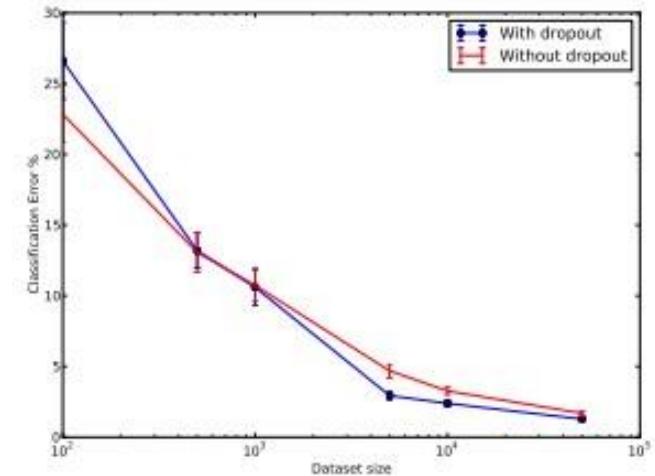
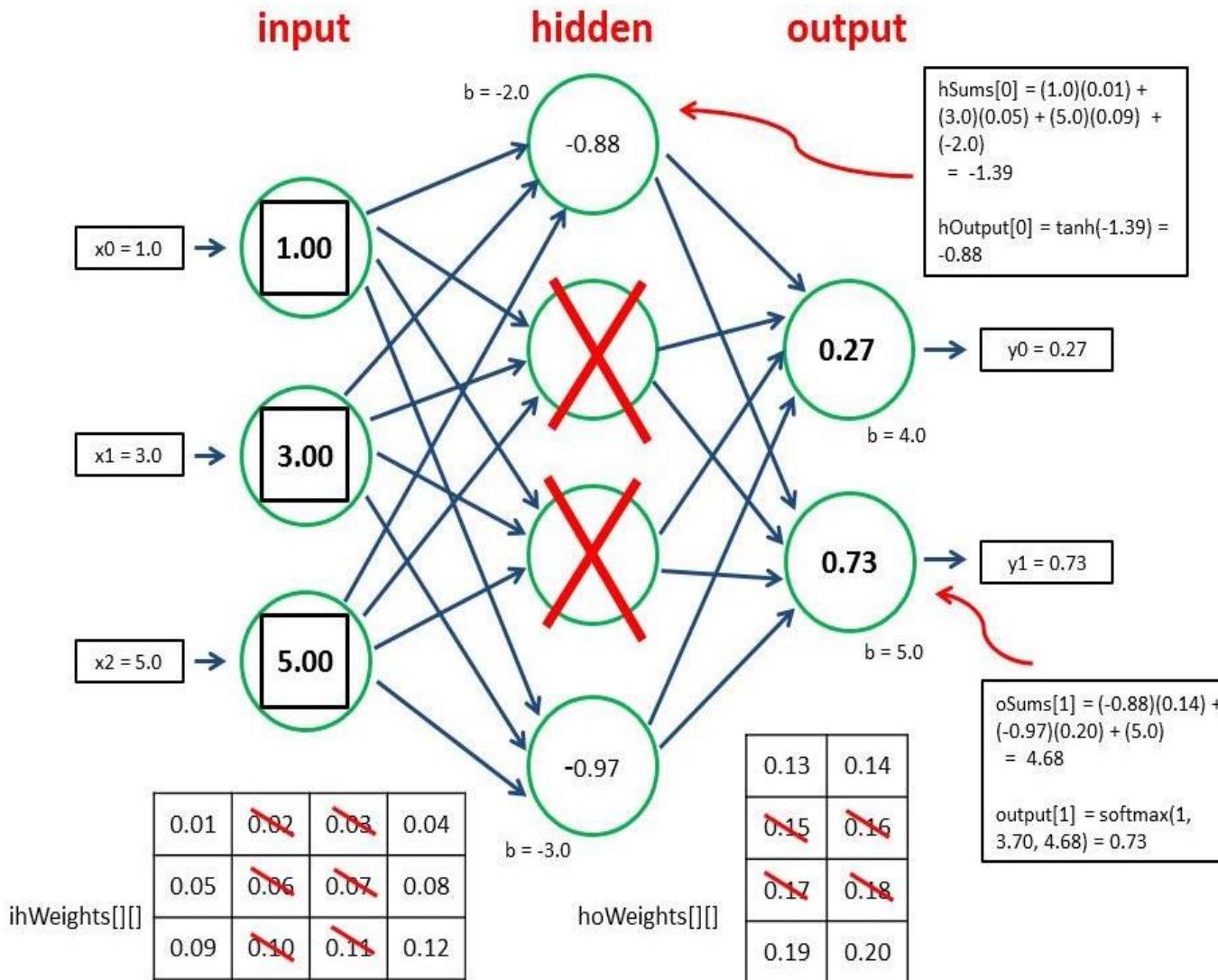


Figure 10: Effect of varying data set size.

Convolutional Neural Network

Convolution



$$s_t = \sum_{a=0}^{\infty} x_{t-a} w_{-a} = (x * w)_t$$

input filter
 convolution

A mathematical equation illustrating convolution. It shows the output s_t as a weighted sum of past input measurements x_{t-a} and a filter w_{-a} . The result is labeled as a convolution of the input and filter at time t .

- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals
- Now suppose our sensor is noisy
- To obtain a less noisy estimate we would like to average several measurements
- More recent measurements are more important so we would like to take a weighted average

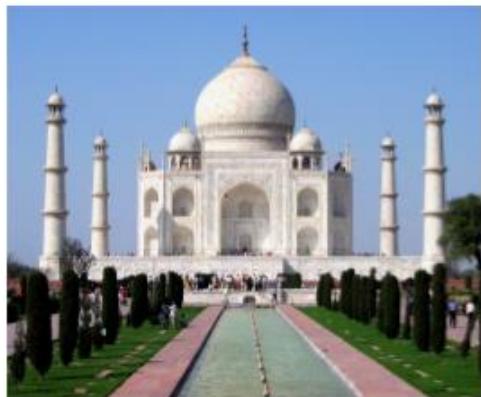
Convolution



- We can think of images as 2D inputs
- We would now like to use a 2D filter ($m \times n$)
- First let us see what the 2D formula looks like
- This formula looks at all the preceding neighbours $(i - a, j - b)$

$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i-a, j-b} K_{a,b}$$

Convolution



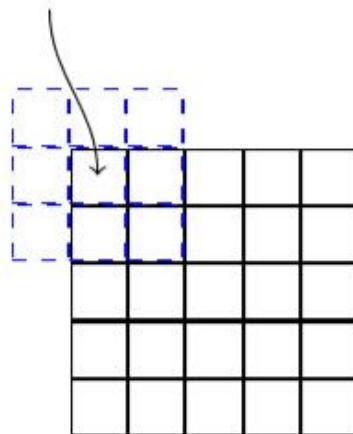
$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a, j+b} K_{a,b}$$

- We can think of images as 2D inputs
- We would now like to use a 2D filter ($m \times n$)
- First let us see what the 2D formula looks like
- This formula looks at all the preceding neighbours ($i - a, j - b$)
- In practice, we use the following formula which looks at the succeeding neighbours

Convolution

$$S_{ij} = (I * K)_{ij} = \sum_{a=\left\lfloor -\frac{m}{2} \right\rfloor}^{\left\lfloor \frac{m}{2} \right\rfloor} \sum_{b=\left\lfloor -\frac{n}{2} \right\rfloor}^{\left\lfloor \frac{n}{2} \right\rfloor} I_{i-a, j-b} K_{\frac{m}{2}+a, \frac{n}{2}+b}$$

pixel of interest



- For the rest of the discussion we will use the following formula for convolution
- In other words we will assume that the kernel is centered on the pixel of interest
- So we will be looking at both preceding and succeeding neighbors

Let us apply this idea to a toy example and see the results

Input

a	b	c	d
e	f	g	h
i	j	k	A

Kernel

w	x
y	z

Output

aw+bx+ey+fz		

Let us apply this idea to a toy example and see the results

Input

a	b	c	d
e	f	g	h
i	j	k	A

Kernel

w	x
y	z

Output

aw+bx+ey+fz	bw+cx+fy+gz	

Let us apply this idea to a toy example and see the results

Input

a	b	c	d
e	f	g	h
i	j	k	A

Kernel

w	x
y	z

Output

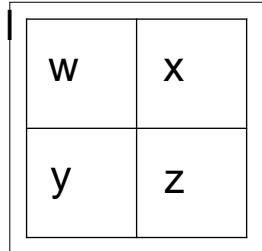
aw+bx+ey+fz	bw+cx+fy+gz	cw+dx+gy+hz

Let us apply this idea to a toy example and see the results

Input

a	b	c	d
e	f	g	h
i	j	k	A

Kerne



Output

$aw+bx+ey+fz$	$bw+cx+fy+gz$	$cw+dx+gy+hz$
$ew+fx+iy+jz$		

Let us apply this idea to a toy example and see the results

Input

a	b	c	d
e	f	g	h
i	j	k	A

Kernel

w	x
y	z

Output

$aw+bx+ey+fz$	$bw+cx+fy+gz$	$cw+dx+gy+hz$
$ew+fx+iy+jz$	$fw+gx+jy+kz$	

Let us apply this idea to a toy example and see the results

Input

a	b	c	d
e	f	g	h
i	j	k	A

Kernel

w	x
y	z

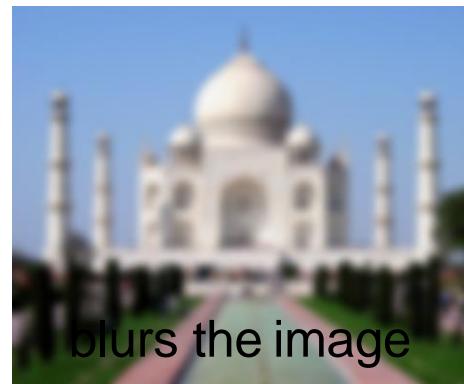
Output

$aw+bx+ey+fz$	$bw+cx+fy+gz$	$cw+dx+gy+hz$
$ew+fx+iy+jz$	$fw+gx+jy+kz$	$gw+hx+ky+Az$

Example of kernel: Blur



$$* \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} =$$



Example of kernel: Edge detection

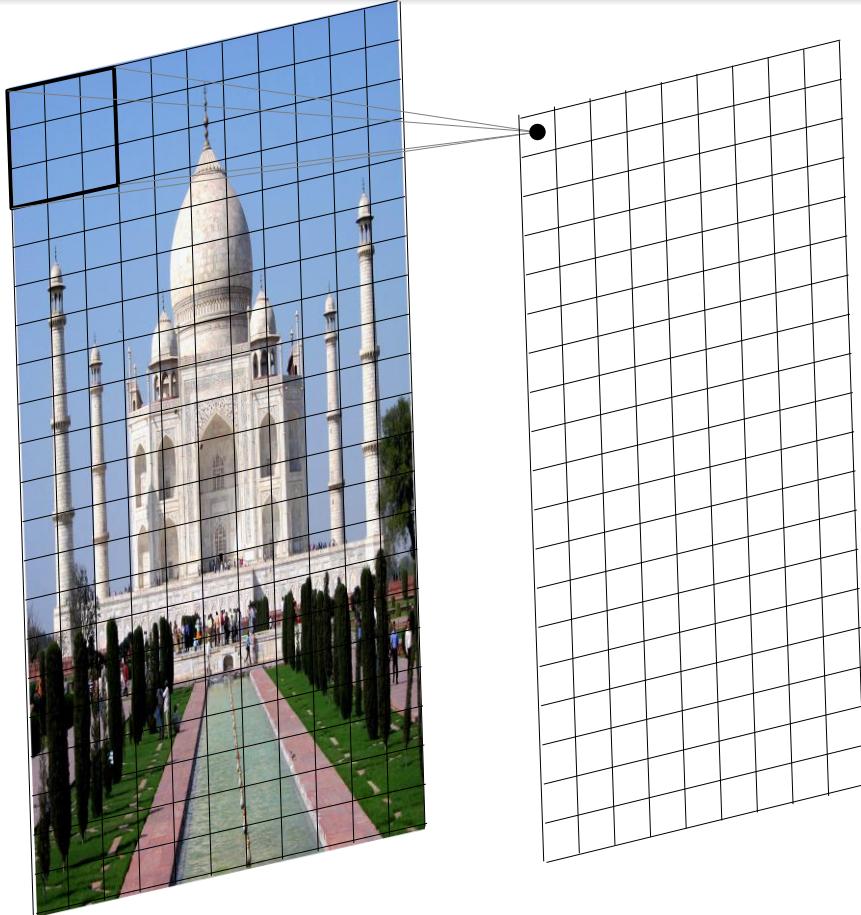


$$\begin{array}{ccc} * & \begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix} & = \end{array}$$



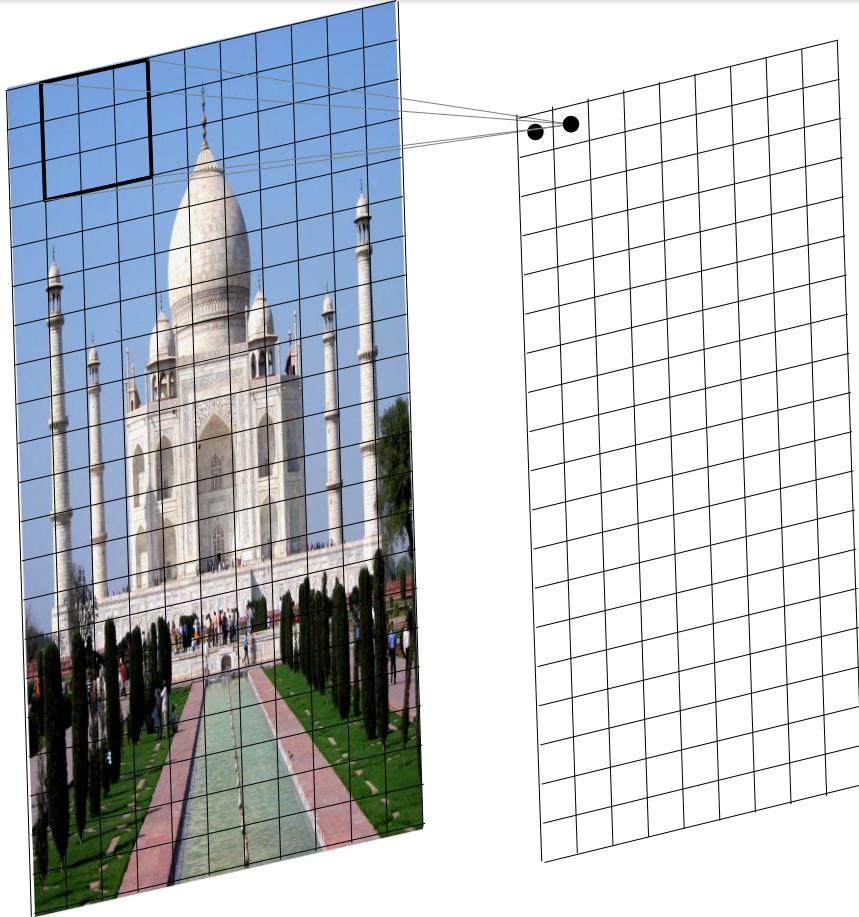
edges

Convolution



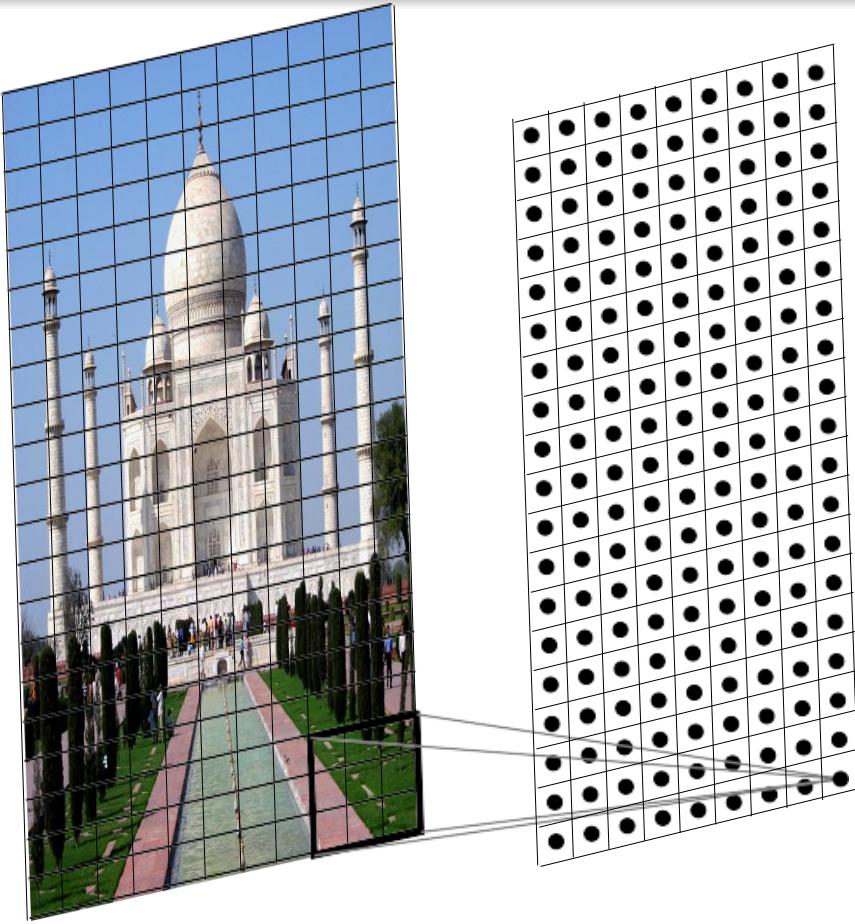
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

Convolution

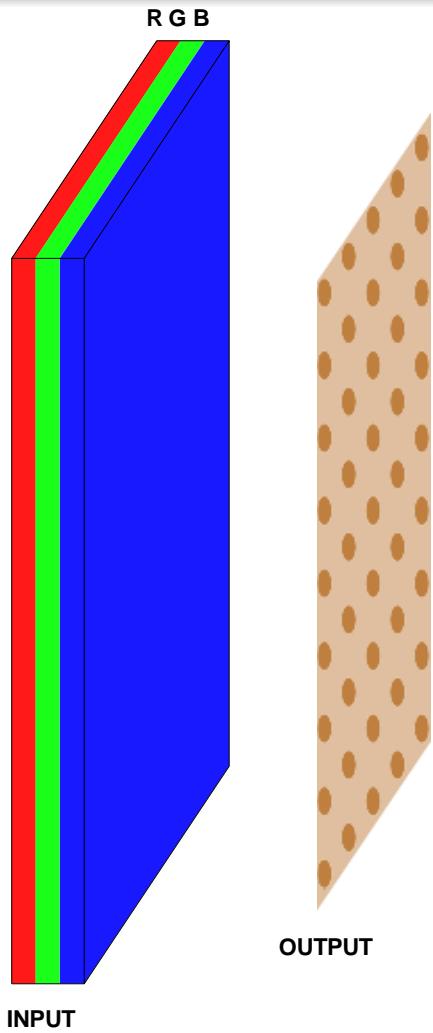


- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

2D convolutions applied to images

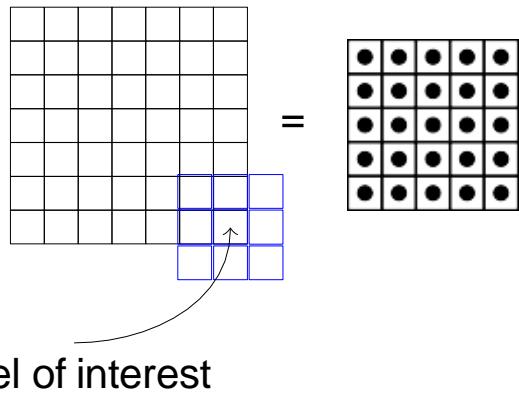


- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output
- The resulting output is called a feature map.
- We can use multiple filters to get multiple feature maps.

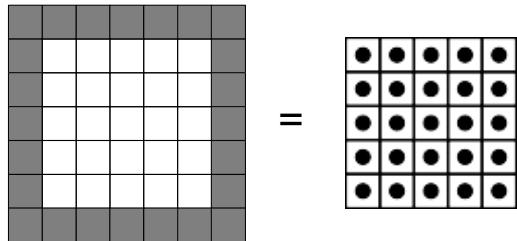


- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation
- We will assume that the filter always extends to the depth of the image

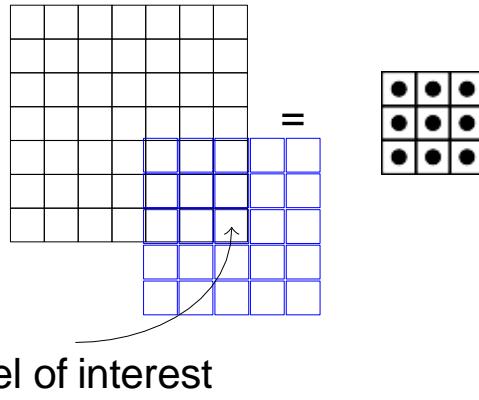
- In effect, we are doing a 2D convolution operation on a 3D input (because the filter moves along the height and the width but not along the depth)
- As a result the output will be 2D (only width and height, no depth)
- Once again we can apply multiple filters to get multiple feature maps



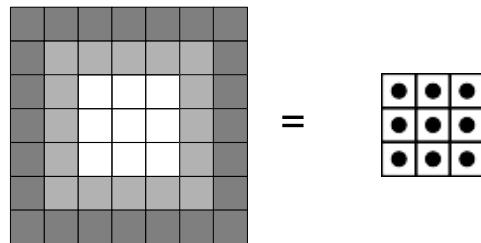
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary



- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels



- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller output now



In general, $W_2 = W_1 - F + 1$

$$H_2 = H_1 - F + 1$$

We will refine this formula further

- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller output now

$$\begin{array}{ccccccccc}
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & & & & & & & & 0 \\
 0 & & & & & & & & 0 \\
 0 & & & & & & & & 0 \\
 0 & & & & & & & & 0 \\
 0 & & & & & & & & 0 \\
 0 & & & & & & & & 0 \\
0 & & & & & & & & 0 \\
0 & & & & & & & & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array} =
 \begin{array}{ccccccccc}
\bullet & \bullet \\
\bullet & \bullet \\
\bullet & \bullet \\
\bullet & \bullet \\
\bullet & \bullet \\
\bullet & \bullet \\
\bullet & \bullet \\
\bullet & \bullet
\end{array}$$

- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right

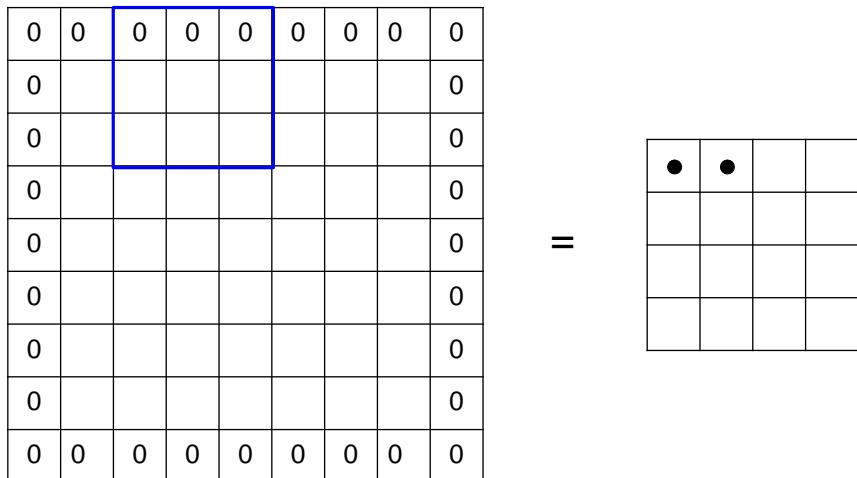
We now have, $W_2 = W_1 - F + 2P + 1$ $H_2 = H_1 - F + 2P + 1$

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

=

•			

- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

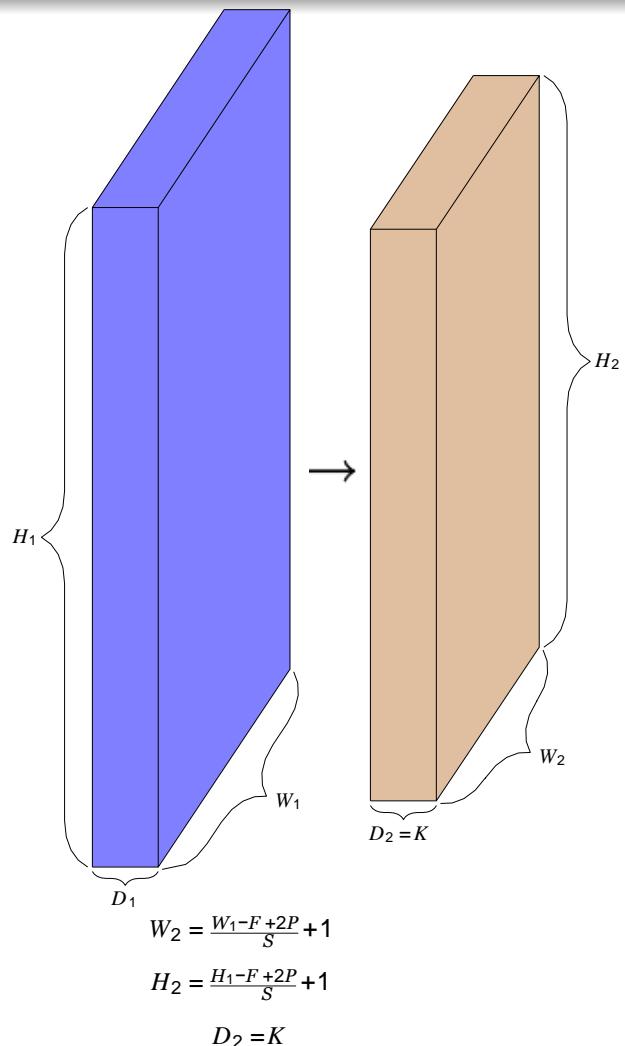


- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

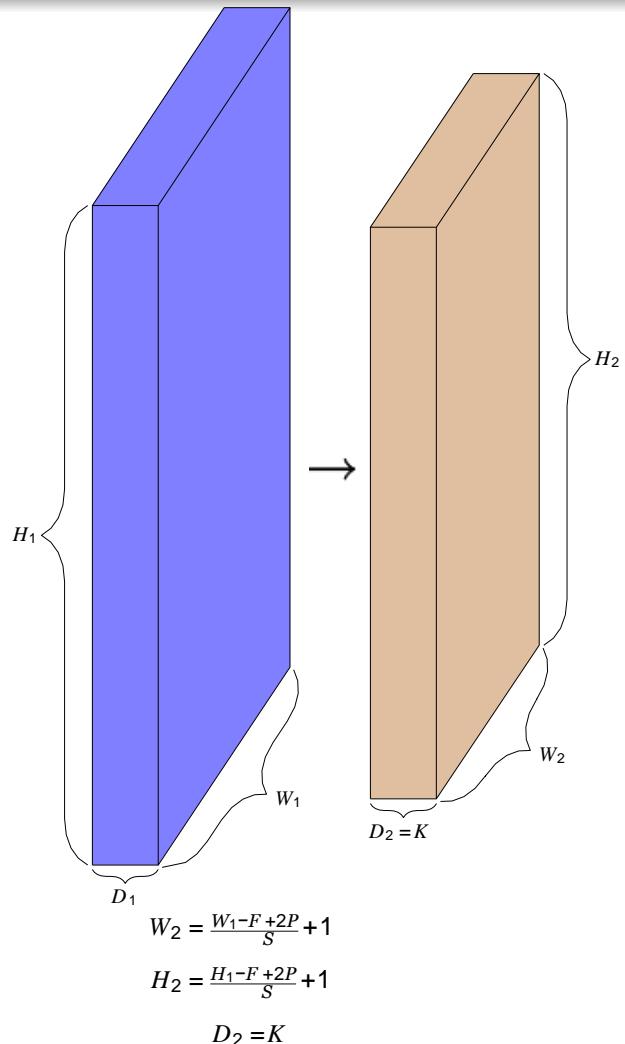
So what should our final formula look like,

$$W = \frac{W_1 - F + 2P}{S} + 1$$

$$H = \frac{H_1 - F + 2P}{S} + 1$$

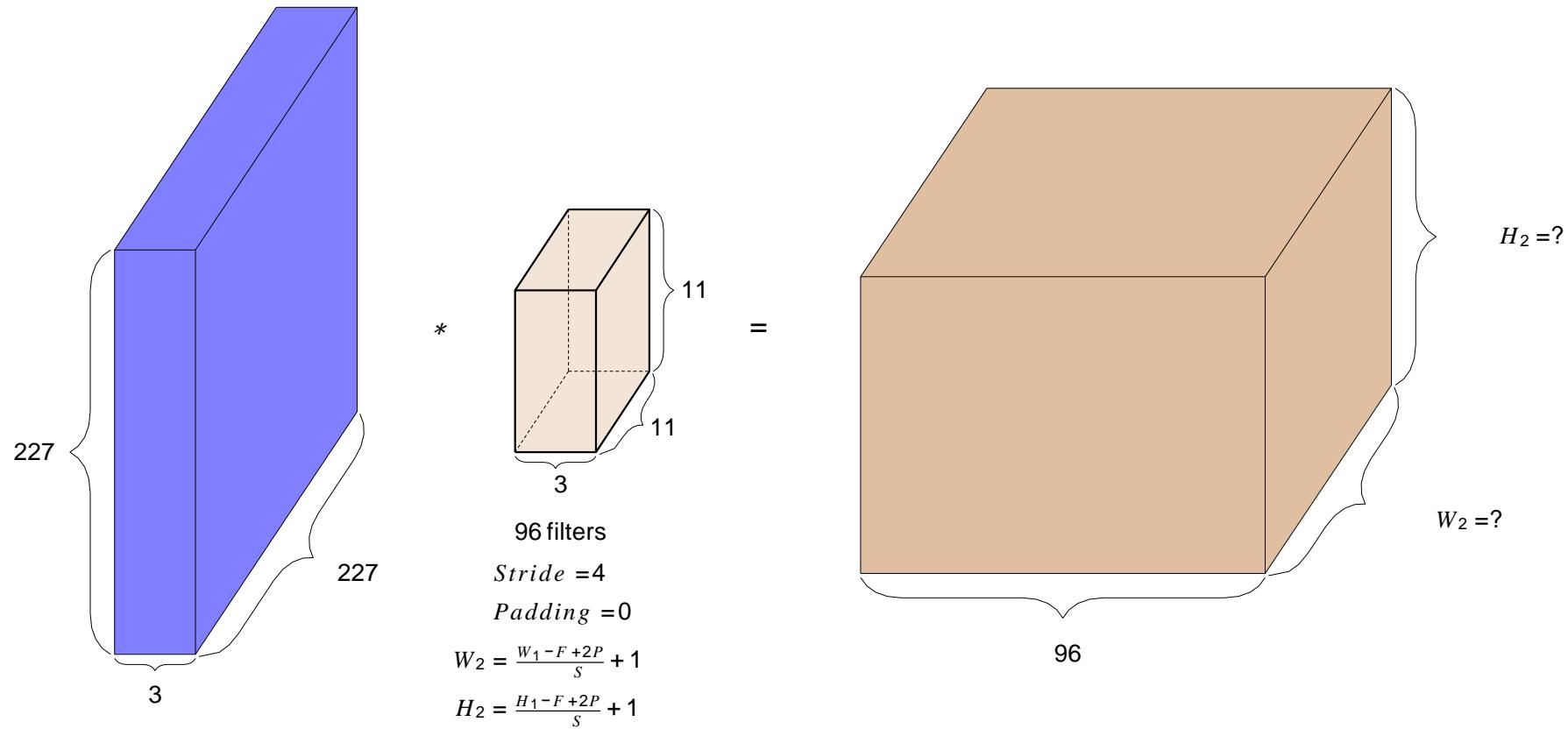


- Finally, coming to the depth of the output.
- Each filter gives us one 2D output.
- K filters will give us K such 2D outputs

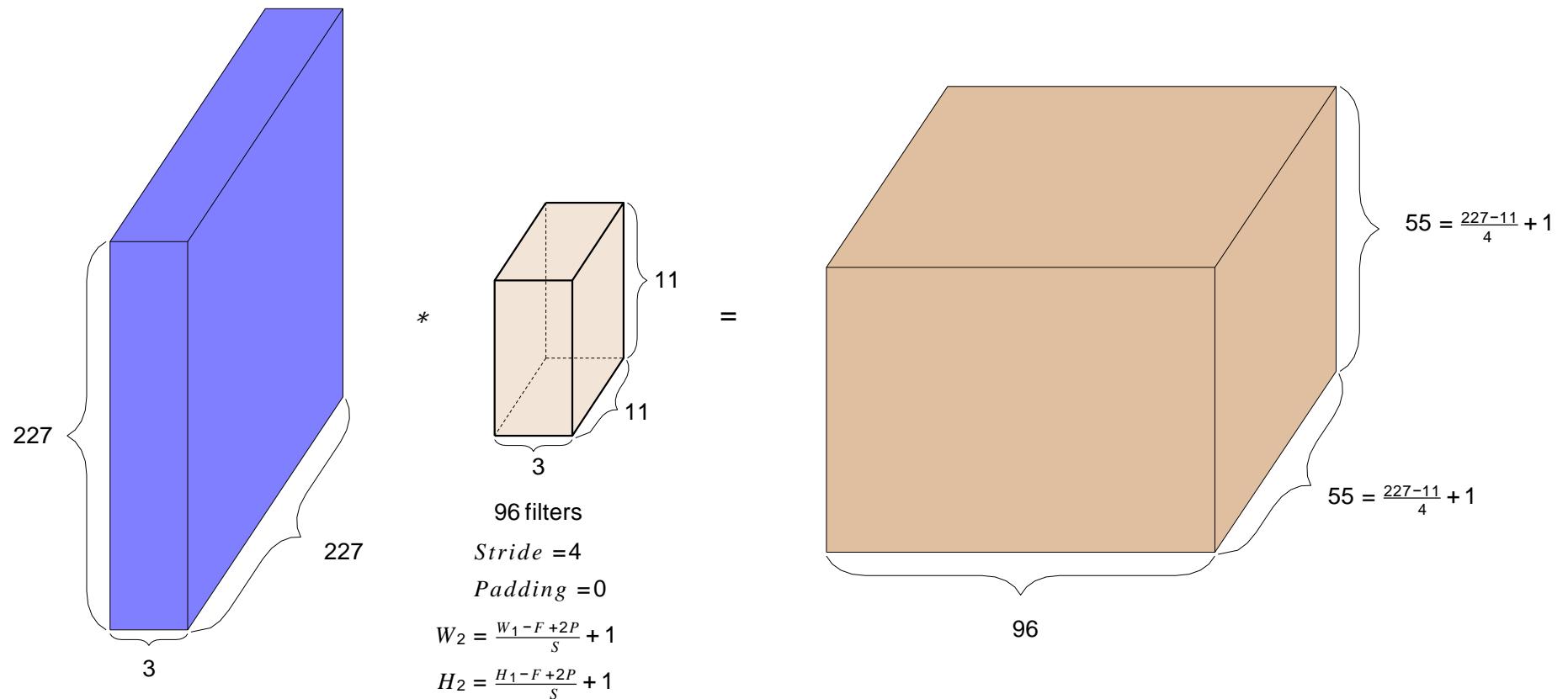


- Finally, coming to the depth of the output.
- Each filter gives us one 2D output.
- K filters will give us K such 2D outputs
- We can think of the resulting output as $K \times W_2 \times H_2$ volume
- Thus $D_2 = K$

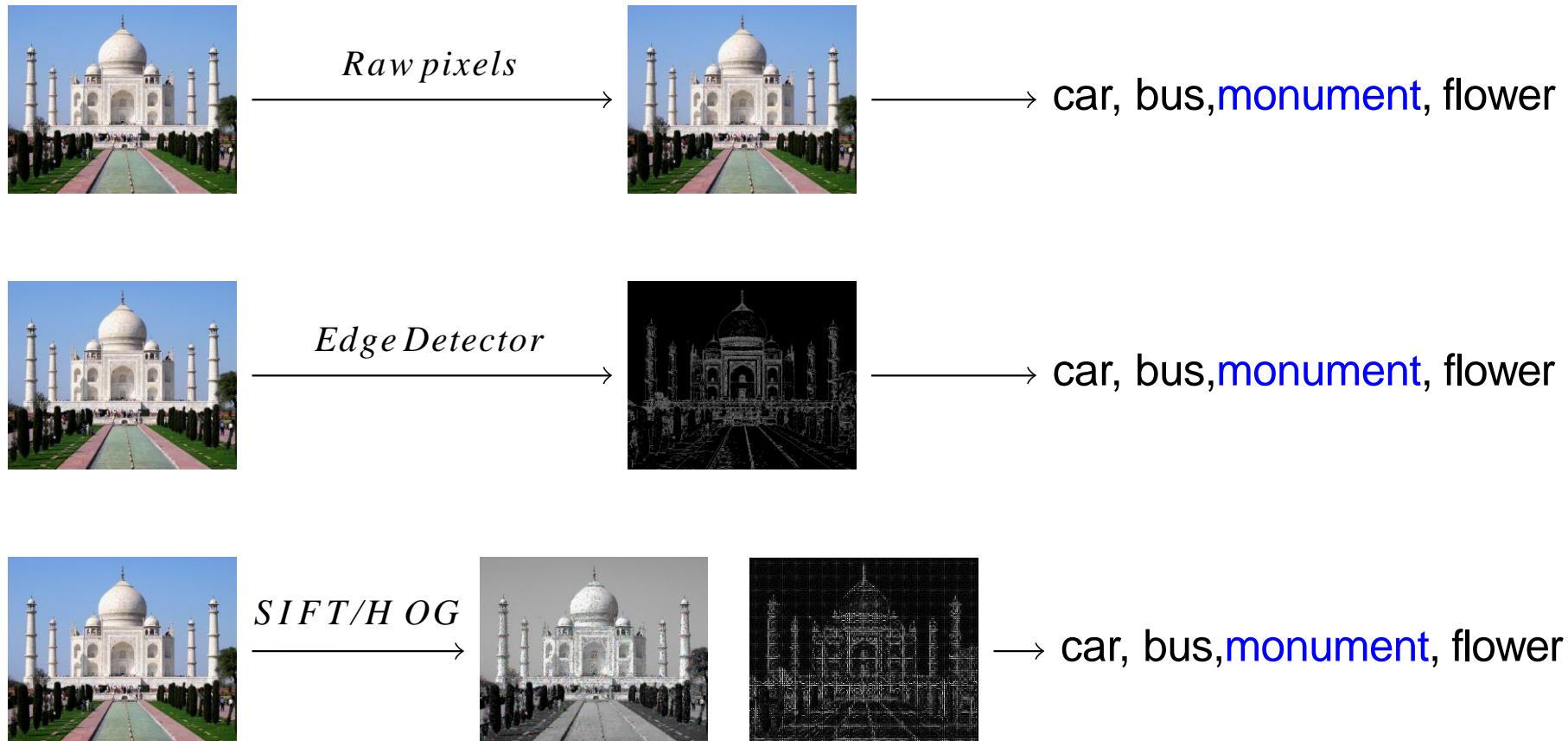
Example

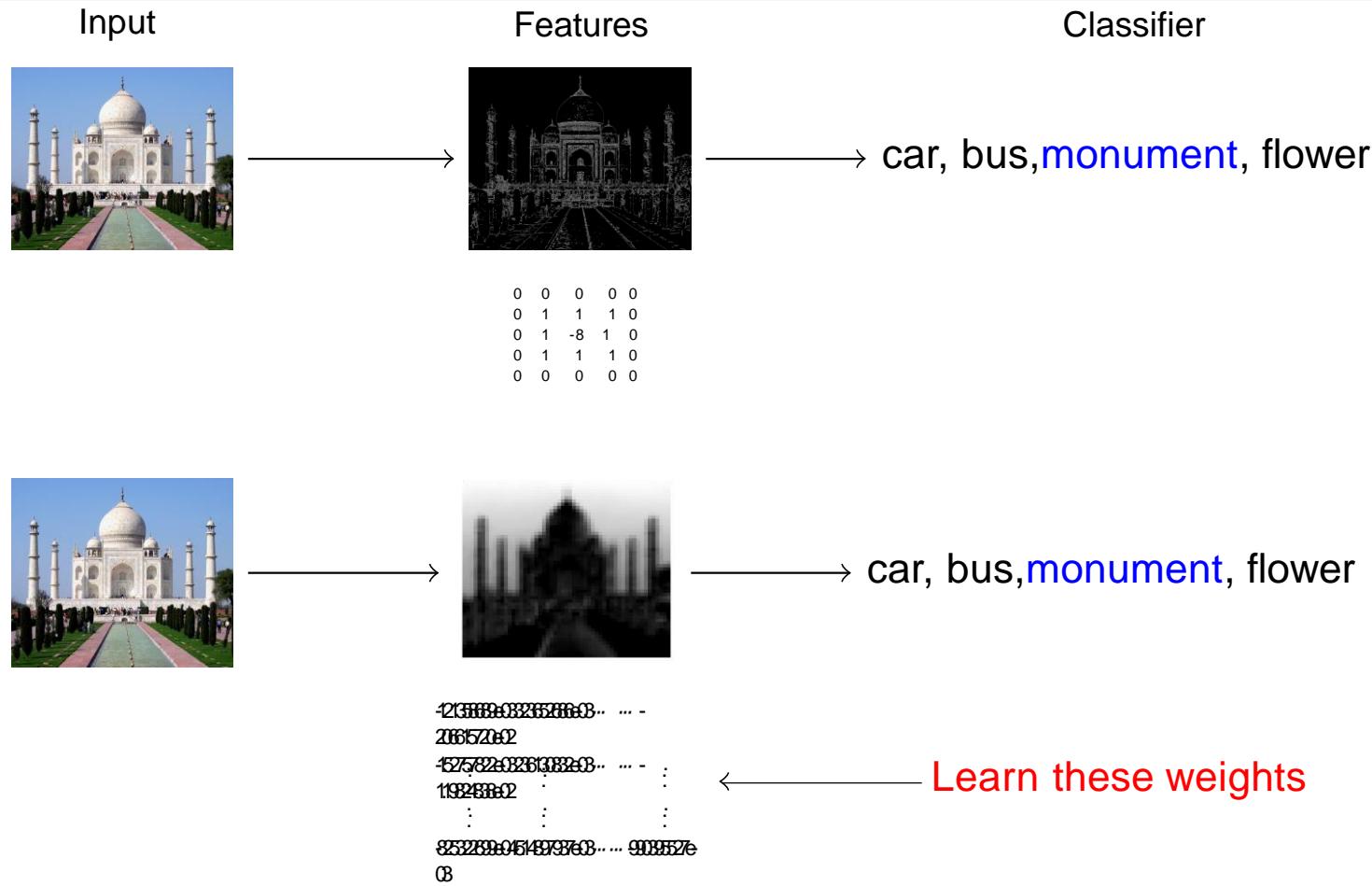


Example

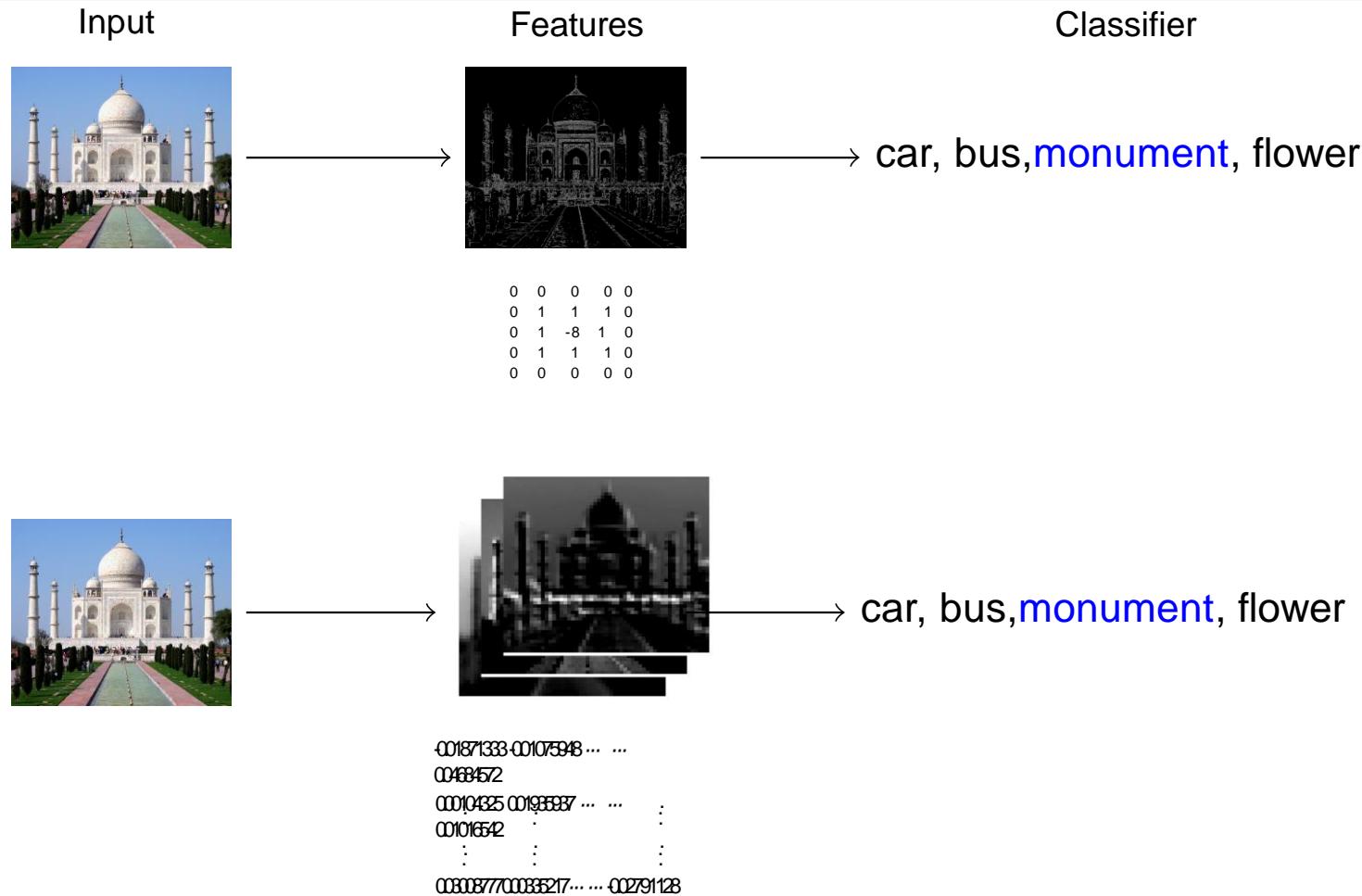


Features



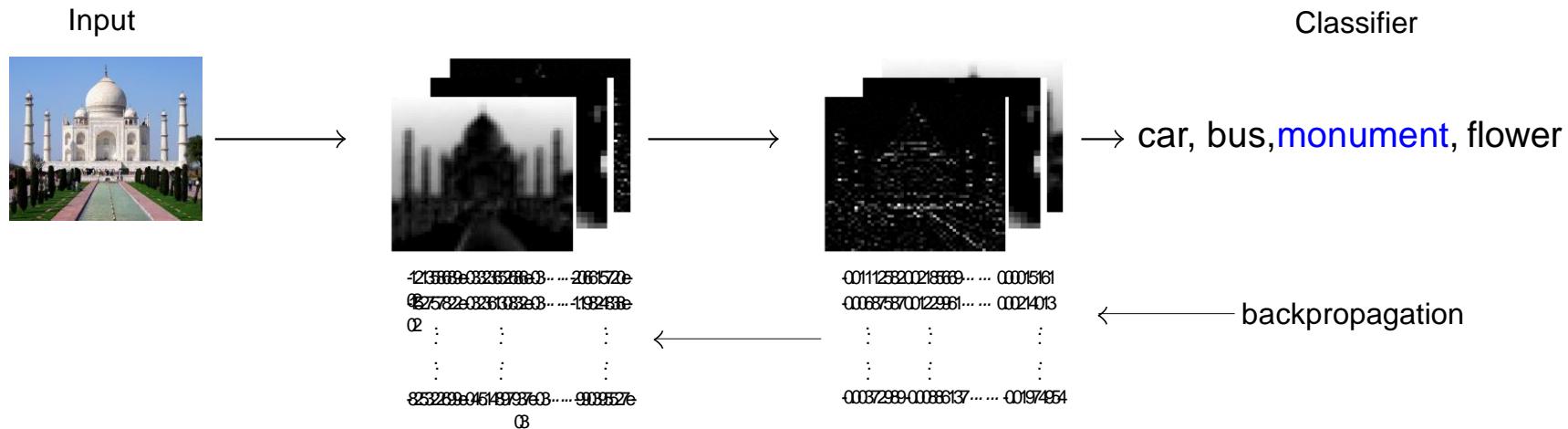


■ Instead of using handcrafted kernels such as edge detectors **can we learn meaningful kernels/filters in addition to learning the weights of the classifier?**

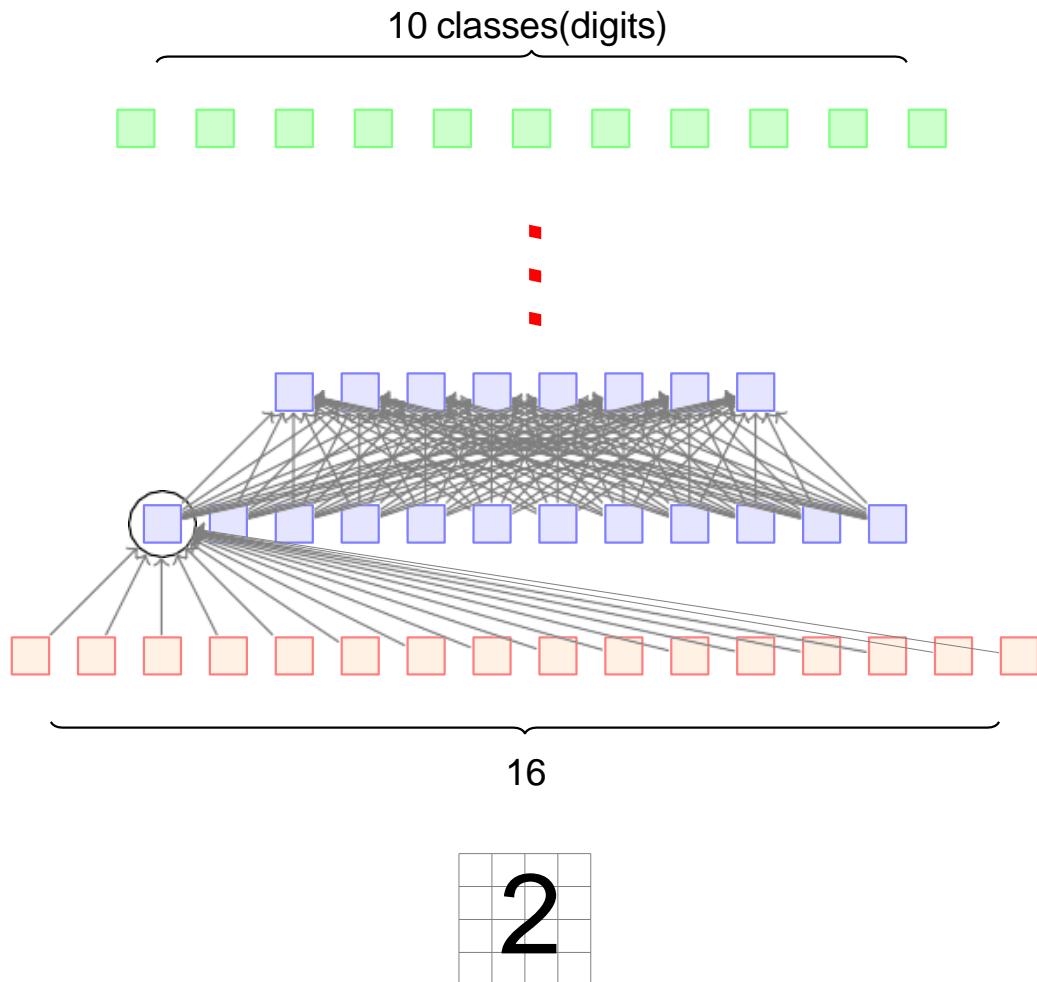


- **Even better:** Instead of using handcrafted kernels (such as edge detectors) **can we learn multiple meaningful kernels/filters in addition to learning the weights of the classifier?**

Convolutional Neural Network

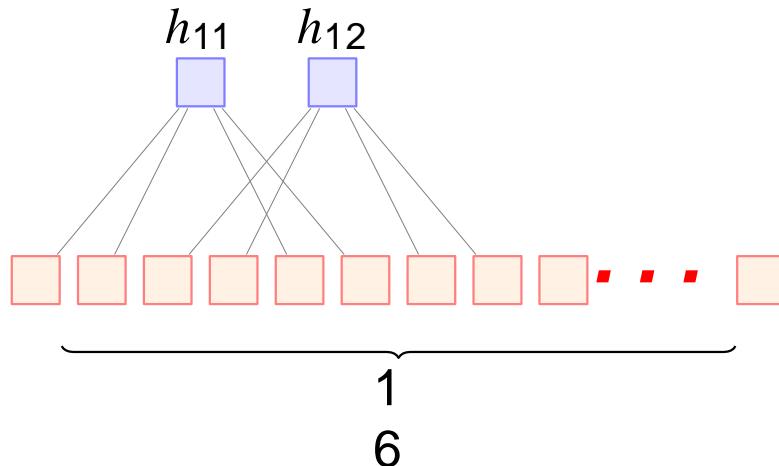


- Can we learn multiple **layers** of meaningful **kernels/filters** in addition to learning the weights of the classifier?
- Yes, we can !
- Simply by treating these kernels as parameters and learning them in addition to the weights of the classifier (using back propagation)
- Such a network is called a Convolutional Neural Network.



- This is what a regular feed-forward neural network will look like
- There are many dense connections here
- For example all the 16 input neurons are contributing to the computation of h_{11}
- Contrast this to what happens in the case of convolution

Sparse Connectivity

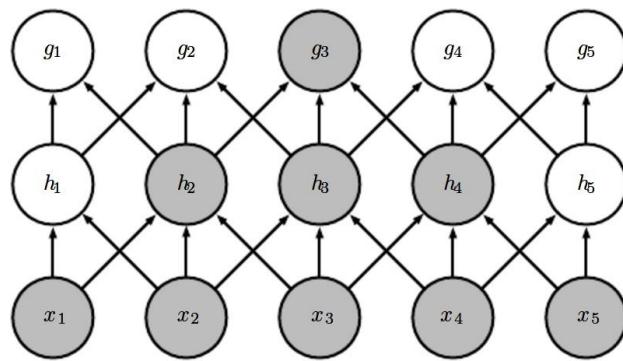


The diagram shows a 4x4 input matrix with a handwritten digit '2' and a 3x3 weight matrix. An asterisk (*) indicates multiplication, and an equals sign (=) indicates the result. The result is a single blue square, representing the output of the neuron h_{11} .

- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}
- The connections are much sparser
- We are taking advantage of the structure of the image (interactions)
- This **sparse connectivity** reduces the number of parameters in the model

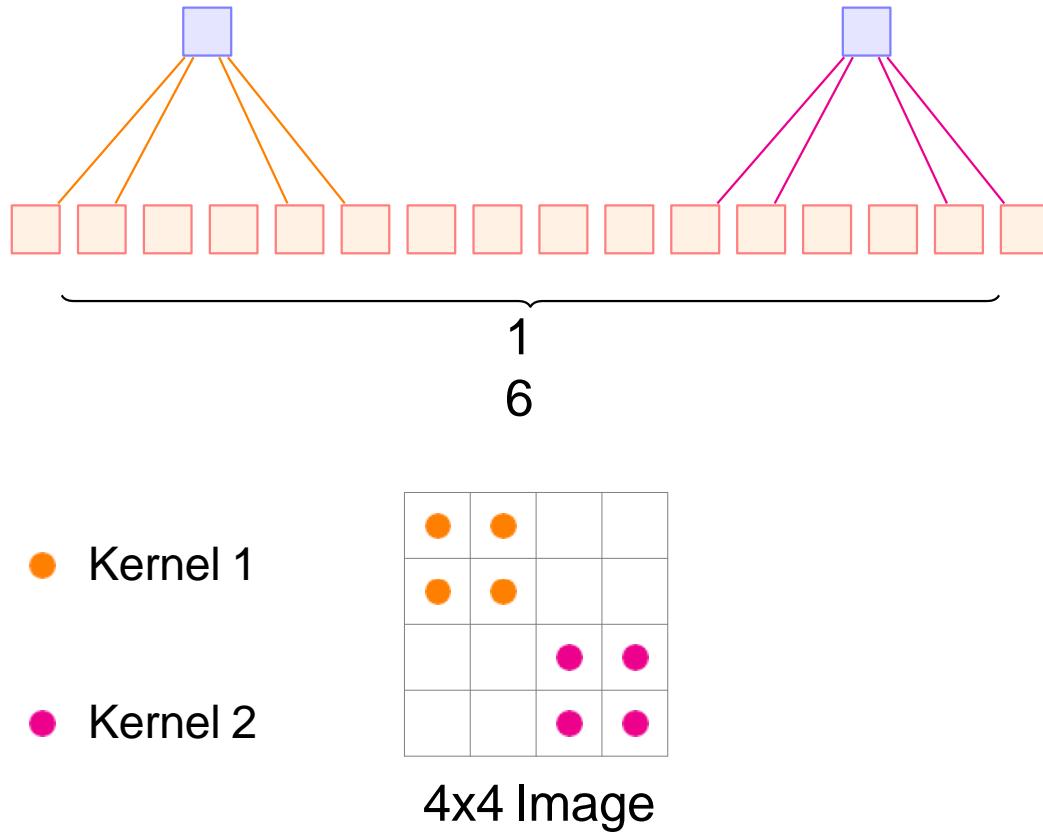
Sparse Connectivity

- But is sparse connectivity really good thing ?
- Aren't we losing information (by losing interactions between some input pixels)
- Well, not really
- The two highlighted neurons (x_1 & x_5) do not interact in *layer 1*
- But they indirectly contribute to the computation of g_3 and hence interact indirectly



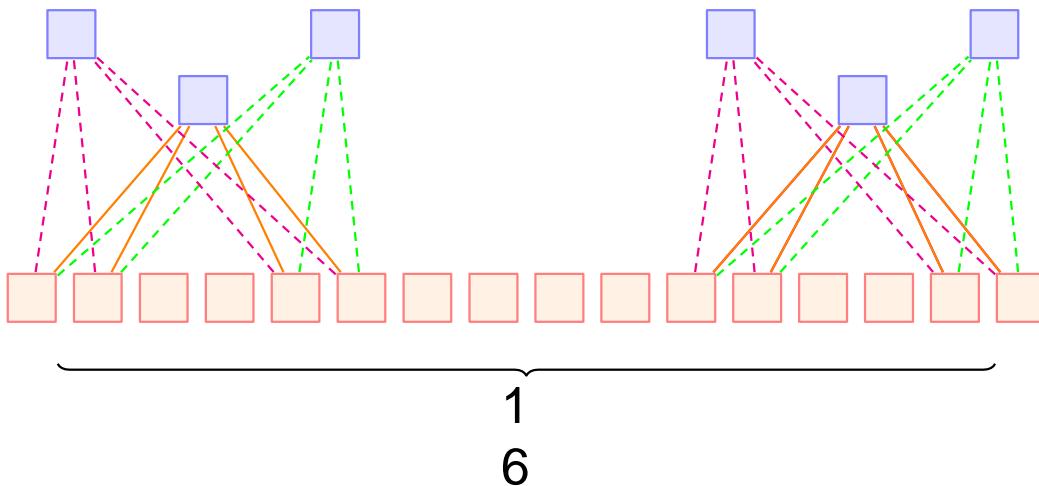
*Goodfellow-et-al-2016

Weight-sharing

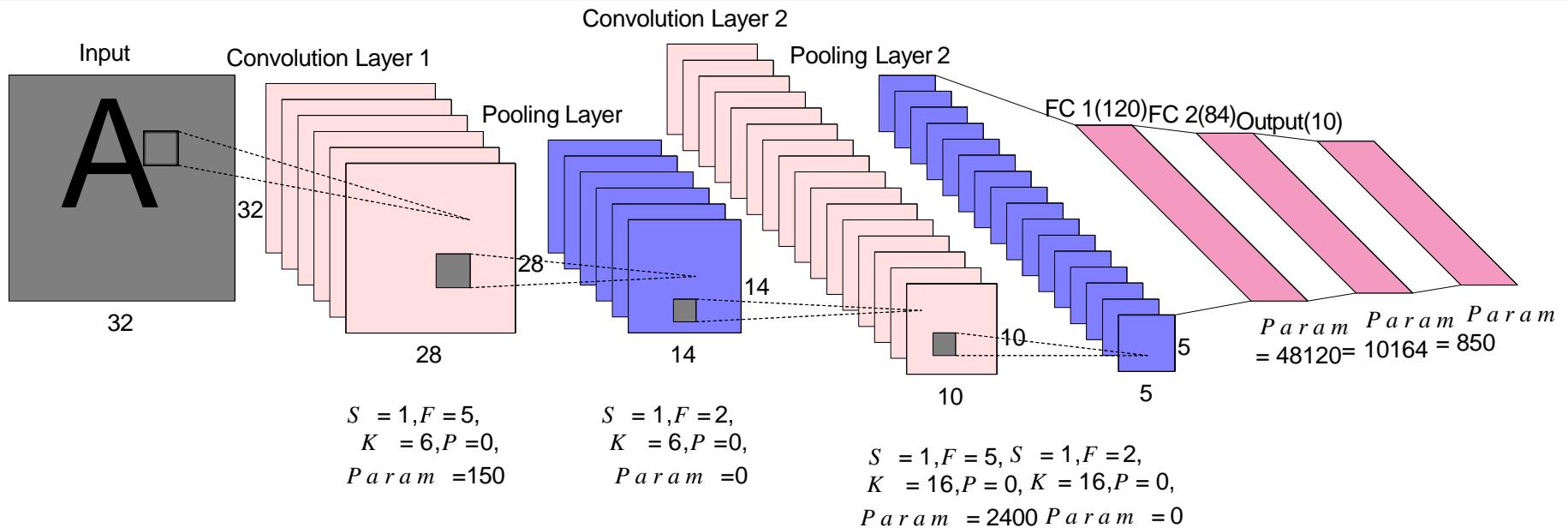


- Another characteristic of CNNs is **weight sharing**
- Consider the following network
- Do we want the kernel weights to be different for different portions of the image?
- Imagine that we are trying to learn a kernel that detects edges
- Shouldn't we be applying the same kernel at all the portions of the image?

Weight-sharing

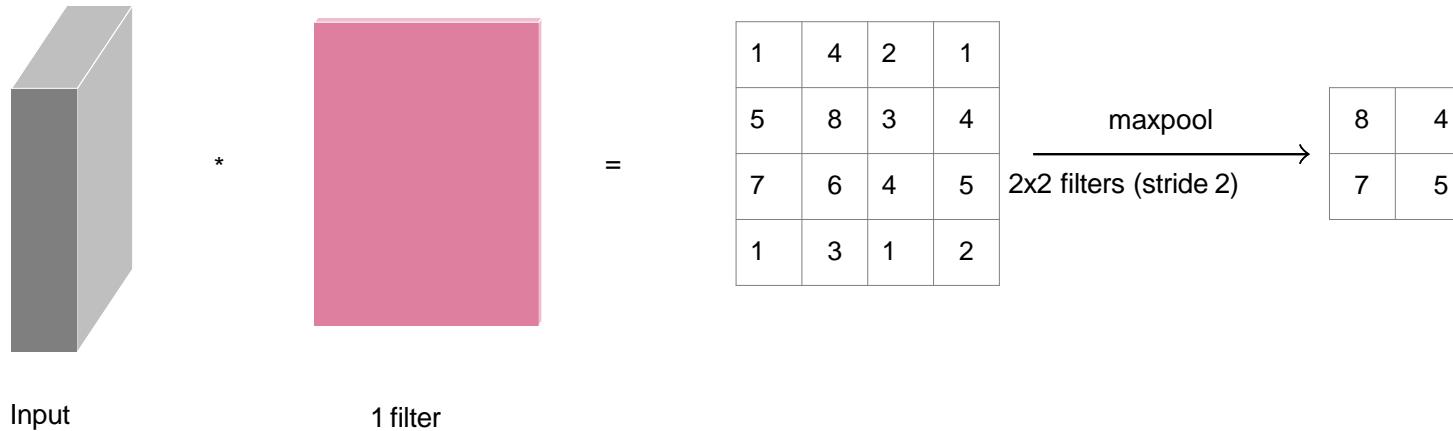


- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?
- No, we can have many such kernels but the kernels will be shared by all locations in the image
- This is called “weight sharing”

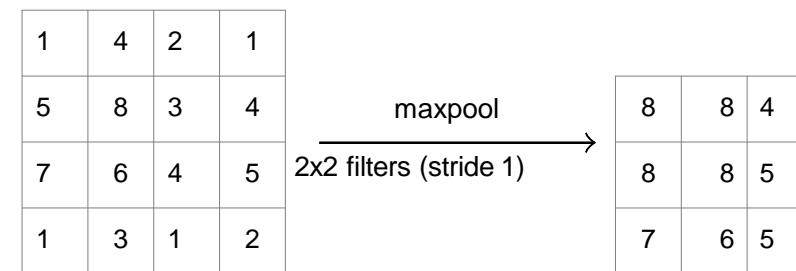
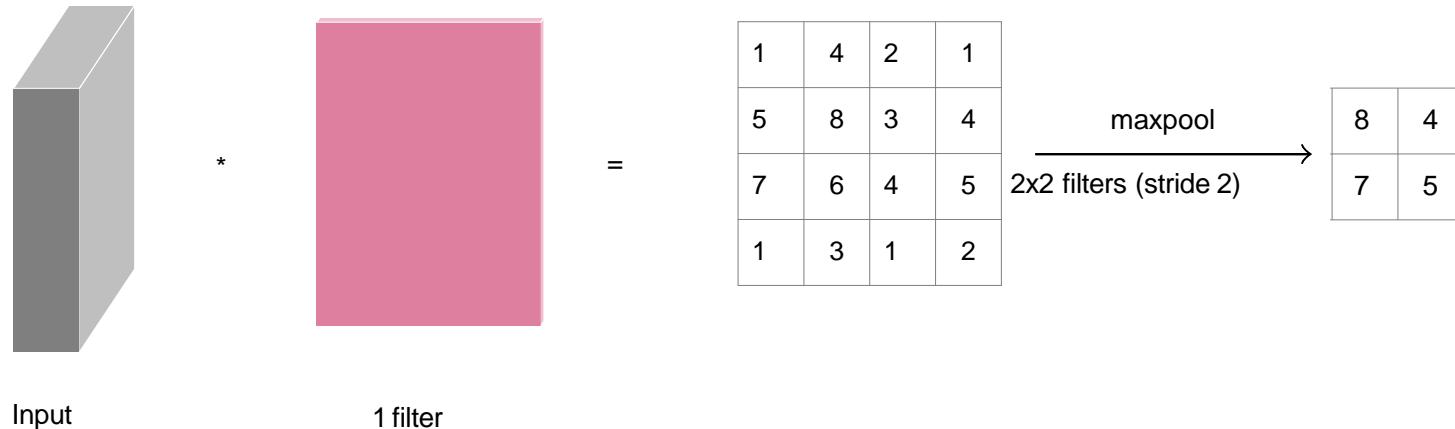


- It has alternate convolution and pooling layers
- What does a pooling layer do?
- Let us see

Pooling



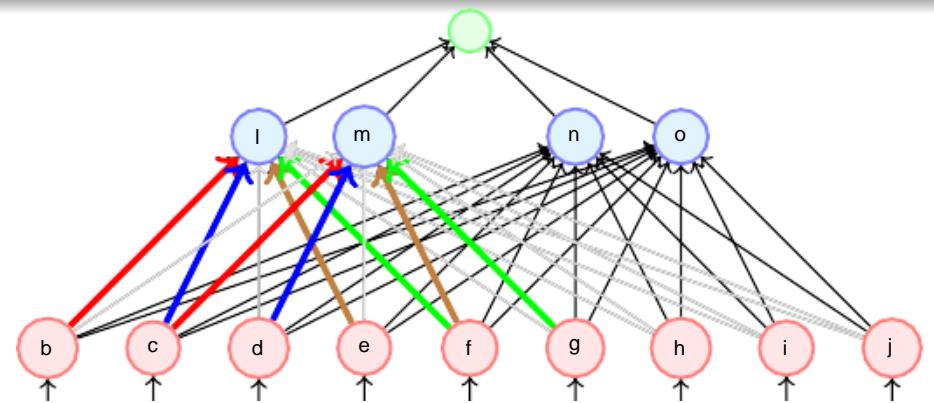
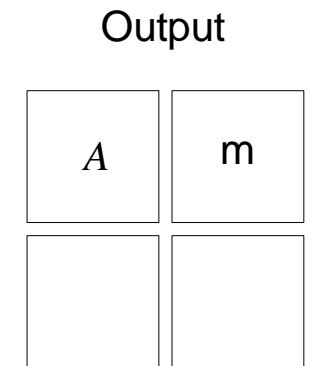
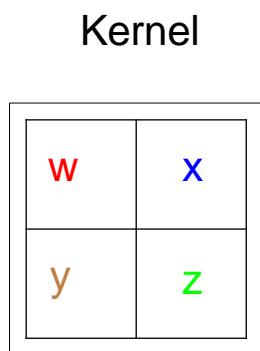
Pooling



- Instead of max pooling we can also do average pooling

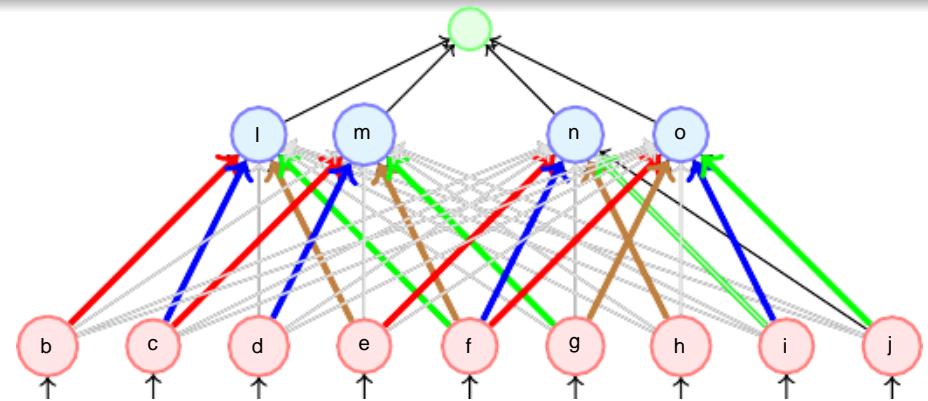
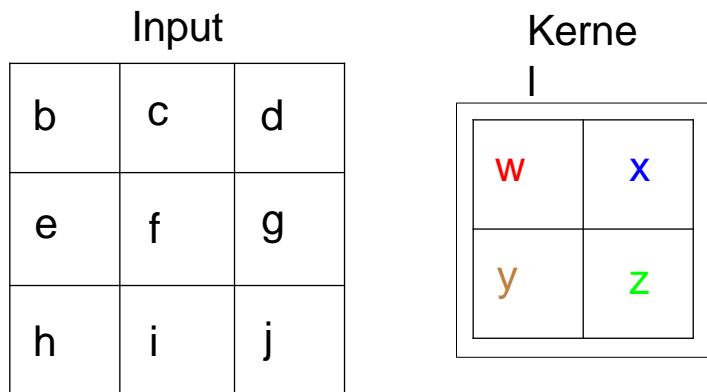
Training CNN

Input		
b	c	d
e	f	g
h	i	j



- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero

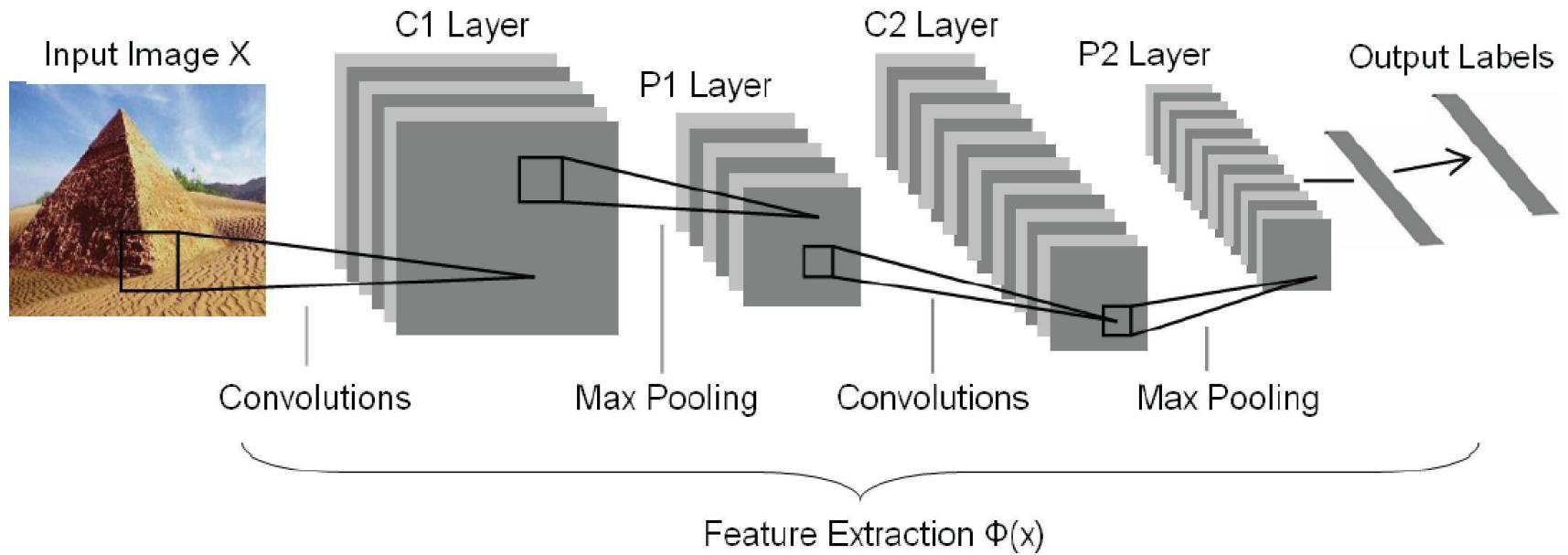
Training CNN



- We can thus train a convolution neural network using backpropagation by thinking of it as a feedforward neural network with sparse connections
- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero

Convolutional Neural Nets for Image Recognition

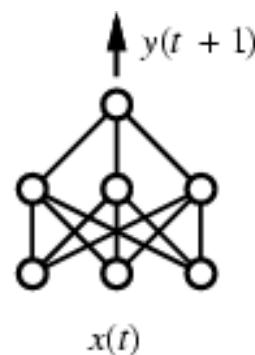
[Le Cun, 1992]



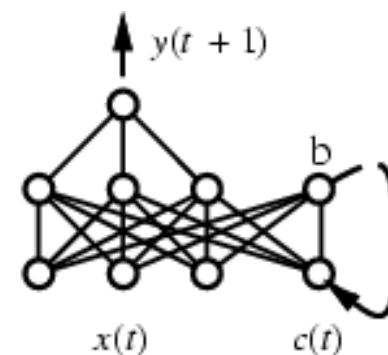
- specialized architecture: mix different types of units, not completely connected, motivated by primate visual cortex
- many shared parameters, stochastic gradient training
- very successful! now many specialized architectures for vision, speech, translation, ...

Recurrent Networks: Time Series

- Suppose we want to predict next state of world
 - and it depends on history of unknown length
 - e.g., robot with forward-facing sensors trying to predict next sensor reading as it moves and turns
- Idea: use hidden layer in network to capture state history

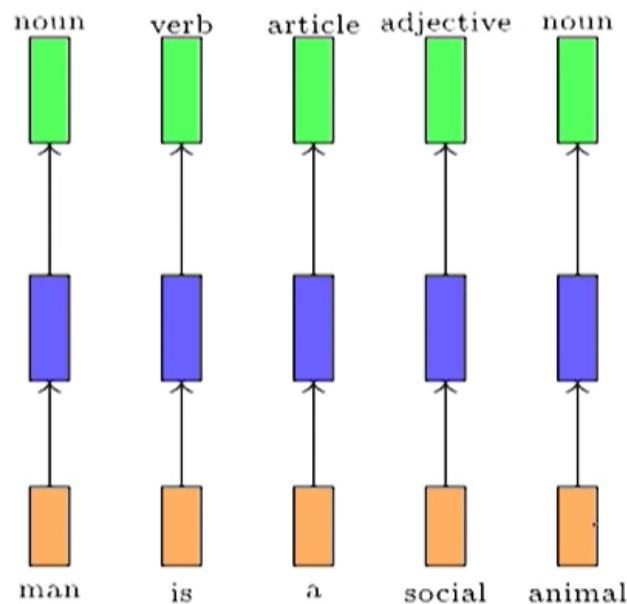


(a) Feedforward network



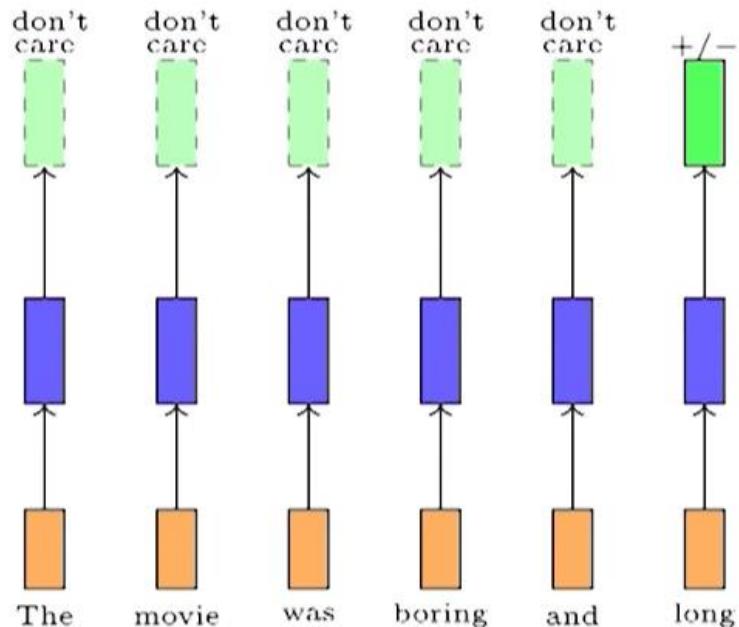
(b) Recurrent network

Recurrent Neural Network



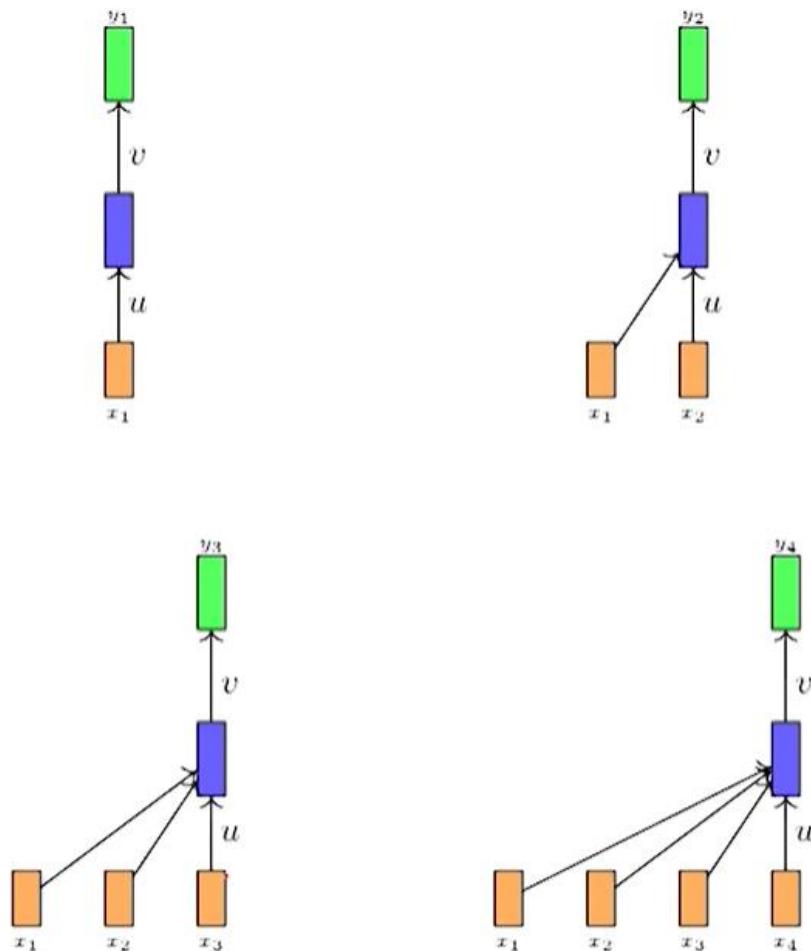
- Consider the task of predicting the part of speech tag (noun, adverb, adjective verb) of each word in a sentence
- Once we see an adjective (social) we are almost sure that the next word should be a noun (man)
- Thus the current output (noun) depends on the current input as well as the previous input
- Further the size of the input is not fixed (sentences could have arbitrary number of words)
- Notice that here we are interested in producing an output at each time step
- Each network is performing the same task (**input** : word, **output** : tag)

Recurrent Neural Network



- Sometimes we may not be interested in producing an output at every stage
- Instead we would look at the full sequence and then produce an output
- For example, consider the task of predicting the polarity of a movie review
- The prediction clearly does not depend only on the last word but also on some words which appear before
- Here again we could think that the network is performing the same task at each step (input : word, output : $+/-$) but it's just that we don't care about intermediate outputs

Recurrent Neural Network



- First, the function being computed at each time-step now is different

$$y_1 = f_1(x_1)$$

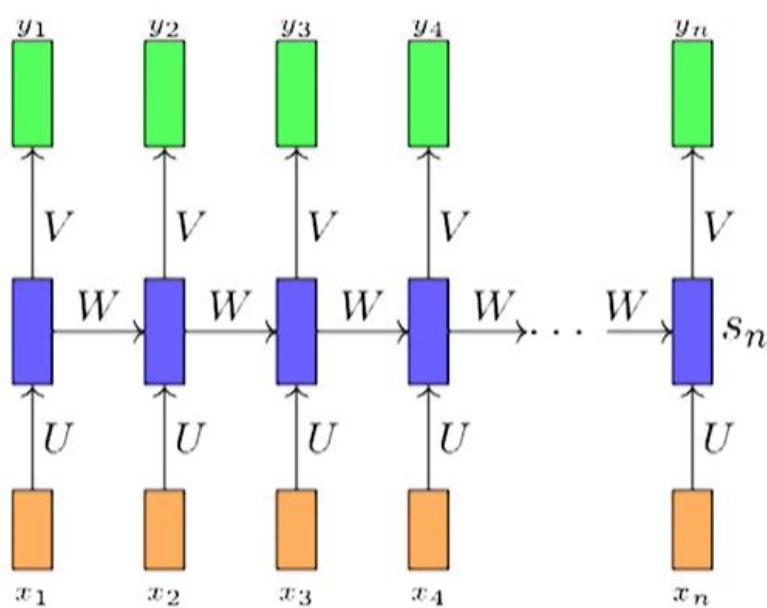
$$y_2 = f_2(x_1, x_2)$$

$$y_3 = f_3(x_1, x_2, x_3)$$

- The network is now sensitive to the length of the sequence
- For example a sequence of length 10 will require f_1, \dots, f_{10} whereas a sequence of length 100 will require f_1, \dots, f_{100}

Recurrent Neural Network

- The solution is to add a recurrent connection in the network,



$$s_i = \sigma(Ux + Ws_{i-1} + b)$$

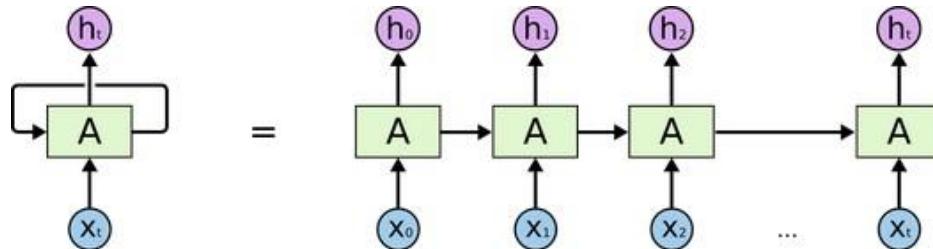
$$y_i = \sigma(Vs_i + c)$$

or

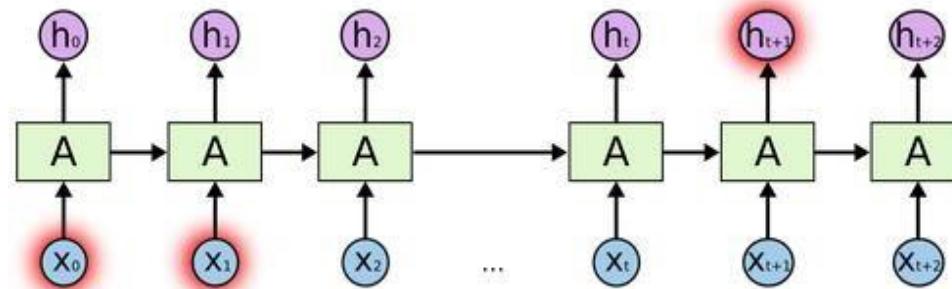
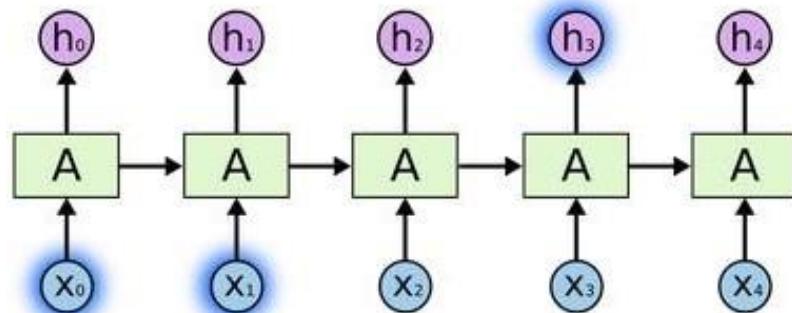
$$y_i = f(x_i, s_i, W, U, V)$$

- s_i is the state of the network at timestep i
- The parameters are W, U, V which are shared across timesteps
- The same network (and parameters) can be used to compute y_1, y_2, \dots, y_{10} or y_{100}

Recurrent Neural Networks



An unrolled recurrent neural network.



Why RNNs:

Traditional neural networks can't have persistence.

For example, imagine you want to classify what kind of event is happening at every point in a movie. It's unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.

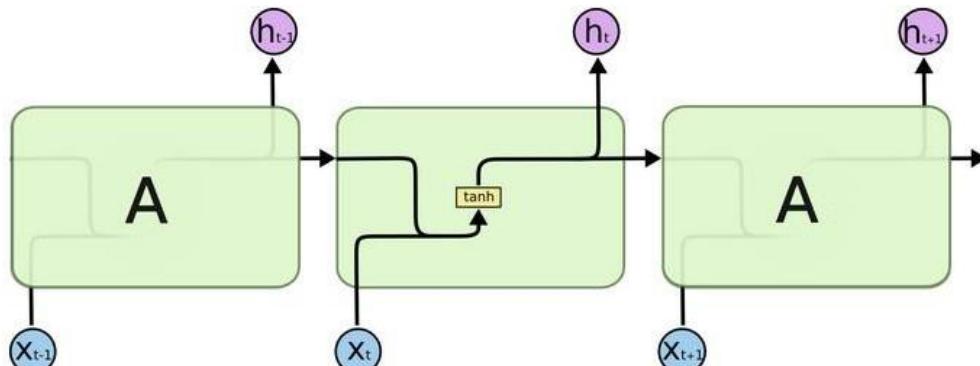
RNNs address this issue. They are networks with loops in them, allowing information to persist.

Problem of Long Term Dependency:

Consider trying to predict the last word in the text "I grew up in France... I speak fluent *French*." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.

Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.

Long Short Term Memory Networks

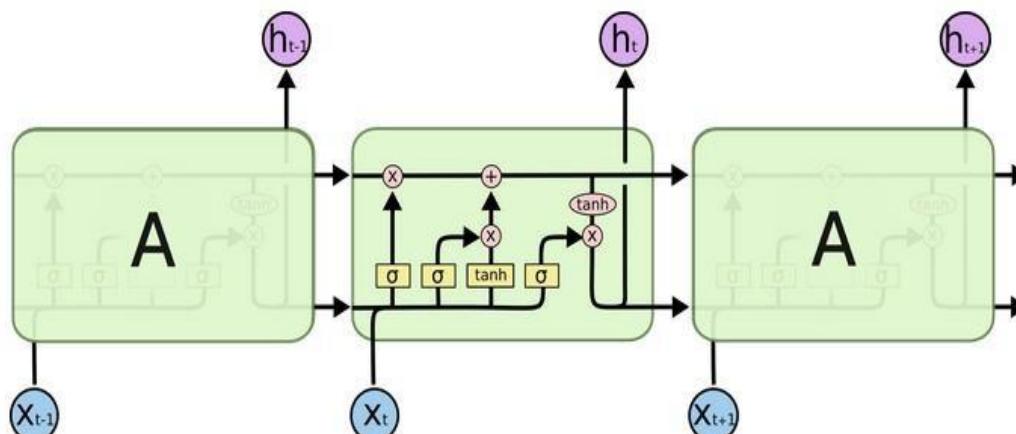


The repeating module in a standard RNN contains a single layer.

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

- Forget gate layer
- Input gate layer
- Control gate layer
- Output gate layer



The repeating module in an LSTM contains four interacting layers.

The LSTM have the ability to remove or add information to the cell state, carefully regulated by structures called **gates**.

The sigmoid layer outputs numbers between 0 and 1, describing how much of each component should be let through. A value of 0 means “let nothing through,” while a value of 1 means “let everything through!”

Artificial Neural Networks: Summary

- Highly non-linear regression/classification
- Hidden layers learn intermediate representations
- Potentially millions of parameters to estimate
- Stochastic gradient descent, local minima problems
- Deep networks have produced real progress in many fields
 - computer vision
 - speech recognition
 - mapping images to text
 - recommender systems
 - ...
- They learn very useful non-linear representations

Good References for understanding Neural Network

Mitesh Khapra

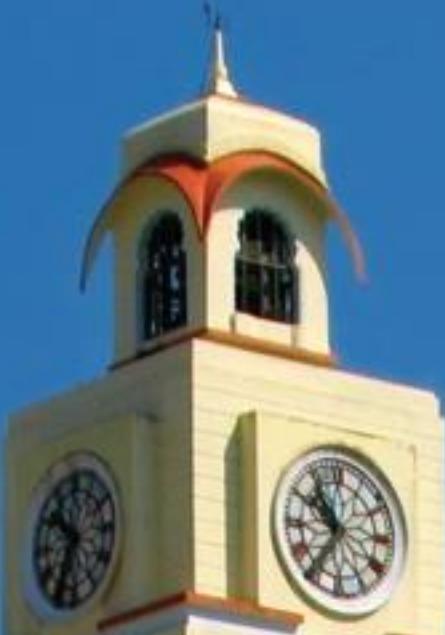
<https://www.youtube.com/watch?v=yw8xwS15Pf4>

Visualization of CNN

<https://www.youtube.com/watch?v=cNBBNAxC8I4>

Back propagation

https://www.youtube.com/watch?v=G5b4jRBKNxw&list=PLZbbT5o_s2xq7Lwl2y8_QtvuXZedL6tQU&index=25



BITS Pilani
Pilani Campus

Instance-based Learning

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in



Text Book(s)

T1	Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition
T2	Tom M. Mitchell: Machine Learning, The McGraw-Hill Companies, Inc..

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Tom Mitchell, Prof.. Burges, Prof. Andrew Moore and many others who made their course materials freely available online.

Topics to be covered

- Instance based learning
 - K-Nearest Neighbour Learning
 - Locally Weighted Regression (LWR)
Learning
 - Radial Basis Functions
-

k-Nearest Neighbor Classifier

- Nearest Neighbour classifier is an instance based classifier
- ‘lazy learning’, as learning is postponed until a new instance is encountered
- Constructs a local approximation to the target function, applicable in the neighbourhood of new instance
- Suitable in cases where target function is complex over the entire input space, but easily describable in local approximations
- Real world applications found in recommendation systems (amazon).
- Caveat is the high cost of classification, which happens at the time of processing rather than before hand (there’s no training phase)

Instance Based Learning

Key idea: just store all training examples $\langle x_i, f(x_i) \rangle$

Nearest neighbor:

- Given query instance x_q , first locate nearest training example x_n , then estimate $f^*(x_q) = f(x_n)$

K-nearest neighbor:

- Given x_q , take vote among its k nearest neighbors (if discrete-valued target function)
- Take mean of f values of k nearest neighbors (if real-valued) $f^*(x_q) = \sum_{i=1}^k f(x_i)/k$

When to Consider Nearest Neighbors

- Instances map to points in \mathbb{R}^N
- Less than 20 attributes per instance
- Lots of training data

Advantages:

- Training is very fast
- Learn complex target functions
- Do not lose information

Disadvantages:

- Slow at query time
- Easily fooled by irrelevant attributes

k-Nearest Neighbor Classifier

- Considers all instances as members of n-dimensional space
- Nearest neighbours of an instance is determined based on Euclidean distance
- Distance between two n-dimensional instances x_i and x_j is given by:

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^n (a_r(x_i) - a_r(x_j))^2}$$

- For nearest neighbour classifier, target function can be discrete or continuous

Distance Measures : Special Cases of Minkowski

- $h = 1$: Manhattan (city block, L₁ norm) distance

$$d(i, j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \dots + |x_{ip} - x_{jp}|$$

- $h = 2$: (L₂ norm) Euclidean distance

$$d(i, j) = \sqrt{(|x_{i1} - x_{j1}|^2 + |x_{i2} - x_{j2}|^2 + \dots + |x_{ip} - x_{jp}|^2)}$$

- $h \rightarrow \infty$. “supremum” (L_{max} norm, L _{∞} norm) distance.
 - This is the maximum difference between any component (attribute) of the vectors

$$d(i, j) = \lim_{h \rightarrow \infty} \left(\sum_{f=1}^p |x_{if} - x_{jf}|^h \right)^{\frac{1}{h}} = \max_f^p |x_{if} - x_{jf}|$$

Standardizing Numeric Data

- X: raw score to be standardized, μ : mean of the population, σ : standard deviation
- the distance between the raw score and the population mean in units of the standard deviation
- negative when the raw score is below the mean, “+” when above

Where

$$z = \frac{x - \mu}{\sigma}$$

Multiple Features

$$y_n = w_0 + w_1 f_{n1} + w_2 f_{n2} + w_3 f_{n3} + w_4 f_{n4}$$

Size (feet ²) f1	Number of bedrooms f2	Number of floors f3	Age of home (years) f4	Price (\$1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...

Feature scaling

Feature scaling of features x_i consists of rescaling the range of features to scale the range in $[0, 1]$ or $[-1, 1]$ (Do not apply to $x_0 = 1$)

E.g. $x_1 = \frac{\text{size} - 1000}{2000}$

Average value of x_1
Maximum value of x_1 – min value of x_1

$$x_2 = \frac{\#\text{bedrooms} - 2}{5}$$

Discrete and Continuous-valued function

- **discrete-valued target function:**

- $f : \mathbb{R}^n \rightarrow V$ where V is the finite set $\{v_1, v_2, \dots, v_s\}$
- the target function value is the most common value among the k nearest training examples

$$\hat{f}(x_q) \leftarrow \operatorname{argmax}_{v \in V} \sum_{i=1}^k \delta(v, f(x_i))$$

where $\delta(a, b) = (a == b)$

- **continuous-valued target function:**

- algorithm has to calculate the mean value instead of the most common value
- $f : \mathbb{R}^n \rightarrow \mathbb{R}$

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^k f(x_i)}{k}$$

k-Nearest Neighbor Classifier

Training algorithm:

- For each training example $(x, f(x))$, add the example to the list *training_examples*

Classification algorithm:

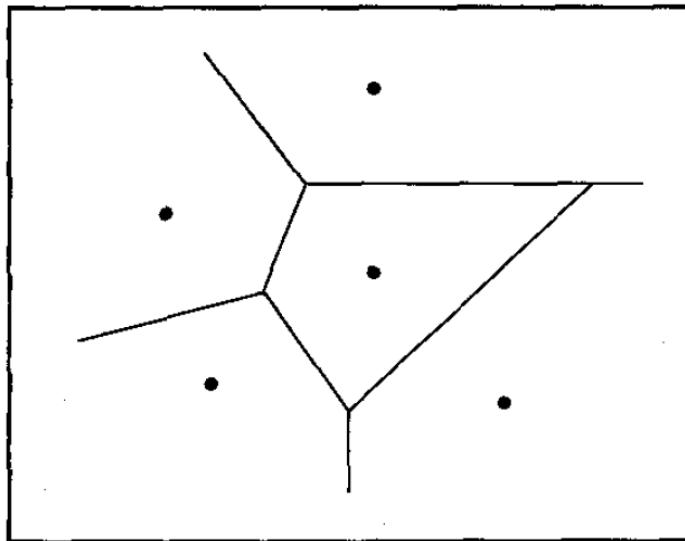
- Given a query instance x_q to be classified,
 - Let $x_1 \dots x_k$ denote the k instances from *training_examples* that are nearest to x_q
 - Return

$$\hat{f}(x_q) \leftarrow \operatorname{argmax}_{v \in V} \sum_{i=1}^k \delta(v, f(x_i))$$

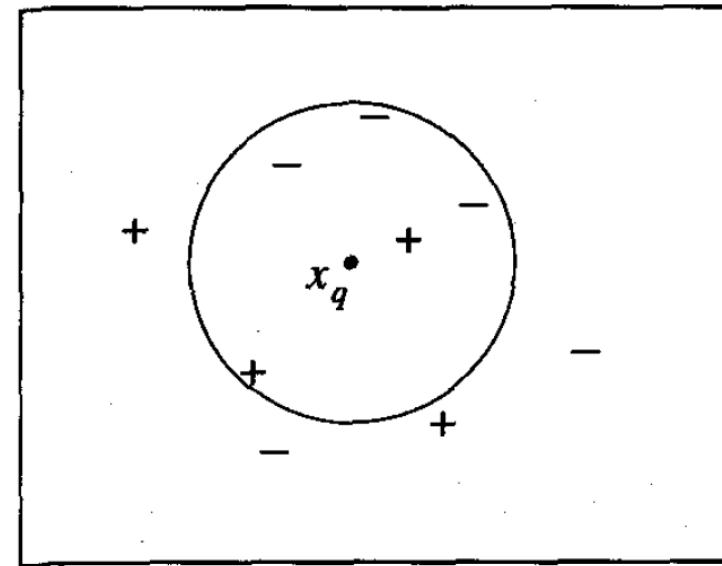
where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

* It can be used for Regression as well.

k-NN examples



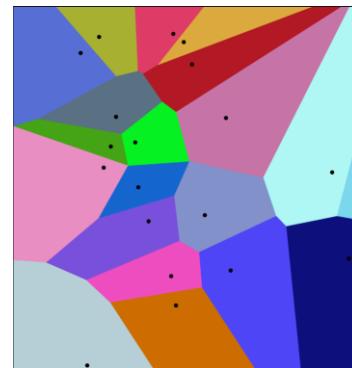
K=1



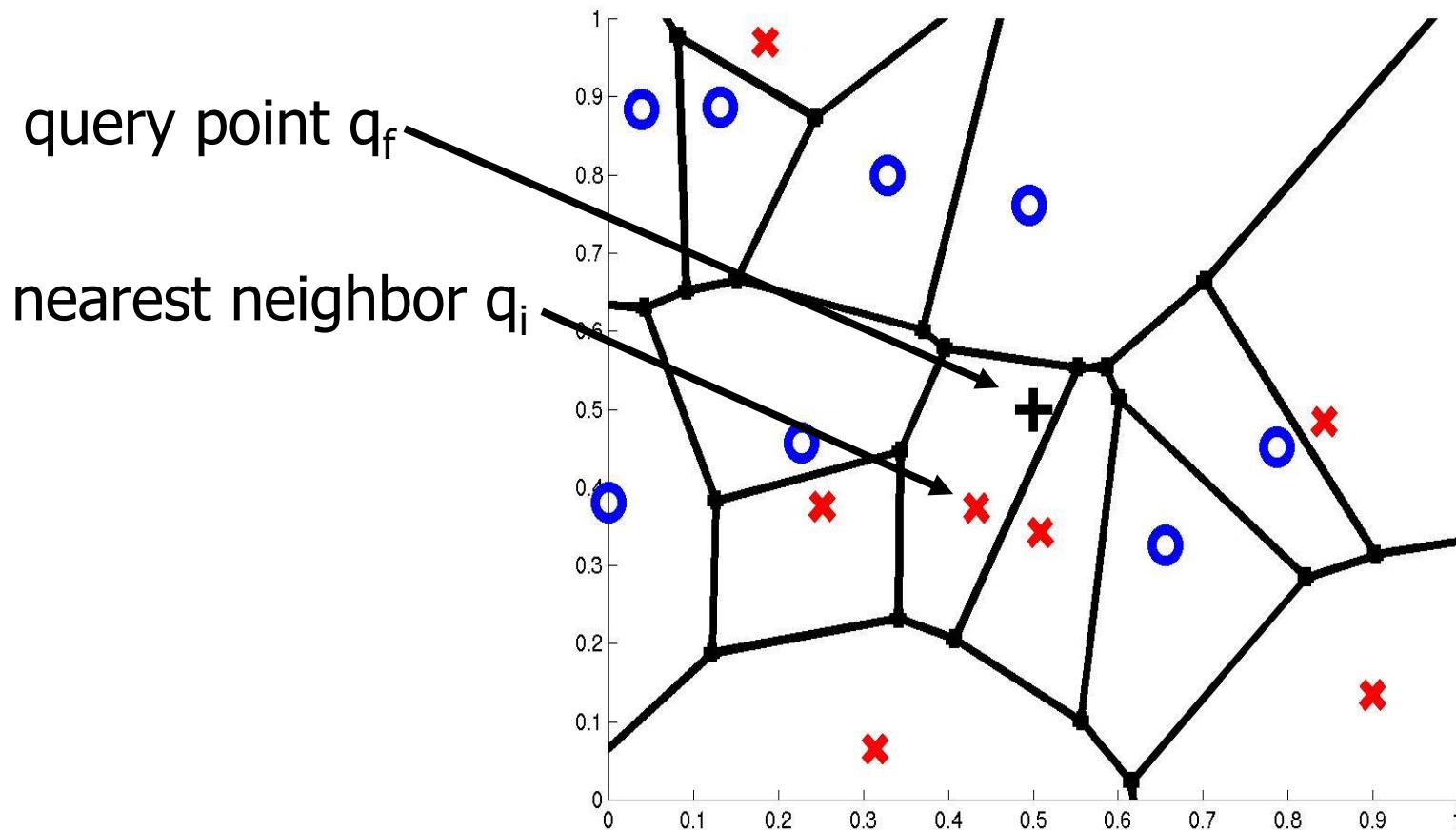
K=5

Voronoi Diagram

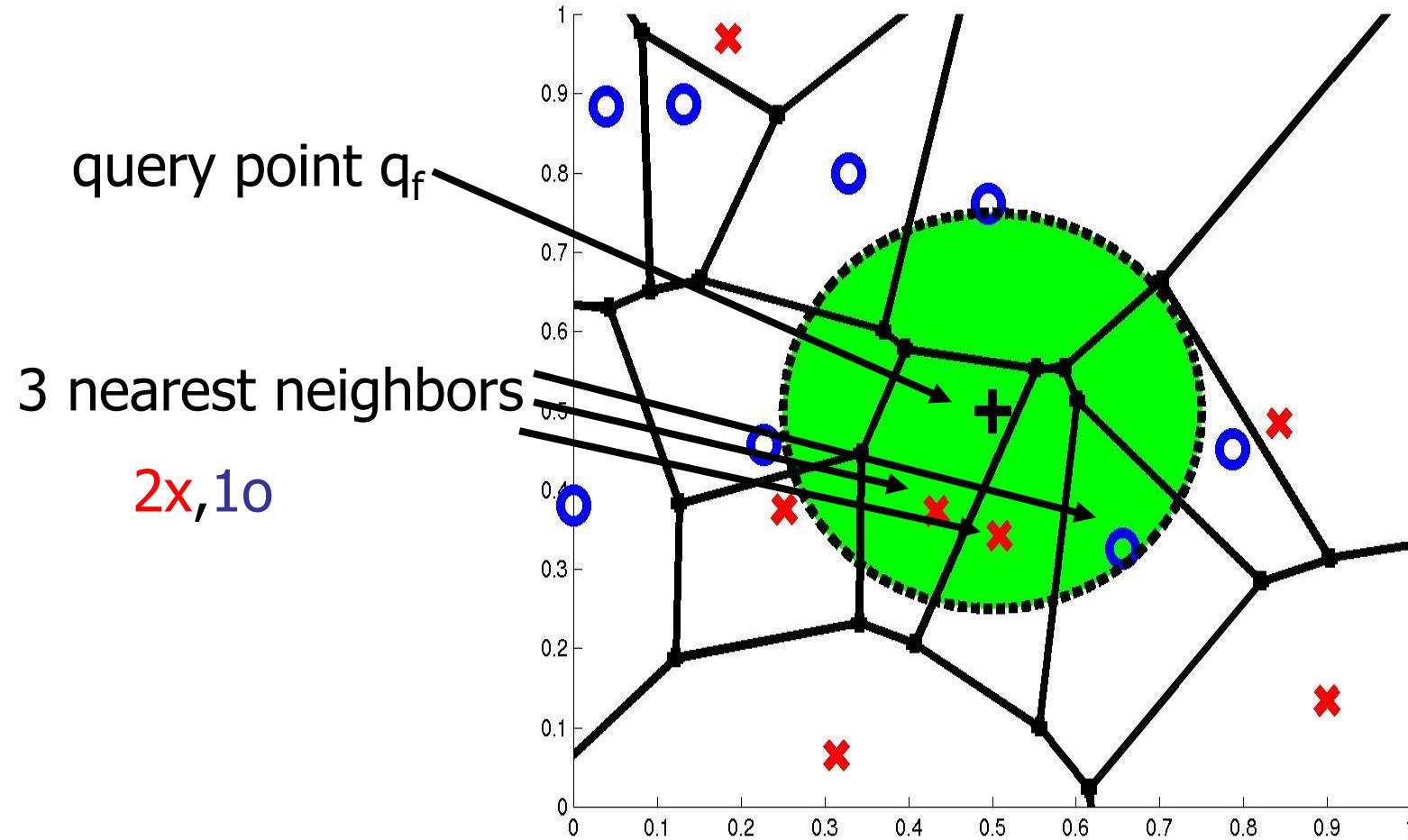
- It is a partition of a plane into regions close to each of a given set of objects.



Voronoi Diagram



3-Nearest Neighbors

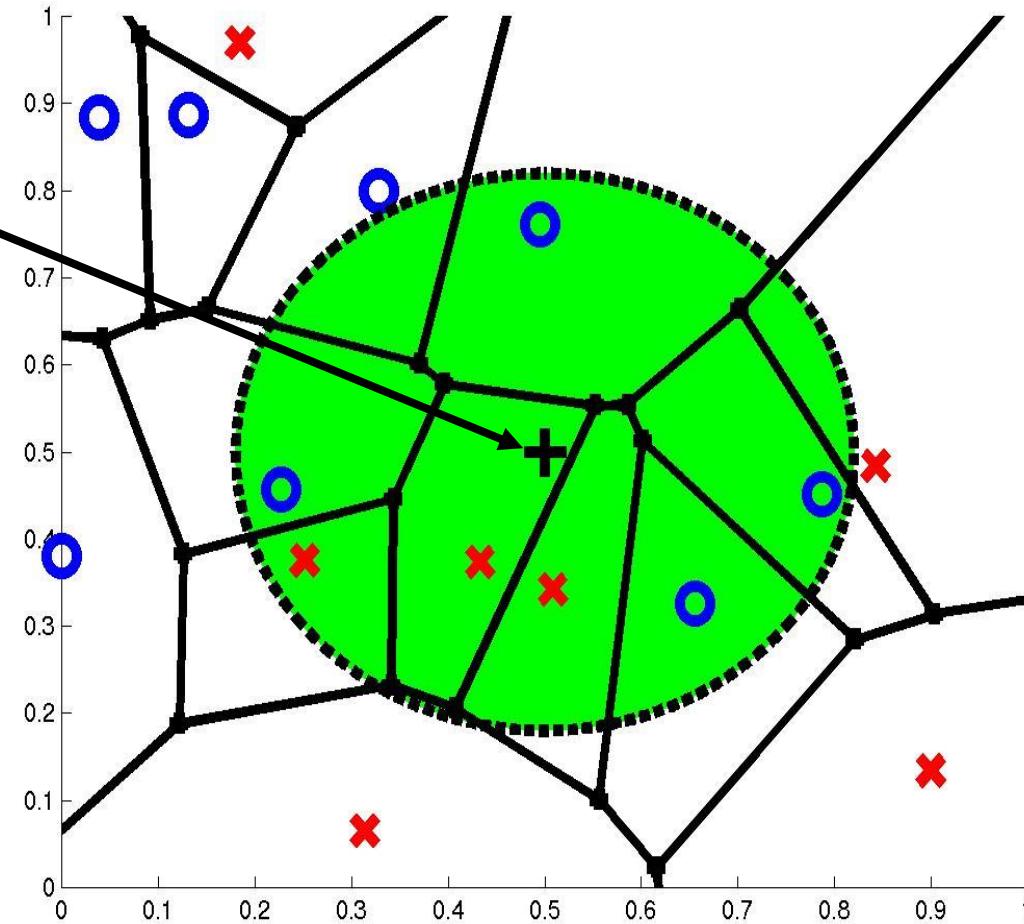


7-Nearest Neighbors

query point q_f

7 nearest neighbors

$3x, 4o$

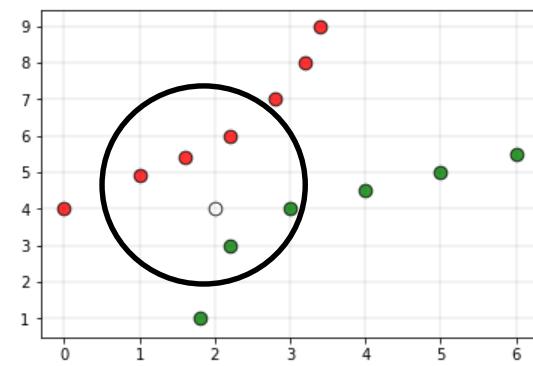


Various issues that affect the performance of kNN:

Performance of a classifier largely depends on the of the hyperparameter k

- Choosing smaller values for K, noise can have a higher influence on the result.
- Larger values of k are computationally expensive

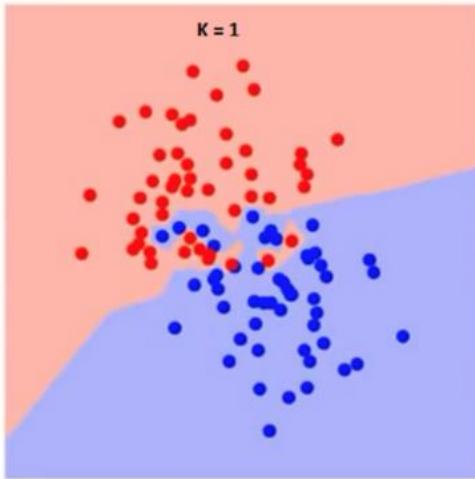
Assigning the class labels can be tricky. For example, in the below case, for (k=5) the point is closer to ‘green’ classification, but gets classified as ‘red’ due to higher red votes/majority voting to ‘red’



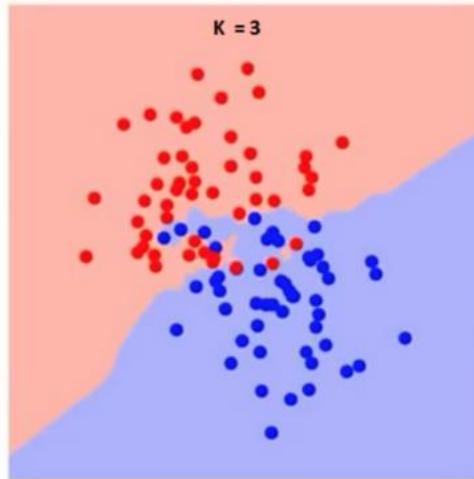
Finding optimal k for kNN classifiers



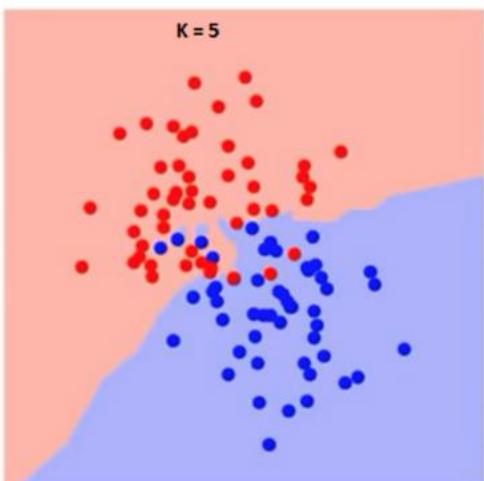
K = 1



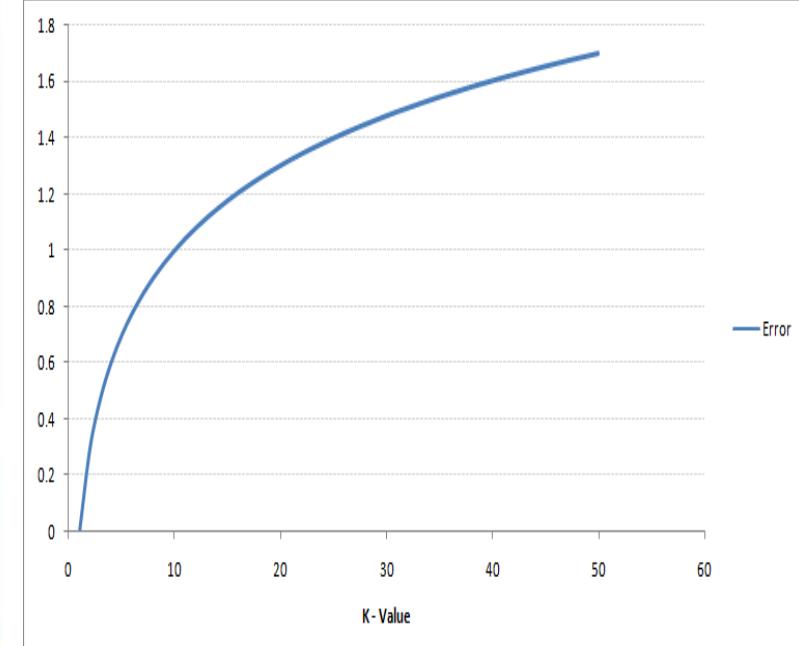
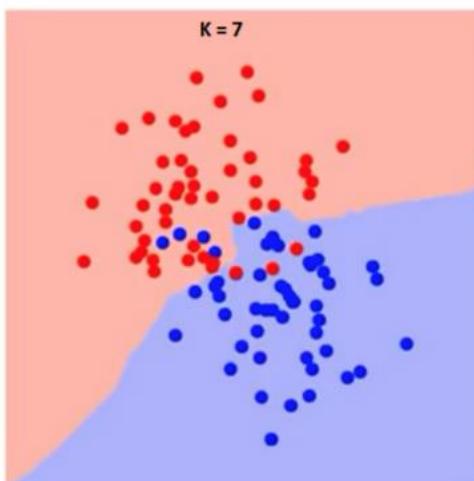
K = 3



K = 5

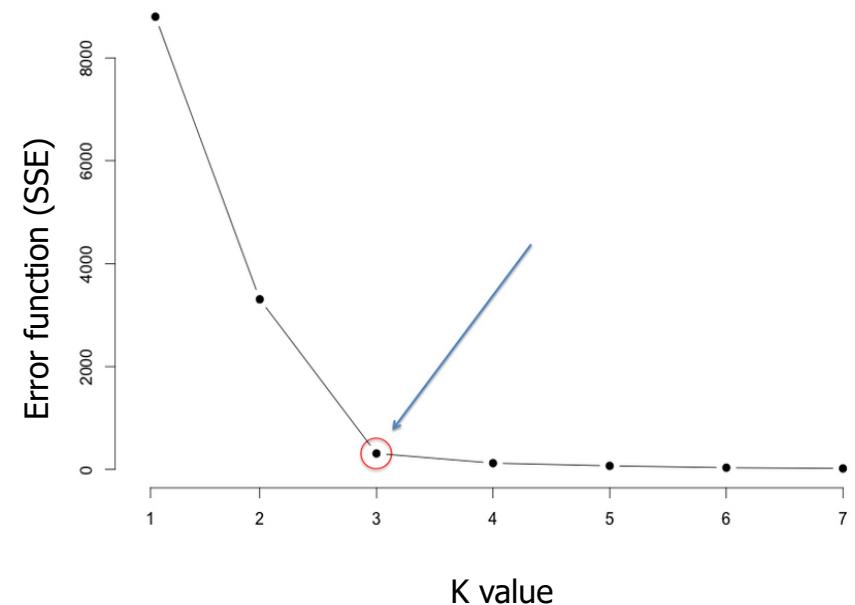


K = 7



Finding K - Elbow method

- Compute sum of squares error (SSE) or any other error function for varying values of K (1 to a reasonable X) and plot against K
- In the plot, the elbow (see pic) gives the value of K beyond which the error function plot almost flattens
- As K approaches the total number of instances in the set, error function drops down to '0', but beyond optimal K, the model becomes too generic



Distance weighted nearest neighbor

- ➊ contribution of each of the k nearest neighbors is weighted accorded to their distance to x_q
 - discrete-valued target functions

$$\hat{f}(x_q) \leftarrow \operatorname{argmax}_{v \in V} \sum_{i=1}^k w_i \delta(v, f(x_i))$$

where $w_i \equiv \frac{1}{d(x_q, x_i)^2}$ and $\hat{f}(x_q) = f(x_i)$ if $x_q = x_i$

- ➋ continuous-valued target function:

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^k w_i f(x_i)}{\sum_{i=1}^k w_i}$$

Distance Weighted k-NN

Give more weight to neighbors closer to the query point

$$- f^*(x_q) = \sum_{i=1}^k w_i f(x_i) / \sum_{i=1}^k w_i \\ ; w_i = K(d(x_q, x_i))$$

and

– $d(x_q, x_i)$ is the distance between x_q and x_i

Variation: Instead of only k-nearest neighbors use all training examples (Shepard's method)

Distance Weighted Average

Weighting the data

$$- f^*(x_q) = \sum_i f(x_i) K(d(x_i, x_q)) / \sum_i K(d(x_i, x_q))$$

Relevance of a data point $(x_i, f(x_i))$ is measured by calculating the distance $d(x_i, x_q)$ between the query x_q and the input vector x_i

Weighting the error criterion

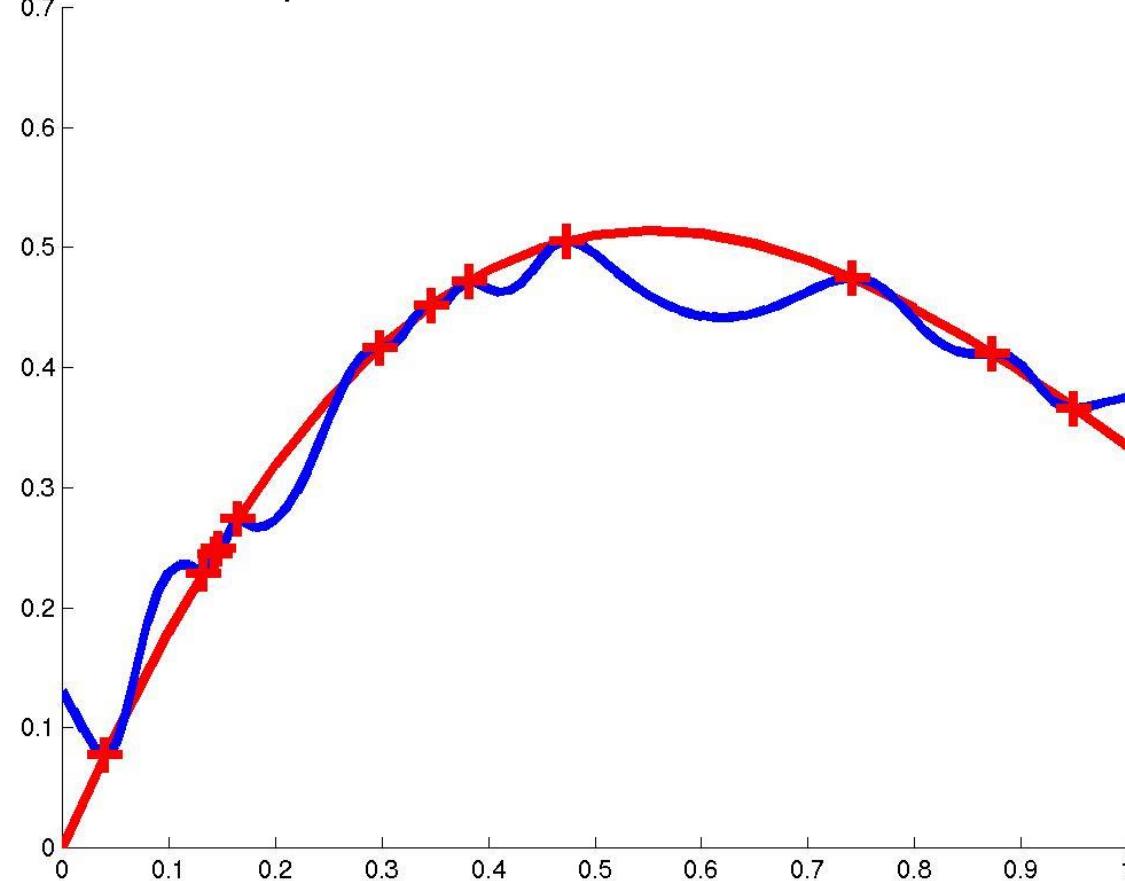
$$- E(x_q) = \sum_i (f^*(x_q) - f(x_i))^2 K(d(x_i, x_q))$$

Best estimate $f^*(x_q)$ will minimize the cost $E(x_q)$, therefore

$$\partial E(x_q) / \partial f^*(x_q) = 0$$

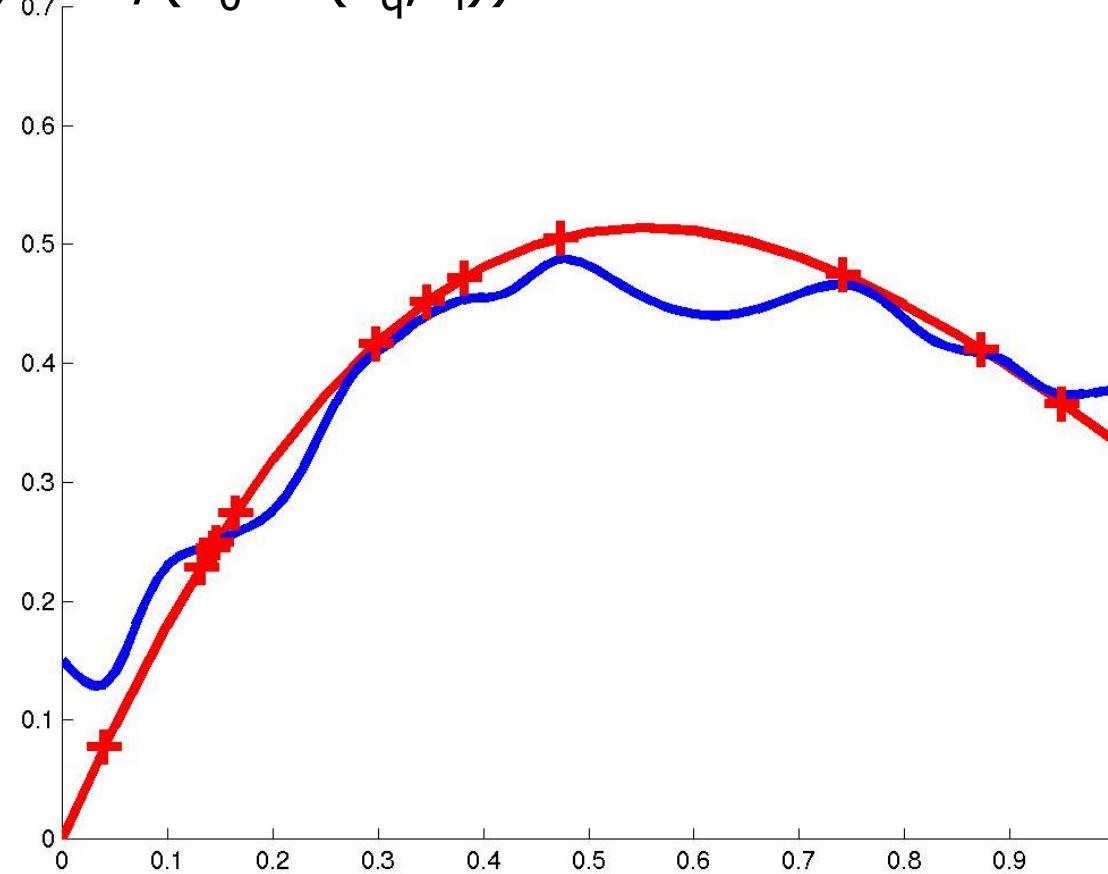
Distance Weighted NN

$$K(d(x_q, x_i)) = 1/d(x_q, x_i)^2$$



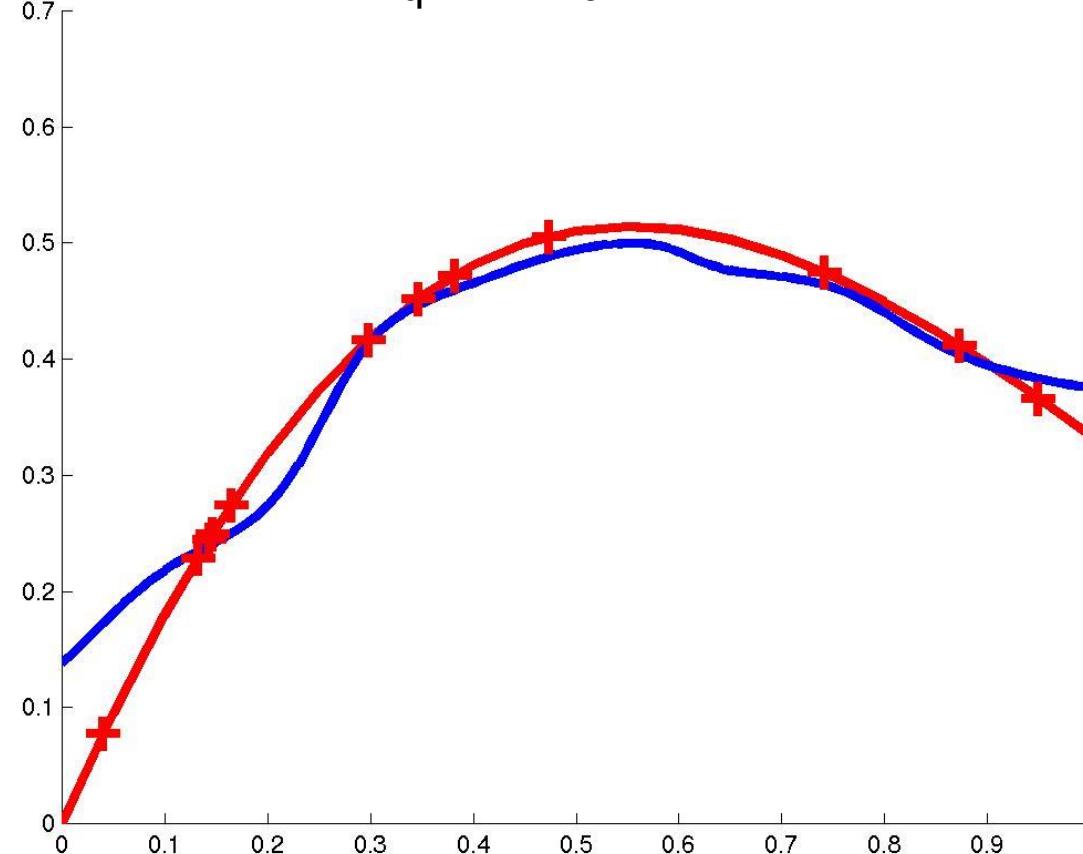
Distance Weighted NN

$$K(d(x_q, x_i)) = 1/(d_0 + d(x_q, x_i))^2$$



Distance Weighted NN

$$K(d(x_q, x_i)) = \exp(-(d(x_q, x_i)/\sigma_0)^2)$$



Curse of Dimensionality

Imagine instances described by 20 attributes but only a few are relevant to target function

Curse of dimensionality: nearest neighbor is easily misled when instance space is high-dimensional

One approach:

Stretch j -th axis by weight z_j , where z_1, \dots, z_n chosen to minimize prediction error

Use cross-validation to automatically choose weights

z_1, \dots, z_n

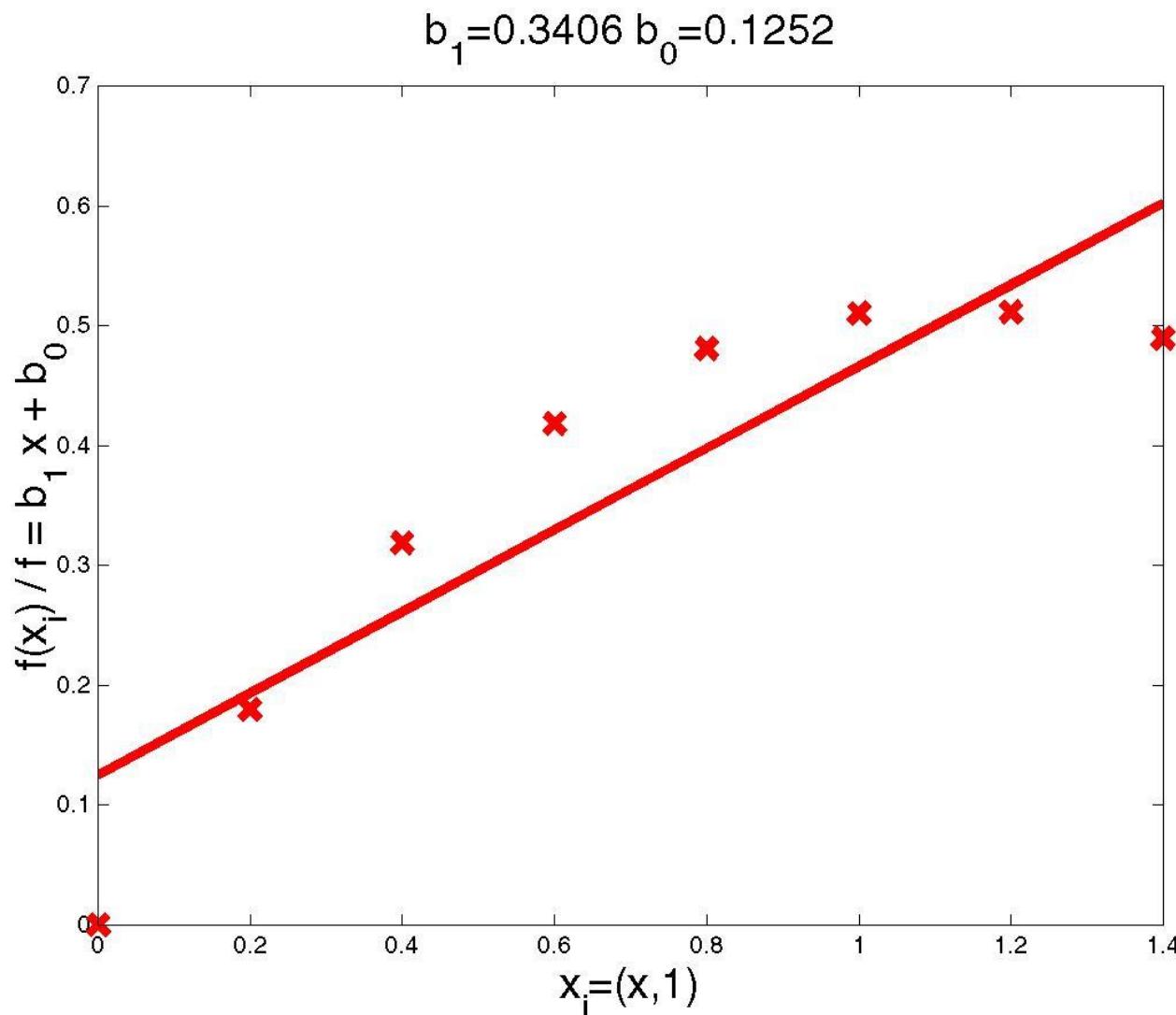
Note setting z_j to zero eliminates this dimension altogether (feature subset selection)

Locally Weighted Regression

Locally Weighted Regression

- Locally – Function approximated based on data near query point
- Weighted – Contribution by each training example is weighted by its distance from query point
- Regression- Approximates real-valued target function

Linear Regression Example



Locally weighted regression

- ➊ a note on terminology:
 - ➊ *Regression* means approximating a real-valued target function
 - ➋ *Residual* is the error $\hat{f}(x) - f(x)$ in approximating the target function
 - ➋ *Kernel function* is the function of distance that is used to determine the weight of each training example. In other words, the kernel function is the function K such that $w_i = K(d(x_i, x_q))$
 - ➋ nearest neighbor approaches can be thought of as approximating the target function at the single query point x_q
 - ⌋ locally weighted regression is a generalization to this approach, because it constructs an explicit approximation of f over a local region surrounding x_q
-

Locally weighted regression

- target function is approximated using a **linear function**
$$\hat{f}(x) = w_0 + w_1 a_1(x) + \dots + w_n a_n(x)$$
 - methods like **gradient descent** can be used to calculate the coefficients w_0, w_1, \dots, w_n to minimize the error in fitting such linear functions
 - ANNs require a global approximation to the target function
 - here, just a local approximation is needed
- ⇒ the error function has to be redefined
-

Locally weighted regression



possibilities to redefine the error criterion E

1. Minimize the squared error over just the k nearest neighbors

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest neighbors}} (f(x) - \hat{f}(x))^2$$

2. Minimize the squared error over the entire set D , while weighting the error of each training example by some decreasing function K of its distance from x_q

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \cdot K(d(x_q, x))$$

3. Combine 1 and 2

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest neighbors}} (f(x) - \hat{f}(x))^2 \cdot K(d(x_q, x))$$

Locally weighted regression

- ➊ choice of the error criterion
 - E_2 is the most esthetically criterion, because it allows every training example to have impact on the classification of x_q
 - however, computational effort grows with the number of training examples
 - E_3 is a good approximation to E_2 with constant effort

$$\Delta w_j = \eta \sum_{x \in k \text{ nearest neighbors}} K(d(x_q, x))(f(x) - \hat{f}(x))a_j$$

- ➋ Remarks on locally weighted linear regression:
 - in most cases, constant, linear or quadratic functions are used
 - costs for fitting more complex functions are prohibitively high
 - simple approximations are good enough over a sufficiently small subregion of X

Design Issues in Local Regression

- Local model order (constant, linear, quadratic)
- Distance function d
 - feature scaling: $d(x,q) = (\sum_{j=1}^d m_j (x_j - q_j)^2)^{1/2}$
 - irrelevant dimensions $m_j = 0$
- kernel function K

See paper by Atkeson [1996] "Locally Weighted Learning"

Radial Basis Function

Radial Basis Function

- closely related to distance-weighted regression and to ANNs
- learned hypotheses have the form

$$\hat{f}(x) = w_0 + \sum_{u=1}^k w_u \cdot K_u(d(x_u, x))$$

where

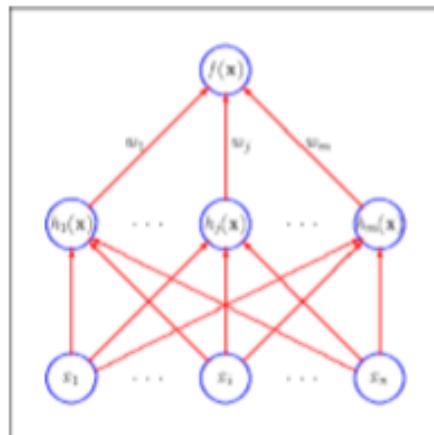
- each x_u is an instance from X and
 - $K_u(d(x_u, x))$ decreases as $d(x_u, x)$ increases and
 - k is a user-provided constant
-
- though $\hat{f}(x)$ is a global approximation to $f(x)$, the contribution of each of the K_u terms is localized to a region nearby the point x_u

Radial Basis Function

- it is common to choose each function $K_u(d(x_u, x))$ to be a Gaussian function centered at x_u with some variance σ^2

$$K_u(d(x_u, x)) = e^{-\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

- the function of $\hat{f}(x)$ can be viewed as describing a two-layer network where the first layer of units computes the various $K_u(d(x_u, x))$ values and the second layer a linear combination of the results



Training Radial Basis Function Networks

How to choose the center x_n for each Kernel function K_n ?

- scatter uniformly across instance space
- use distribution of training instances (clustering)

How to train the weights?

- Choose mean x_n and variance σ_n for each K_n
non-linear optimization or EM
- Hold K_n fixed and use local linear regression to
compute the optimal weights w_n

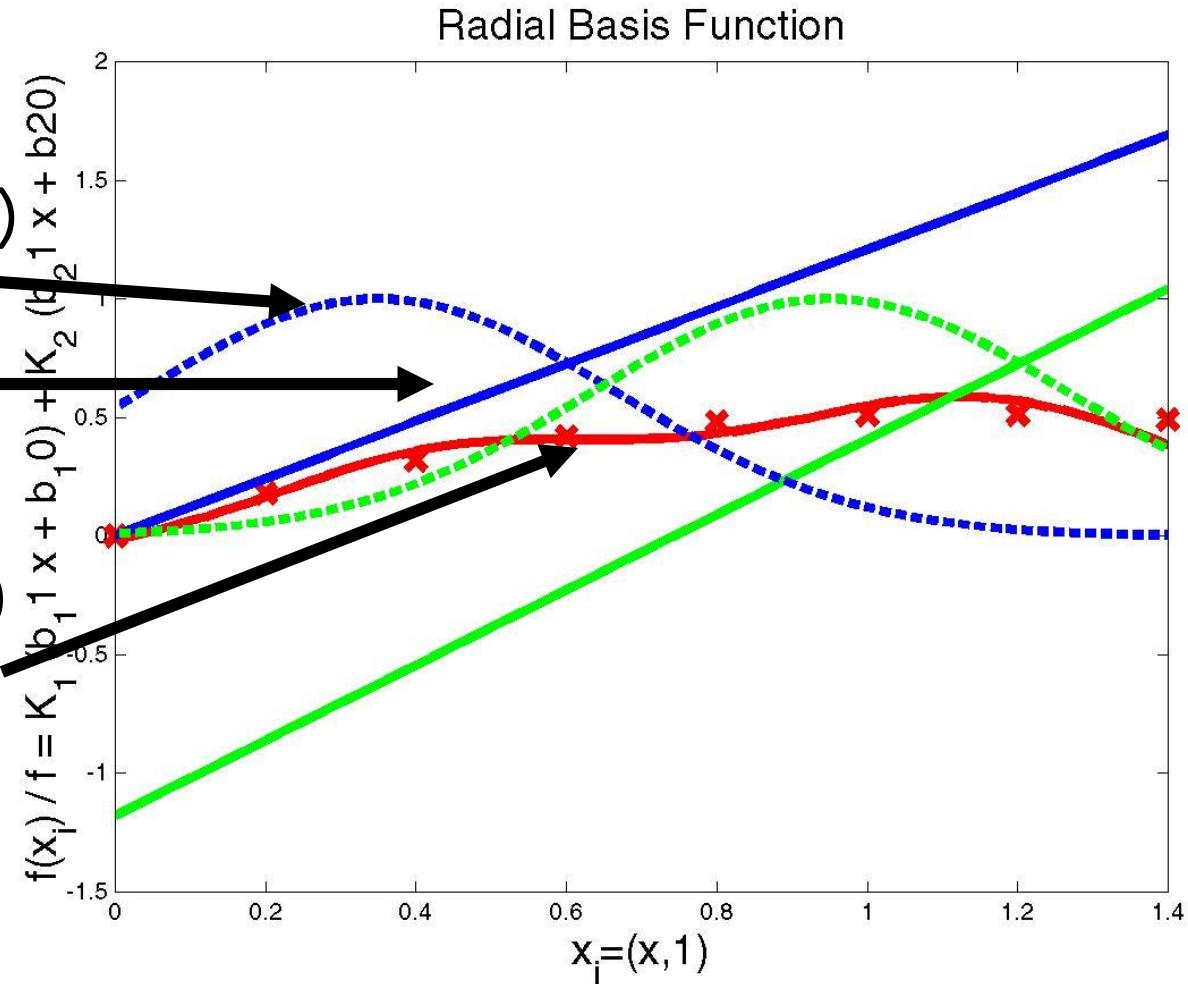
Radial Basis Network Example

$$K_1(d(x_1, x)) =$$

$$\exp(-1/2 d(x_1, x)^2 / \sigma^2)$$

$$w_1 x + w_0$$

$$\hat{f}(x) = K_1(w_1 x + w_0) + K_2(w_3 x + w_2)$$



Lazy and Eager Learning

Lazy: wait for query before generalizing

- k-nearest neighbors, weighted linear regression

Eager: generalize before seeing query

- Radial basis function networks, decision trees, back-propagation
- Eager learner must create global approximation

Lazy learner can create local approximations

If they use the same hypothesis space, lazy can represent more complex functions ($H=$ linear functions)

Confusion Matrix

- **True Positive (TP):** It refers to the number of predictions where the classifier correctly predicts the positive class as positive.
- **True Negative (TN):** It refers to the number of predictions where the classifier correctly predicts the negative class as negative.
- **False Positive (FP):** It refers to the number of predictions where the classifier incorrectly predicts the negative class as positive.
- **False Negative (FN):** It refers to the number of predictions where the classifier incorrectly predicts the positive class as negative.

Predicted class ->	C_1	$\neg C_1$
Actual class \downarrow		
C_1	True Positives (TP)	False Negatives (FN)
$\neg C_1$	False Positives (FP)	True Negatives (TN)

Classifier Evaluation Metrics: Accuracy, Error Rate

Classifier Accuracy, or recognition rate: percentage of test set tuples that are correctly classified

$$\text{Accuracy} = (\text{TP} + \text{TN})/\text{All}$$

most effective when the class distribution is relatively balanced

Classification Error/ Misclassification rate

Misclassification rate = $1 - \text{accuracy}$,
 $= (\text{FP} + \text{FN})/\text{All}$

A\P	C	$\neg C$	
C	TP	FN	P
$\neg C$	FP	TN	N
	P'	N'	All

Metrics - Basics

True positive rate (TPR) or **sensitivity** is defined as the fraction of positive examples predicted correctly by the model

$$TPR = TP / (TP + FN)$$

True negative rate (TNR) or **specificity** is defined as the fraction of negative examples predicted correctly by the model

$$TNR = TN / (TN + FP)$$

False positive rate (FPR) is the fraction of negative examples predicted as a positive class

$$FPR = FP / (TN + FP)$$

False negative rate (FNR) is the fraction of positive examples predicted as a negative class

$$FNR = FN / (TP + FN)$$

Precision, Recall & F1

- Precision determines the fraction of records that actually turns out to be positive in the group the classifier has declared as a positive class
 - The higher the precision is, the lower the number of false positive errors committed by the classifier

$$\text{Precision, } p = \frac{TP}{TP + FP}$$

- Recall (same as TPR) measures the fraction of positive examples correctly predicted by the classifier

$$\text{Recall, } r = \frac{TP}{TP + FN}$$

- Classifiers with large recall have very few positive examples misclassified as the negative class
- Harmonic between precision & recall is known as F_1 measure

$$F_1 = \frac{2}{\frac{1}{r} + \frac{1}{p}}$$

$$F_1 = \frac{2rp}{r + p} = \frac{2 \times TP}{2 \times TP + FP + FN}$$

High value of F_1 measure ensures that both precision and recall are high

Evaluating Classifier Accuracy: Holdout & Cross-Validation Methods



Holdout method

- Given data is randomly partitioned into two independent sets
 - Training set (e.g., 2/3) for model construction
 - Test set (e.g., 1/3) for accuracy estimation
- Random sampling: a variation of holdout
 - Repeat holdout k times, accuracy = avg. of the accuracies obtained

Cross-validation (k -fold, where $k = 10$ is most popular)

- Randomly partition the data into k *mutually exclusive* subsets, each approximately equal size
- At i -th iteration, use D_i as test set and others as training set
- The Accuracy of the model is the average of the accuracy of each fold.

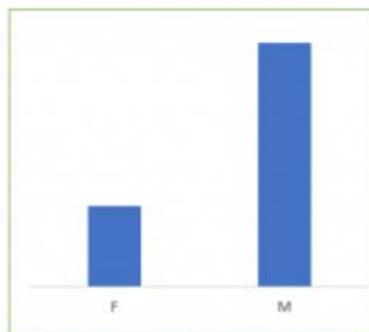
Cross Validation

k-Fold Cross Validation:

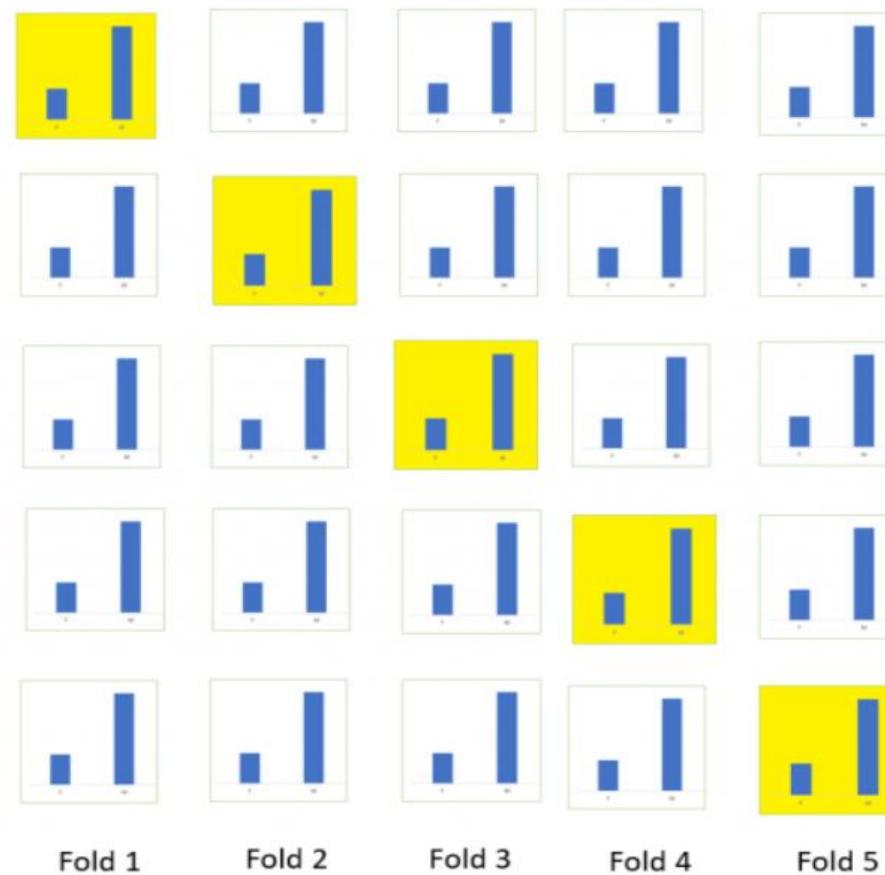


Stratified Cross Validation

Stratified K-Fold
Cross Validation
(K=5)



Class Distributions



Literature & Software

- T. Mitchell, “Machine Learning”, chapter 8, “Instance-Based Learning”
- “Locally Weighted Learning”, Christopher Atkeson, Andrew Moore, Stefan Schaal
- R. Duda et al., “Pattern recognition”, chapter 4 “Non-Parametric Techniques”

Netlab toolbox

- k-nearest neighbor classification
- Radial basis function networks

Thank You



BITS Pilani
Pilani Campus

Ensemble Learning

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in



Text Book(s)

- | | |
|----|--|
| T1 | Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition |
| T2 | Tom M. Mitchell: Machine Learning, The McGraw-Hill Companies, Inc.. |

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Tom Mitchell, Prof. Burges, Prof. Andrew Moore and many others who made their course materials freely available online.

Contents

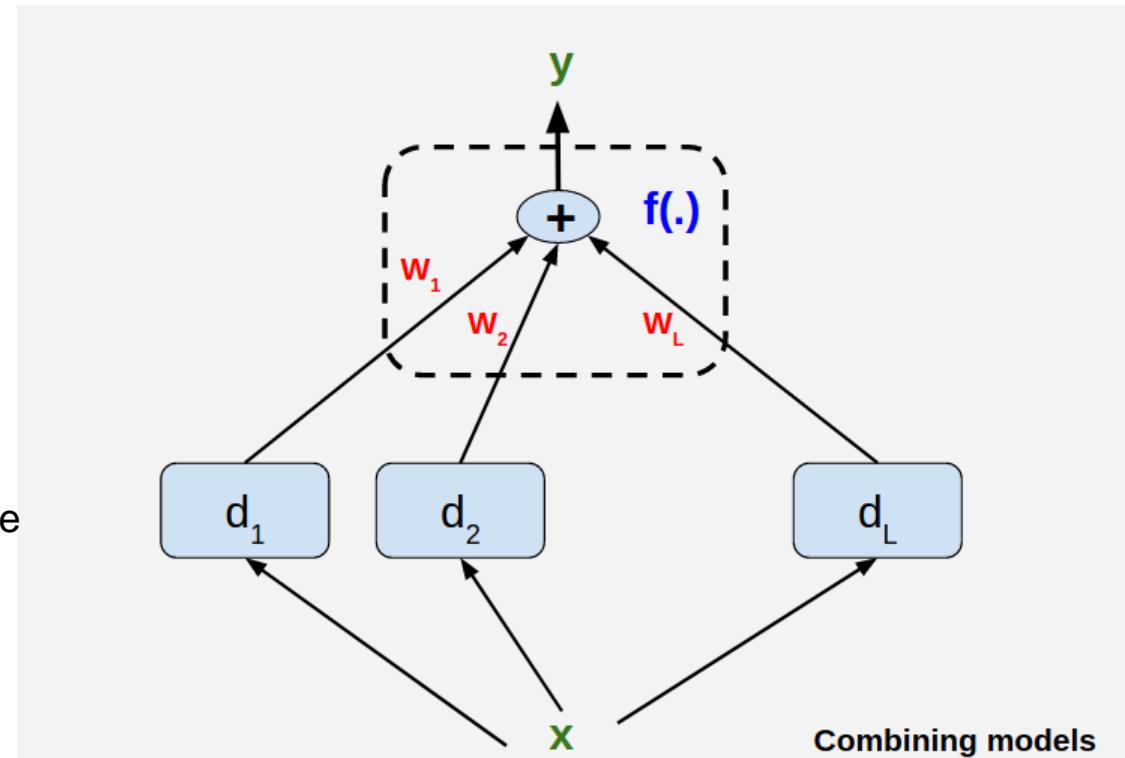
- Combining classifiers
- Bagging
- Boosting
- Random Forest Algorithm
- AdaBoost Algorithm
- Gradient Boosting

Getting Started

- No Free Lunch Theorem: There is no algorithm that is always the most accurate
- Each learning algorithm dictates a certain model that comes with a set of assumptions
 - Each algorithm converges to a different solution and fails under different circumstances
 - The best tuned learners could miss some examples and there could be other learners which works better on (may be only) those !
 - In the absence of a single expert (*a superior model*) , a committee (*combinations of models*) can do better !
 - A committee can work in many ways ...

Committee of Models

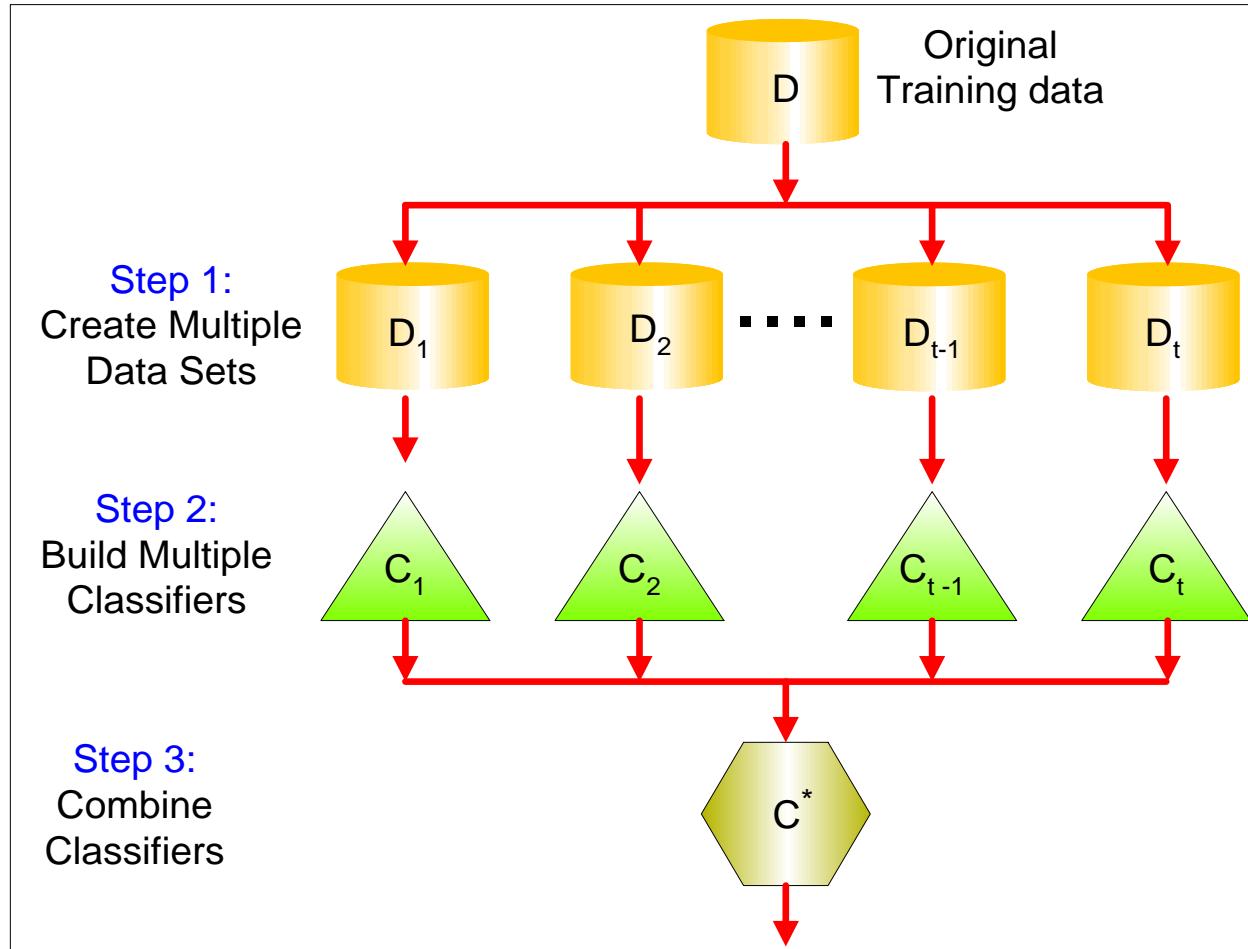
- Committee Members are base learners !
- Major challenges dealing with this committee
 - Expertise of each of the members (Does it help / not?)
 - Combining the results from the members for better performance



Ensemble Methods

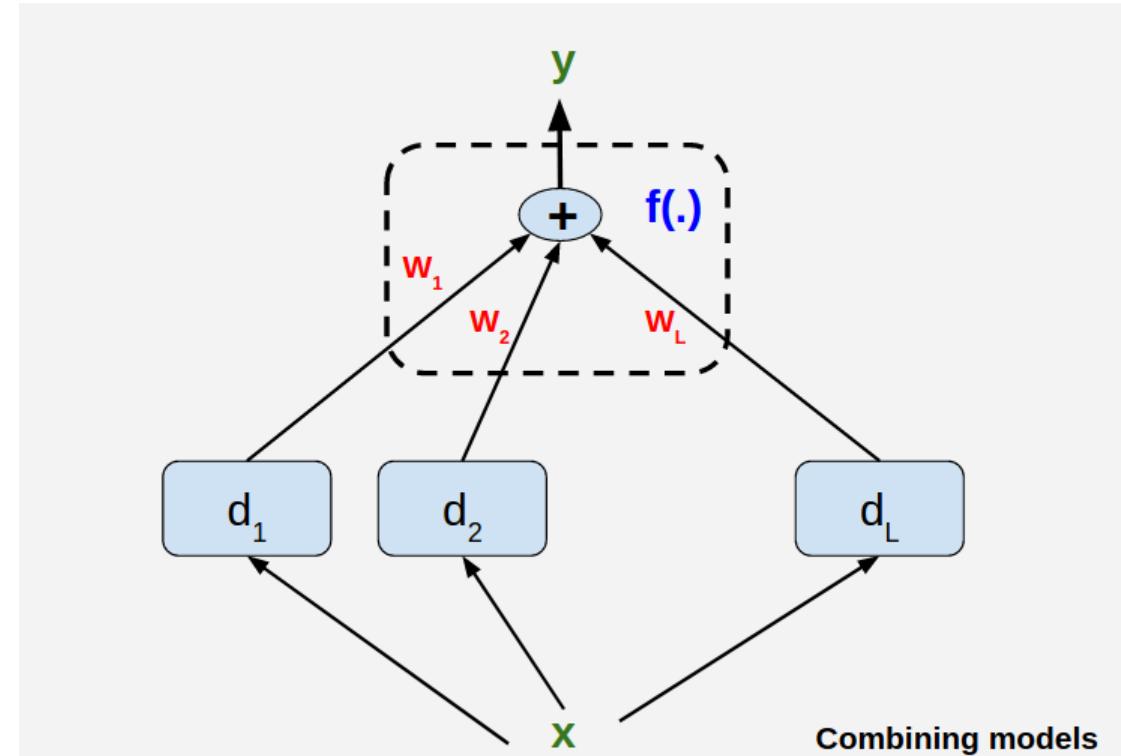
- **Ensemble methods** use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone
- Construct a set of classifiers from the training data
- Predict class label of test records by combining the predictions made by multiple classifiers
- Tend to reduce problems related to over-fitting of the training data.

General Approach



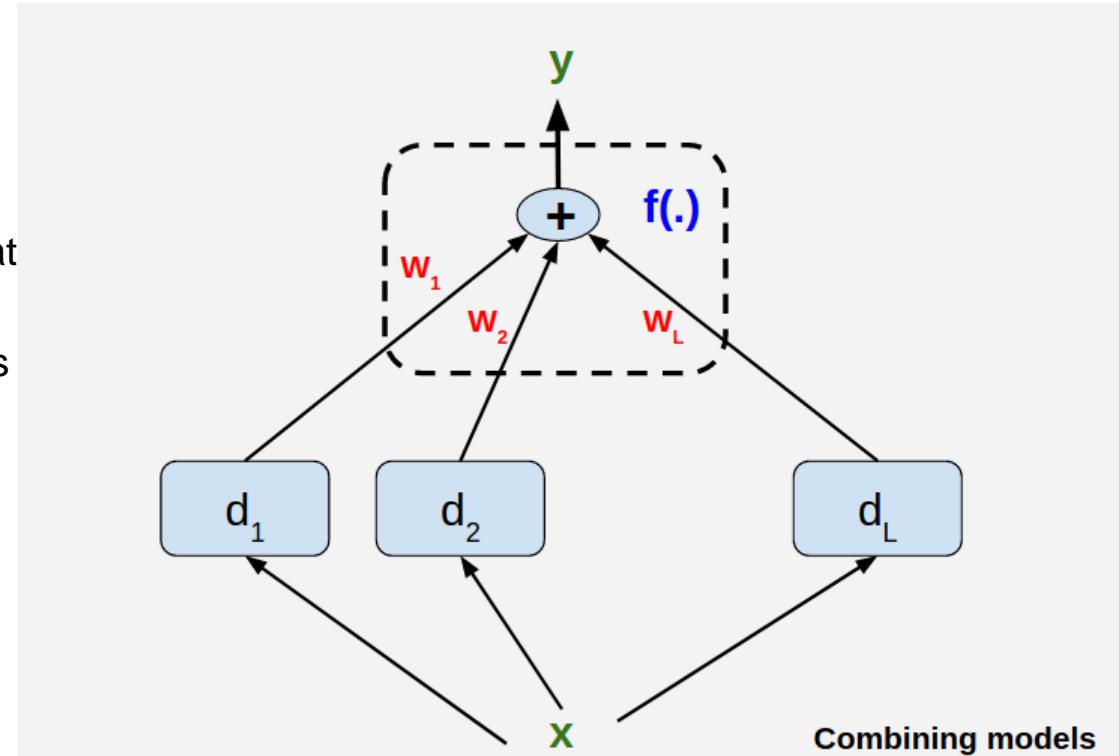
Issue 1 : On the members (Base Learners)

- It does not help if all learners are good/bad at roughly same thing
 - Need Diverse Learners



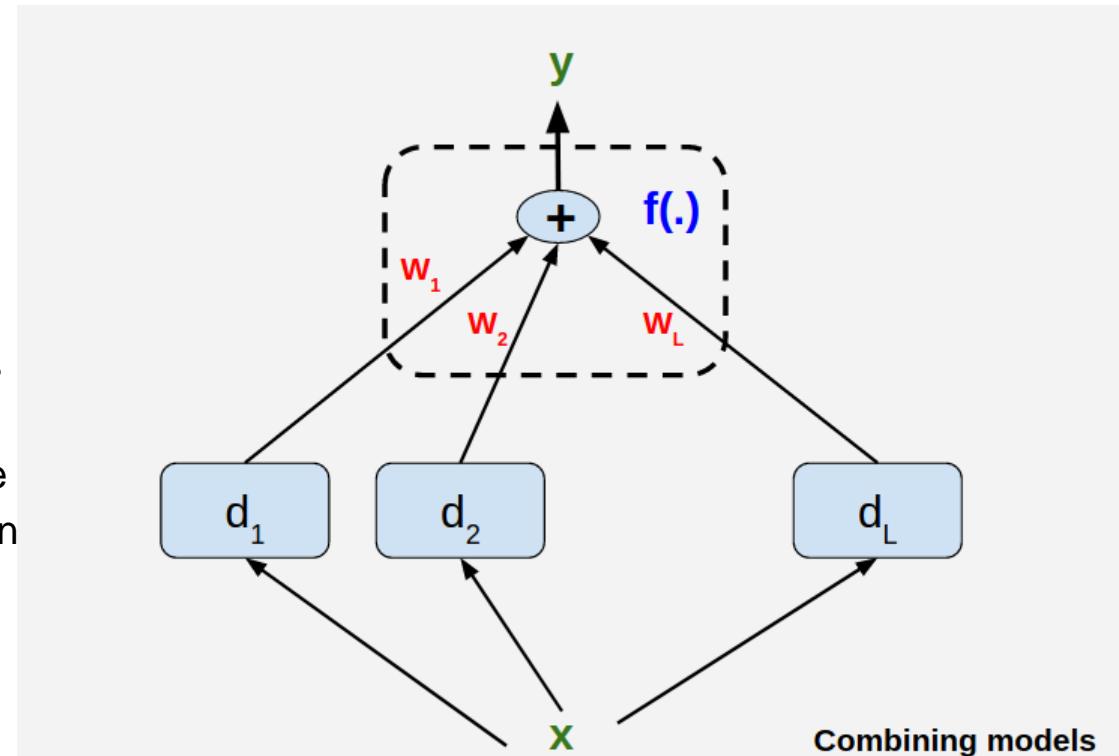
Issue 1 : On the members (Base Learners)

- Use Different Algorithms
 - Different algorithms make different assumptions
- Use Different Hyperparameters, that is ,
 - vary the structure of neural nets



Issue 1 : On the members (Base Learners)

- Different input representations
 - Uttered words + video information of speakers clips
 - image + text annotations
- Different training sets
 - Draw different random samples of data
 - Partition data in the input space and have learners specialized in those spaces (mixture of experts)



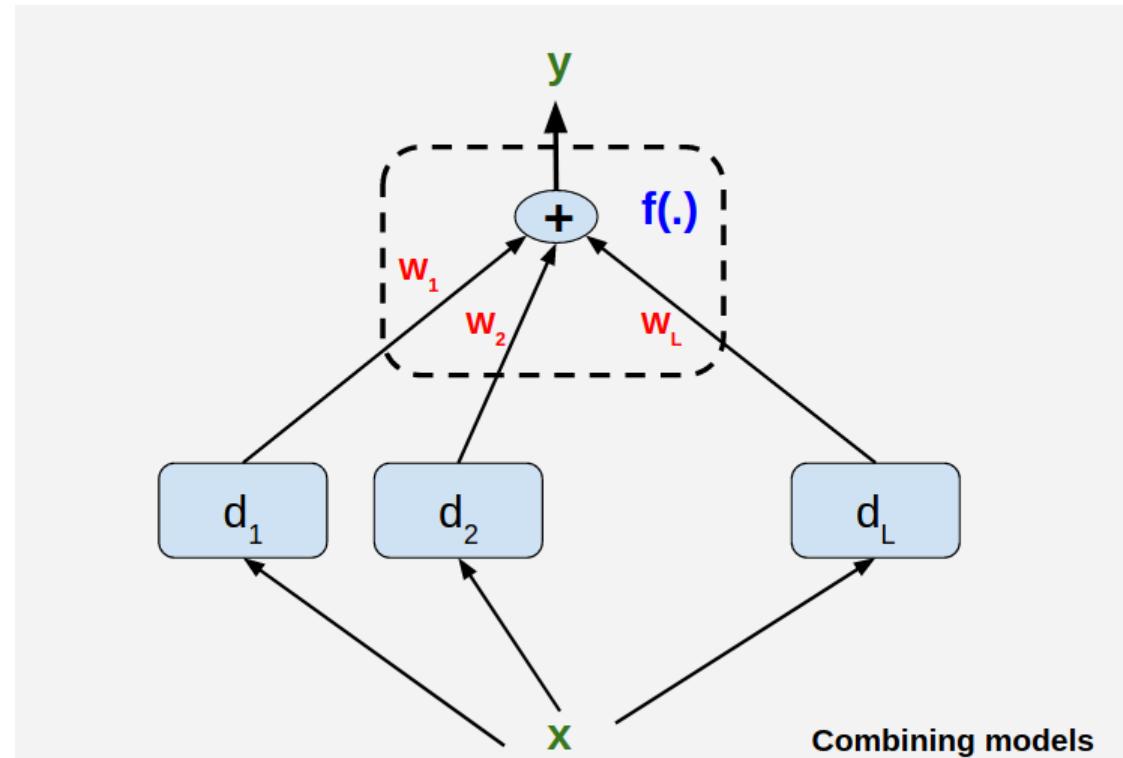
Issue -2 : Combining Results

$$y = f(d_1, d_2, \dots, d_L | \Phi)$$

A Simple Combination Scheme:

$$y = \sum_{j=1}^L w_j d_j$$

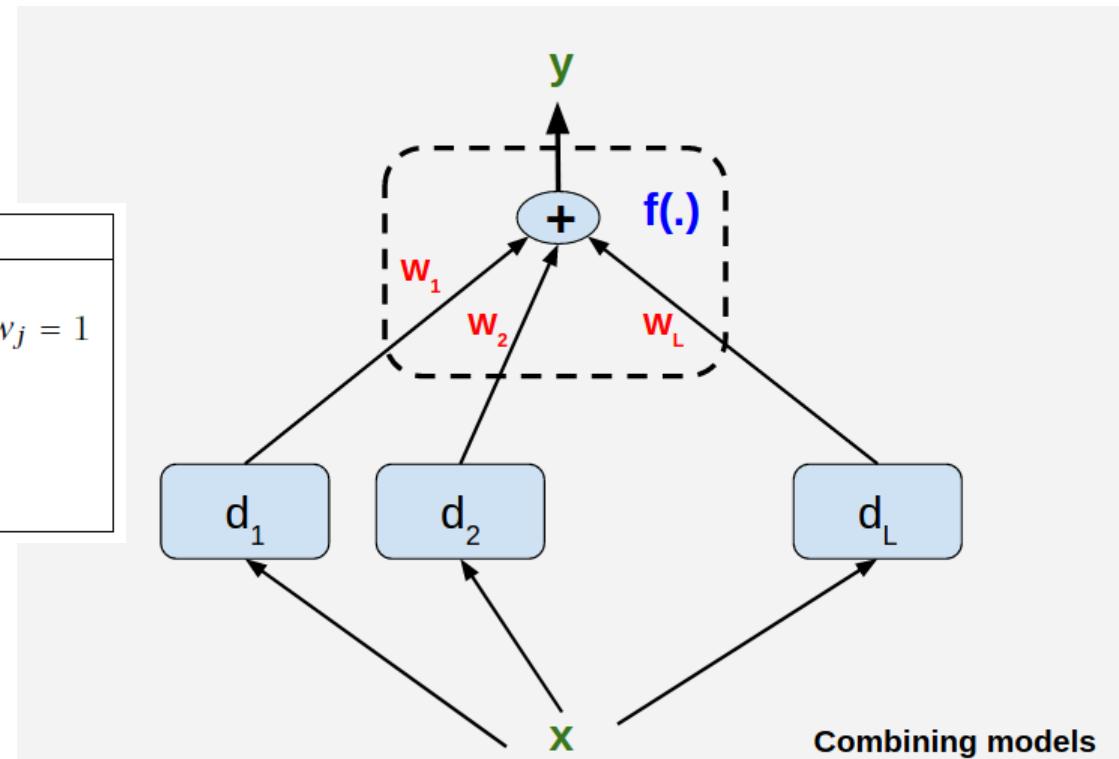
$$w_j \geq 0 \text{ and } \sum_{j=1}^L w_j = 1$$



Issue -2 : Combining Results

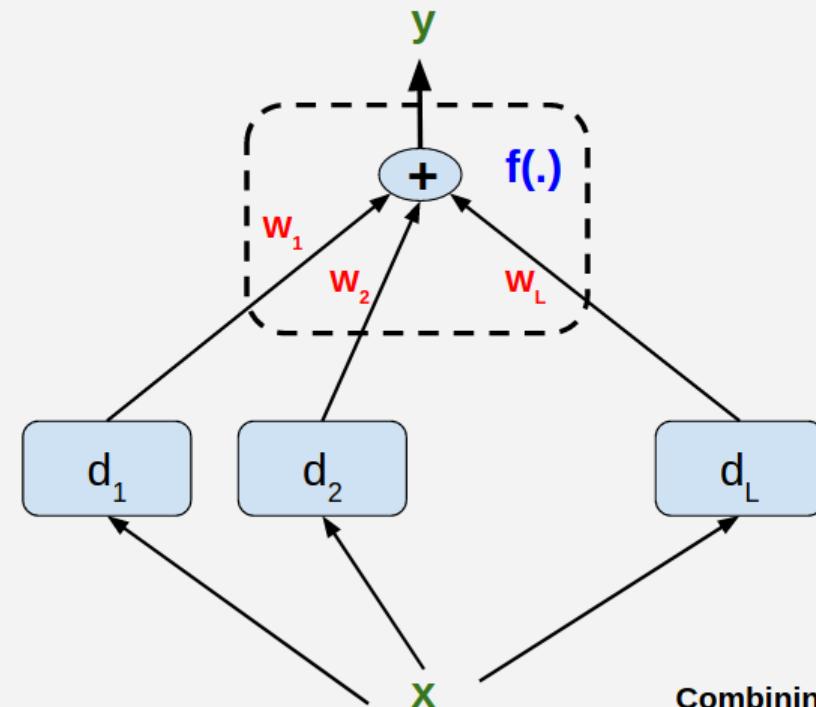
$$y = f(d_1, d_2, \dots, d_L | \Phi)$$

Rule	Fusion function $f(\cdot)$
Sum	$y_i = \frac{1}{L} \sum_{j=1}^L d_{ji}$
Weighted sum	$y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$
Median	$y_i = \text{median}_j d_{ji}$
Minimum	$y_i = \min_j d_{ji}$
Maximum	$y_i = \max_j d_{ji}$
Product	$y_i = \prod_j d_{ji}$



Issue -2 : Combining Results

	C_1	C_2	C_3
d_1	0.2	0.5	0.3
d_2	0.0	0.6	0.4
d_3	0.4	0.4	0.2
Sum	0.2	0.5	0.3
Median	0.2	0.5	0.4
Minimum	0.0	0.4	0.2
Maximum	0.4	0.6	0.4
Product	0.0	0.12	0.032

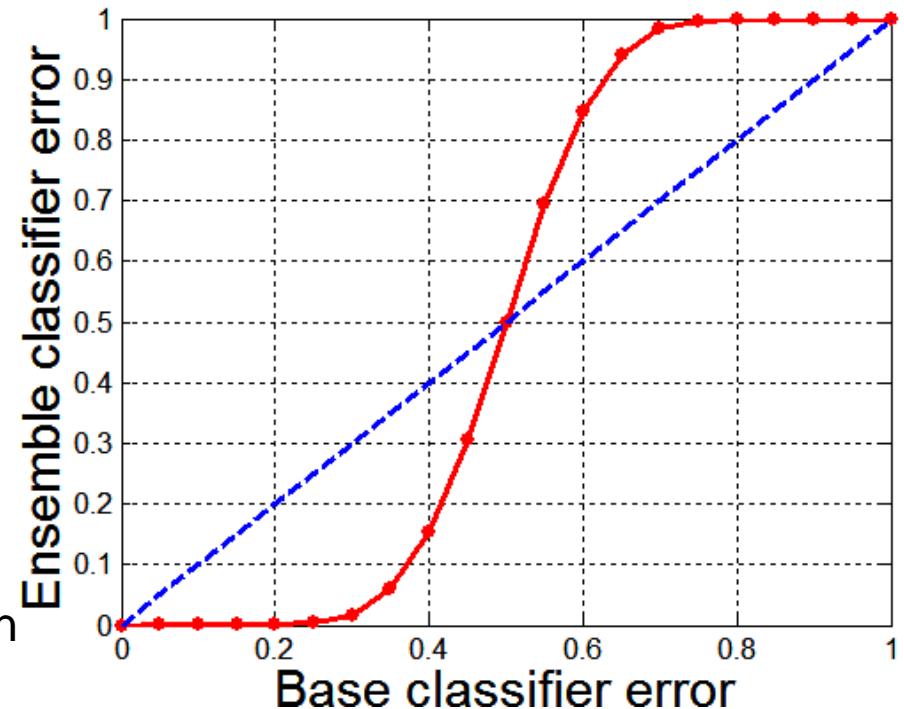


When does Ensemble work?

- Ensemble classifier performs better than the base classifiers when e is smaller than 0.5
- Necessary conditions for an ensemble classifier to perform better than a single classifier:
 - Base classifiers should be independent of each other
 - Base classifiers should do better than a classifier that performs random guessing

Why Ensemble Methods work?

- 25 base classifiers
- Each classifier has error rate, $\varepsilon = 0.35$
- If base classifiers are identical, then the ensemble will misclassify the same examples predicted incorrectly by the base classifiers depicted by dotted line
- Assume errors made by classifiers are uncorrelated
- Ensemble makes a wrong prediction only if base classifiers error is more than 0.5



Types of Ensemble Methods

Manipulate data distribution

- Example: bagging, boosting

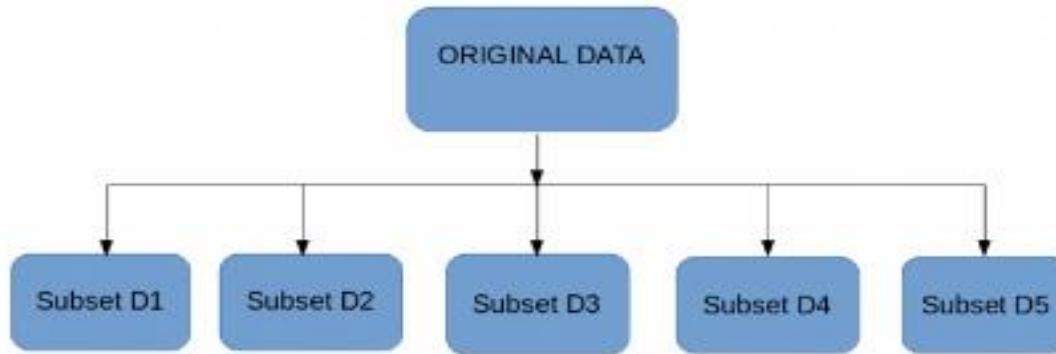
Manipulate input features

- Example: random forests

Bagging (Bootstrap Aggregating)

- Technique uses these subsets (bags) to get a fair idea of the distribution (complete set).
- The size of subsets created for bagging may be less than the original set.
- Bootstrapping is a sampling technique in which we create subsets of observations from the original dataset, **with replacement**.
- When you sample with replacement, items are independent. One item does not affect the outcome of the other. You have $1/7$ chance of choosing the first item and a $1/7$ chance of choosing the second item.
- If the two items are **dependent**, or linked to each other. When you choose the first item, you have a $1/7$ probability of picking a item. Assuming you don't replace the item, you only have six items to pick from. That gives you a $1/6$ chance of choosing a second item.

Bagging



- Multiple subsets are created from the original dataset, selecting observations with replacement.
- A base model (weak model) is created on each of these subsets.
- The models run in parallel and are independent of each other.
- The final predictions are determined by combining the predictions from all the models.

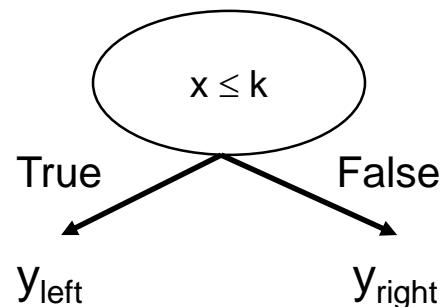
Bagging Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
 - Decision rule: $x \leq k$ versus $x > k$
 - Split point k is chosen based on entropy



Bagging Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

$x \leq 0.35 \rightarrow y = 1$
 $x > 0.35 \rightarrow y = -1$

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

$x \leq 0.01 \rightarrow y = -1$
 $x > 0.01 \rightarrow y = 1$

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1	1

$x \leq 0.35 \rightarrow y = 1$
 $x > 0.35 \rightarrow y = -1$

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

$x \leq 0.3 \rightarrow y = 1$
 $x > 0.3 \rightarrow y = -1$

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

$x \leq 0.35 \rightarrow y = 1$
 $x > 0.35 \rightarrow y = -1$

Bagging Example

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

$$x \leq 0.75 \rightarrow y = -1$$
$$x > 0.75 \rightarrow y = 1$$

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

$$x \leq 0.75 \rightarrow y = -1$$
$$x > 0.75 \rightarrow y = 1$$

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

$$x \leq 0.75 \rightarrow y = -1$$
$$x > 0.75 \rightarrow y = 1$$

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

$$x \leq 0.75 \rightarrow y = -1$$
$$x > 0.75 \rightarrow y = 1$$

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

$$x \leq 0.05 \rightarrow y = 1$$
$$x > 0.05 \rightarrow y = 1$$

Bagging Example

- Summary of Training sets:

Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

Bagging Example

- Assume test set is the same as the original data
- Use majority vote to determine class of ensemble classifier

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted
Class

Bagging Algorithm

Algorithm 5.6 Bagging Algorithm

- 1: Let k be the number of bootstrap samples.
 - 2: for $i = 1$ to k do
 - 3: Create a bootstrap sample of size n , D_i .
 - 4: Train a base classifier C_i on the bootstrap sample D_i .
 - 5: end for
 - 6: $C^*(x) = \arg \max_y \sum_i \delta(C_i(x) = y)$, $\{\delta(\cdot) = 1 \text{ if its argument is true, and } 0 \text{ otherwise.}\}$
-

Boosting

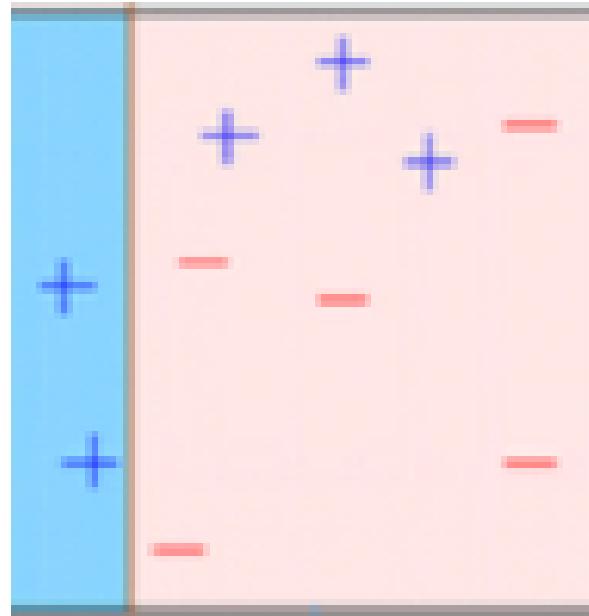
- What if a data point is incorrectly predicted by the first model, and then the next (probably all models), will combining the predictions provide better results? Such situations are taken care of by boosting.
- Boosting is a sequential process, where each subsequent model attempts to correct the errors of the previous model.
- The succeeding models are dependent on the previous model.

Boosting

- An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records
 - Initially, all N records are assigned equal weights
 - Unlike bagging, weights may change at the end of each boosting round

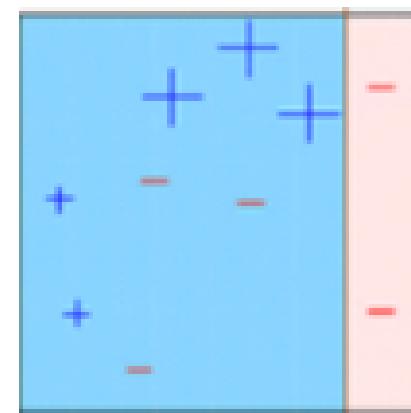
Boosting

- A subset is created from the original dataset.
- Initially, all data points are given equal weights.
- A base model is created on this subset.
- This model is used to make predictions on the whole dataset.



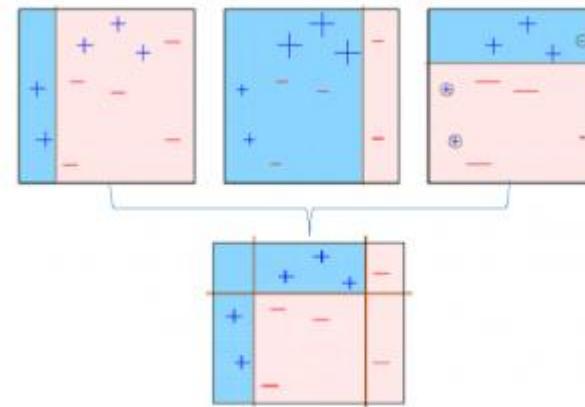
Boosting

- Errors are calculated using the actual values and predicted values.
- The observations which are incorrectly predicted, are given higher weights. (Here, the three misclassified blue-plus points will be given higher weights)
- Another model is created and predictions are made on the dataset. (This model tries to correct the errors from the previous model)



Boosting

- Similarly, multiple models are created, each correcting the errors of the previous model.
- The final model (strong learner) is the weighted mean of all the models (weak learners).



- Individual models would not perform well on the entire dataset, but they work well for some part of the dataset. Thus, each model actually boosts the performance of the ensemble.

Algorithms based on Bagging and Boosting

Bagging algorithms:

- Random forest

Boosting algorithms:

- AdaBoost

Random Forest

- Random Forest is ensemble machine learning algorithm that follows the bagging technique.
- The base estimators in random forest are decision trees.
- Random forest randomly selects a set of features which are used to decide the best split at each node of the decision tree.

Random Forest

- Random subsets are created from the original dataset (bootstrapping).
- At each node in the decision tree, only a random set of features are considered to decide the best split.
- A decision tree model is fitted on each of the subsets.
- The final prediction is calculated by averaging the predictions from all decision trees.

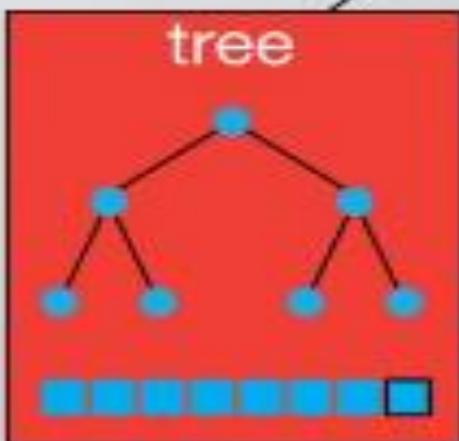
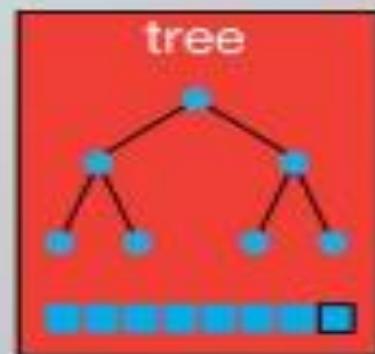
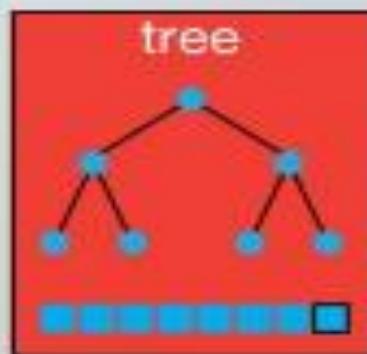
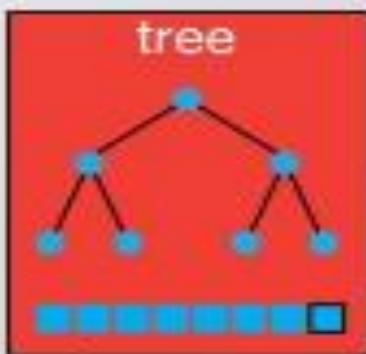
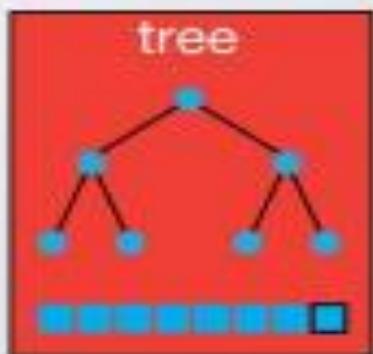
All Data

random subset

random subset

random subset

random subset



At each node:
choose some ballsubset of variables at random
find a variable (and a value for that variable) which optimizes the split

Advantages of Random Forest

- Algorithm can solve both type of problems i.e. classification and regression
- Power to handle large data set with higher dimensionality.
- It can handle thousands of input variables and identify most significant variables so it is considered as one of the dimensionality reduction methods.
- Model outputs **Importance of variable**, which can be a very handy feature (on some random data set).

Disadvantages of Random Forest

- May over-fit data sets that are particularly noisy.
- Random Forest can feel like a black box approach for statistical modelers – you have very little control on what the model does. You can at best – try different parameters and random seeds!

AdaBoost

- Adaptive boosting or AdaBoost is one of the simplest boosting algorithms. Usually, decision trees are used for modelling. Multiple sequential models are created, each correcting the errors from the last model.
- AdaBoost assigns weights to the observations which are incorrectly predicted and the subsequent model works to predict these values correctly.

AdaBoost Algorithm

- Initially, all observations (n) in the dataset are given equal weights ($1/n$).
- A model is built on a subset of data.
- Using this model, predictions are made on the whole dataset.
- Errors are calculated by comparing the predictions and actual values.
- While creating the next model, higher weights are given to the data points which were predicted incorrectly.

Adaboost Algorithm

- Weights can be determined using the error value. For instance, higher the error more is the weight assigned to the observation.
- This process is repeated until the error function does not change, or the maximum limit of the number of estimators is reached.

AdaBoost



- Base classifiers C_i : C_1, C_2, \dots, C_T
- Error rate:
 - N input samples

$$\varepsilon_i = \frac{1}{N} \sum_{j=1}^N w_j \delta(C_i(x_j) \neq y_j)$$

- Importance of a classifier:

$$\alpha_i = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_i}{\varepsilon_i} \right)$$

https://en.wikipedia.org/wiki/AdaBoost#Choosing_at

AdaBoost: Weight Update

Weight Update:

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases} \quad \text{Eqn:5.88}$$

where Z_j is the normalization factor

$$C^*(x) = \arg \max_y \sum_{j=1}^T \alpha_j \delta(C_j(x) = y)$$

- Reduce weight if correctly classified else increase
- If any intermediate rounds produce error rate higher than 50%, the weights are reverted back to $1/n$ and the resampling procedure is repeated

AdaBoost Algorithm

Algorithm 5.7 AdaBoost Algorithm

```

1:  $w = \{w_j = 1/n \mid j = 1, 2, \dots, n\}$ . {Initialize the weights for all  $n$  instances.}
2: Let  $k$  be the number of boosting rounds.
3: for  $i = 1$  to  $k$  do
4:   Create training set  $D_i$  by sampling (with replacement) from  $D$  according to  $w$ .
5:   Train a base classifier  $C_i$  on  $D_i$ .
6:   Apply  $C_i$  to all instances in the original training set,  $D$ .
7:    $\epsilon_i = \frac{1}{n} [\sum_j w_j \delta(C_i(x_j) \neq y_j)]$  {Calculate the weighted error}
8:   if  $\epsilon_i > 0.5$  then
9:      $w = \{w_j = 1/n \mid j = 1, 2, \dots, n\}$ . {Reset the weights for all  $n$  instances.}
10:    Go back to Step 4.
11:   end if
12:    $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$ .
13:   Update the weight of each instance according to equation (5.88).
14: end for
15:  $C^*(x) = \arg \max_y \sum_{j=1}^T \alpha_j \delta(C_j(x) = y)$ .

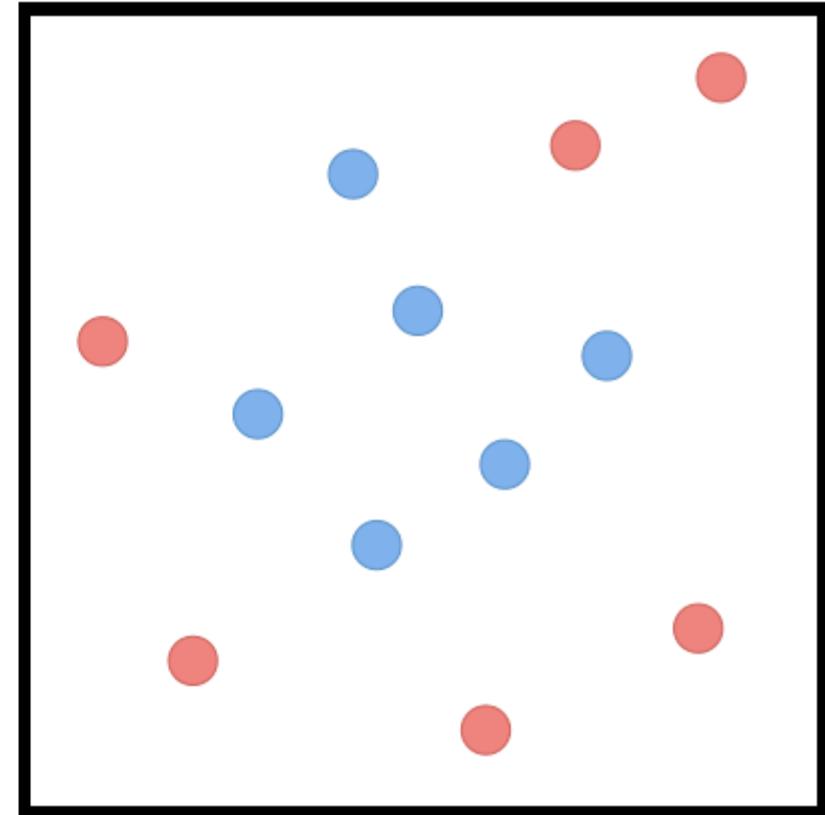
```

AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$


```



- Size of point represents the instance's weight

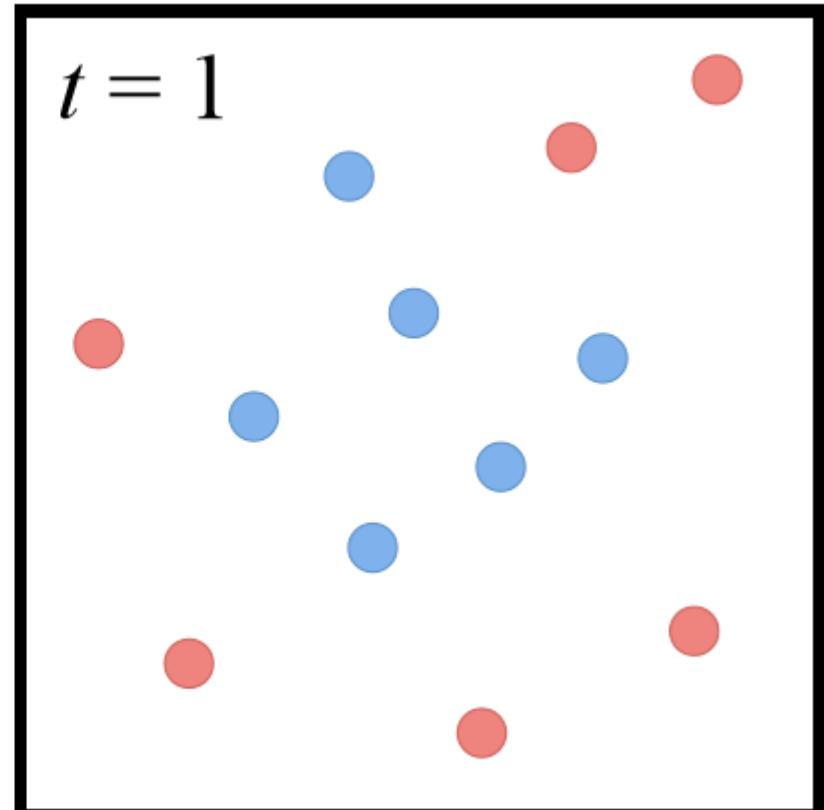
α in earlier slide same as β = weight of class

AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
```

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

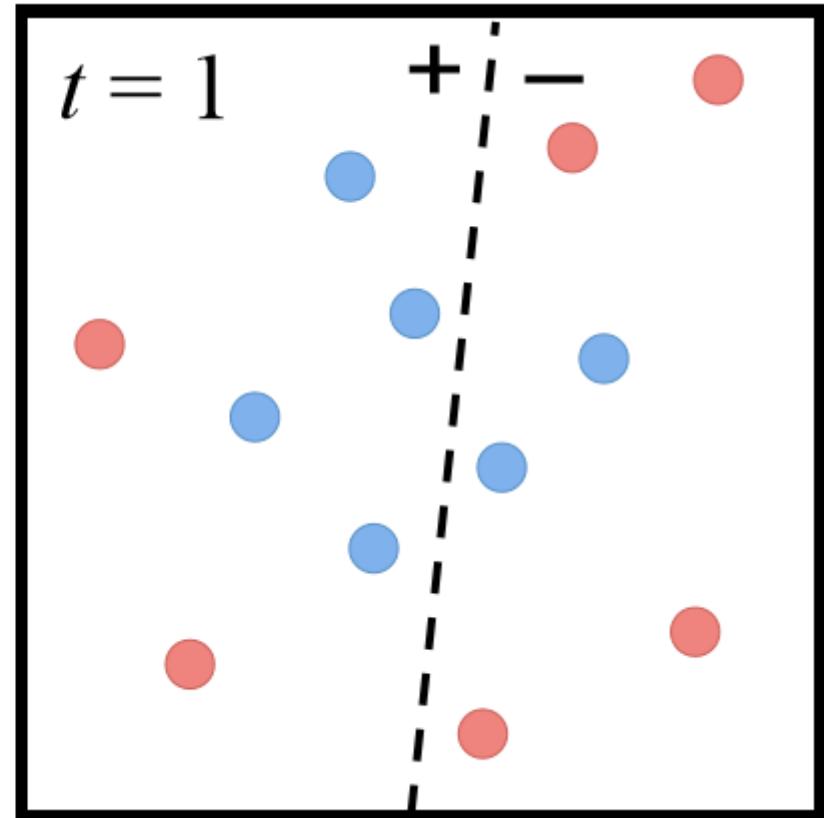


AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
```

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

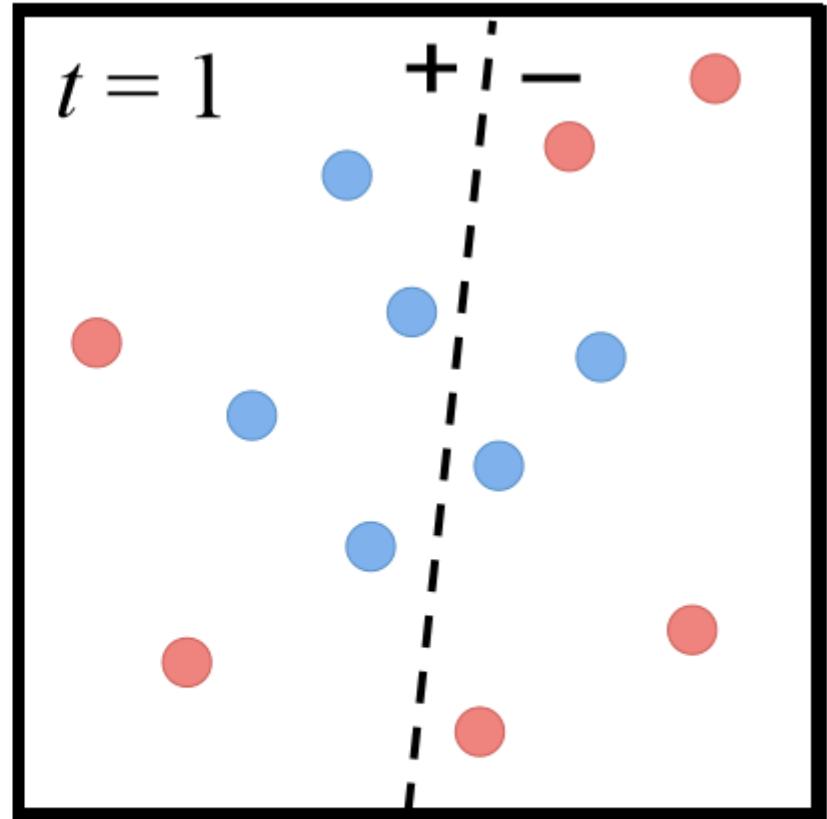


AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp (-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
```

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



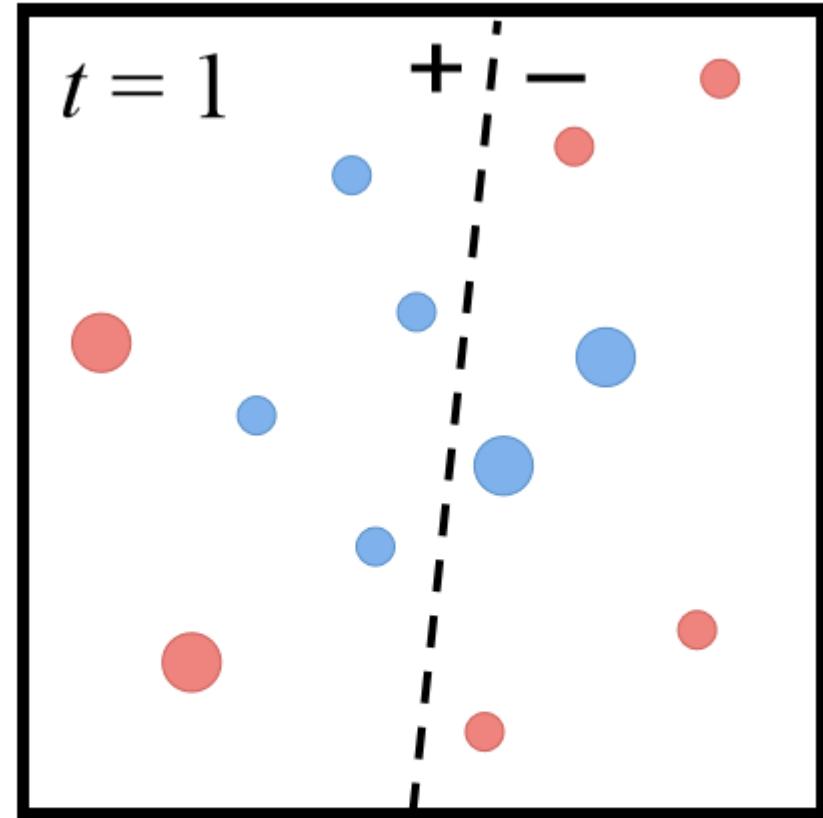
- β_t measures the importance of h_t
- If $\epsilon_t \leq 0.5$, then $\beta_t \geq 0$ (β_t grows as ϵ_t gets smaller)

AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
```

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



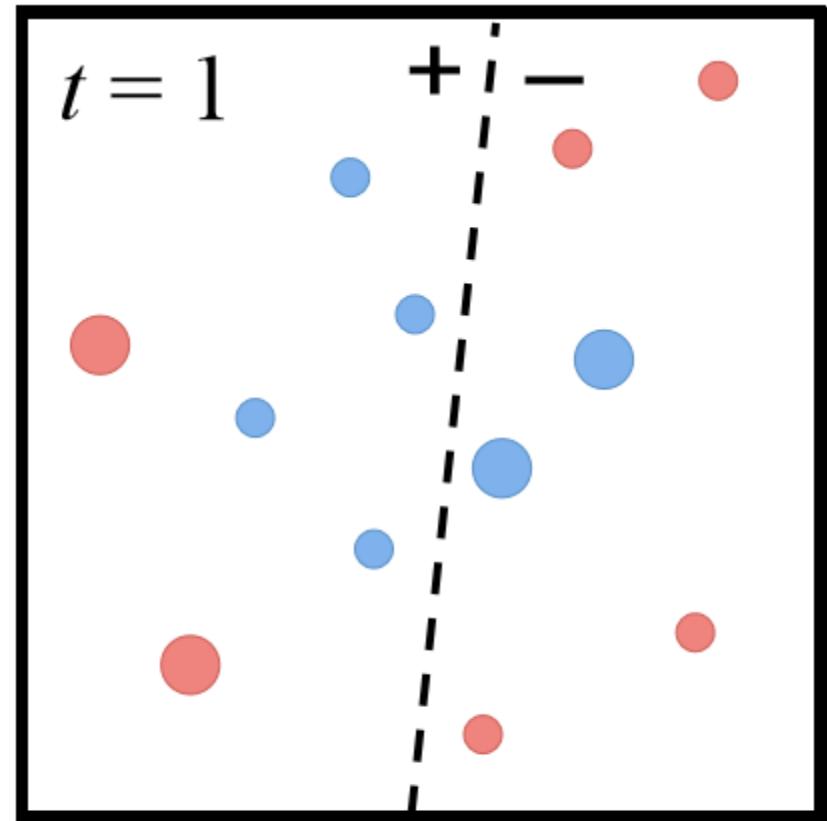
- Weights of correct predictions are multiplied by $e^{-\beta_t} \leq 1$
- Weights of incorrect predictions are multiplied by $e^{\beta_t} \geq 1$

AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
```

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



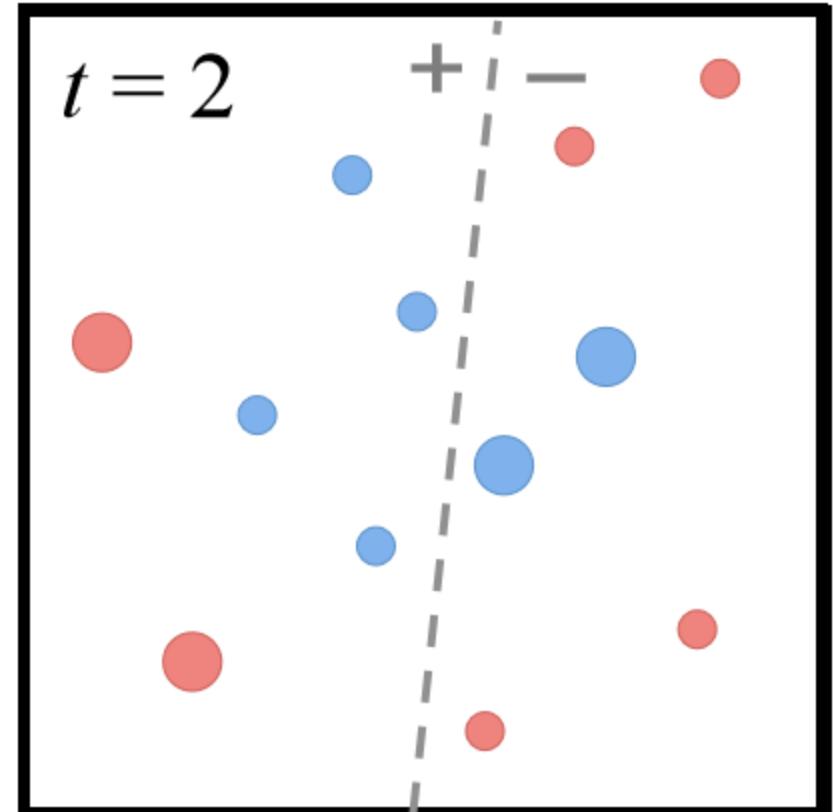
Disclaimer: Note that resized points in the illustration above are not necessarily to scale with β_t

AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
```

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

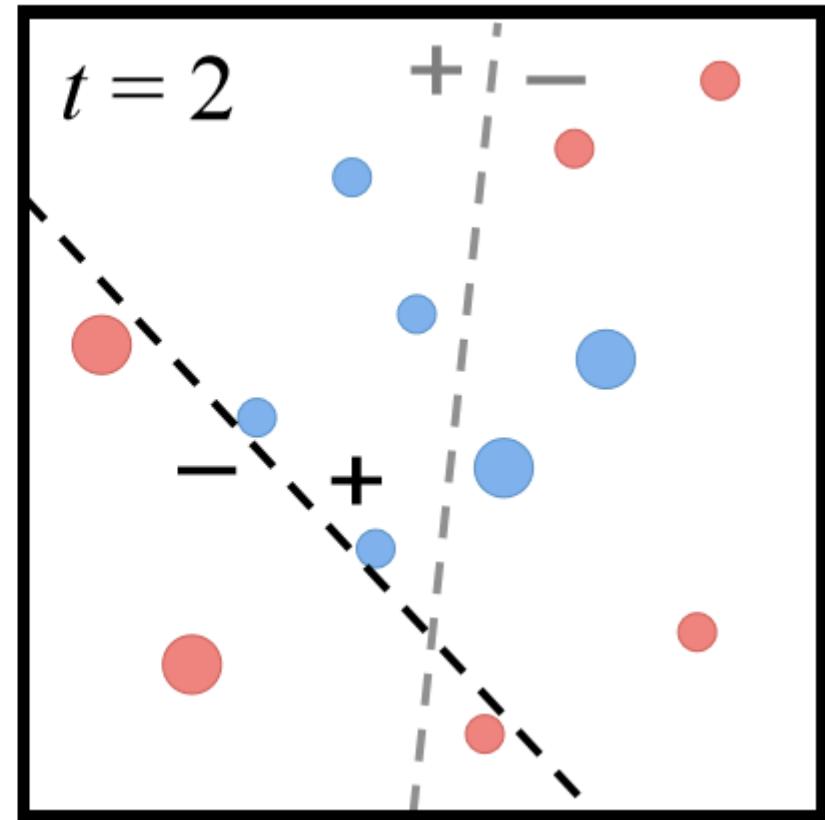


AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
   $H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$ 

```

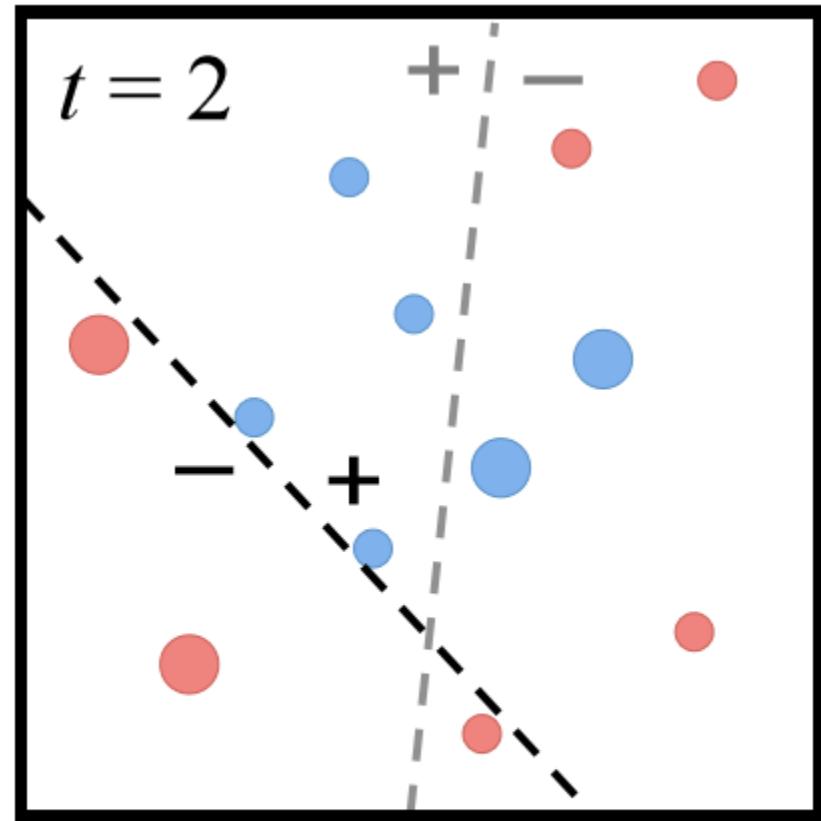


AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
```

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



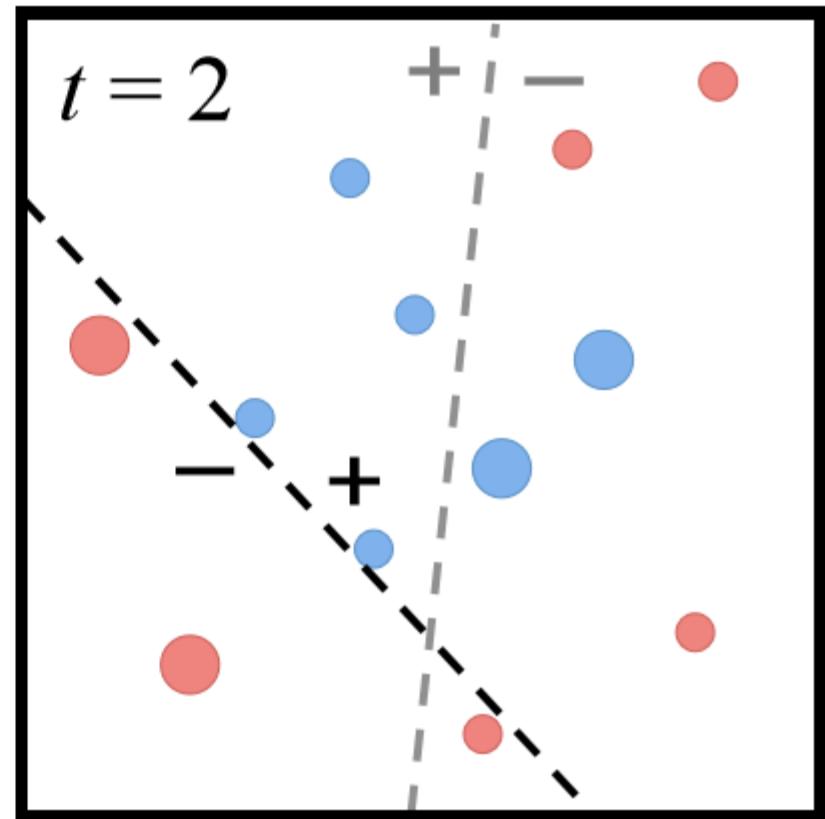
- β_t measures the importance of h_t
- If $\epsilon_t \leq 0.5$, then $\beta_t \geq 0$ (β_t grows as ϵ_t gets smaller)

AdaBoost Algorithm

- 1: Initialize a vector of n uniform weights \mathbf{w}_1
- 2: **for** $t = 1, \dots, T$
- 3: Train model h_t on X, y with weights \mathbf{w}_t
- 4: Compute the weighted training error of h_t
- 5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$$
- 7: Normalize \mathbf{w}_{t+1} to be a distribution
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



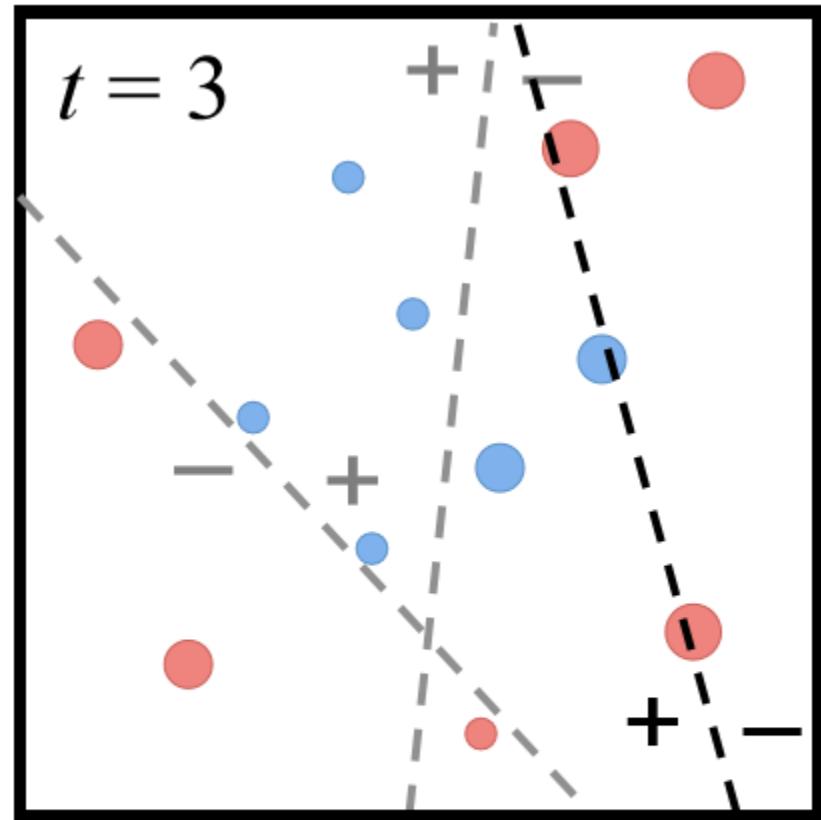
- Weights of correct predictions are multiplied by $e^{-\beta_t} \leq 1$
- Weights of incorrect predictions are multiplied by $e^{\beta_t} \geq 1$

AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
   $H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$ 

```



- β_t measures the importance of h_t
- If $\epsilon_t \leq 0.5$, then $\beta_t \geq 0$ (β_t grows as ϵ_t gets smaller)

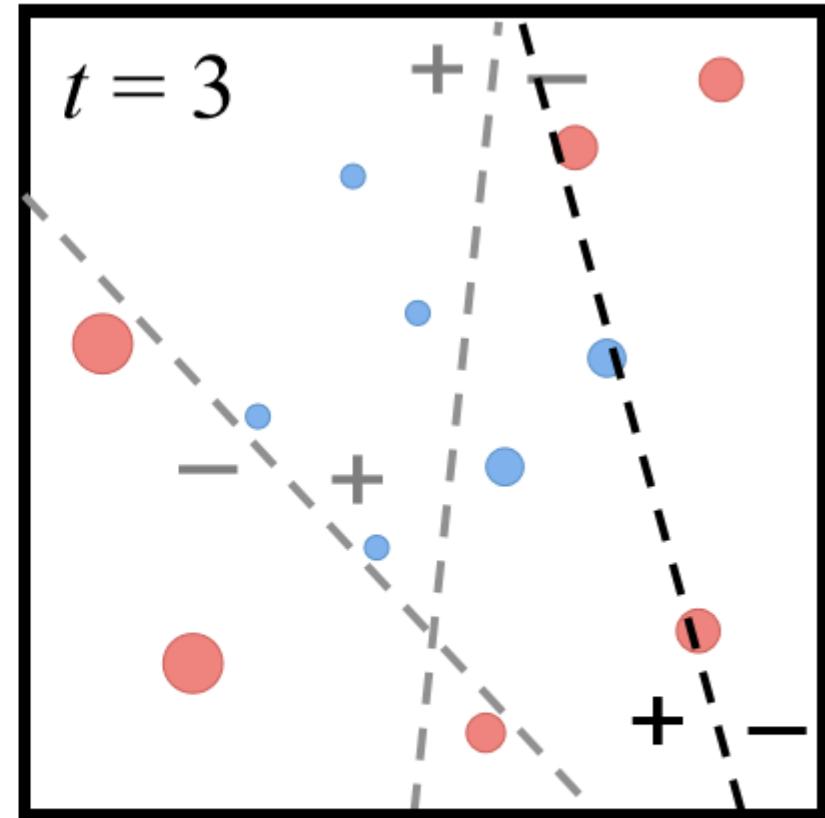
AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis

```

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



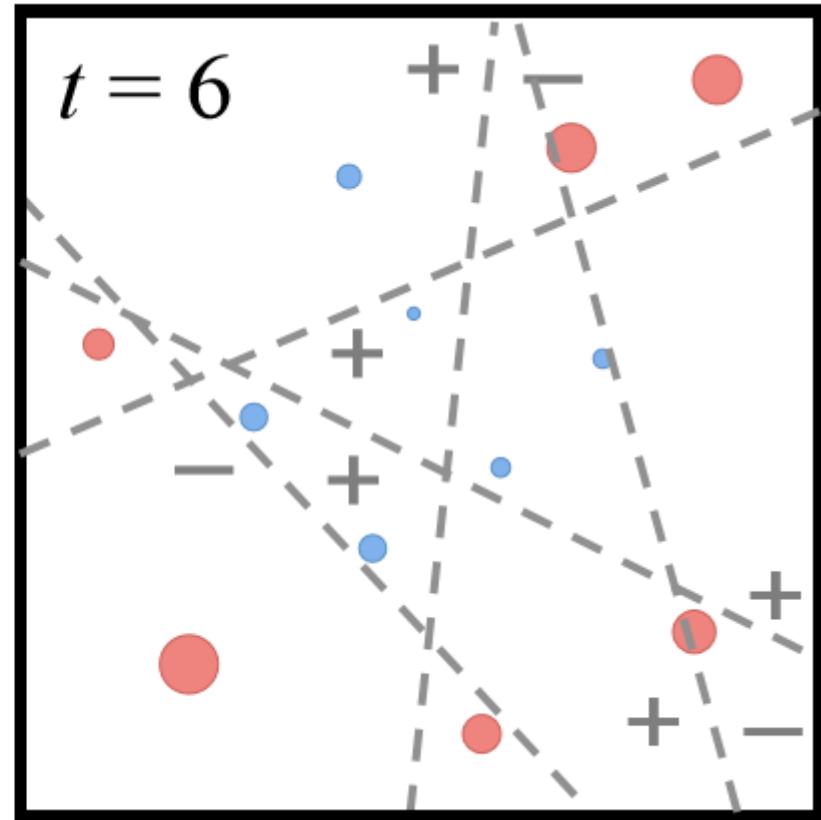
- Weights of correct predictions are multiplied by $e^{-\beta_t} \leq 1$
- Weights of incorrect predictions are multiplied by $e^{\beta_t} \geq 1$

AdaBoost Algorithm

```

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$ 
2: for  $t = 1, \dots, T$ 
3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$ 
4:   Compute the weighted training error of  $h_t$ 
5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
6:   Update all instance weights:
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$ 
7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
8: end for
9: Return the hypothesis
  
```

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



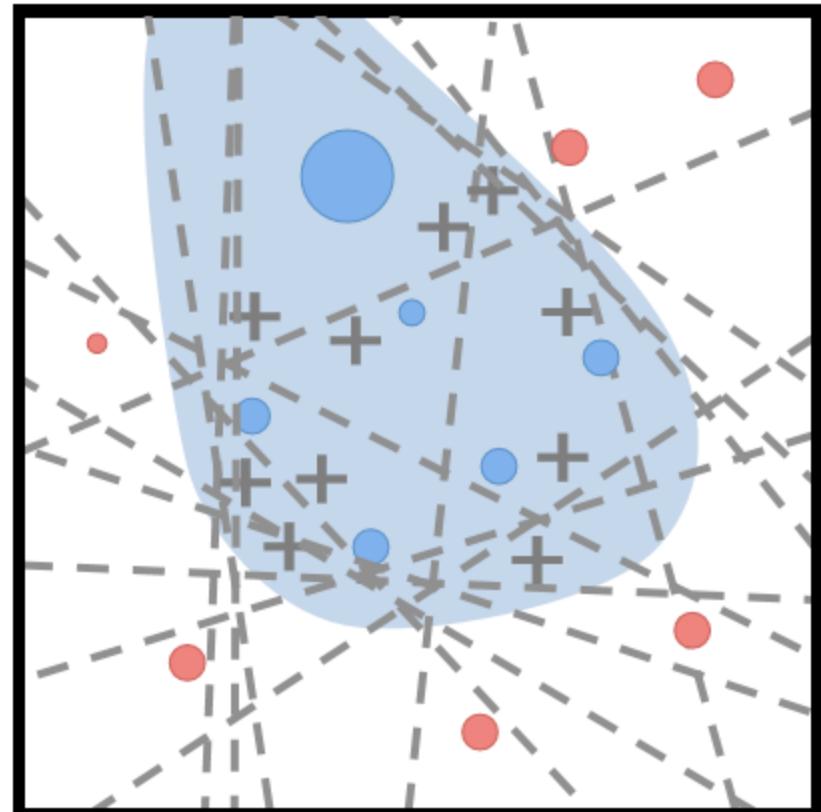
AdaBoost Algorithm

- 1: Initialize a vector of n uniform weights \mathbf{w}_1
- 2: **for** $t = 1, \dots, T$
- 3: Train model h_t on X, y with weights \mathbf{w}_t
- 4: Compute the weighted training error of h_t
- 5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
- 6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$$
- 7: Normalize \mathbf{w}_{t+1} to be a distribution
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

$t = T$



- Final model is a **weighted combination** of members
 - Each member weighted by its importance

AdaBoost Algorithm

INPUT: training data $X, y = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$,
the number of iterations T

1: Initialize a vector of n uniform weights $\mathbf{w}_1 = [\frac{1}{n}, \dots, \frac{1}{n}]$

2: **for** $t = 1, \dots, T$

3: Train model h_t on X, y with instance weights \mathbf{w}_t

4: Compute the weighted training error rate of h_t :

$$\epsilon_t = \sum_{i:y_i \neq h_t(\mathbf{x}_i)} w_{t,i}$$

5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$

6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i)) \quad \forall i = 1, \dots, n$$

7: Normalize \mathbf{w}_{t+1} to be a distribution:

$$w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^n w_{t+1,j}} \quad \forall i = 1, \dots, n$$

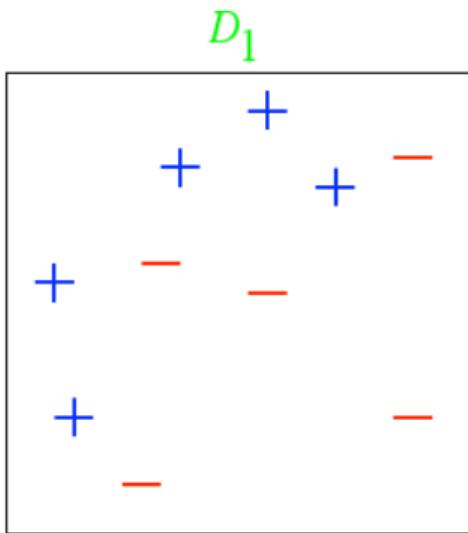
8: **end for**

9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

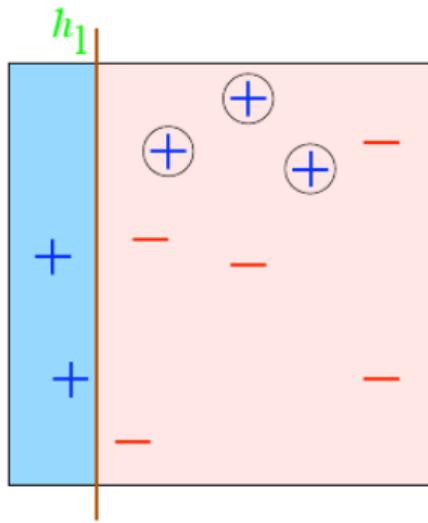
Member classifier with less error are given more weight in final ensemble hypothesis. Final prediction is a weighted combination of each members prediction

Example



From, Léon Bottou

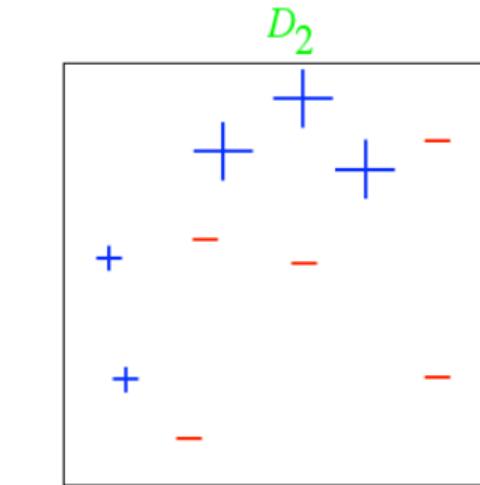
Example



$$\epsilon_1 = 0.30$$

$$\alpha_1 = 0.42$$

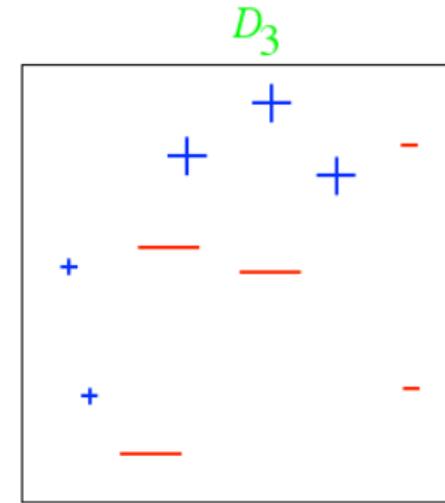
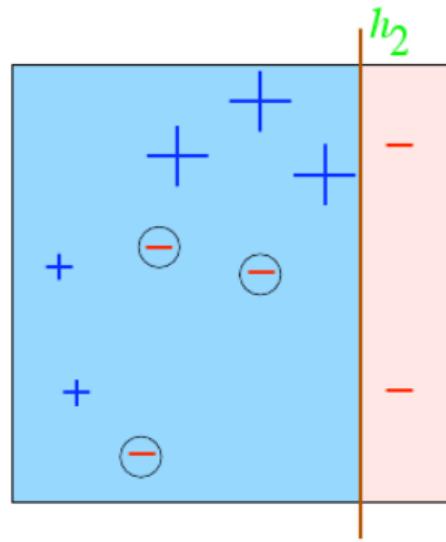
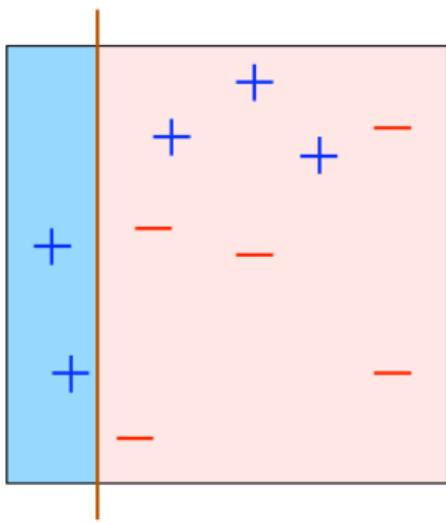
From, Léon Bottou



$$\epsilon_i = \frac{1}{N} \sum_{j=1}^N w_j \delta(C_i(x_j) \neq y_j)$$

$$\alpha_i = \frac{1}{2} \ln \left(\frac{1 - \epsilon_i}{\epsilon_i} \right)$$

Example



$$\varepsilon_2 = 0.21$$

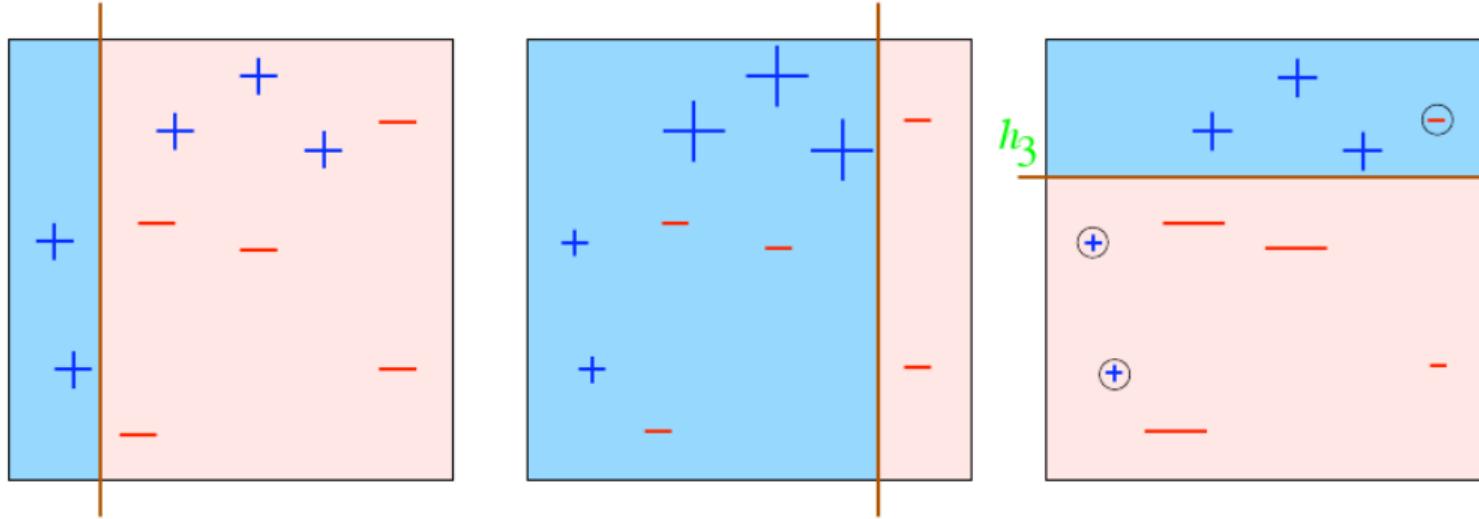
$$\alpha_2 = 0.65$$

$$\varepsilon_i = \frac{1}{N} \sum_{j=1}^N w_j \delta(C_i(x_j) \neq y_j)$$

$$\alpha_i = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_i}{\varepsilon_i} \right)$$

From, Léon Bottou

Example



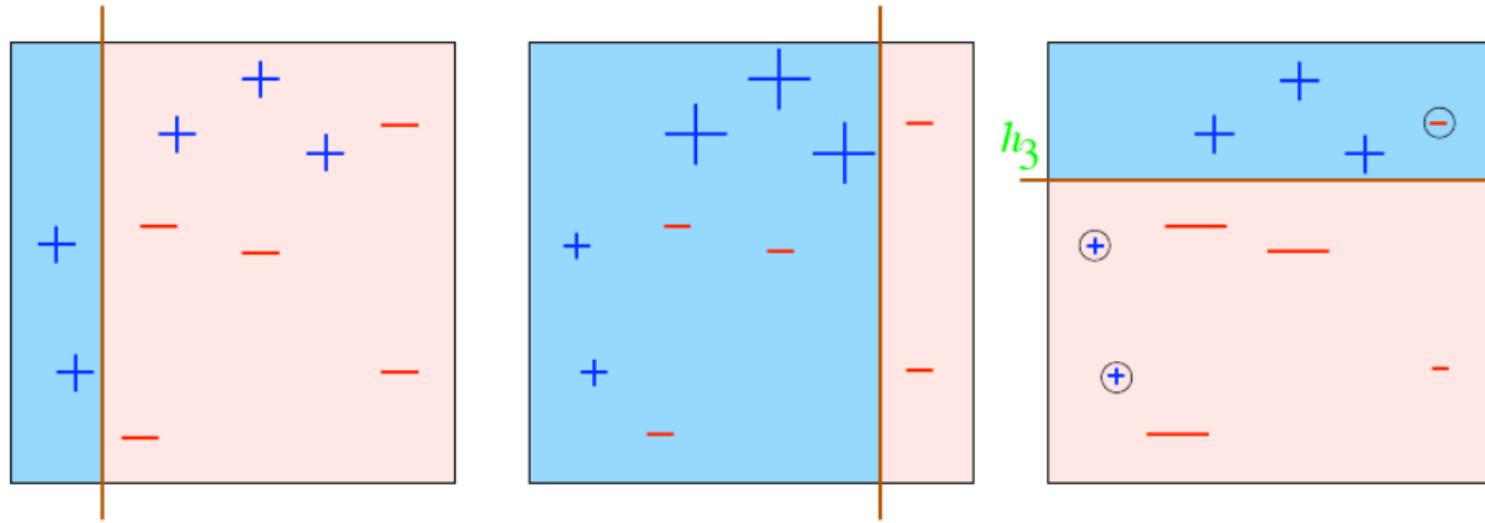
$$\varepsilon_3 = 0.14$$

$$\alpha_3 = 0.92$$

From, Léon Bottou

$$\alpha_i = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_i}{\varepsilon_i} \right)$$

Example



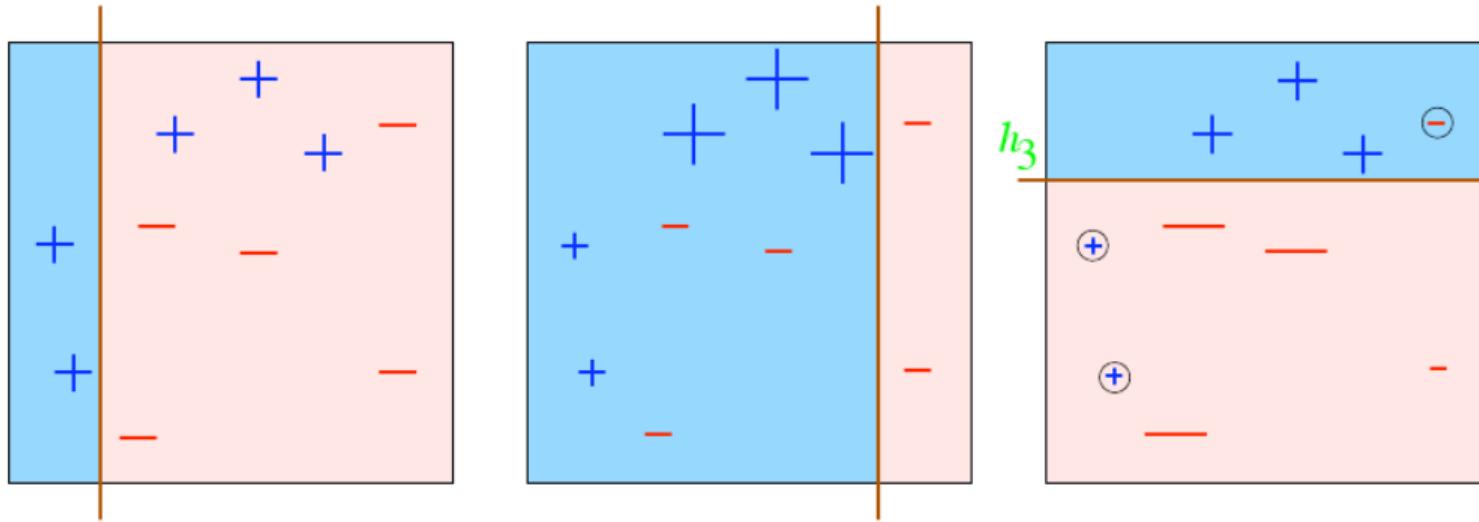
$$\varepsilon_3 = 0.14$$

$$\alpha_3 = 0.92$$

From, Léon Bottou

Example

How do we combine the results now?



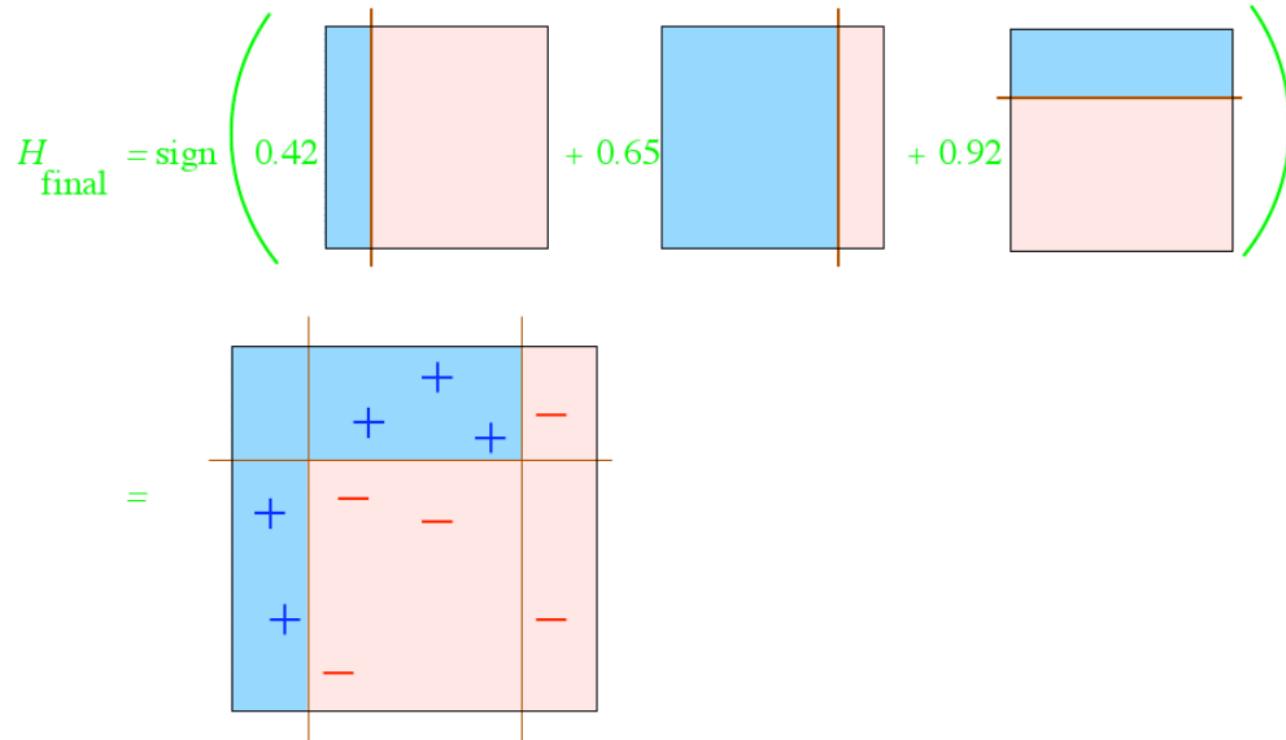
$$\varepsilon_3 = 0.14$$

$$\alpha_3 = 0.92$$

From, Léon Bottou

Example

How do we combine the results now?



From, Léon Bottou

AdaBoost Example

- Training sets for the first 3 boosting rounds:

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

- Summary:

Round	Split Point	Left Class	Right Class	alpha
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

AdaBoost Example

- Weights

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

- Classification

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

AdaBoost error function takes into account the fact that only the sign of the final result is used, thus sum can be far larger than 1 without increasing error

AdaBoost base learners

- AdaBoost works best with “weak” learners
 - Should not be complex
 - Typically high bias classifiers
 - Works even when weak learner has an error rate just slightly under 0.5 (i.e., just slightly better than random)
 - Can prove training error goes to 0 in $O(\log n)$ iterations
 - Examples:
 - Decision stumps (1 level decision trees)
 - Depth-limited decision trees
 - Linear classifiers
-

AdaBoost in practice

Strengths:

- Fast and simple to program
- No parameters to tune (besides T)
- No assumptions on weak learner

When boosting can fail:

- Given insufficient data
- Overly complex weak hypotheses
- Can be susceptible to noise
- When there are a large number of outliers

Fine Tuning Ensembles

- Model combination does not always guaranteed to decrease error, unless
 - base-learners are diverse and accurate
- Ignore poor base learners
 - Use accuracy as a cut-off
 - Introduce some pruning with which at each iteration remove poor learners / learners whose absence lead to improvement (if any)
 - Modify iterations to allow both additions / deletions of learners
 - Discarding appropriately leads to better performance

Gradient Boosting

- In Gradient Boosting, "shortcomings" are identified by gradients.
- Recall that, in Adaboost, “shortcomings” are identified by high-weight data points.
- Both high-weight data points and gradients tell us how to improve our model.

Gradient Boosting

- Gradient Boosting for Different Problems
Difficulty: regression ==> classification
==> ranking

Gradient Boosting

- You are given $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, and the task is to fit a model $F(x)$ to minimize square loss
- There are some mistakes: $F(x_1) = 0.8$, while $y_1 = 0.9$, and $F(x_2) = 1.4$ while $y_2 = 1.3\dots$ How can you improve this model?
- Rules:
 - You are not allowed to remove anything from F or change any parameter in F .
 - You can add an additional model (regression tree)⁷¹ h to F , so the new prediction will be $F(x) + h(x)$.

Gradient Boosting

- You wish to improve the model such that
 - $F(x_1) + h(x_1) = y_1$
 - $F(x_2) + h(x_2) = y_2 \dots$
 - $F(x_n) + h(x_n) = y_n$

Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$

$$h(x_2) = y_2 - F(x_2) \dots$$

$$h(x_n) = y_n - F(x_n)$$

Fit a regression tree h to data

$$(x_1, y_1 - F(x_1)), (x_2, y_2 - F(x_2)), \dots, (x_n, y_n - F(x_n))$$

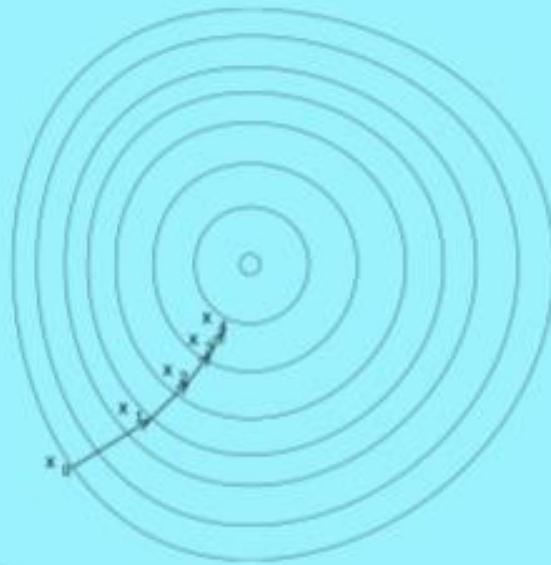
Gradient Boosting

- Simple solution: $y_i - F(x_i)$ are called residuals. These are the parts that existing model F cannot do well.
- The role of h is to compensate the shortcoming of existing model F .
- If the new model $F + h$ is still not satisfactory, we can add another regression tree...
- We are improving the predictions of training data, is the procedure also useful for test data?

Gradient Descent

Minimize a function by moving in the opposite direction of the gradient.

$$\theta_i := \theta_i - \rho \frac{\partial J}{\partial \theta_i}$$



Gradient Boosting for regression

Loss function $L(y, F(x)) = (y - F(x))^2/2$

We want to minimize $J = \sum_i L(y_i, F(x_i))$ by adjusting $F(x_1), F(x_2), \dots, F(x_n)$.

Notice that $F(x_1), F(x_2), \dots, F(x_n)$ are just some numbers. We can treat $F(x_i)$ as parameters and take derivatives

$$\frac{\partial J}{\partial F(x_i)} = \frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = F(x_i) - y_i$$

So we can interpret residuals as negative gradients.

$$y_i - F(x_i) = -\frac{\partial J}{\partial F(x_i)}$$

$$F(x_i) := F(x_i) + h(x_i)$$

$$F(x_i) := F(x_i) + y_i - F(x_i)$$

$$F(x_i) := F(x_i) - 1 \frac{\partial J}{\partial F(x_i)}$$

$$\theta_i := \theta_i - \rho \frac{\partial J}{\partial \theta_i}$$

For regression with **square loss**,

residual \Leftrightarrow negative gradient

fit h to residual \Leftrightarrow fit h to negative gradient

update F based on residual \Leftrightarrow update F based on negative gradient

So we are actually updating our model using **gradient descent!**

Gradient Boosting Algorithm

- It involves three elements
 - A loss function to be optimized (minimizes expected value)

$$\hat{F} = \arg \min_F \mathbb{E}_{x,y} [L(y, F(x))]$$

- Approximation of $F(x)$ in terms of weighted sum of base(weak) learners $h_i(x)$ to make

$$\hat{F}(x) = \sum_{i=1}^M \gamma_i h_i(x) + \text{const}$$

- An additive model to minimize the loss function, starting with $F_0(x)$ and incrementally expanding in greedy fashion

$$F_0(x) = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, \gamma),$$

$$F_m(x) = F_{m-1}(x) + \arg \min_{h_m \in \mathcal{H}} \left[\sum_{i=1}^n L(y_i, F_{m-1}(x_i) + h_m(x_i)) \right]$$

Why XGBoost so popular?

- **Speed** : faster than other ensemble classifiers.
- **Core algorithm is parallelizable**: harness the power of multi-core computers and networks of computers enabling to train on very large datasets **Consistently outperforms other algorithm methods** : It has shown better performance on a variety of machine learning benchmark datasets.
- **Wide variety of tuning parameters** : cross-validation, regularization, missing values, tree parameters, etc
- XGBoost (Extreme Gradient Boosting) uses the gradient boosting (GBM) framework at its core.

References

The-Morgan-Kaufmann-Series-in-Data-Management-
Systems-Jiawei-Han-Micheline-Kamber-Jian-Pei-Data-
Mining.-Concepts-and-Techniques-3rd-Edition-Morgan-
Kaufmann-2011

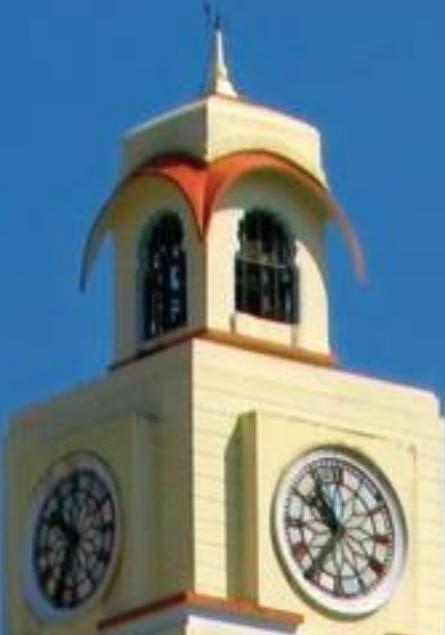
Bishop - Pattern Recognition And Machine Learning -
Springer 2006

A Gentle Introduction to Gradient Boosting
Cheng Li chengli@ccs.neu.edu College of Computer and
Information Science Northeastern University

https://www.youtube.com/watch?time_continue=647&v=LsK-xG1cLYA&feature=emb_logo



Thank You!



BITS Pilani
Pilani Campus

Support Vector Machines

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in



Text Book(s)

- | | |
|----|--|
| T1 | Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition |
| T2 | Tom M. Mitchell: Machine Learning, The McGraw-Hill Companies, Inc.. |

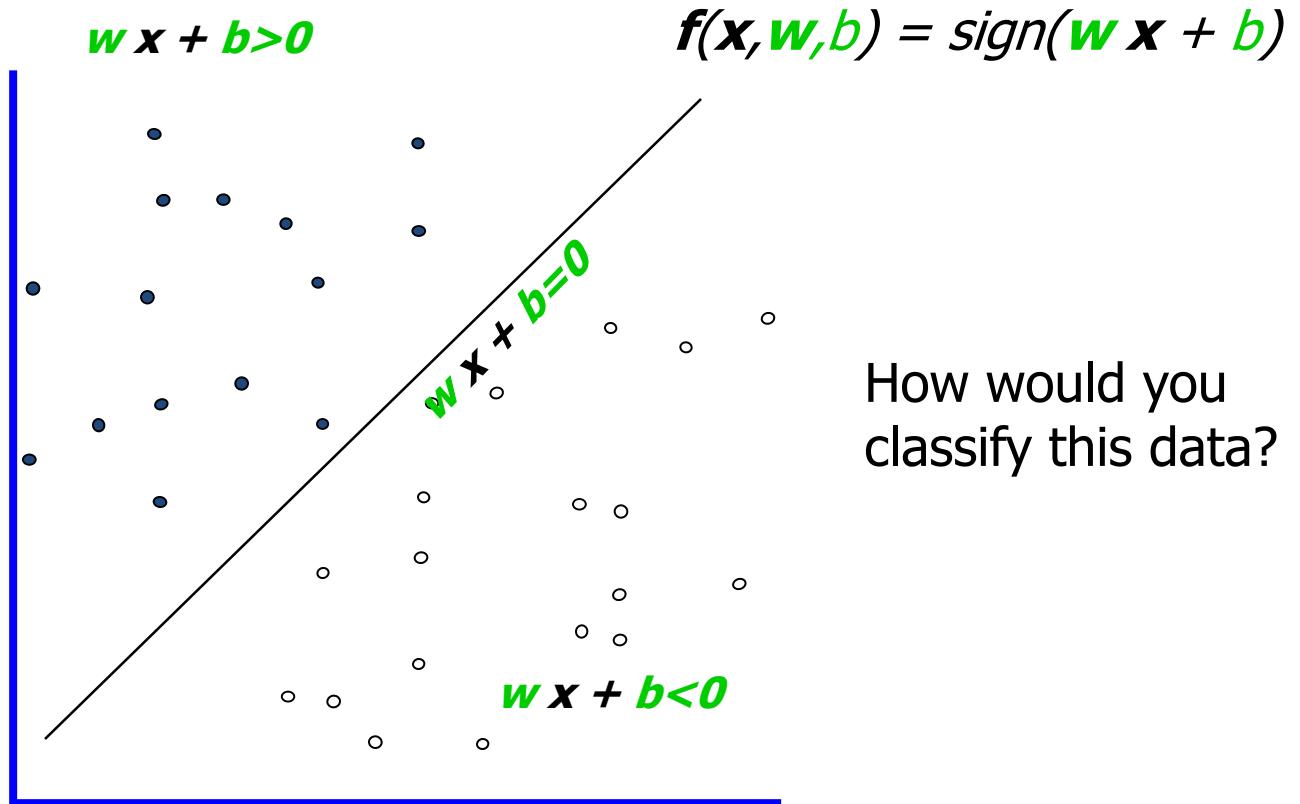
These slides are prepared by the instructor, with grateful acknowledgement of Prof. Tom Mitchell, Prof. Burges, Prof. Andrew Moore and many others who made their course materials freely available online.

Topics to be covered

- Linear Classifiers
- Maximum Margin Classification
- Linear SVM
- SVM optimization problem
- Soft Margin SVM

Linear Classifiers

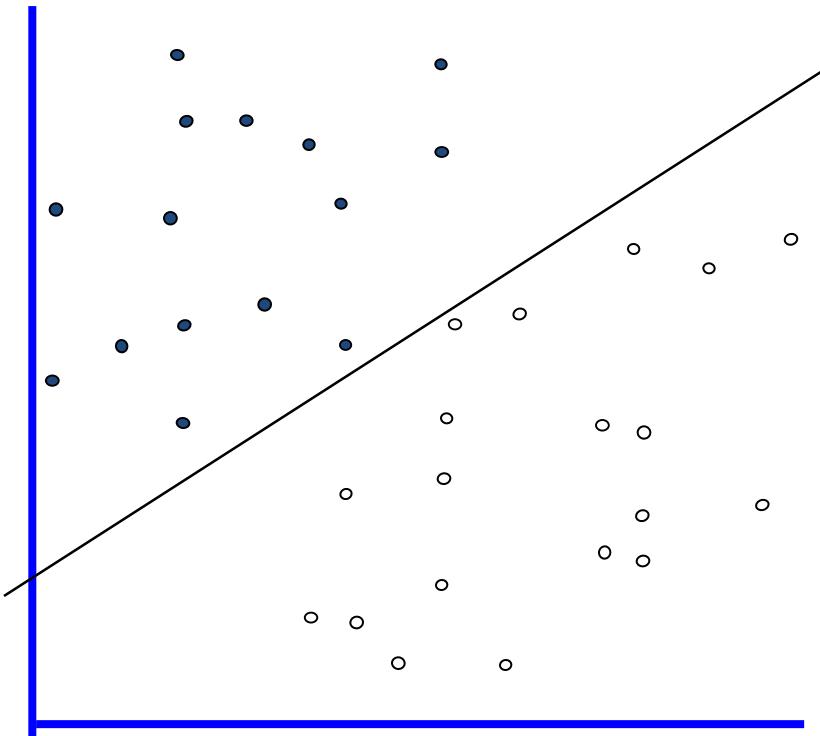
- denotes +1
- denotes -1



Linear Classifiers

$$\mathbf{f}(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

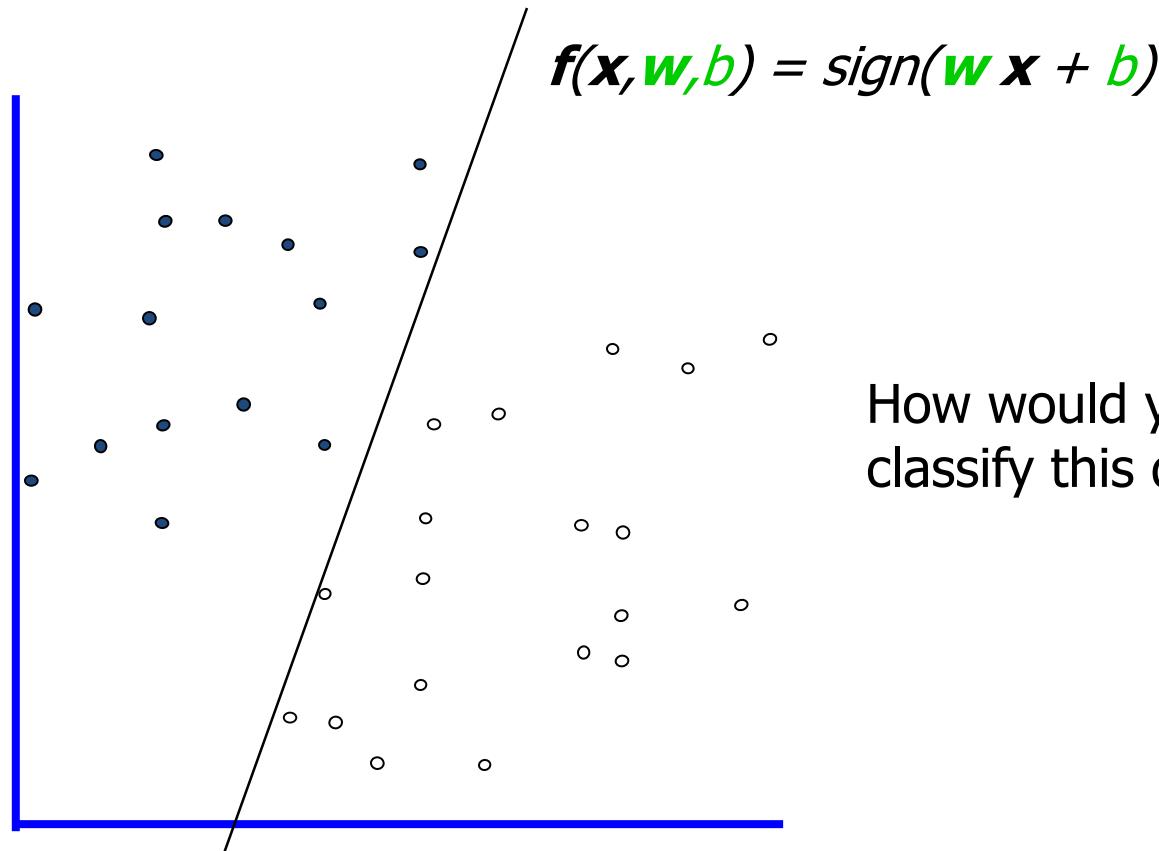
- denotes +1
- denotes -1



How would you
classify this data?

Linear Classifiers

- denotes +1
- denotes -1

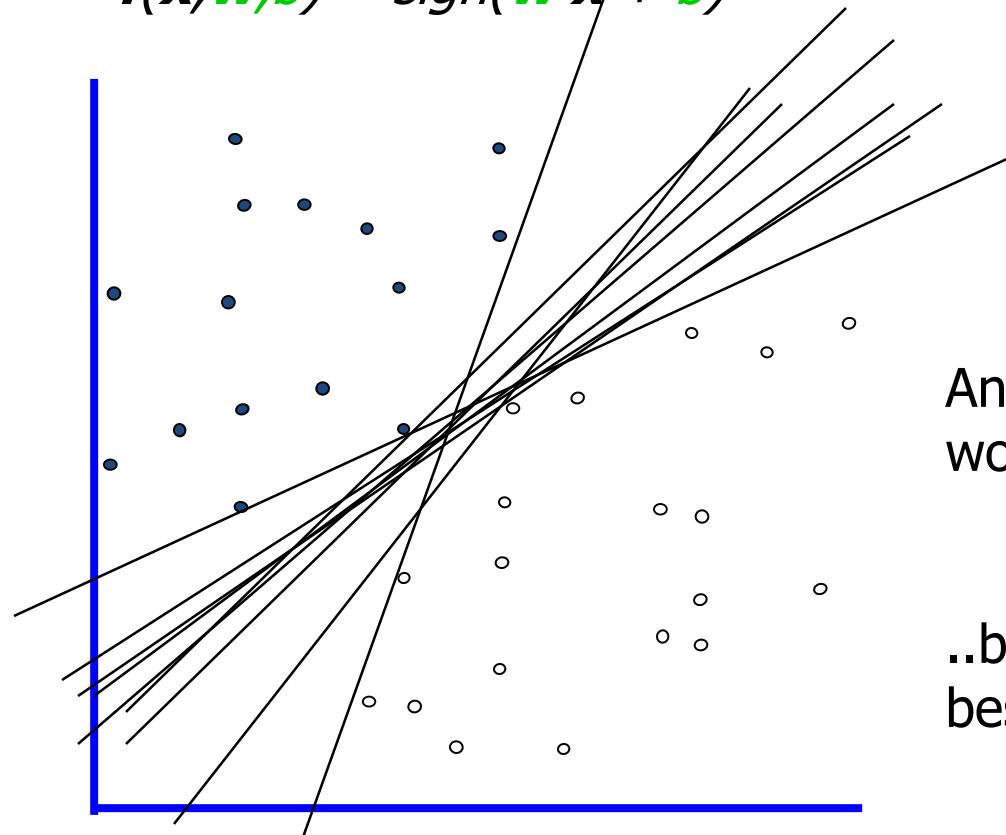


How would you
classify this data?

Linear Classifiers

$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

- denotes +1
- denotes -1

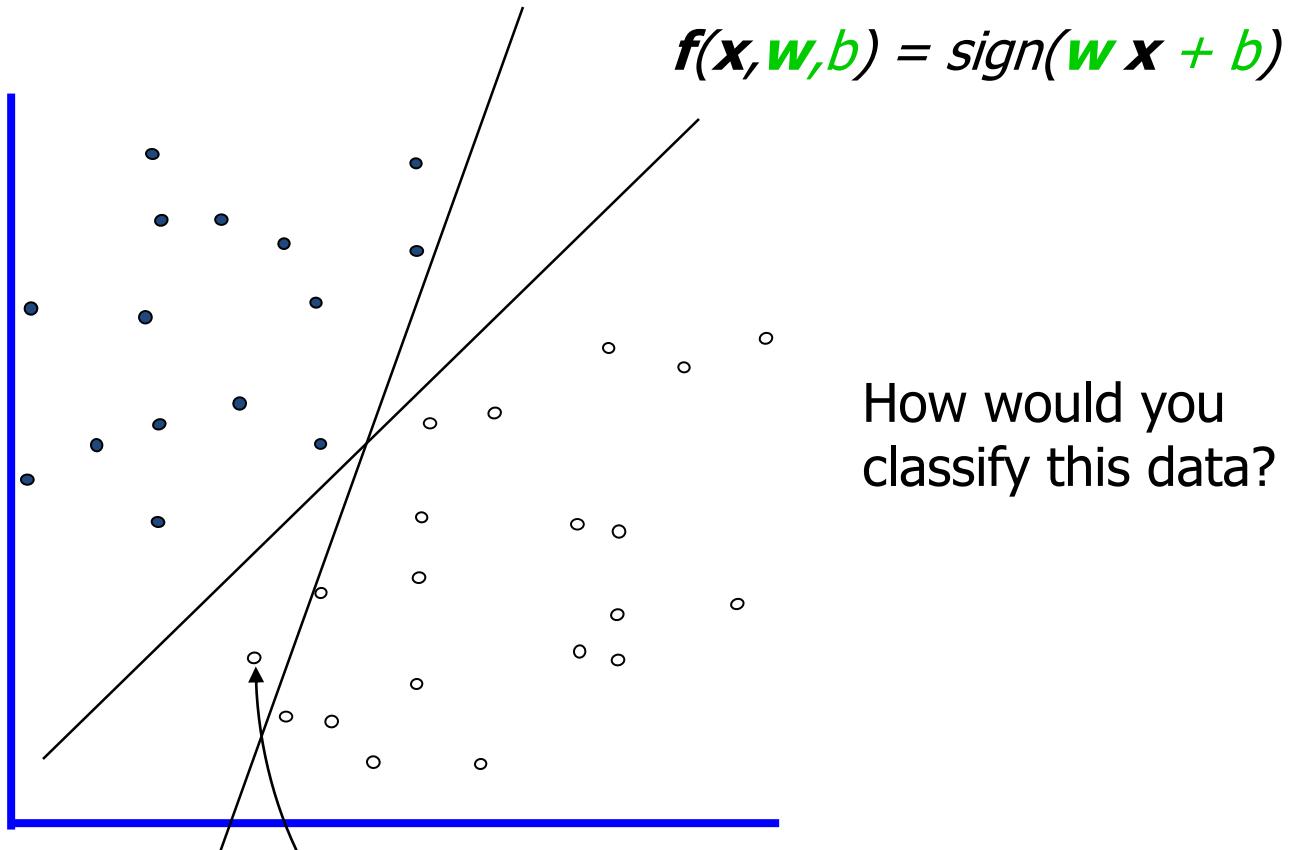


Any of these
would be fine..

..but which is
best?

Linear Classifiers

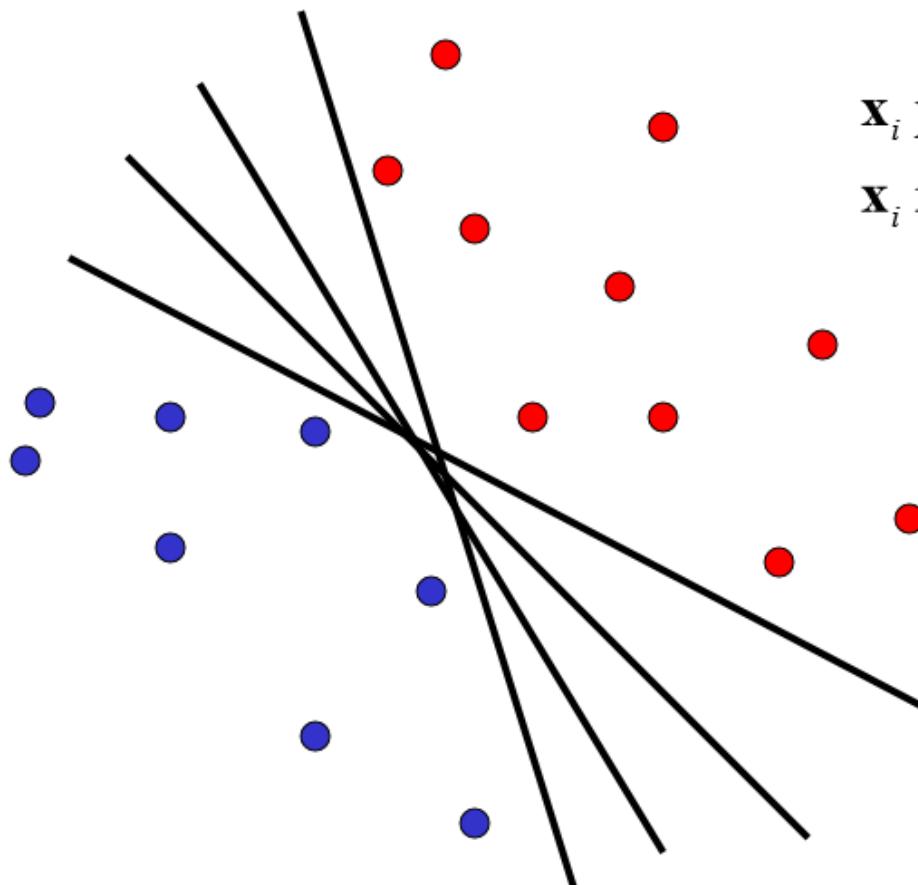
- denotes +1
- denotes -1



Misclassified
to +1 class

Linear Classifier

- Find linear function to separate positive and negative examples

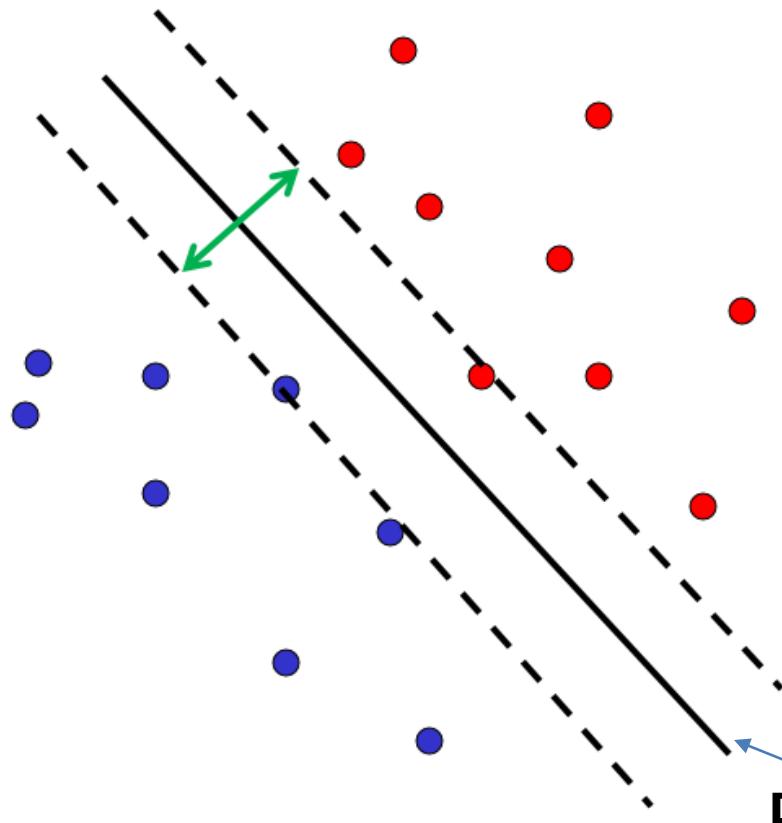


\mathbf{x}_i positive : $\mathbf{x}_i \cdot \mathbf{w} + b \geq 0$

\mathbf{x}_i negative : $\mathbf{x}_i \cdot \mathbf{w} + b < 0$

Which line
is best?

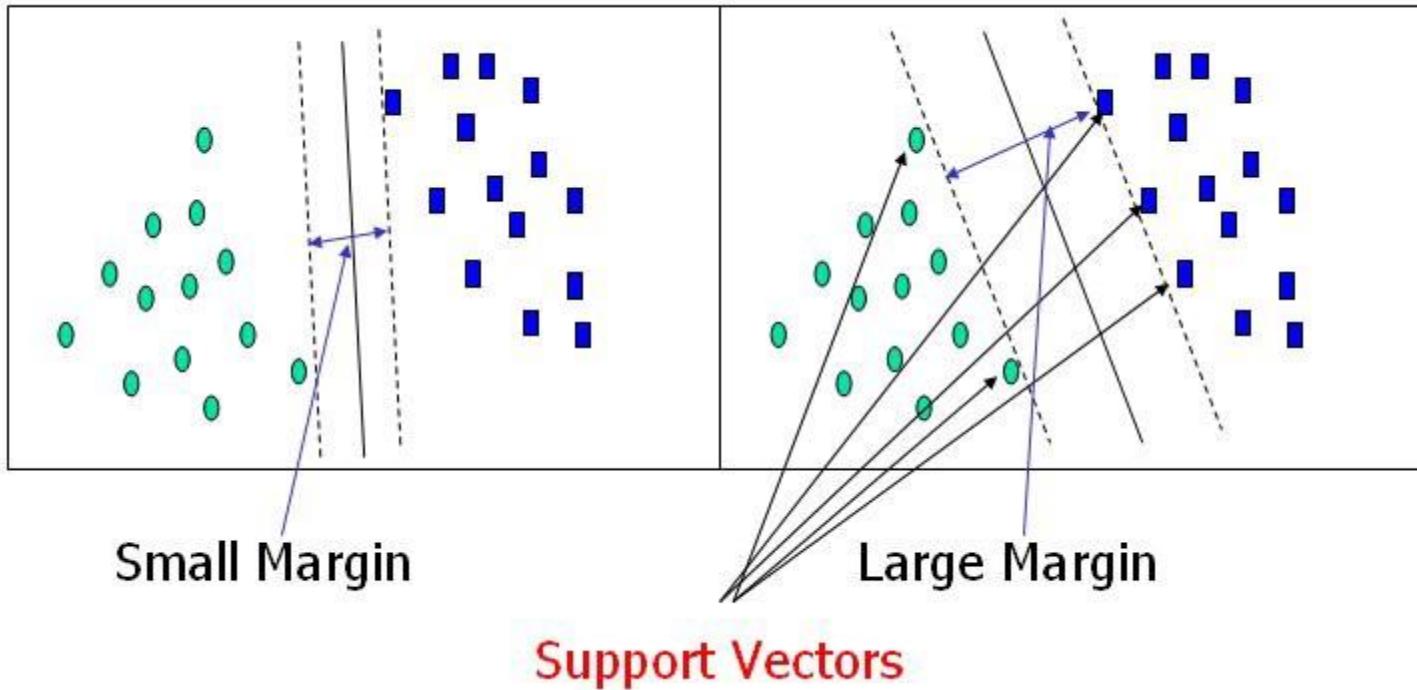
Linear Classifier



- Discriminative classifier based on *optimal separating line (for 2d case)*
- Maximize the *margin* between the positive and negative training examples

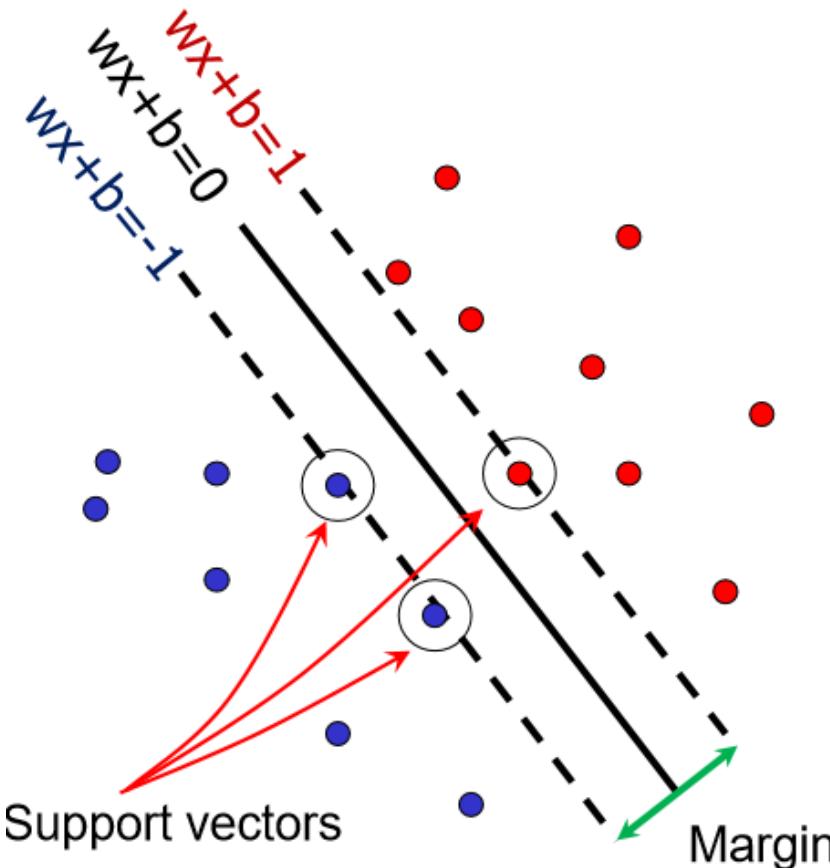
C. Burges, [A Tutorial on Support Vector Machines for Pattern Recognition](#), Data Mining and Knowledge Discovery, 1998

Large margin and support vectors



Support Vector Machines

- Want line that maximizes the margin.



\mathbf{x}_i positive ($y_i = 1$): $\mathbf{x}_i \cdot \mathbf{w} + b \geq 1$

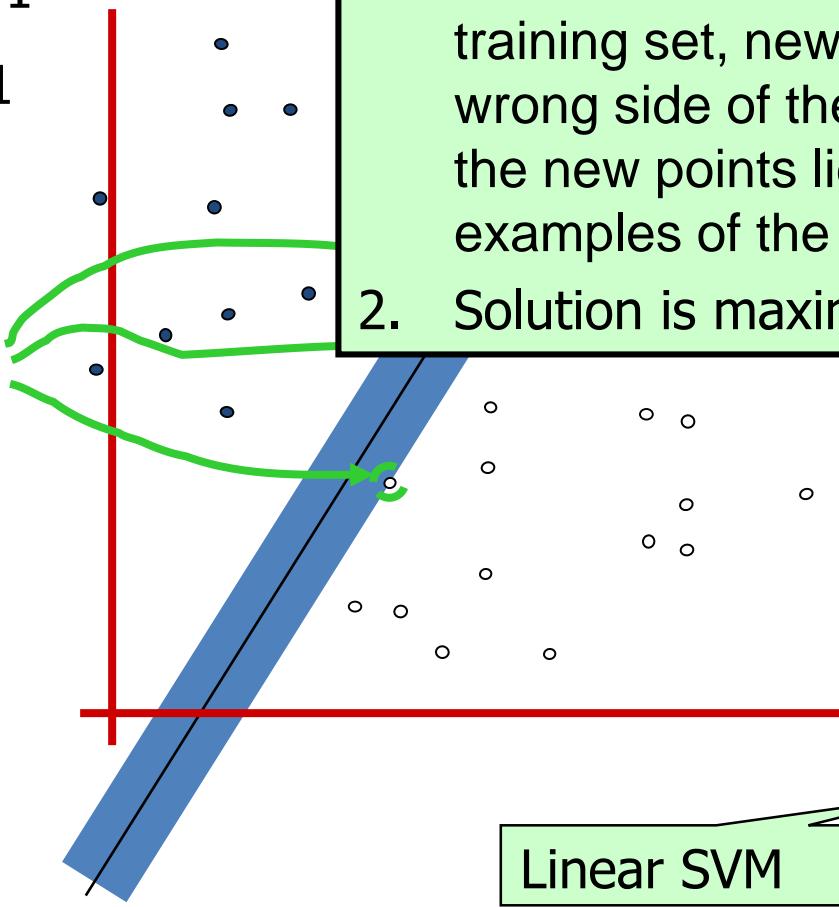
\mathbf{x}_i negative ($y_i = -1$): $\mathbf{x}_i \cdot \mathbf{w} + b \leq -1$

For support vectors, $\mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

Maximum Margin

• denotes +1
 • denotes -1

Support Vectors
 are those
 datapoints that
 the margin
 pushes up
 against



1. If hyperplane is oriented such that it is close to some of the points in your training set, new data may lie on the wrong side of the hyperplane, even if the new points lie close to training examples of the correct class.
2. Solution is maximizing the margin with the, maximum margin.

This is the simplest kind of SVM (Called an LSVM)

Support Vectors

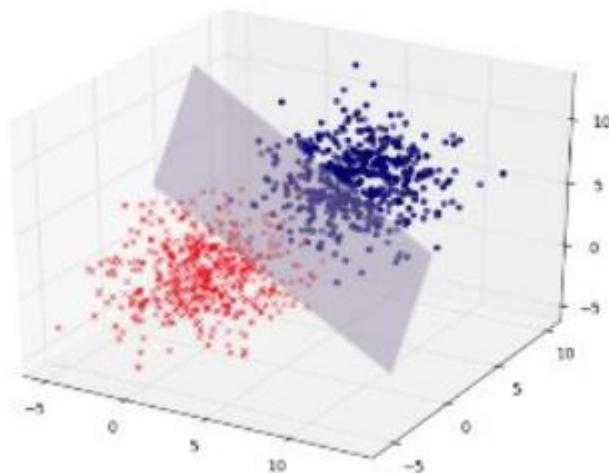
- Geometric description of SVM is that the max-margin hyperplane is completely determined by those points that lie nearest to it.
- Points that lie on this margin are the support vectors.
- The points of our data set which if removed, would alter the position of the dividing hyperplane

Example

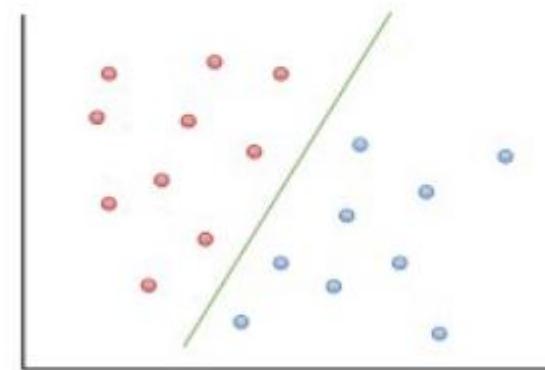
$$\mathbf{w}^T \mathbf{x} = 0$$

$$y = ax + b$$

Hyperplane



Line



Weight vector is perpendicular to the hyperplane

Consider the points x_a and x_b , which lie on the decision boundary.

This gives us two equations:

$$w^T x_a + b = 0$$

$$w^T x_b + b = 0$$

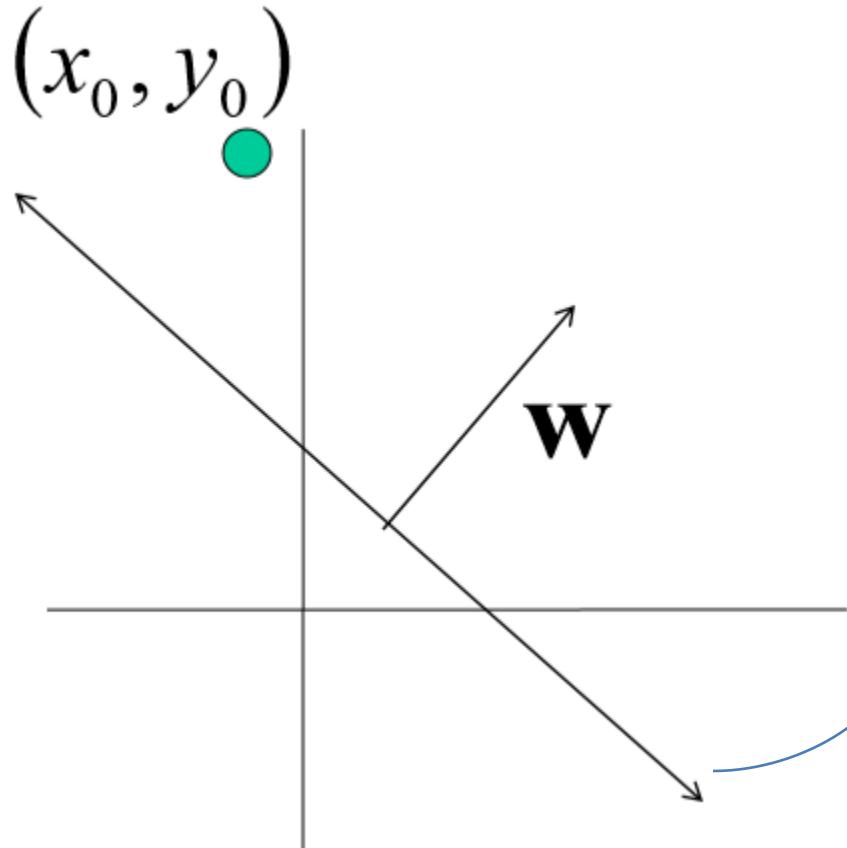
Subtracting these two equations gives us

$$w^T.(x_a - x_b) = 0$$

Note that the vector $x_a - x_b$ lies on the decision boundary, and it is directed from x_b to x_a .

Since the dot product $w^T.(x_a - x_b)$ is zero, w^T must be orthogonal to $x_a - x_b$ and in turn, to the decision boundary.

Line with 2 features: R2



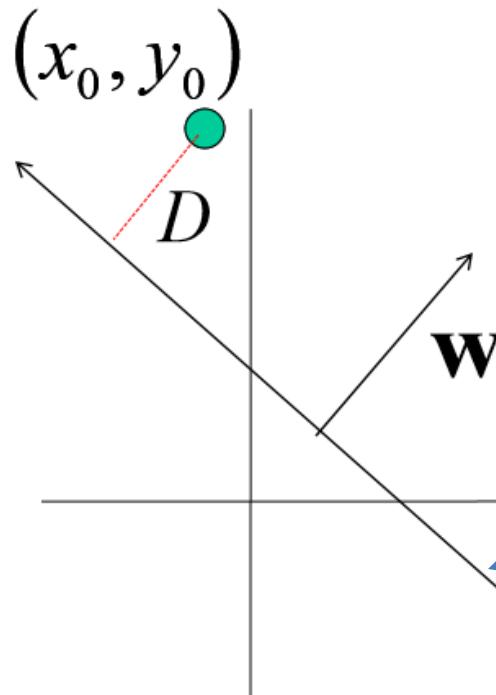
Let $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$ $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$



$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

Line with 2 features: R2



Let $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$ $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$

↔

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

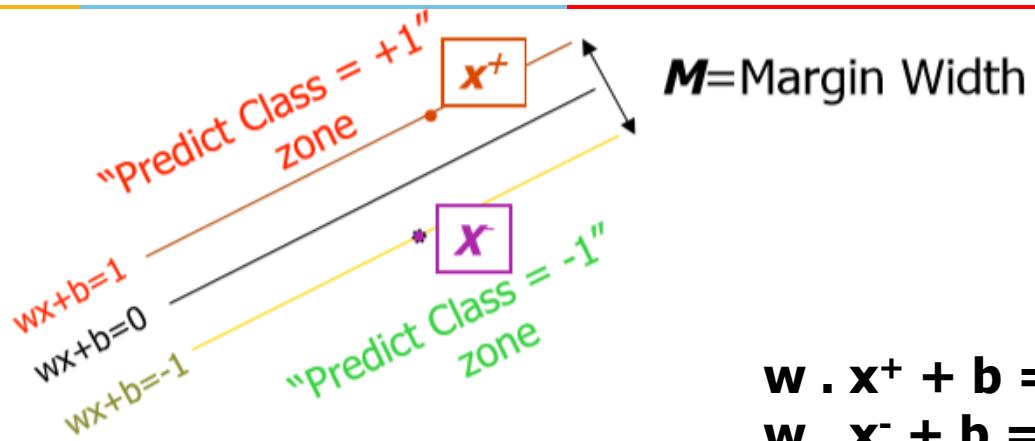
$$D = \frac{|ax_0 + cy_0 + b|}{\sqrt{a^2 + c^2}} = \frac{|\mathbf{w}^\top \mathbf{x} + b|}{\|\mathbf{w}\|}$$

} distance from
point to line

Kristen Grauman

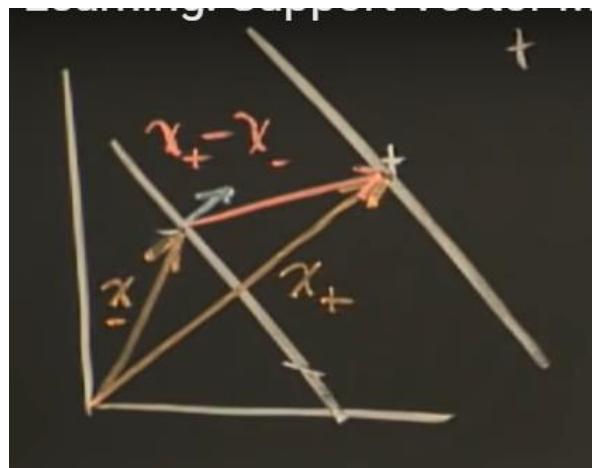
<https://brilliant.org/wiki/dot-product-distance-between-point-and-a-line/>

Linear SVM Mathematically



$$\mathbf{w} \cdot \mathbf{x}^+ + \mathbf{b} = +1$$

$$\mathbf{w} \cdot \mathbf{x}^- + \mathbf{b} = -1$$



Margin width

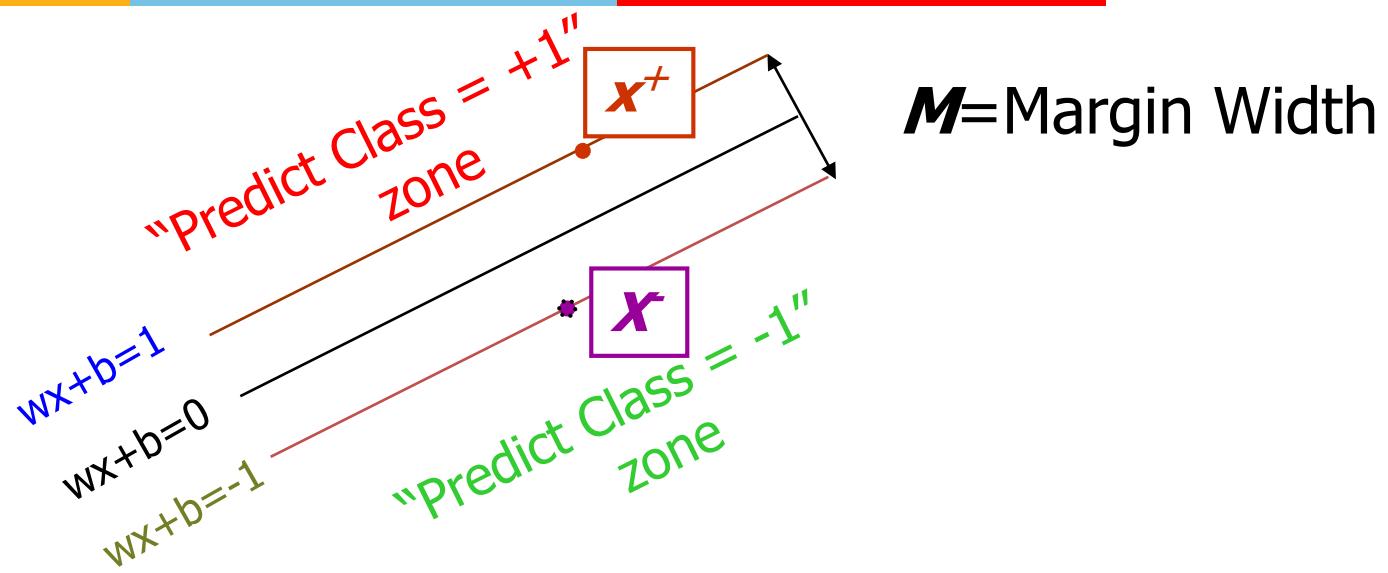
$$= \mathbf{x}^+ - \mathbf{x}^- \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|}$$

$$= \frac{\mathbf{w} \cdot \mathbf{x}^+ - \mathbf{w} \cdot \mathbf{x}^-}{\|\mathbf{w}\|}$$

$$= (1-\mathbf{b}) - (-1-\mathbf{b}) / \|\mathbf{w}\|$$

$$= \frac{2}{\|\mathbf{w}\|}$$

Linear SVM Mathematically



Distance between lines given by solving linear equation:

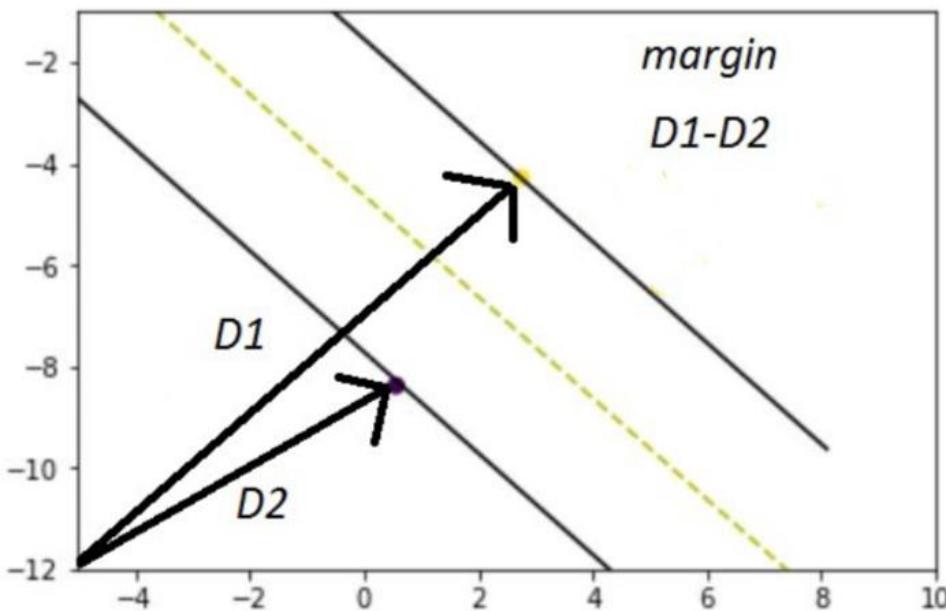
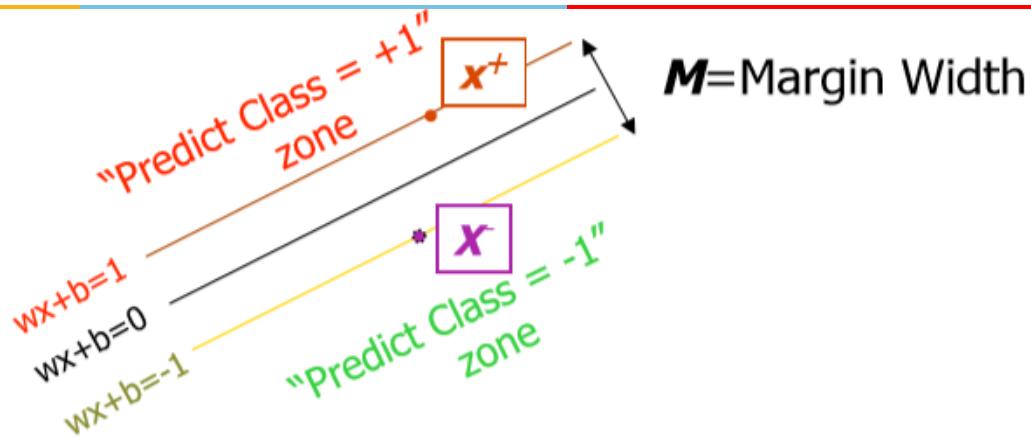
What we know:

- $\mathbf{w} \cdot \mathbf{x}^+ + b = +1$
- $\mathbf{w} \cdot \mathbf{x}^- + b = -1$

$$\text{Maximize margin: } M = \frac{2}{\|\mathbf{w}\|}$$

$$\text{Equivalent to minimize: } \frac{1}{2} \|\mathbf{w}\|^2$$

Linear SVM Mathematically



$$D1 = w^T x + b = 1 \quad w^T x + b - 1 = 0$$

$$D2 = w^T x + b = -1 \quad w^T x + b + 1 = 0$$

$$w^T x + b - 1 - w^T x + b + 1$$



Solve algebraically

$$\frac{2}{|w|}$$

Solving the Optimization Problem

1. Maximize margin $2/\|\mathbf{w}\|$
2. Correctly classify all training data points:

$$\mathbf{x}_i \text{ positive } (y_i = 1) : \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1) : \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

Quadratic optimization problem:

Find \mathbf{w} and b such that

$\Phi(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2$ is minimized;

and for all $\{(\mathbf{x}_i, y_i)\}$: $y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$$

$$+1 (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$$

$$-1 (\mathbf{w}^T \mathbf{x}_i + b) \leq 1$$

$$\text{same as } (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$$

Solving the Optimization Problem



Find w and b such that

$\Phi(w) = \frac{1}{2}\|w\|^2$ is minimized; Type equation here.

and for all $\{(x_i, y_i)\}$: $y_i (w^T x_i + b) \geq 1$

← Primal

- Need to optimize a *quadratic function subject to linear inequality* constraints.
- All constraints in SVM are linear
- Quadratic optimization problems are a well-known class of mathematical programming problems, and many (rather intricate) algorithms exist for solving them.
- The solution involves constructing a *unconstrained problem* where a *Lagrange multiplier α_i* is associated with every constraint in the primary problem:

Optimization Problem

- Optimization problem is typically written:

Minimize $f(x)$

subject to

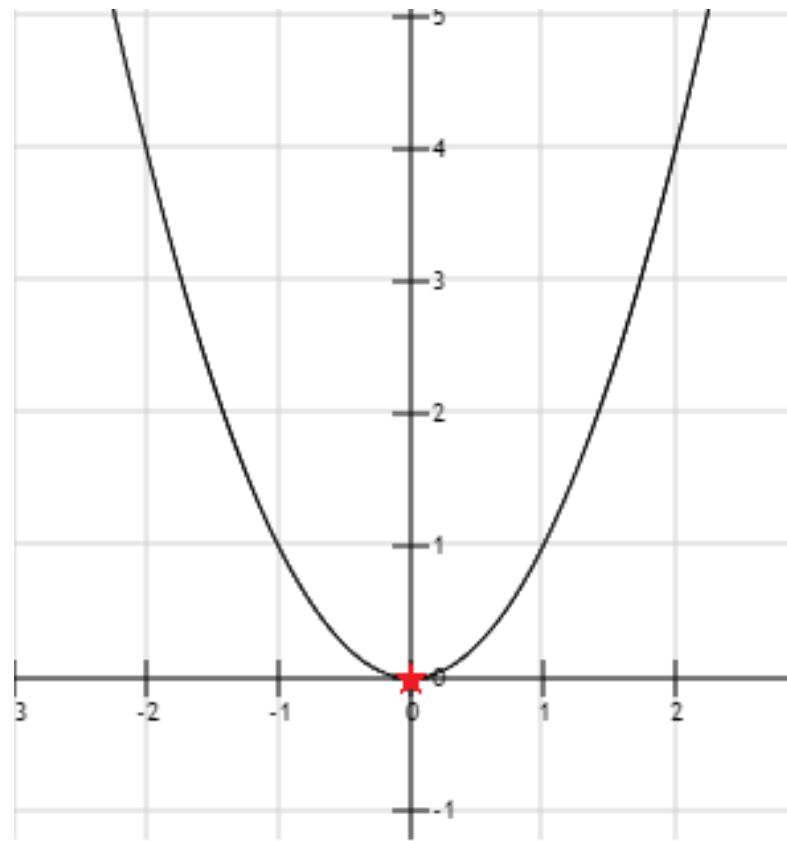
$$g_i(x) = 0, \quad i=1, \dots, p$$

$$h_i(x) \leq 0, \quad i=1, \dots, m$$

- $f(x)$ is called the objective function
 - By changing x (the optimization variable) we wish to find a value x^* for which $f(x)$ is at its minimum.
 - p functions of g_i define equality constraints and
 - m functions h_i define inequality constraints.
 - The value we find MUST respect these constraints!
-

Unconstrained Optimization

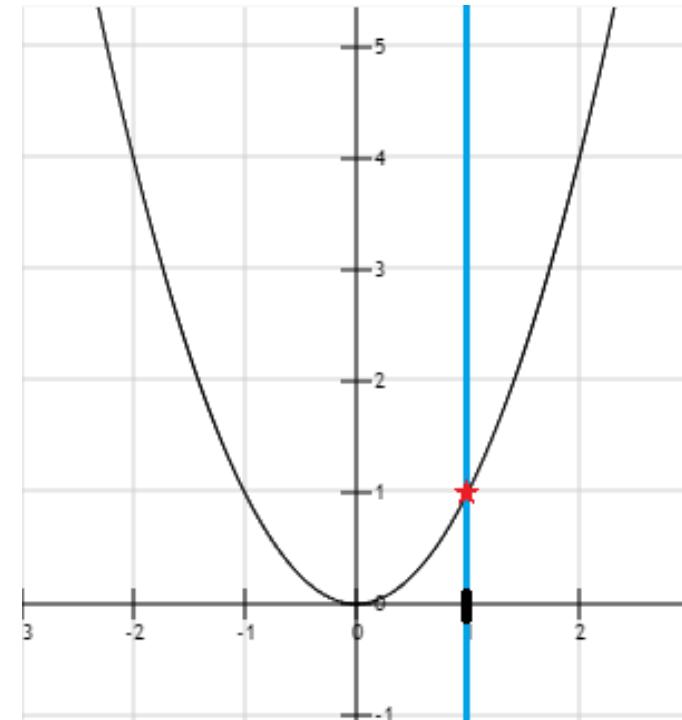
- Minimize x^2



Constrained Optimization -Equality Constraint

Minimize x^2

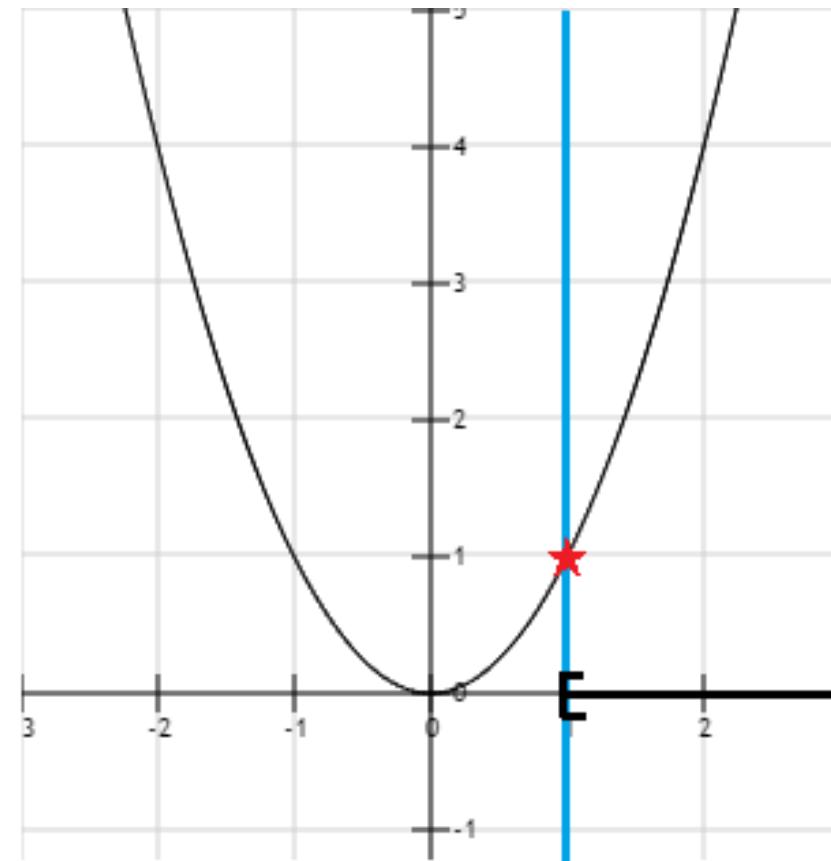
Subject to $x = 1$



Constrained Optimization - Inequality Constraint

Minimize x^2

Subject to $x \geq 1$



Constrained optimization

- We can also have mix equality and inequality constraints together.
- Only restriction is that if we use contradictory constraints, we can end up with a problem which does not have a feasible set

Minimize x^2

Subject to

$$x = 1$$

$$x < 0$$

Impossible for x to be equal 1 and less than zero at the same

Constrained optimization

- A solution is an assignment of values to variables.
- A feasible solution is an assignment of values to variables such that all the constraints are satisfied.
- The objective function value of a solution is obtained by evaluating the objective function at the given solution.
- An optimal solution (assuming minimization) is one whose objective function value is less than or equal to that of all other feasible solutions.

Lagrange Multipliers

- **How do we find the solution to an optimization problem with constraints?**
- Constrained maximization (minimization) problem is rewritten as a Lagrange function whose optimal point is a saddle point, i.e. a global maximum (minimum)
- *Lagrange function use Lagrange multipliers is a strategy for finding the local maxima and minima of a function subject to constraints*

Constrained to Unconstrained Optimization: Lagrange Multiplier

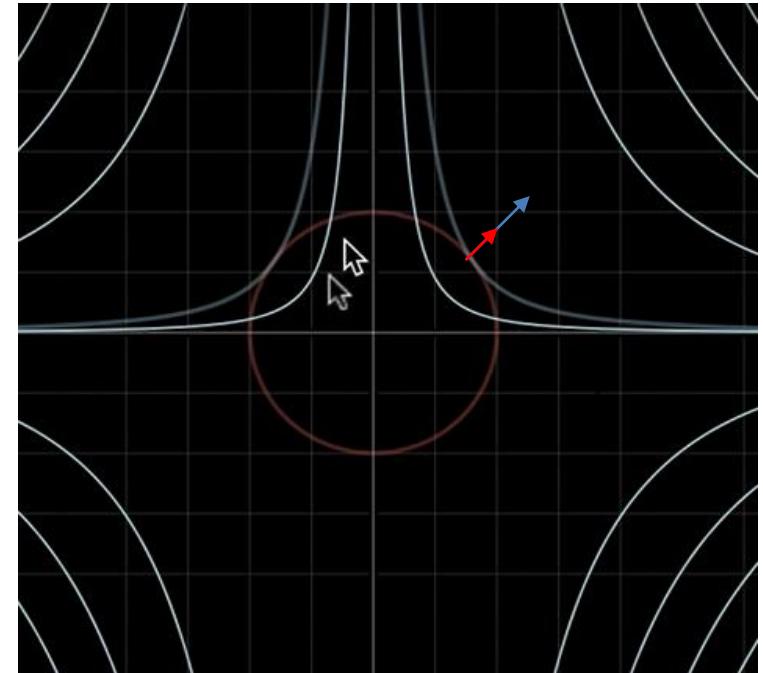
Maximize

$$f(x,y) = x^2 y$$

Subject to

$$g(x, y) : x^2 + y^2 = 1$$

- Maximum of $f(x,y)$ under constraint $g(x, y)$ is obtained when their gradients point to same direction (when they are tangent to each other).
 - Introduce a Lagrange multiplier λ for the equality constraint
 - Mathematically,
- $$\nabla f(x,y) = \lambda \nabla g(x,y)$$



Example:

$$\max_{x,y} xy \text{ subject to } x + y = 6$$

- Introduce a Lagrange multiplier λ for constraint
- Construct the Lagrangian

$$L(x, y) = xy - \lambda(x + y - 6)$$

- Stationary points

$$\frac{\partial L(x, y)}{\partial \lambda} = x + y - 6 = 0$$

$$\begin{cases} \frac{\partial L(x, y)}{\partial x} = y - \lambda = 0 \\ \frac{\partial L(x, y)}{\partial y} = x - \lambda = 0 \end{cases} \Rightarrow x = y = \lambda$$

$$\Rightarrow x = y = 3$$

x and y values remain same even if you take $+\lambda$ or $-\lambda$ for equality constraint

$$\begin{aligned} 2x &= 6 \\ x &= y = 3 \\ \lambda &= 3 \end{aligned}$$

Karush–Kuhn–Tucker (KKT) theorem

- KKT approach to nonlinear programming (quadratic) generalizes the method of Lagrange multipliers, which allows only equality constraints.
- KKT allows inequality constraints

Karush-Kuhn-Tucker (KKT) conditions

- Start with

$\max f(x)$ subject to

$g_i(x) = 0$ and $h_j(x) \geq 0$ for all i, j

- Make the Lagrangian function

$$\mathcal{L} = f(x) - \sum_i \lambda_i g_i(x) - \sum_j \mu_j h_j(x)$$

- Take gradient and set to 0 – but other conditions also.

KKT conditions

- Make the Lagrangian function

$$\mathcal{L} = f(x) - \sum_i \lambda_i g_i(x) - \sum_j \mu_j h_j(x)$$

- Necessary conditions to have a minimum are

$$\nabla_x \mathcal{L}(x^*, \lambda^*, \mu^*) = 0$$

$$g_i(x^*) = 0 \text{ for all } i$$

$$h_j(x^*) \geq 0 \text{ for all } j$$

$$\mu_j \geq 0 \text{ for all } j$$

$$\mu_j^* h_j(x^*) = 0 \text{ for all } j$$

Solving the Optimization Problem

Find w and b such that

$\Phi(w) = \frac{1}{2}||w||^2$ is minimized;

and for all $\{(x_i, y_i)\}$: $y_i (w^T x_i + b) \geq 1$

- Need to optimize a *quadratic* function subject to *linear inequality* constraints.
- The solution involves constructing a *dual problem* where a *Lagrange multiplier α_i* is associated with every constraint in the primal problem

Solving the Optimization Problem

- The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every constraint in the primary problem:

$$L(w, b, \alpha_i) = \frac{1}{2} \|w\|^2 - \sum \alpha_i [y_i (w^T x_i + b) - 1]$$

- Taking partial derivative with respect to w , $\frac{\partial L}{\partial w} = 0$**
 - $w - \sum \alpha_i y_i x_i = 0$
 - $w = \sum \alpha_i y_i x_i$
- Taking partial derivative with respect to b , $\frac{\partial L}{\partial b} = 0$**
 - $-\sum \alpha_i y_i = 0$
 - $\sum \alpha_i y_i = 0$

Solving the Optimization Problem

$$L(w, b, \alpha_i) = \frac{1}{2} \|w\|^2 - \sum \alpha_i [y_i (w \cdot x_i + b) - 1]$$

- Expanding above equation:

$$L(w, b, \alpha_i) = \frac{1}{2} \|w\|^2 - \sum \alpha_i y_i w \cdot x_i - \sum \alpha_i y_i b + \sum \alpha_i$$

- Substituting $w = \sum \alpha_i y_i x_i$ and $\sum \alpha_i y_i = 0$ in above equation

$$L(w, b, \alpha_i) = \frac{1}{2} (\sum_i \alpha_i y_i x_i) (\sum_j \alpha_j y_j x_j) - (\sum_i \alpha_i y_i x_i) (\sum_j \alpha_j y_j x_j) + \sum \alpha_i$$

$$L(w, b, \alpha_i) = \sum \alpha_i - \frac{1}{2} (\sum_i \alpha_i y_i x_i) (\sum_j \alpha_j y_j x_j)$$

$$L(w, b, \alpha_i) = \sum \alpha_i - \frac{1}{2} (\sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i \cdot x_j)$$

Support Vectors

Using KKT conditions :

$$\alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1] = 0$$

For this condition to be satisfied
either $\alpha_i = 0$ and $y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 > 0$

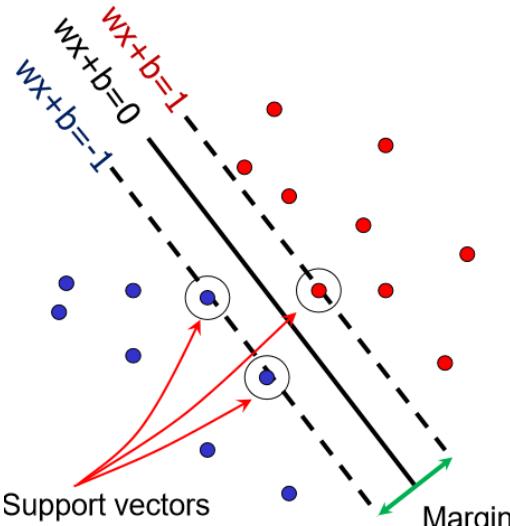
OR

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 = 0 \text{ and } \alpha_i > 0$$

For support vectors:
 $y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 = 0$

For all points other than
support vectors:
 $\alpha_i = 0$

- Want line that maximizes the margin.



$$\mathbf{x}_i \text{ positive } (y_i = 1) : \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

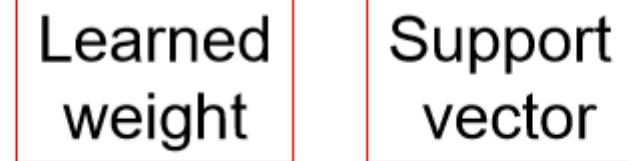
$$\mathbf{x}_i \text{ negative } (y_i = -1) : \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

$$\text{For support vectors, } \mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$$

$$L(\mathbf{w}, b, \alpha_i) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum \alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1]$$

Solving the Optimization Problem

- Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$



Learned weight Support vector

Solving the Optimization Problem

- Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$
 $b = y_i - \mathbf{w} \cdot \mathbf{x}_i$ (for any support vector)

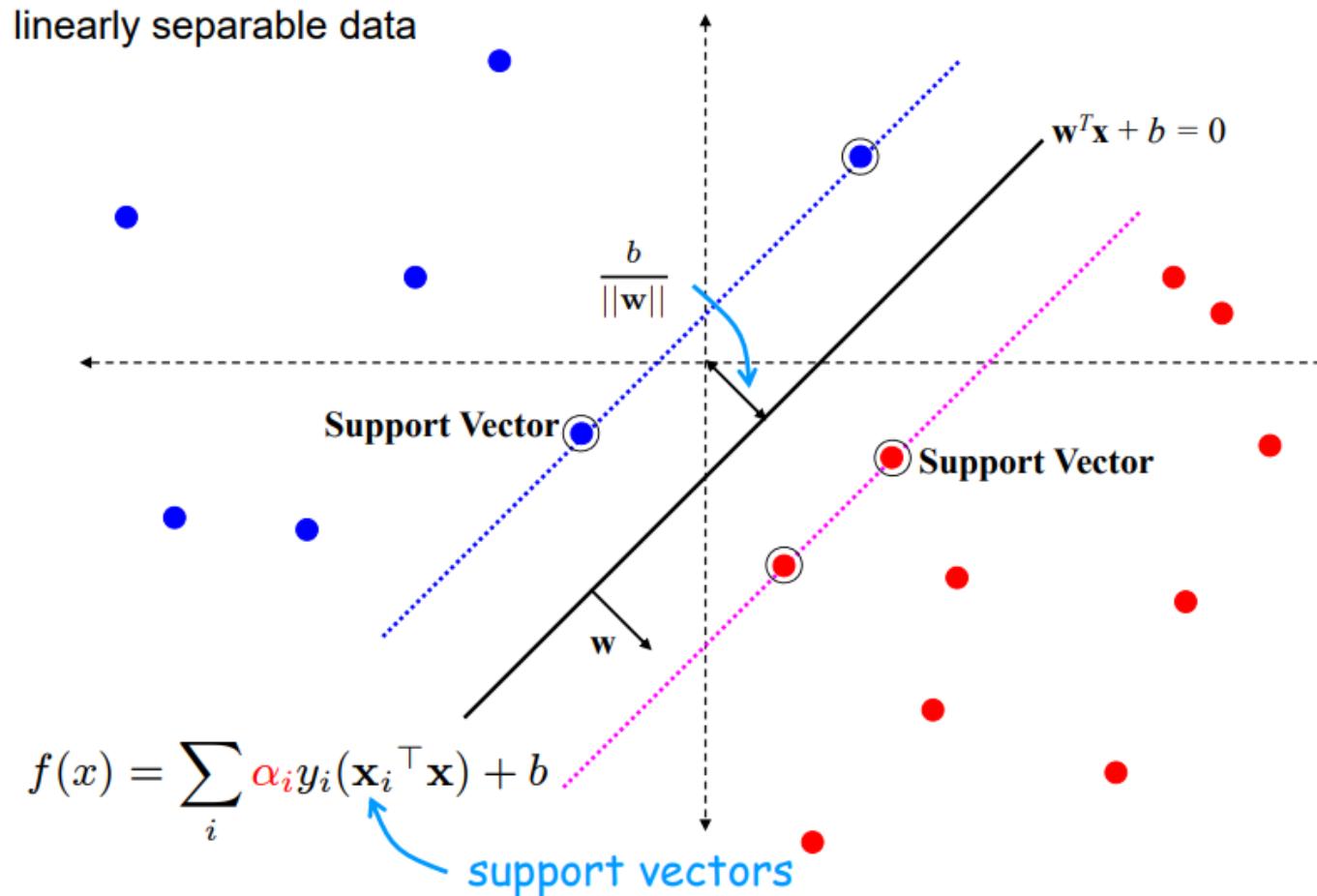
- Classification function:

$$\begin{aligned} f(\mathbf{x}) &= \text{sign} (\mathbf{w} \cdot \mathbf{x} + b) \\ &= \text{sign} \left(\sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b \right) \end{aligned}$$

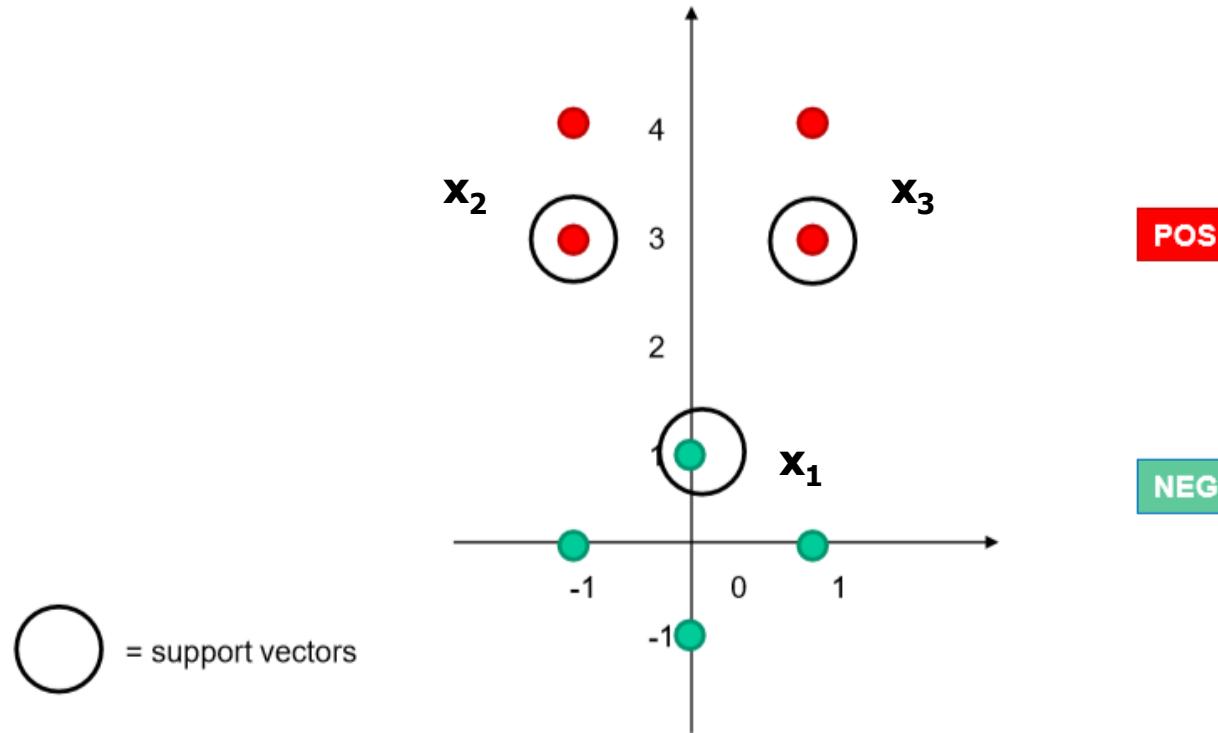
If $f(\mathbf{x}) < 0$, classify as negative, otherwise classify as positive.

- Notice that it relies on an *inner product* between the test point \mathbf{x} and the support vectors \mathbf{x}_i
- (Solving the optimization problem also involves computing the inner products $\mathbf{x}_i \cdot \mathbf{x}_j$ between all pairs of training points)

Substituting w in support vectors function



Example



Example adapted from Dan Ventura

Solving for α

- We know that for the support vectors, $f(x) = 1$ or -1 exactly
- Add a 1 in the feature representation for the bias
- The support vectors have coordinates and labels:
 - $x_1 = [0 \ 1 \ 1]$, $y_1 = -1$
 - $x_2 = [-1 \ 3 \ 1]$, $y_2 = +1$
 - $x_3 = [1 \ 3 \ 1]$, $y_3 = +1$
- Thus we can form the following system of linear equations:

Solving for α

- System of linear equations:

$$\alpha_1 y_1 \text{dot}(x_1, x_1) + \alpha_2 y_2 \text{dot}(x_1, x_2) + \alpha_3 y_3 \text{dot}(x_1, x_3) = y_1$$

$$\alpha_1 y_1 \text{dot}(x_2, x_1) + \alpha_2 y_2 \text{dot}(x_2, x_2) + \alpha_3 y_3 \text{dot}(x_2, x_3) = y_2$$

$$\alpha_1 y_1 \text{dot}(x_3, x_1) + \alpha_2 y_2 \text{dot}(x_3, x_2) + \alpha_3 y_3 \text{dot}(x_3, x_3) = y_3$$

$$-2 * \alpha_1 + 4 * \alpha_2 + 4 * \alpha_3 = -1$$

$$-4 * \alpha_1 + 11 * \alpha_2 + 9 * \alpha_3 = +1$$

$$-4 * \alpha_1 + 9 * \alpha_2 + 11 * \alpha_3 = +1$$

$$\alpha_i [-1 (\mathbf{w} \cdot \mathbf{x}_i + b)] = -1$$

$$\alpha_i [+1 (\mathbf{w} \cdot \mathbf{x}_i + b)] = 1$$

- Solution: $\alpha_1 = 3.5$, $\alpha_2 = 0.75$, $\alpha_3 = 0.75$

Solving for w and b

We know $w = \alpha_1 y_1 x_1 + \dots + \alpha_N y_N x_N$ where $N = \# \text{ SVs}$

$$\text{Thus } w = -3.5 * [0 \ 1 \ 1] + 0.75 [-1 \ 3 \ 1] + 0.75 [1 \ 3 \ 1] = \\ [0 \ 1 \ -2]$$

Separating out weights and bias, we have: $w = [0 \ 1]$ and $b = -2$

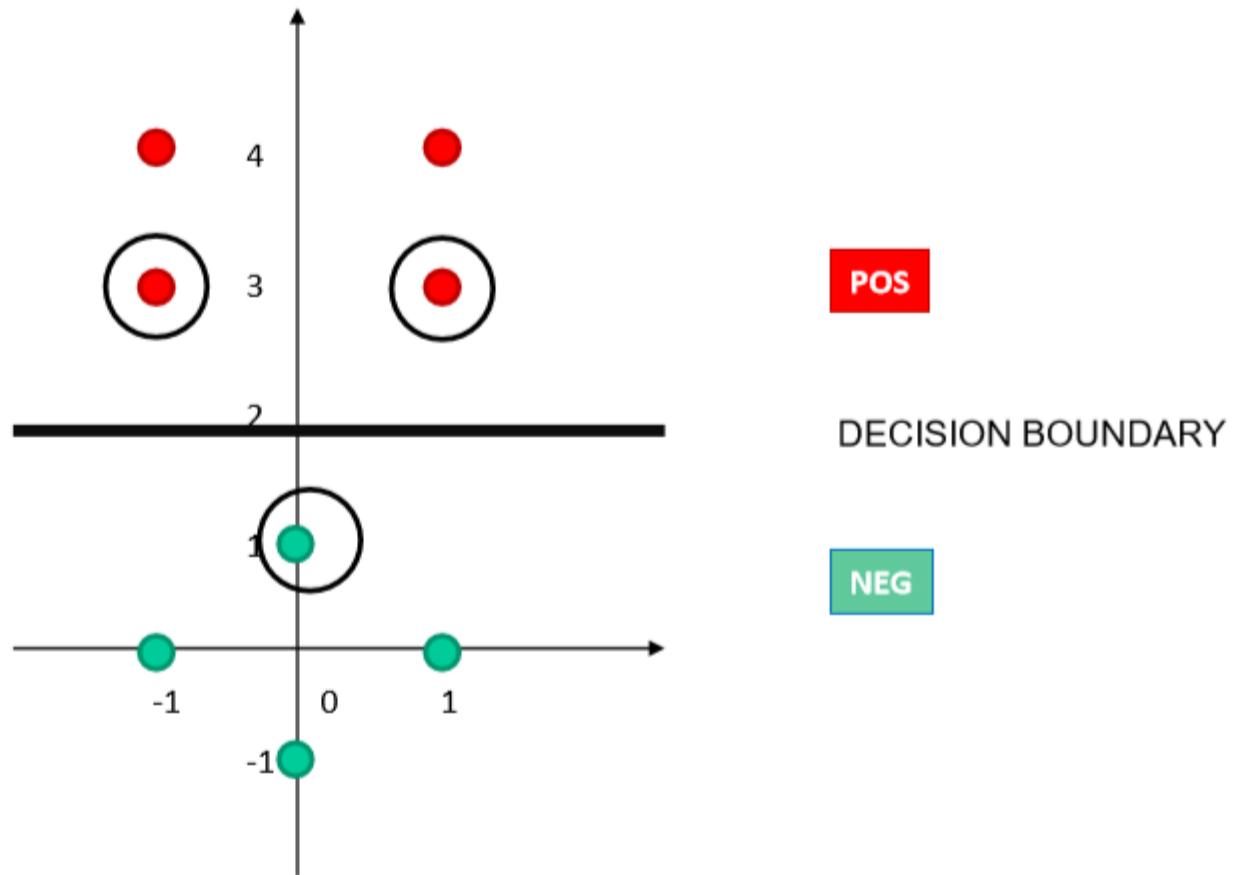
For SVMs, we used this eq for a line: $ax + cy + b = 0$
where $w = [a \ c]$

$$\text{Thus } ax + b = -cy \rightarrow y = (-a/c)x + (-b/c)$$

$$\text{Thus y-intercept is } -(-2)/1 = 2$$

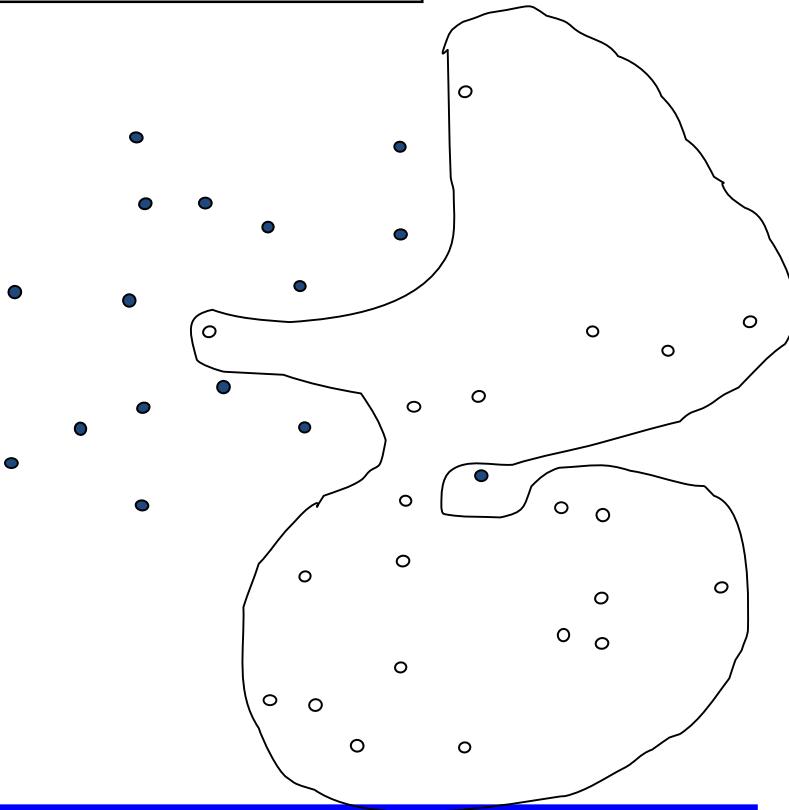
The decision boundary is perpendicular to w and it has slope $-0/1 = 0$

Decision boundary



Dataset with noise

- denotes +1
- denotes -1



- **Hard Margin:** So far we require all data points be classified correctly
 - No training error
- **What if the training set is noisy?**

Soft Margin Classification

Slack variables ξ_i can be added to allow misclassification of difficult or noisy examples.

What should our quadratic optimization criterion be?

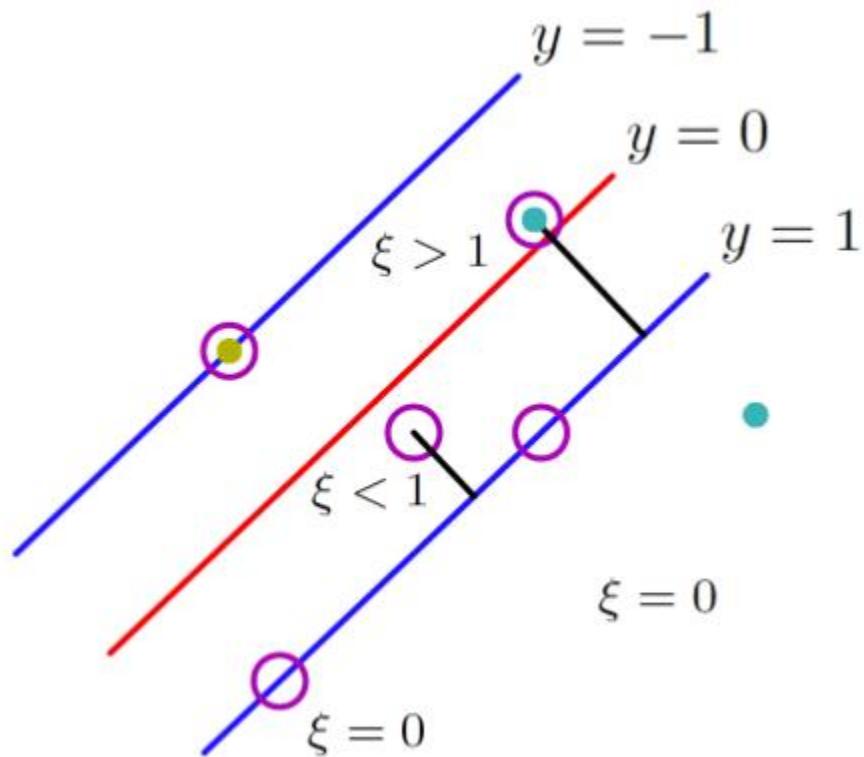
Minimize

$$\frac{1}{2} \mathbf{w} \cdot \mathbf{w} + C \sum_{k=1}^R \xi_k$$

Slack Variable

- **Slack variable** as giving the classifier some leniency when it comes to moving around points near the **margin**.
- When C is large, larger slacks penalize the objective function of SVM's more than when C is small.

Soft margin example



Soft Margin

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i$$

The \mathbf{w} that minimizes...

Maximize margin Minimize misclassification

Misclassification cost # data samples Slack variable

subject to $y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i,$

$\xi_i \geq 0, \quad \forall i = 1, \dots, N$

Hard Margin versus Soft Margin

- **Hard Margin:**

Find \mathbf{w} and b such that

$$\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} \text{ is minimized and for all } \{(\mathbf{x}_i, y_i)\}$$

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$$

- **Soft Margin incorporating slack variables:**

Find \mathbf{w} and b such that

$$\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum \xi_i \text{ is minimized and for all } \{(\mathbf{x}_i, y_i)\}$$

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0 \text{ for all } i$$

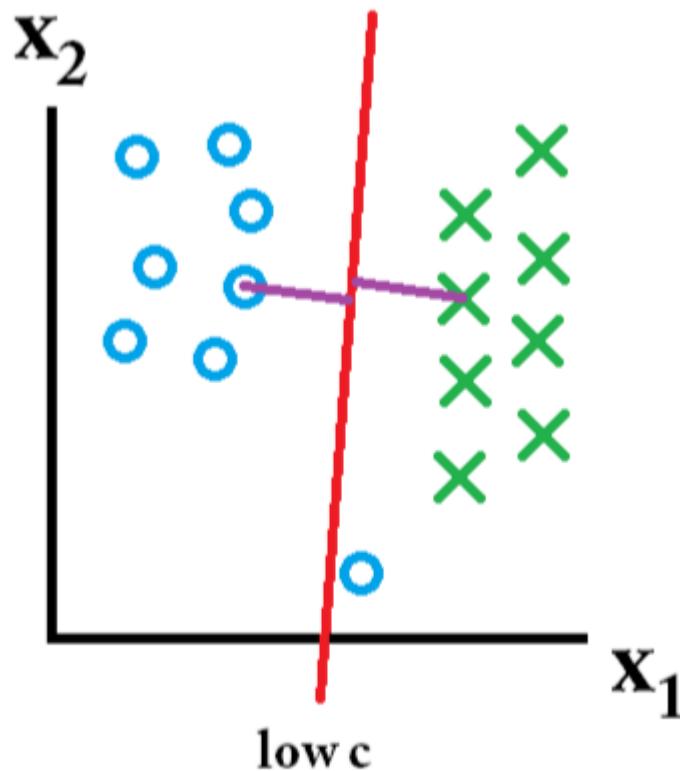
- **Parameter C can be viewed as a way to control overfitting.**

Value of C parameter

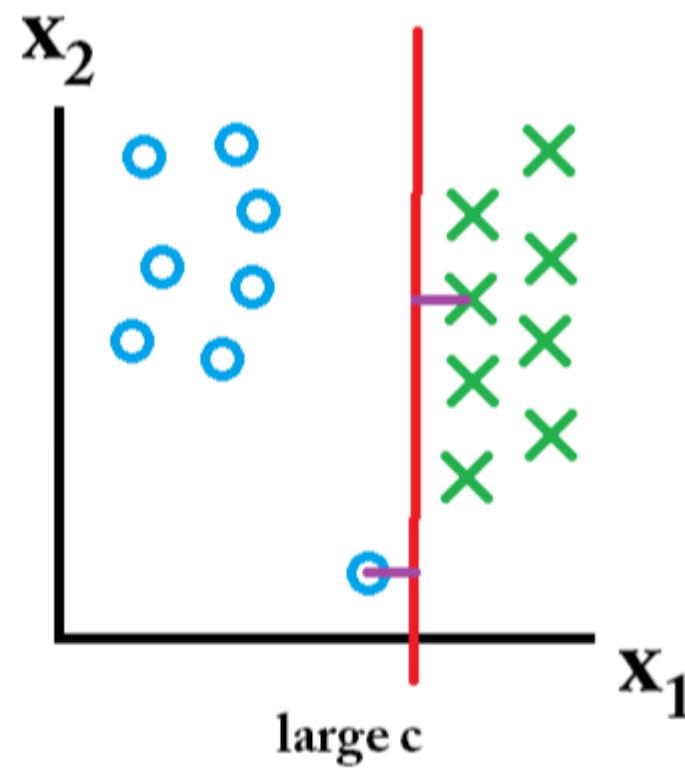
- C parameter tells the SVM optimization how much you want to avoid misclassifying each training example.
 - For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly.
 - Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.
-

Effect of Margin size v/s misclassification cost

Training set



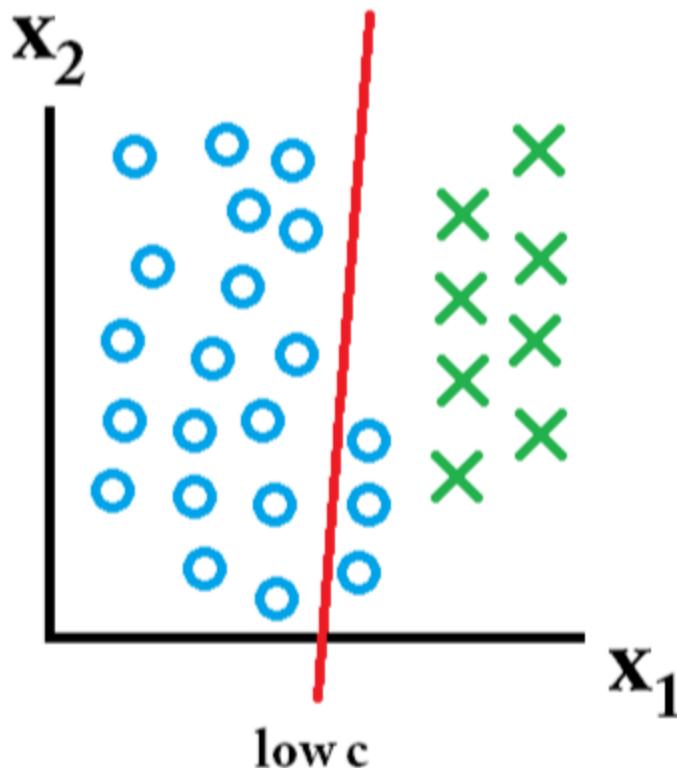
Misclassification ok, want large margin



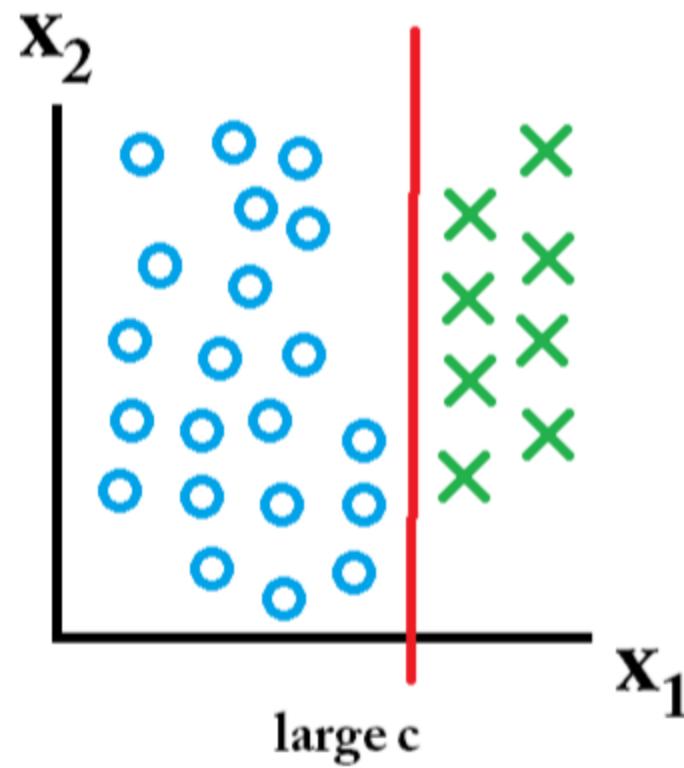
Misclassification not ok

Effect of Margin size v/s misclassification cost

Including test set A



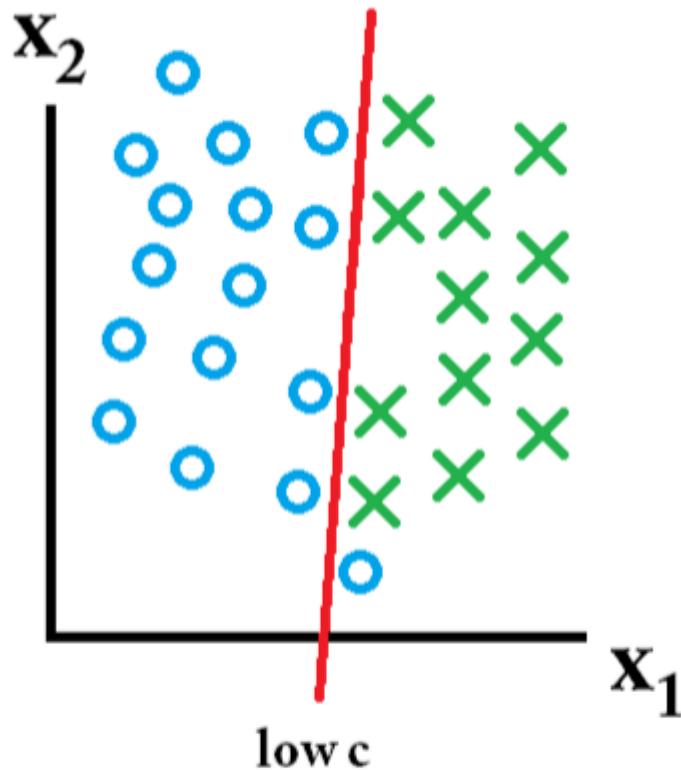
Misclassification ok, want large margin



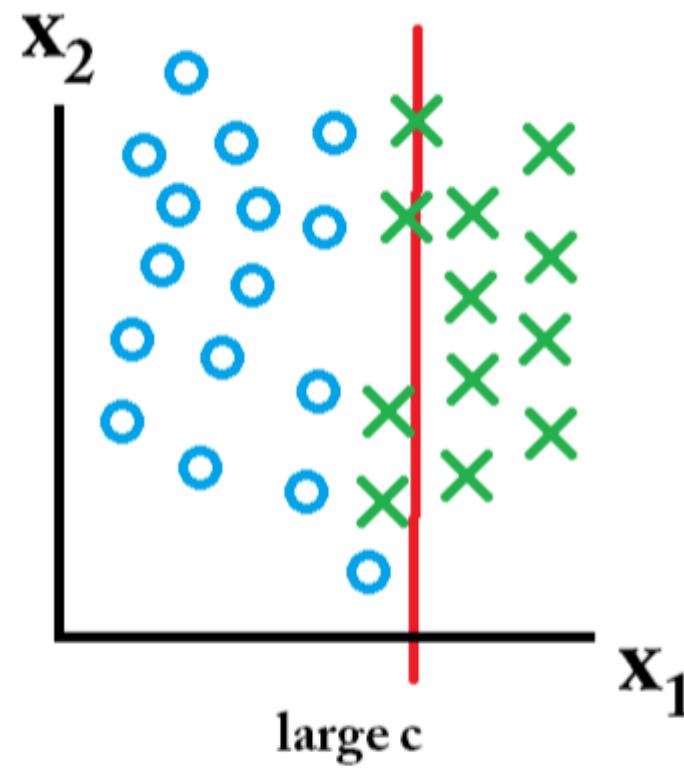
Misclassification not ok

Effect of Margin size v/s misclassification cost

Including test set B



Misclassification ok, want large margin



Misclassification not ok

Linear SVMs: Overview



- The classifier is a *separating hyperplane*.
- Most “important” training points are support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points x_i are support vectors with non-zero Lagrangian multipliers α_i ,

$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$$

Good Web References for SVM

- **Text categorization with Support Vector Machines: learning with many relevant features** - T. Joachims, ECML
- **A Tutorial on Support Vector Machines for Pattern Recognition**, Kluwer Academic Publishers - Christopher J.C. Burges
- <http://www.cs.utexas.edu/users/mooney/cs391L/>
- <https://www.coursera.org/learn/machine-learning/home/week/7>
- <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>
- <https://data-flair.training/blogs/svm-kernel-functions/>
- [MIT 6.034 Artificial Intelligence, Fall 2010](https://mit-6.034-artificial-intelligence-fall-2010.readthedocs.io/en/latest/lectures/lec10.html)
- <https://stats.stackexchange.com/questions/30042/neural-networks-vs-support-vector-machines-are-the-second-definitely-superior>
- <https://www.sciencedirect.com/science/article/abs/pii/S0893608006002796>
- <https://medium.com/deep-math-machine-learning-ai/chapter-3-support-vector-machine-with-math-47d6193c82be>
- [Radial basis kernel](#)



Thank You



BITS Pilani
Pilani Campus

Support Vector Machines

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in



Text Book(s)

- | | |
|----|--|
| T1 | Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition |
| T2 | Tom M. Mitchell: Machine Learning, The McGraw-Hill Companies, Inc.. |

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Tom Mitchell, Prof. Burges, Prof. Andrew Moore and many others who made their course materials freely available online.

Topics to be covered

- Nonlinear SVM
 - Kernel Trick
 - SVM Kernels
 - Multi-Class Problem
 - SVM vs Logistic Regression
 - SVM Applications
-

Solving the Optimization Problem

1. Maximize margin $2/\|\mathbf{w}\|$
2. Correctly classify all training data points:

$$\mathbf{x}_i \text{ positive } (y_i = 1) : \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1) : \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

Quadratic optimization problem:

Find \mathbf{w} and b such that

$\Phi(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2$ is minimized;

and for all $\{(\mathbf{x}_i, y_i)\}$: $y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$

Hinge Loss

- Hinge loss is a loss function used for training classifiers.
- The hinge loss is used for "maximum-margin" classification, most notably for support vector machines (SVMs).
- For an intended output $t = \pm 1$ and a classifier score y , the hinge loss of the prediction y is defined as

$$\ell(y) = \max(0, 1 - t \cdot y)$$

- Hinge loss $\ell(y)=0$ when $|y| >= 0$
- Hinge loss $\ell(y)$ increases linearly with y when $|y| < 0$

Solving the Optimization Problem

- The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every constraint in the primary problem:

$$L(w, b, \alpha_i) = \frac{1}{2} \|w\|^2 - \sum \alpha_i [y_i (w^T x_i + b) - 1]$$

- Taking partial derivative with respect to w , $\frac{\partial L}{\partial w} = 0$
 - $w - \sum \alpha_i y_i x_i = 0$
 - $w = \sum \alpha_i y_i x_i$
- Taking partial derivative with respect to b , $\frac{\partial L}{\partial b} = 0$
 - $-\sum \alpha_i y_i = 0$
 - $\sum \alpha_i y_i = 0$

Solving the Optimization Problem

$$L(w, b, \alpha_i) = \frac{1}{2} \|w\|^2 - \sum \alpha_i [y_i (w^T x_i + b) - 1]$$

- Expanding above equation:

$$L(w, b, \alpha_i) = \frac{1}{2} w^T w - \sum \alpha_i y_i w^T x_i + \sum \alpha_i y_i b + \sum \alpha_i$$

- Substituting $w = \sum \alpha_i y_i x_i$ and $\sum \alpha_i y_i = 0$ in above equation

$$L(w, b, \alpha_i) = \frac{1}{2} (\sum_i \alpha_i y_i x_i)(\sum_j \alpha_j y_j x_j) - (\sum_i \alpha_i y_i x_i)(\sum_j \alpha_j y_j x_j) + \sum \alpha_i$$

$$L(w, b, \alpha_i) = \sum \alpha_i - \frac{1}{2} (\sum_i \alpha_i y_i x_i)(\sum_j \alpha_j y_j x_j)$$

$$L(w, b, \alpha_i) = \sum \alpha_i - \frac{1}{2} (\sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i \cdot x_j)$$

The Dual Problem

- The new objective function is in terms of α_i only
- It is known as the dual problem: if we know \mathbf{w} , we know all α_i ; if we know all α_i , we know \mathbf{w}
- The original problem is known as the primal problem
- The objective function of the dual problem needs to be maximized (comes out from the KKT theory)
- The dual problem is therefore:

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

subject to $\alpha_i \geq 0$,

$$\sum_{i=1}^n \alpha_i y_i = 0$$

Properties of α_i when we introduce the Lagrange multipliers

The result when we differentiate the original Lagrangian w.r.t. b

Optimization Problem

Find \mathbf{w} and b such that

$\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} = \frac{1}{2} \|\mathbf{w}\|^2$ is minimized;

and for all $\{(\mathbf{x}_i, y_i)\}$: $y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$

$$L(w, b, \alpha_i) = \frac{1}{2} \|w\|^2 - \sum \alpha_i [y_i (w^T x_i + b) - 1]$$

Find $\alpha_1 \dots \alpha_N$ such that

$Q(a) = \sum \alpha_i - \frac{1}{2} \left(\sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \right)$ is

maximized and

$$(1) \sum \alpha_i y_i = 0$$

$$(2) \alpha_i \geq 0 \text{ for all } \alpha_i$$

Support Vectors

Using KKT conditions :

$$\alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1] = 0$$

For this condition to be satisfied
either $\alpha_i = 0$ and $y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 > 0$

OR

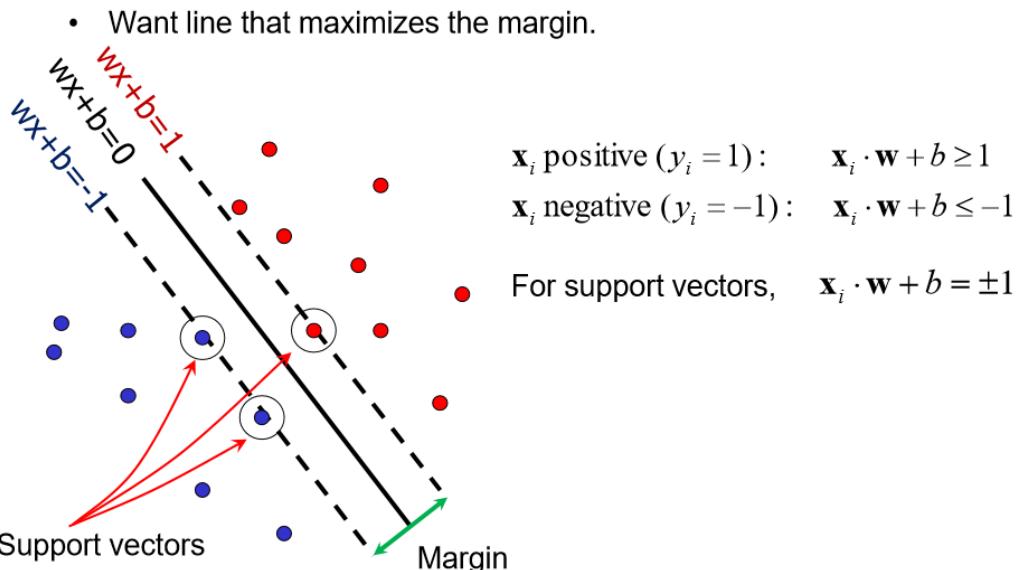
$$y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 = 0 \text{ and } \alpha_i > 0$$

For support vectors:

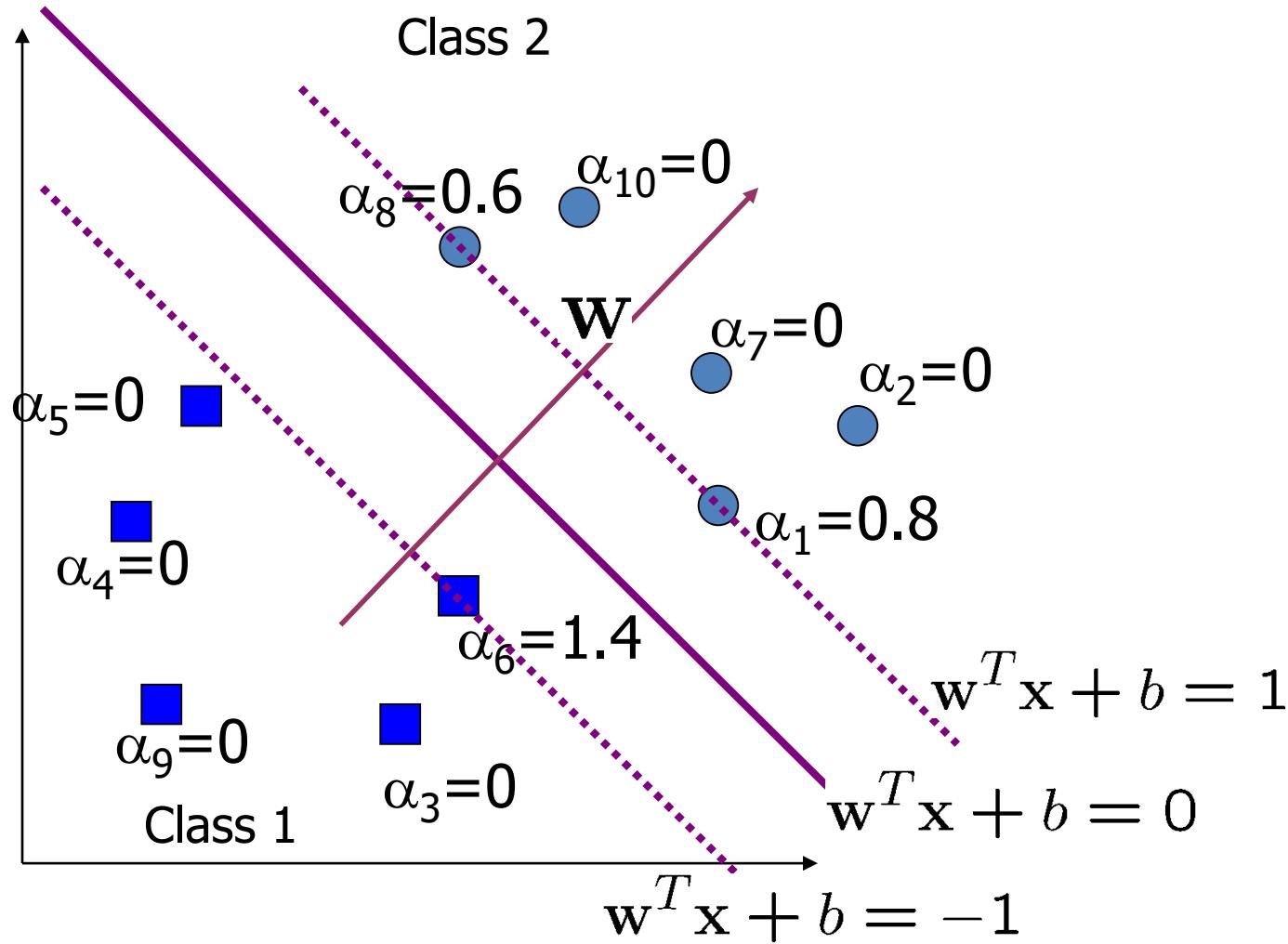
$$y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 = 0$$

For all points other than
support vectors:

$$\alpha_i = 0$$

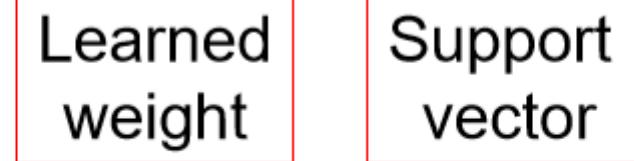


A Geometrical Interpretation



Solving the Optimization Problem

-
- Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$



Solving the Optimization Problem

- Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$
 $b = y_i - \mathbf{w} \cdot \mathbf{x}_i$ (for any support vector)

- Classification function:

$$\begin{aligned}f(x) &= \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) \\&= \text{sign}\left(\sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b\right)\end{aligned}$$

If $f(x) < 0$, classify as negative, otherwise classify as positive.

- Notice that it relies on an *inner product* between the test point \mathbf{x} and the support vectors \mathbf{x}_i
- (Solving the optimization problem also involves computing the inner products $\mathbf{x}_i \cdot \mathbf{x}_j$ between all pairs of training points)

Linear SVMs: Overview

- The classifier is a *separating hyperplane*.
- Most “important” training points are support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points x_i are support vectors with non-zero Lagrangian multipliers α_i .
- Both in the dual formulation of the problem and in the solution training points appear only inside dot products:

Find $\alpha_1 \dots \alpha_N$ such that

$Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j x_i^T x_j$ is maximized and

$$(1) \sum \alpha_i y_i = 0$$

$$(2) 0 \leq \alpha_i \leq C \text{ for all } \alpha_i$$

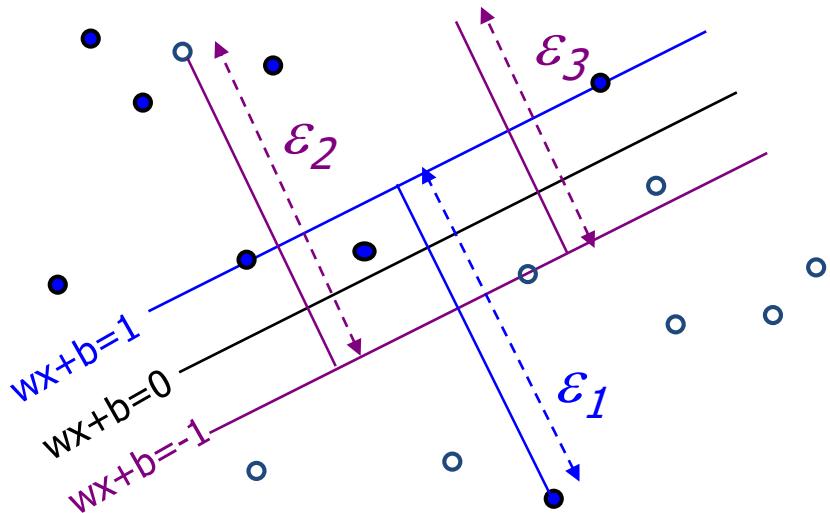
$$f(x) = \sum \alpha_i y_i x_i^T x + b$$

Soft Margin Classification

Slack variables ξ_i can be added to allow misclassification of difficult or noisy examples.

What should our quadratic optimization criterion be?

Minimize



$$\frac{1}{2} \mathbf{w} \cdot \mathbf{w} + C \sum_{k=1}^R \varepsilon_k$$

Soft Margin

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i$$

The \mathbf{w} that minimizes...

Maximize margin Minimize misclassification

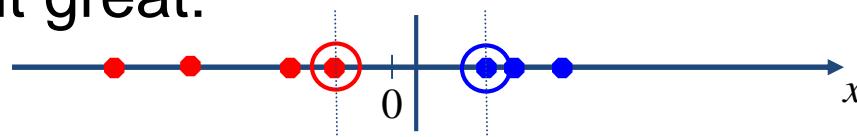
Misclassification cost # data samples Slack variable

subject to $y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i,$

$\xi_i \geq 0, \quad \forall i = 1, \dots, N$

Non-linear SVMs

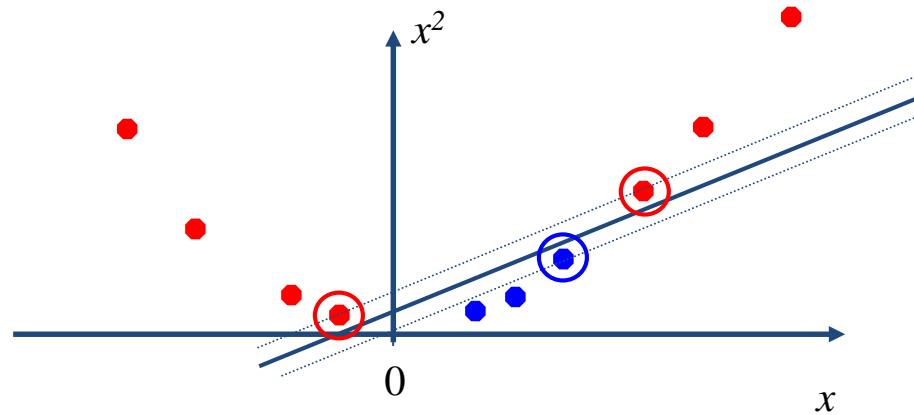
- Datasets that are linearly separable with some noise soft margin work out great:



- But what are we going to do if the dataset is just too hard?



- How about... mapping data to a higher-dimensional space:



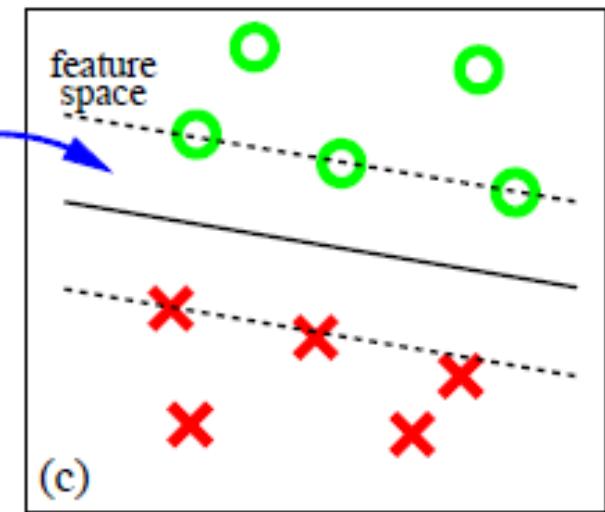
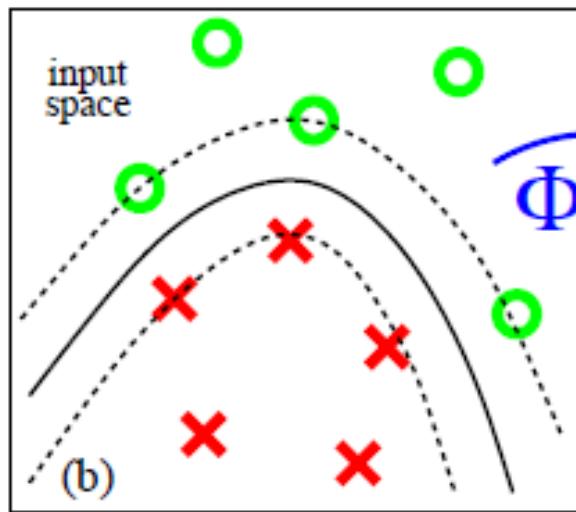
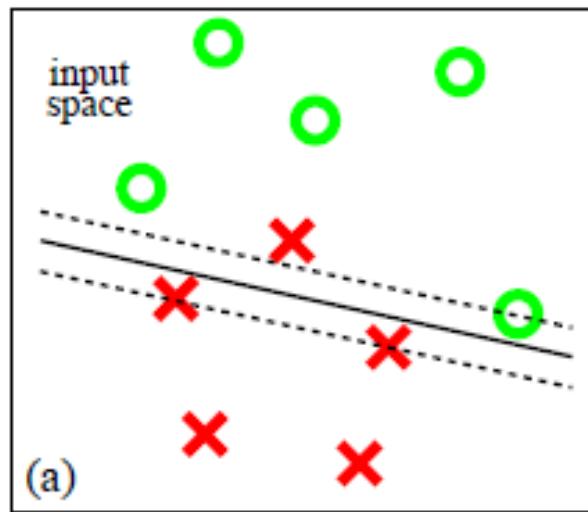
The “Kernel Trick”

- The linear classifier relies on dot product between vectors
 - $\mathbf{x}_i^T \cdot \mathbf{x}_j$
- If every data point is mapped into high-dimensional space via some transformation $\Phi: \mathbf{x} \rightarrow \phi(\mathbf{x})$, the dot product becomes:
$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$
- A *kernel function* is some function that corresponds to an inner product in some expanded feature space.

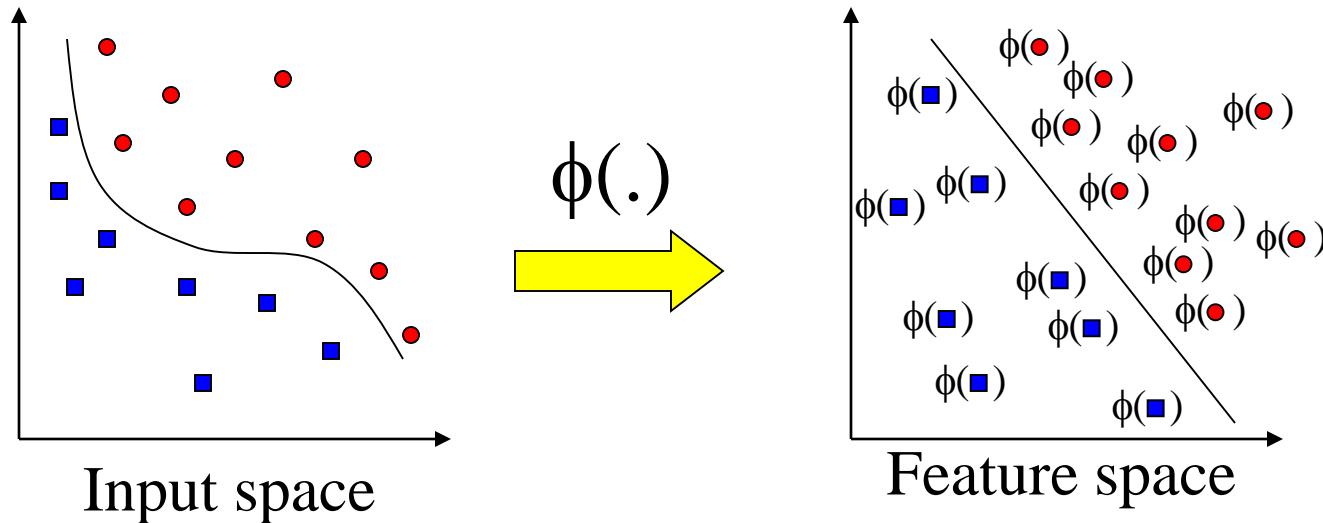
SVM Kernels

- SVM algorithms use a set of mathematical functions that are defined as the kernel.
- Function of kernel is to take data as input and transform it into the required form.
- Different SVM algorithms use different types of kernel functions. Example *linear, nonlinear, polynomial, and sigmoid etc.*

Find a feature space



Transforming the Data

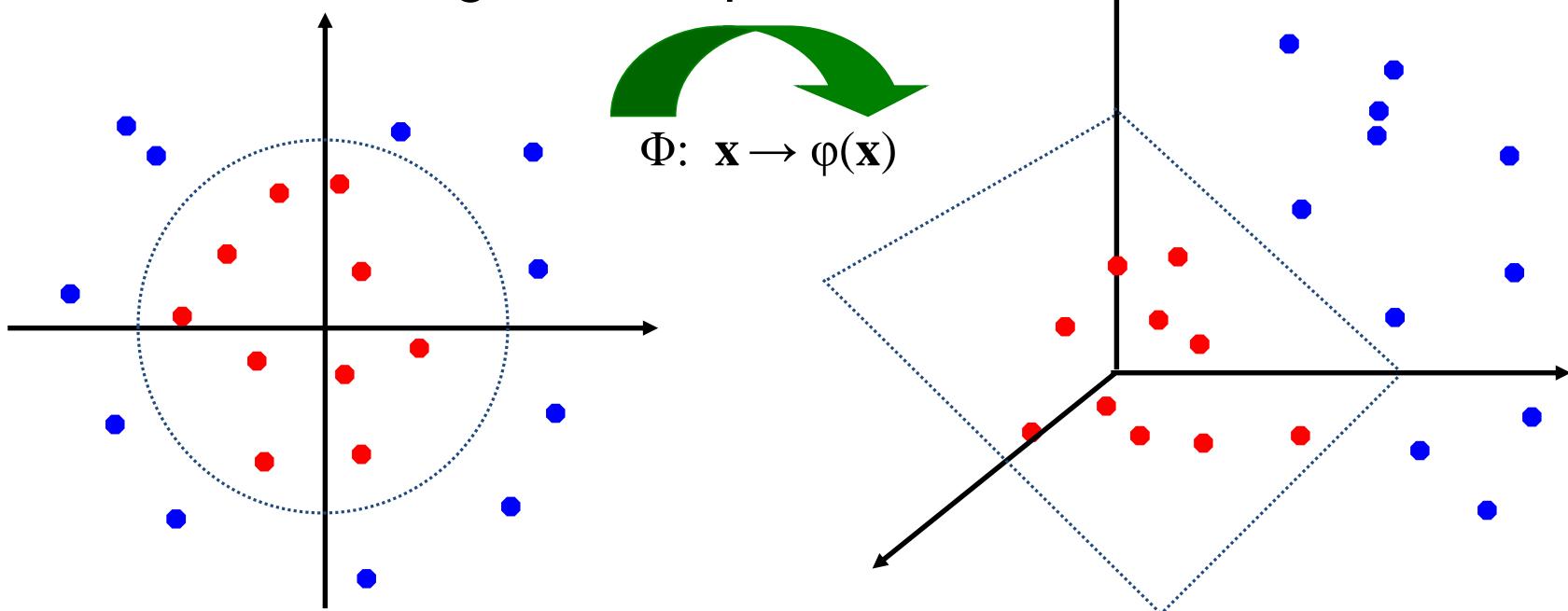


Note: feature space is of higher dimension than the input space in practice

- Computation in the feature space can be costly because it is high dimensional
 - The feature space is typically infinite-dimensional!
- The kernel trick comes to rescue

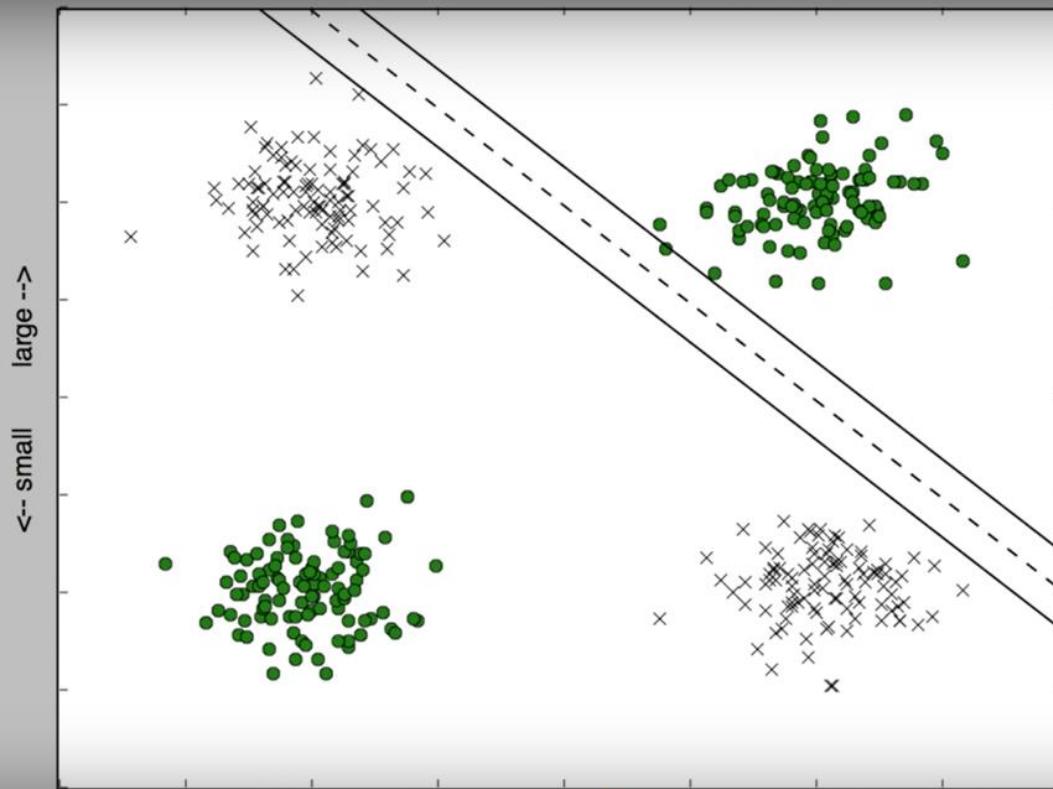
Non-linear SVMs: Feature spaces

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:



Non-linear SVMs

How Support Vector Machines work / How to open a black box



https://www.youtube.com/watch?v=Lpr_X8zuE8

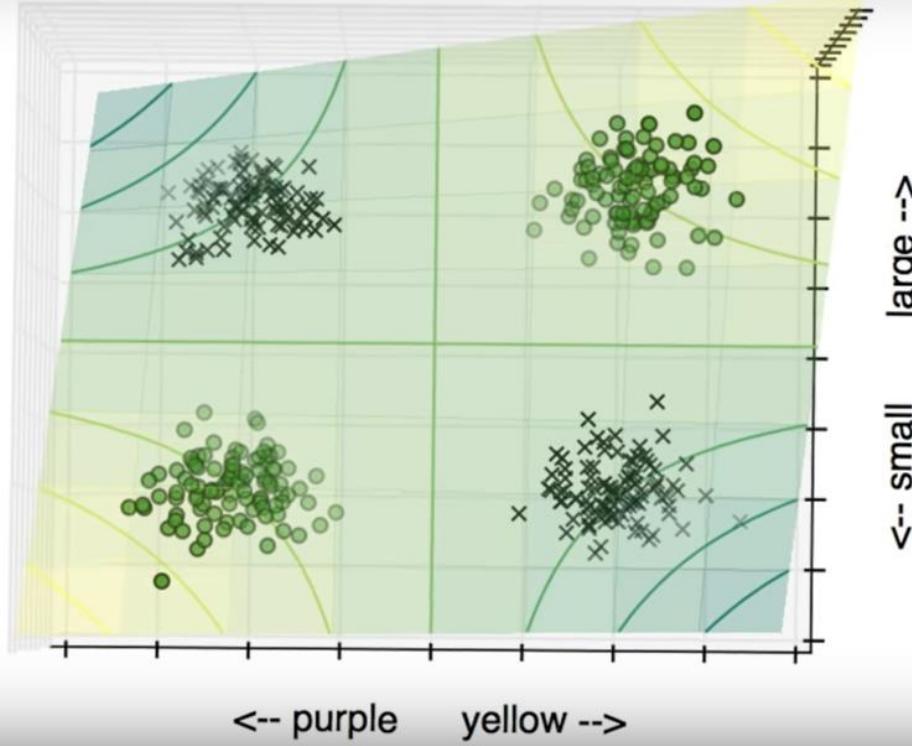
9:38 / 17:53

<-- purple yellow -->

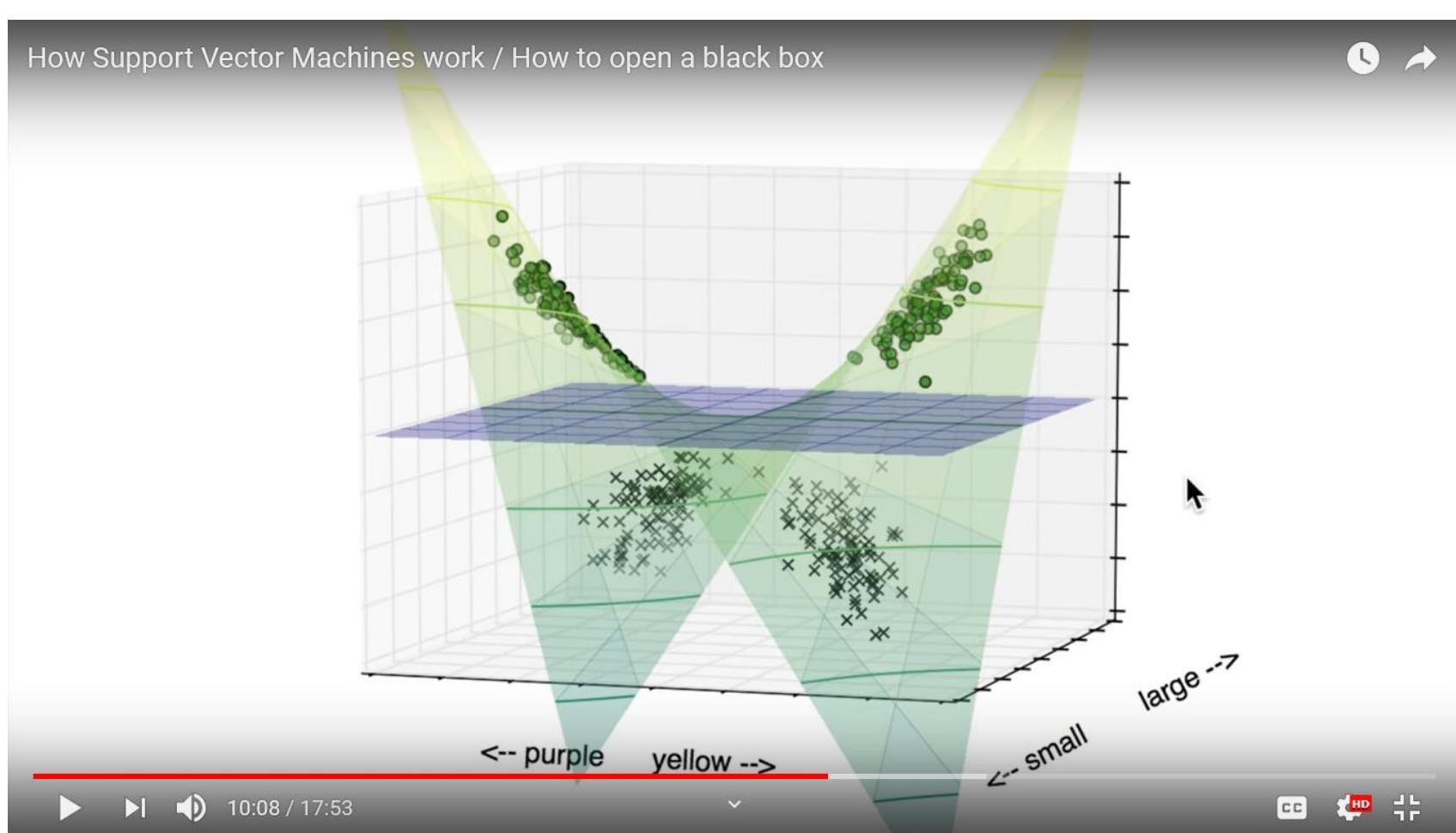
CC HD

Non-linear SVMs: Feature spaces

How Support Vector Machines work / How to open a black box



Non-linear SVMs: Feature spaces



SVM – Overlapping Class Scenario

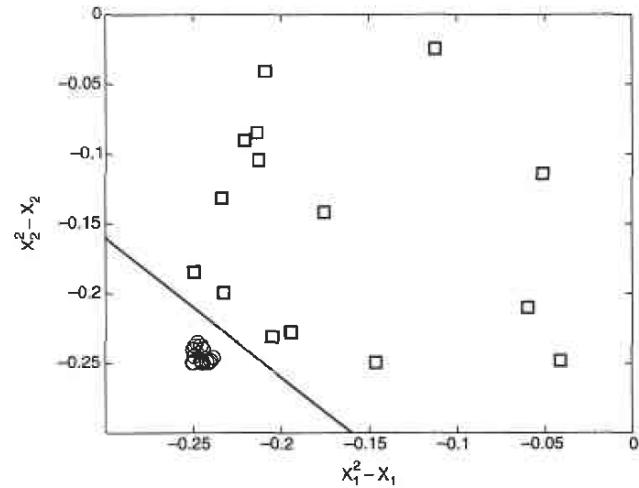
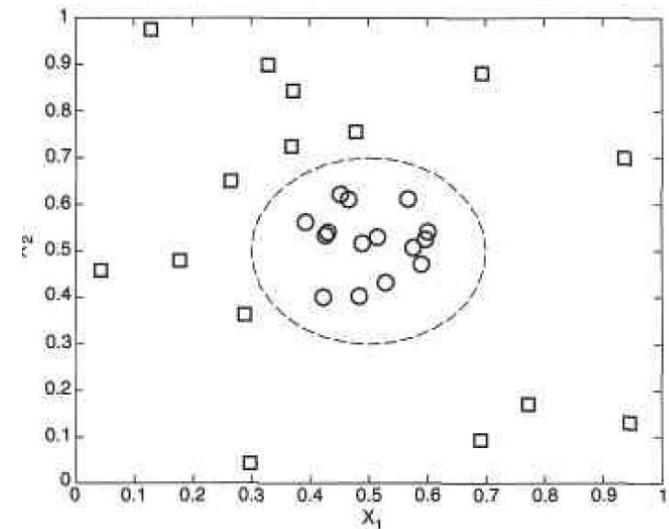
- Data is not separable linearly
- Margin will become inefficient
- Data needs to be transformed from original coordinate space \mathbf{x} to a new space $\Phi(\mathbf{x})$, so that linear decision boundary can be applied
- A non-linear transformation function is needed, like, ex:

$$\Phi : (x_1, x_2) \longrightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$$

- In the transformed space we can choose $w = (w_0, w_1, \dots, w_4)$ such that

$$w_4x_1^2 + w_3x_2^2 + w_2\sqrt{2}x_1 + w_1\sqrt{2}x_2 + w_0 = 0.$$

- The linear decision boundary in the transformed space has the following form: $w \cdot \Phi(\mathbf{x}) + b = 0$



What Functions are Kernels?

- Kernel is a continuous function $k(x,y)$ that takes two arguments x and y (real numbers, functions, vectors, etc.) and maps them to a real value independent of the order of the arguments, i.e., $k(x,y)=k(y,x)$.
- For some functions $K(x_i, x_j)$ checking that $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ can be cumbersome.
- Mercer's theorem:
Every positive-semidefinite symmetric function is a kernel

What Functions are Kernels?

1) We can *construct kernels from scratch*:

- For any $\varphi : \mathcal{X} \rightarrow \mathbb{R}^m$, $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathbb{R}^m}$ is a kernel.
- If $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a *distance function*, i.e.
 - $d(x, x') \geq 0$ for all $x, x' \in \mathcal{X}$,
 - $d(x, x') = 0$ only for $x = x'$,
 - $d(x, x') = d(x', x)$ for all $x, x' \in \mathcal{X}$,
 - $d(x, x') \leq d(x, x'') + d(x'', x')$ for all $x, x', x'' \in \mathcal{X}$,

then $k(x, x') := \exp(-d(x, x'))$ is a kernel.

2) We can *construct kernels from other kernels*:

- if k is a kernel and $\alpha > 0$, then αk and $k + \alpha$ are kernels.
- if k_1, k_2 are kernels, then $k_1 + k_2$ and $k_1 \cdot k_2$ are kernels.

Examples of Kernel Functions

- Linear: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
- Polynomial of power p : $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$
- Gaussian (radial-basis function network):

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

- Sigmoid: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta_0 \mathbf{x}_i^T \mathbf{x}_j + \beta_1)$

Name	Function	Type problem
Polynomial Kernel	$(x_i^T x_j + 1)^q$ q is degree of polynomial	Best for Image processing
Sigmoid Kernel	$\tanh(ax_i^T x_j + k)$ k is offset value	Very similar to neural network
Gaussian Kernel	$\exp(-\ x_i - x_j\ ^2 / 2\sigma^2)$	No prior knowledge on data
Linear Kernel	$(1 + x_i^T x_j) \min(x_i, x_j) - \frac{(x_i + x_j)}{2} \min(x_i, x_j)^2 + \frac{\min(x_i, x_j)^3}{3}$	Text Classification
Laplace Radial Basis Function (RBF)	$(e^{-\lambda \ x_i - x_j\ }), \lambda >= 0$	No prior knowledge on data

There are many more kernel functions.

Non-linear SVMs Mathematically

- The solution is:

$$f(\mathbf{x}) = \sum \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_j) + b$$

- Optimization techniques for finding α_i 's remain the same!

Non-linear SVM using kernel

1. Select a kernel function.
2. Compute pairwise kernel values between labeled examples.
3. Use this “kernel matrix” to solve for SVM support vectors & alpha weights.
4. To classify a new example: compute kernel values between new input and support vectors, apply alpha weights, check sign of output.

Nonlinear SVM - Overview

- SVM locates a separating hyperplane in the feature space and classify points in that space
- It does not need to represent the space explicitly, simply by defining a kernel function
- The kernel function plays the role of the dot product in the feature space.

Multi-Class Problem

Instead of just two classes, we now have C classes

- E.g. predict which movie genre a viewer likes best
- Possible answers: action, drama, indie, thriller, etc.

Two approaches:

- One-vs-all
- One-vs-one

Multi-Class Problem

Instead of just two classes, we now have C classes

- E.g. predict which movie genre a viewer likes best
- Possible answers: action, drama, indie, thriller, etc.

Two approaches:

- One-vs-all
- One-vs-one

Multi-Class Problem

One-vs-all (a.k.a. one-vs-others)

- Train C classifiers
- In each, pos = data from class i , neg = data from classes other than i
- The class with the most confident prediction wins
- Example:
 - You have 4 classes, train 4 classifiers
 - 1 vs others: score 3.5
 - 2 vs others: score 6.2
 - 3 vs others: score 1.4
 - 4 vs other: score 5.5
 - Final prediction: class 2
- Issues?

Multi-Class Problem

One-vs-one (a.k.a. all-vs-all)

- Train $C(C-1)/2$ binary classifiers (all pairs of classes)
- They all vote for the label
- Example:
 - You have 4 classes, then train 6 classifiers
 - 1 vs 2, 1 vs 3, 1 vs 4, 2 vs 3, 2 vs 4, 3 vs 4
 - Votes: 1, 1, 4, 2, 4, 4
 - Final prediction is class 4

SVM versus Logistic Regression

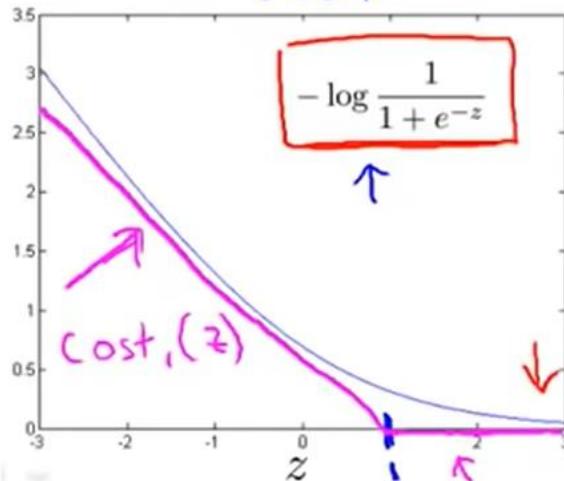
Alternative view of logistic regression

Cost of example: $-(y \log h_\theta(x) + (1 - y) \log(1 - h_\theta(x))) \leftarrow$

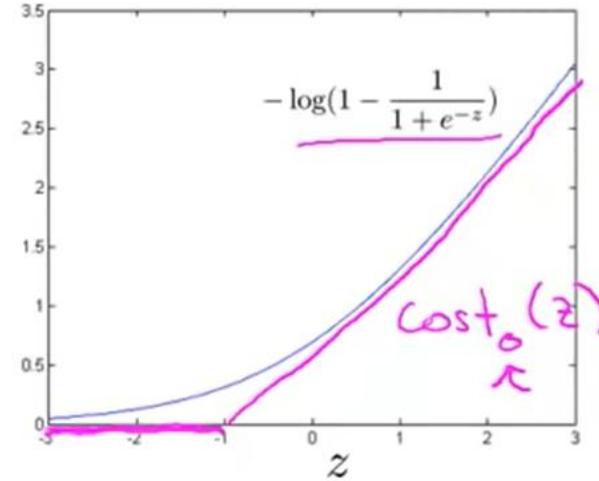
$$= \boxed{-y \log \frac{1}{1 + e^{-\theta^T x}}} - \boxed{(1 - y) \log(1 - \frac{1}{1 + e^{-\theta^T x}})} \leftarrow$$

If $y = 1$ (want $\theta^T x \gg 0$):

$$z = \Theta^T x$$



If $y = 0$ (want $\theta^T x \ll 0$):



Andrew Ng

SVM versus Logistic Regression

Support vector machine

Logistic regression:

$$\min_{\theta} \frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \underbrace{\left(-\log h_{\theta}(x^{(i)}) \right)}_{\text{cost}_1(\theta^T x^{(i)})} + (1 - y^{(i)}) \underbrace{\left(-\log(1 - h_{\theta}(x^{(i)})) \right)}_{\text{cost}_0(\theta^T x^{(i)})} \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$


Support vector machine:

$$\min_{\theta} \underbrace{C \sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)}) \text{cost}_0(\theta^T x^{(i)})}_{A} + \frac{1}{2} \sum_{j=0}^n \theta_j^2 \quad B$$

$$\min_u \frac{(u-s)^2 + 1}{10} \rightarrow u=5 \quad | \quad A + \frac{\lambda}{2} B \leftarrow C = \frac{1}{\lambda}$$

$$\min_u \log(u-s)^2 + 10 \rightarrow u=5 \quad | \quad C \leftarrow A + B \leftarrow$$

$$\min_{\theta} C \sum_{i=1}^m \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^n \theta_j^2$$

SVM Hypothesis

SVM hypothesis

$$\rightarrow \min_{\theta} C \sum_{i=1}^m \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

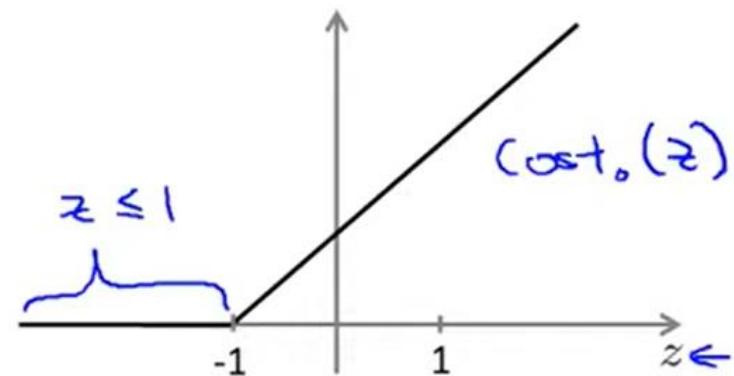
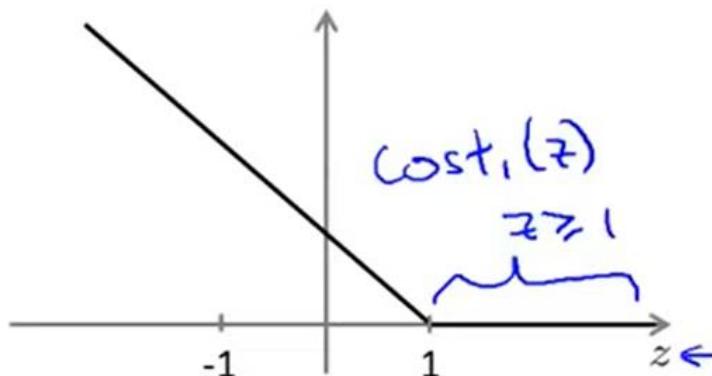
Hypothesis:

$$h_{\theta}(x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

SVM Cost/Loss Function

Support Vector Machine

$$\rightarrow \min_{\theta} C \sum_{i=1}^m \left[y^{(i)} \underline{\text{cost}_1(\theta^T x^{(i)})} + (1 - y^{(i)}) \underline{\text{cost}_0(\theta^T x^{(i)})} \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$



→ If $y = 1$, we want $\underline{\theta^T x \geq 1}$ (not just ≥ 0)

$\theta^T x \geq \cancel{0} 1$

→ If $y = 0$, we want $\underline{\theta^T x \leq -1}$ (not just < 0)

$\theta^T x \leq \cancel{0} -1$

$$C = 100,000$$

SVM Decision boundary

SVM Decision Boundary

$$\min_{\theta} C \sum_{i=1}^m \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

Whenever $y^{(i)} = 1$: $\theta^T x^{(i)} \geq 0$

$$\theta^T x^{(i)} \geq 1$$

$$\min \cancel{C \times 0} + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

$$\text{s.t. } \theta^T x^{(i)} \geq 1 \quad \text{if } y^{(i)} = 1$$

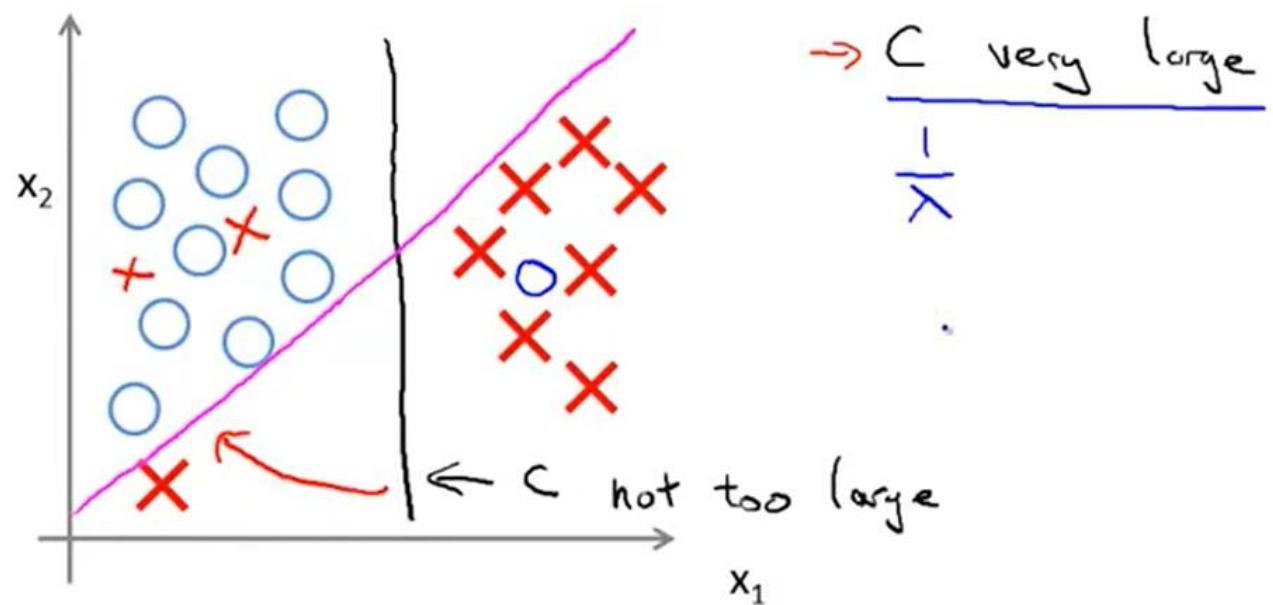
$$\theta^T x^{(i)} \leq -1 \quad \text{if } y^{(i)} = 0$$

Whenever $y^{(i)} = 0$:

$$\theta^T x^{(i)} \leq -1$$

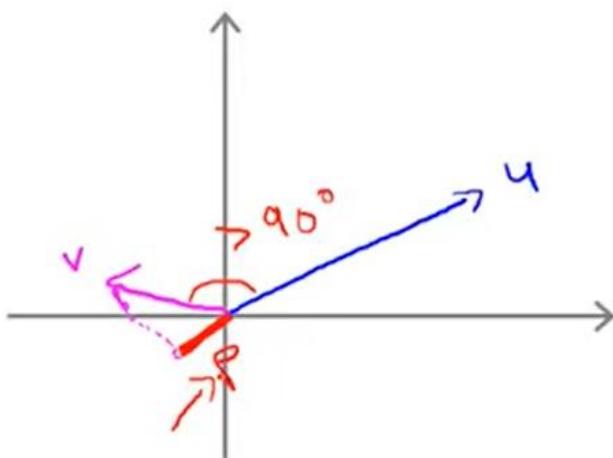
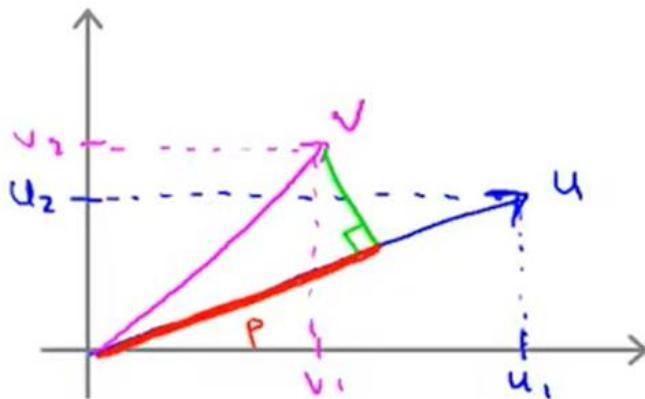
Handling outliers

Large margin classifier in presence of outliers



Norm of a vector

Vector Inner Product



$$\rightarrow u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad \rightarrow v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$u^T v = ? \quad [u_1 \ u_2] \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$\|u\| = \text{length of vector } u \\ = \sqrt{u_1^2 + u_2^2} \in \mathbb{R}$$

$p = \text{length of projection of } v \text{ onto } u.$

$$u^T v = \frac{p \cdot \|u\|}{\|u\|} \leftarrow = v^T u$$

Signed

$$= u_1 v_1 + u_2 v_2 \leftarrow p \in \mathbb{R}$$

$$u^T v = p \cdot \|u\|$$

$$p < 0$$

Andrew Ng

SVM Decision boundary

$$\omega = (\sqrt{\omega^T \omega})^2$$

SVM Decision Boundary

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2 = \frac{1}{2} (\Theta_1^2 + \Theta_2^2) = \frac{1}{2} \left(\sqrt{\Theta_1^2 + \Theta_2^2} \right)^2 = \frac{1}{2} \|\theta\|^2$$

s.t. $\theta^T x^{(i)} \geq 1 \quad \text{if } y^{(i)} = 1$

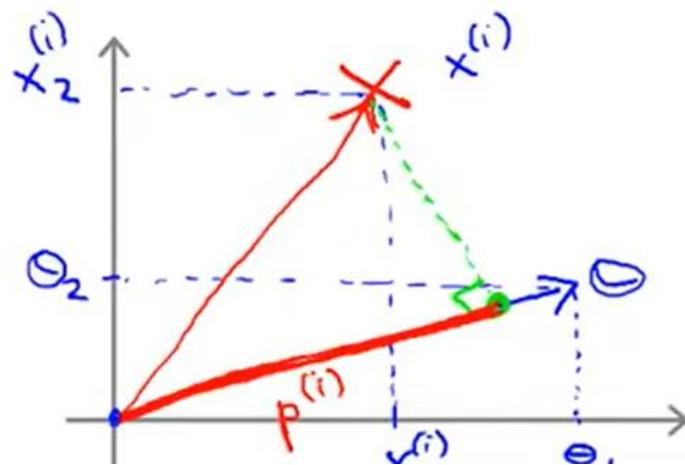
$\rightarrow \theta^T x^{(i)} \leq -1 \quad \text{if } y^{(i)} = 0$

Simplification: $\Theta_0 = 0$. n=2

$$\begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}, \Theta_0 = 0$$

$$\theta^T x^{(i)} = ?$$

↑
U^T V



$$\begin{aligned} \theta^T x^{(i)} &= p^{(i)} \|\theta\| \leftarrow \\ &= \Theta_1 x_1^{(i)} + \Theta_2 x_2^{(i)} \leftarrow \end{aligned}$$

SVM Decision boundary

SVM Decision Boundary

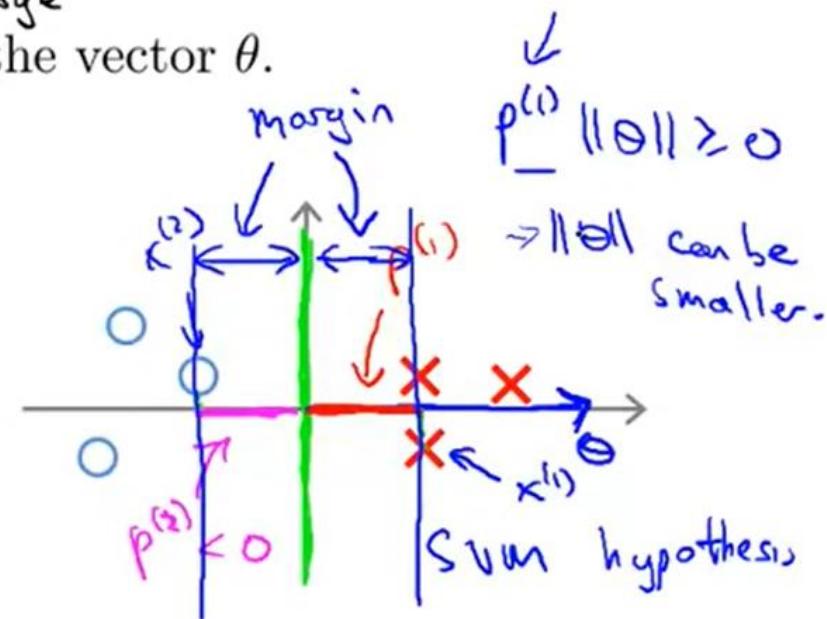
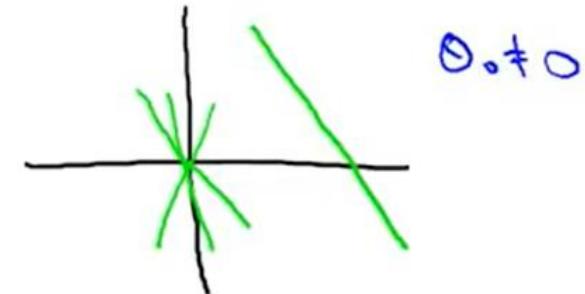
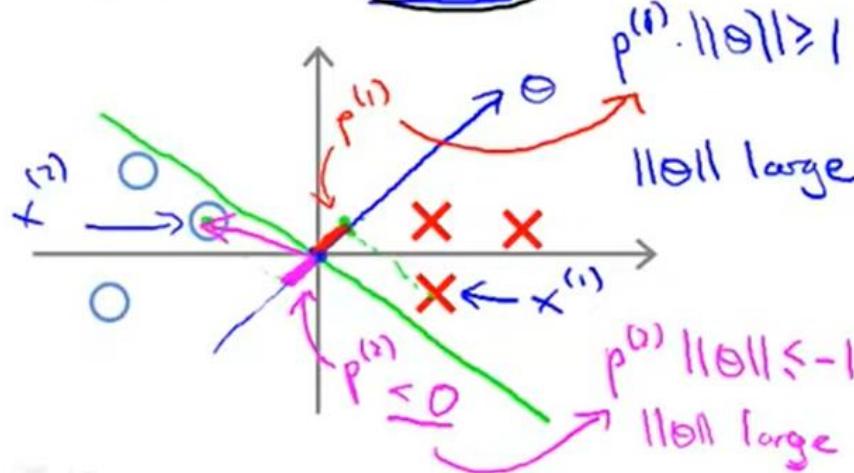
$$\Rightarrow \min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2 = \frac{1}{2} \|\theta\|^2 \leftarrow$$

s.t. $\boxed{p^{(i)} \cdot \|\theta\| \geq 1}$ if $y^{(i)} = 1$

 $p^{(i)} \cdot \|\theta\| \leq -1 \quad \text{if } y^{(i)} = -1$

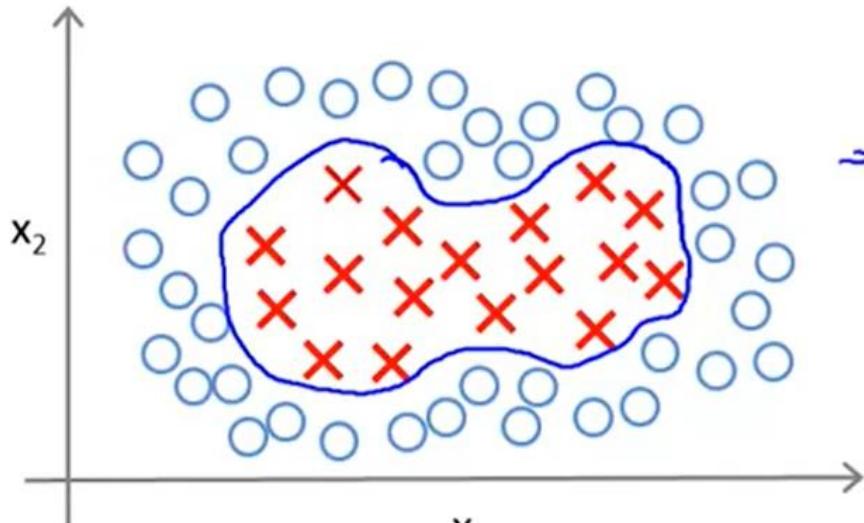
where $p^{(i)}$ is the projection of $x^{(i)}$ onto the vector θ .

Simplification: $\theta_0 = 0 \leftarrow$



SVM Non-linear Decision boundary

Non-linear Decision Boundary



Predict $y = 1$ if

$$\theta_0 + \theta_1 \underline{x_1} + \theta_2 \underline{x_2} + \theta_3 \underline{x_1 x_2} + \theta_4 \underline{x_1^2} + \theta_5 \underline{x_2^2} + \dots \geq 0$$

$$h_{\theta}(x) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \dots \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \dots$$

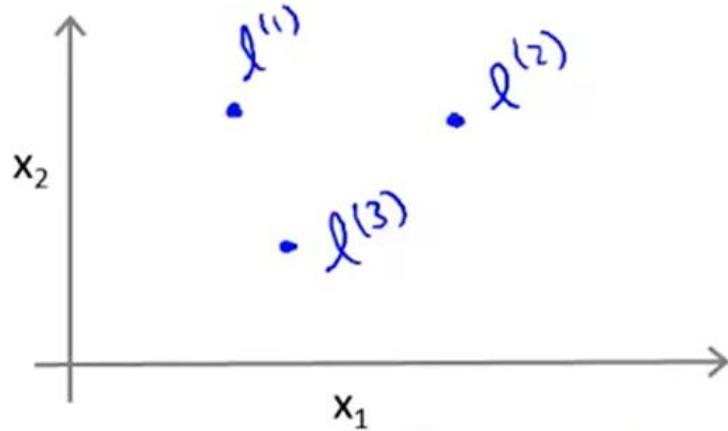
$$f_1 = x_1, \quad f_2 = x_2, \quad f_3 = x_1 x_2, \quad f_4 = x_1^2, \quad f_5 = x_2^2, \dots$$

Is there a different / better choice of the features f_1, f_2, f_3, \dots ?

Andrew Ng

SVM Non-linear Decision boundary

Kernel



Given x , compute new feature depending on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$

Given x :

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

$$f_2 = \text{similarity}(x, l^{(2)}) = \exp\left(-\frac{\|x - l^{(2)}\|^2}{2\sigma^2}\right)$$

$$f_3 = \text{similarity}(x, l^{(3)}) = \exp(\dots)$$

kernel (Gaussian kernels)

$$k(x, l^{(i)})$$

$$\|w\|$$

$$\|x - l^{(1)}\|^2$$

$$\|x - l^{(1)}\|^2$$

SVM Kernels

Kernels and Similarity

$$f_1 = \text{similarity}(x, \underline{l^{(1)}}) = \exp\left(-\frac{\|x - \underline{l^{(1)}}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^n (x_j - l_j^{(1)})^2}{2\sigma^2}\right)$$

If $x \approx l^{(1)}$:

$$f_1 \underset{\uparrow}{\approx} \exp\left(-\frac{0^2}{2\sigma^2}\right) \underset{\downarrow}{\approx} 1$$

$$\begin{aligned} l^{(1)} &\rightarrow f_1 \\ l^{(2)} &\rightarrow f_2 \\ l^{(3)} &\rightarrow f_3. \\ \uparrow & \uparrow \\ x & \end{aligned}$$

If x if far from $\underline{l^{(1)}}$:

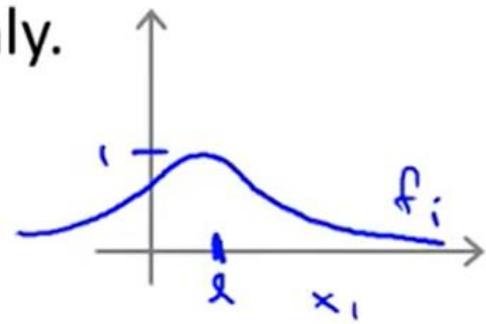
$$f_1 = \exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \underset{\uparrow}{\approx} 0.$$

SVM parameters:

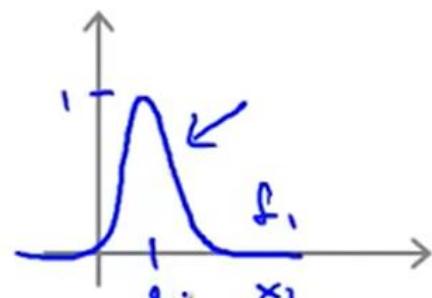
$C \left(= \frac{1}{\lambda} \right)$. \rightarrow Large C: Lower bias, high variance. (small λ)
 \rightarrow Small C: Higher bias, low variance. (large λ)

σ^2 $\underline{\text{Large } \sigma^2}$: Features f_i vary more smoothly.
 \rightarrow Higher bias, lower variance.

$$\exp \left(- \frac{\|x - \mu^{(i)}\|^2}{2\sigma^2} \right)$$



$\underline{\text{Small } \sigma^2}$: Features f_i vary less smoothly.
 Lower bias, higher variance.

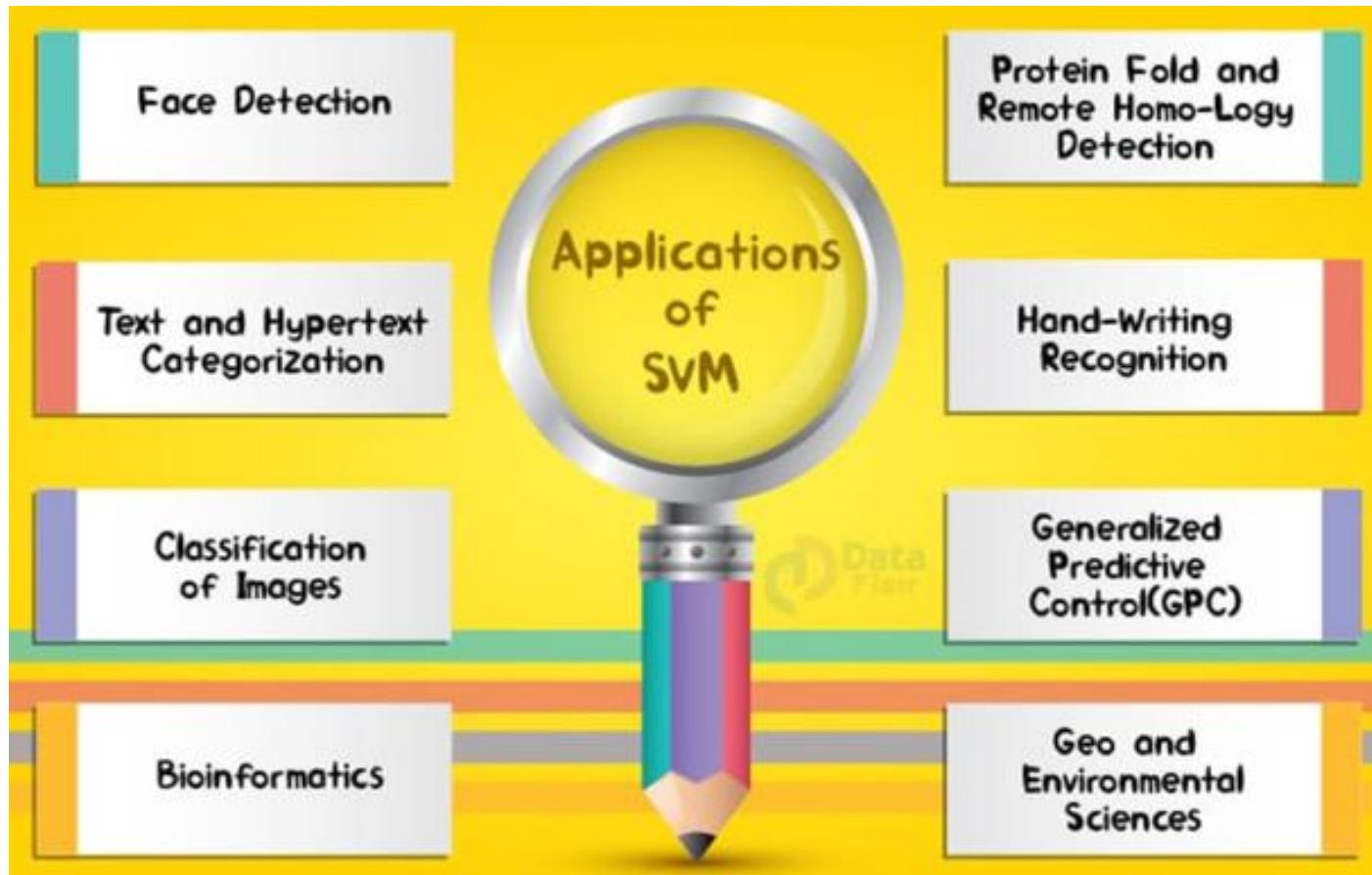


Properties of SVM

- **Flexibility in choosing a similarity function**
- **Sparseness of solution when dealing with large data sets**
 - Only support vectors are used to specify the separating hyperplane
 - Therefore SVM also called sparse kernel machine.
- **Ability to handle large feature spaces**
 - complexity does not depend on the dimensionality of the feature space
- **Overfitting can be controlled by soft margin approach**
- **Nice math property: a simple convex optimization problem which is guaranteed to converge to a single global solution**
- **Feature Selection**

SVM Applications

SVM has been used successfully in many real-world problems



Application : Text Categorization

- Task: The classification of natural text (or hypertext) documents into a fixed number of predefined categories based on their content. A document can be assigned to more than one category, so this can be viewed as a series of binary classification problems, one for each category

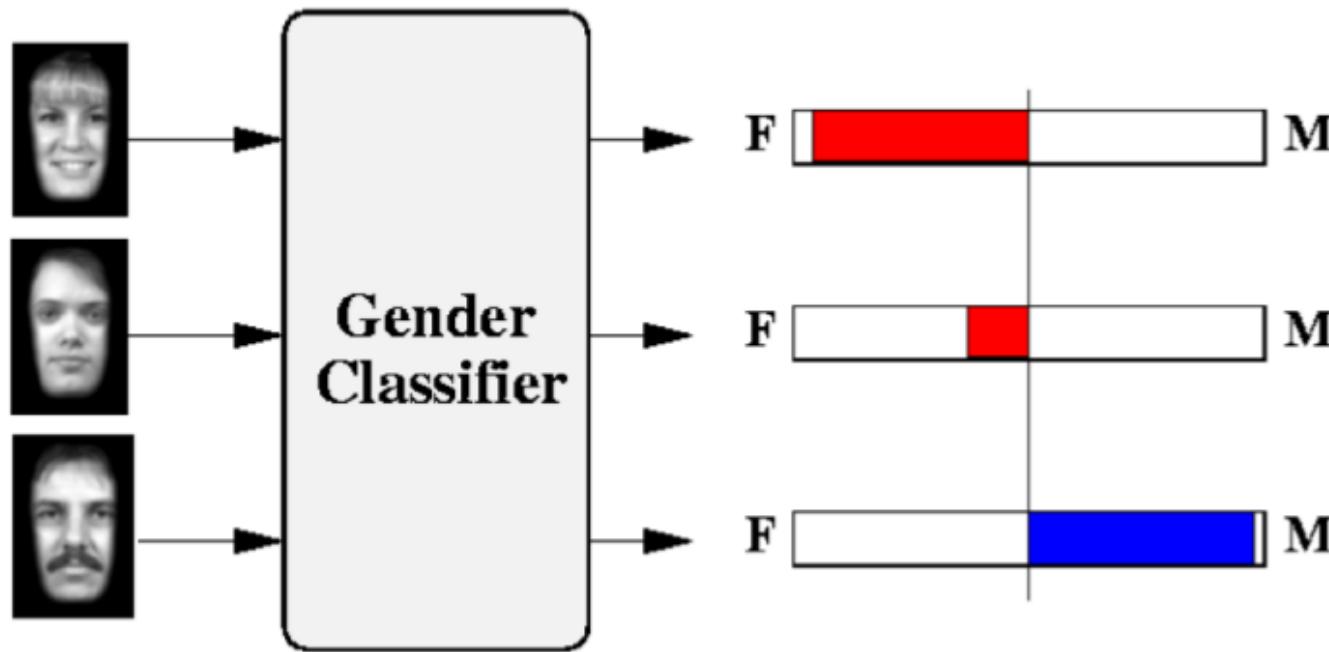
Text Categorization using SVM

- The distance between two documents is $\phi(x) \cdot \phi(z)$
- $K(x,z) = \phi(x) \cdot \phi(z)$ is a valid kernel, SVM can be used with $K(x,z)$ for discrimination.
- Why SVM?
 - High dimensional input space
 - Few irrelevant features (dense concept)
 - Sparse document vectors (sparse instances)
 - Text categorization problems are linearly separable

Using SVM

1. Select a kernel function.
 2. Compute pairwise kernel values between labeled examples.
 3. Use this “kernel matrix” to solve for SVM support vectors & alpha weights.
 4. To classify a new example: compute kernel values between new input and support vectors, apply alpha weights, check sign of output.
-

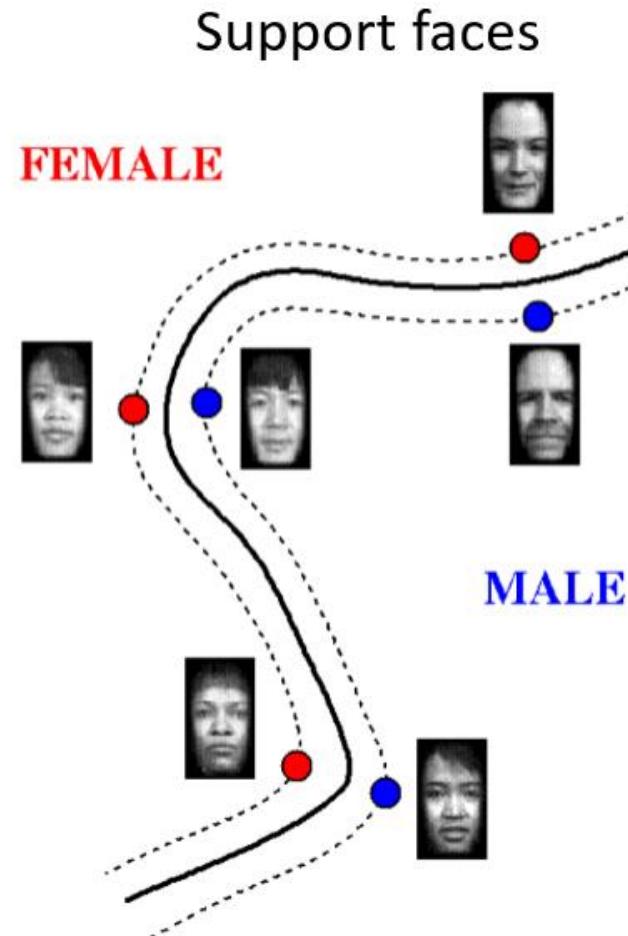
Learning Gender from image with SVM



Moghaddam and Yang, Learning Gender with Support Faces, TPAMI 2002

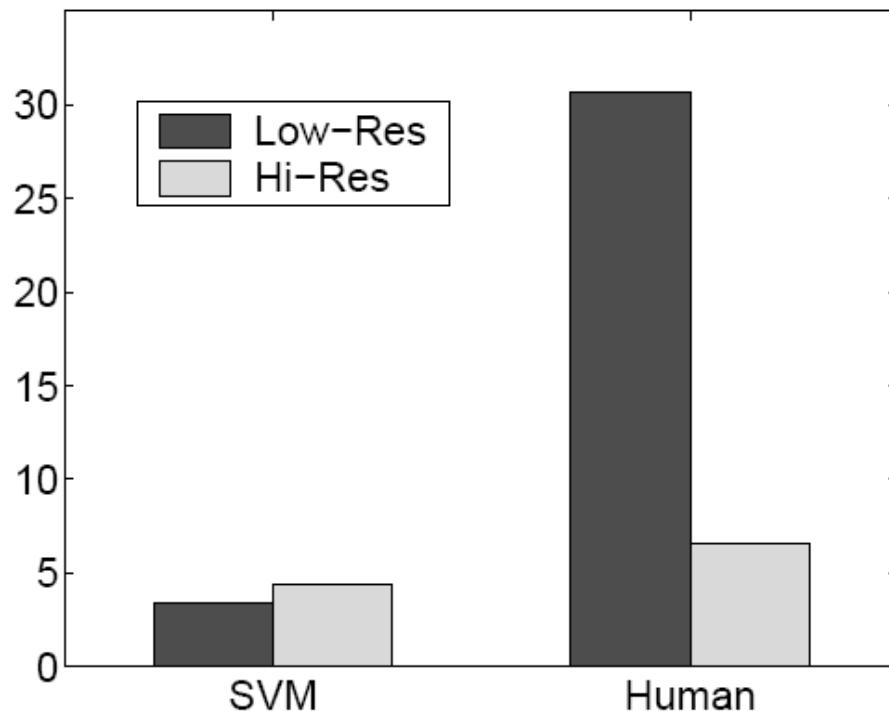
Moghaddam and Yang, Face & Gesture 2000

Support faces



Accuracy of SVM Classifier

% Error Rates



- SVMs performed better than humans, at either resolution

Figure 6. SVM vs. Human performance

Some Issues

- **Sensitive to noise**
 - A relatively small number of mislabeled examples can dramatically decrease the performance
 - **Choice of kernel**
 - Gaussian or polynomial kernel is default
 - if ineffective, more elaborate kernels are needed
 - domain experts can give assistance in formulating appropriate similarity measures
 - **Choice of kernel parameters**
 - e.g. σ in Gaussian kernel
 - σ is the distance between closest points with different classifications
 - In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.
 - **Optimization criterion** – Hard margin v.s. Soft margin
 - a lengthy series of experiments in which various parameters are tested
-

Reference

- **Support Vector Machine Classification of Microarray Gene Expression Data**, Michael P. S. Brown William Noble Grundy, David Lin, Nello Cristianini, Charles Sugnet, Manuel Ares, Jr., David Haussler
- **Text categorization with Support Vector Machines: learning with many relevant features**
T. Joachims, ECML - 98
- Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition
- **A Tutorial on Support Vector Machines for Pattern Recognition**, Kluwer Academic Publishers - Christopher J.C. Burges

Good Web References for SVM

- <http://www.cs.utexas.edu/users/mooney/cs391L/>
- <https://www.coursera.org/learn/machine-learning/home/week/7>
- <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>
- <https://data-flair.training/blogs/svm-kernel-functions/>
- [MIT 6.034 Artificial Intelligence, Fall 2010](#)
- <https://stats.stackexchange.com/questions/30042/neural-networks-vs-support-vector-machines-are-the-second-definitely-superior>
- <https://www.sciencedirect.com/science/article/abs/pii/S0893608006002796>
- <https://medium.com/deep-math-machine-learning-ai/chapter-3-support-vector-machine-with-math-47d6193c82be>
- [Radial basis kernel](#)
- <http://www.engr.mun.ca/~baxter/Publications/LagrangeForSVMs.pdf>



Thank You



BITS Pilani
Pilani Campus

Unsupervised Learning

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in



Text Book(s)

T1	Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition
T2	Tom M. Mitchell: Machine Learning, The McGraw-Hill Companies, Inc..

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Bishop and many others who made their course materials freely available online.

Topics to be covered



Ref: Christopher Bishop: Chapter 9

- Unsupervised learning
- Clustering
- K-means Clustering
- Gaussian Mixture Models
- EM algorithm

Unsupervised Learning

- We only use the features X, not the labels Y
- This is useful because we may not have any labels but we can still detect patterns
- For example:
 - We can detect that news articles revolve around certain topics, and group them accordingly
 - Discover a distinct set of objects appear in a given environment, even if we don't know their names, then ask humans to label each group
 - Identify health factors that correlate with a disease

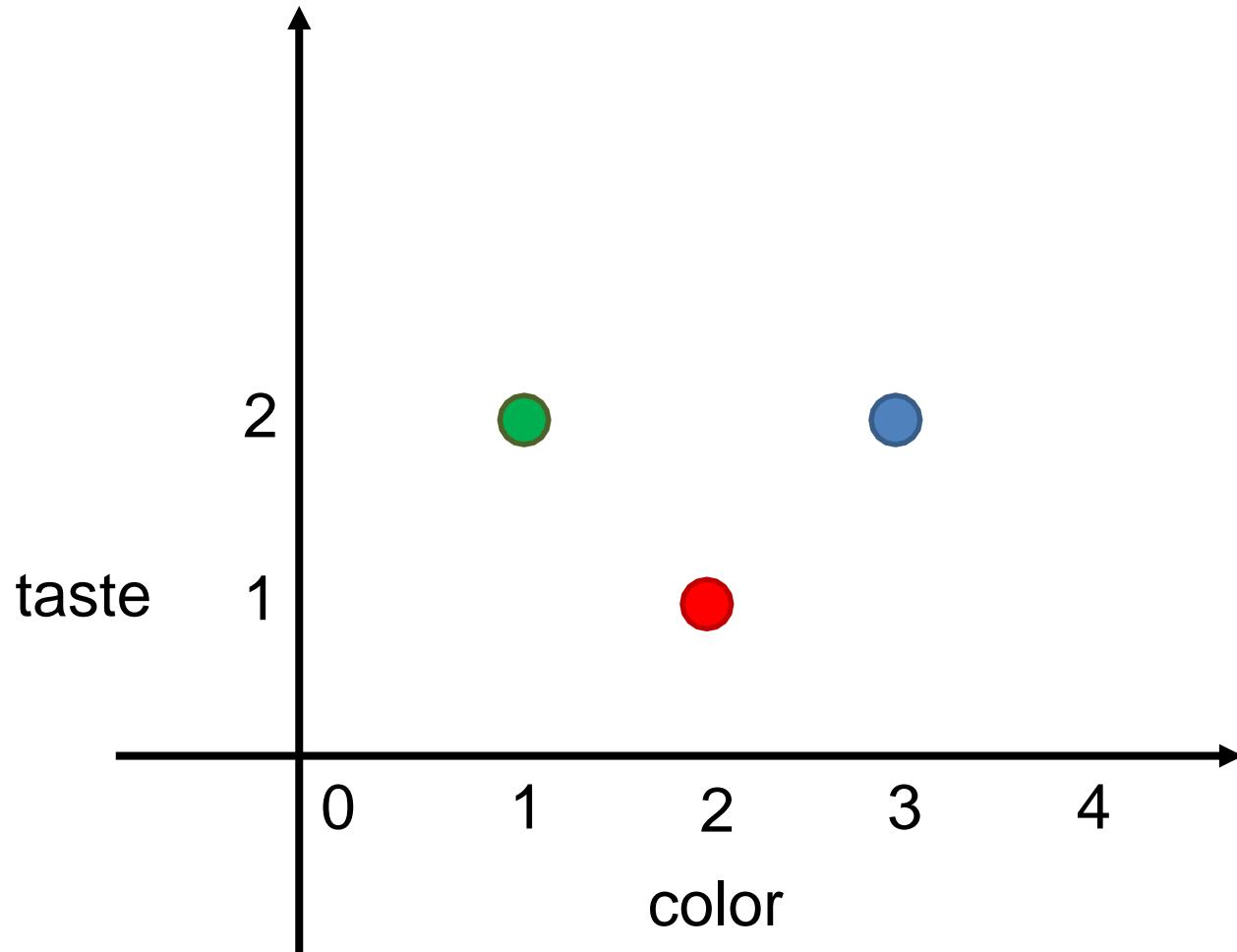
What is clustering?

- Grouping items that “belong together” (i.e. have similar features)

Feature representation (x)

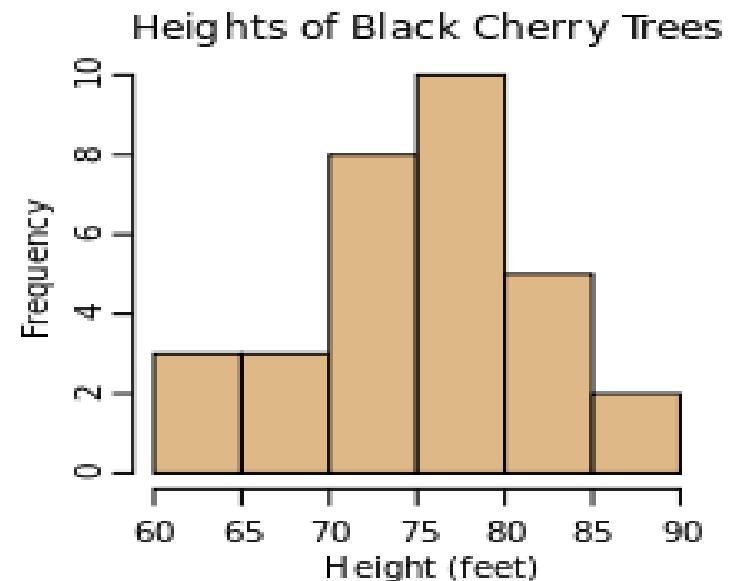
- A vector representing measurable characteristics of a data sample we have
- E.g. a glass of juice can be represented via its color = {yellow=1, red=2, green=3, purple=4} and taste = {sweet=1, sour=2}
- For a given glass i , this can be represented as a vector: $x_i = [3 \ 2]$ represents sour green juice
- For D features, this defines a D -dimensional space where we can measure similarity between samples

Feature representation (x)



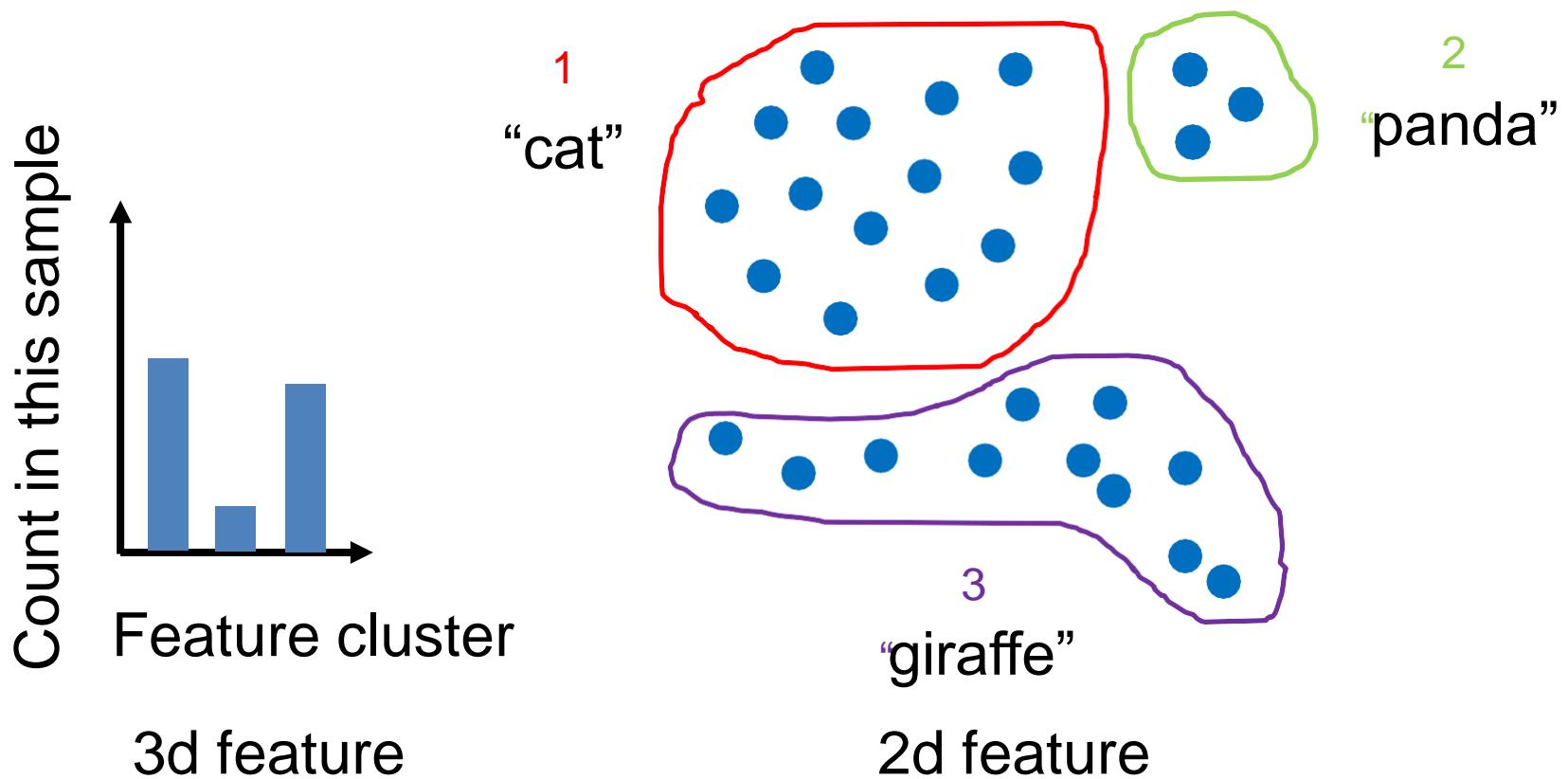
Why do we cluster?

- Counting
 - Feature histograms: by grouping similar features and counting how many of each a data sample has
- Summarizing data
 - Look at large amounts of data
 - Represent a large continuous vector with the cluster number
- Prediction
 - Data points in the same cluster may have the same labels
 - Ask a human to label the clusters

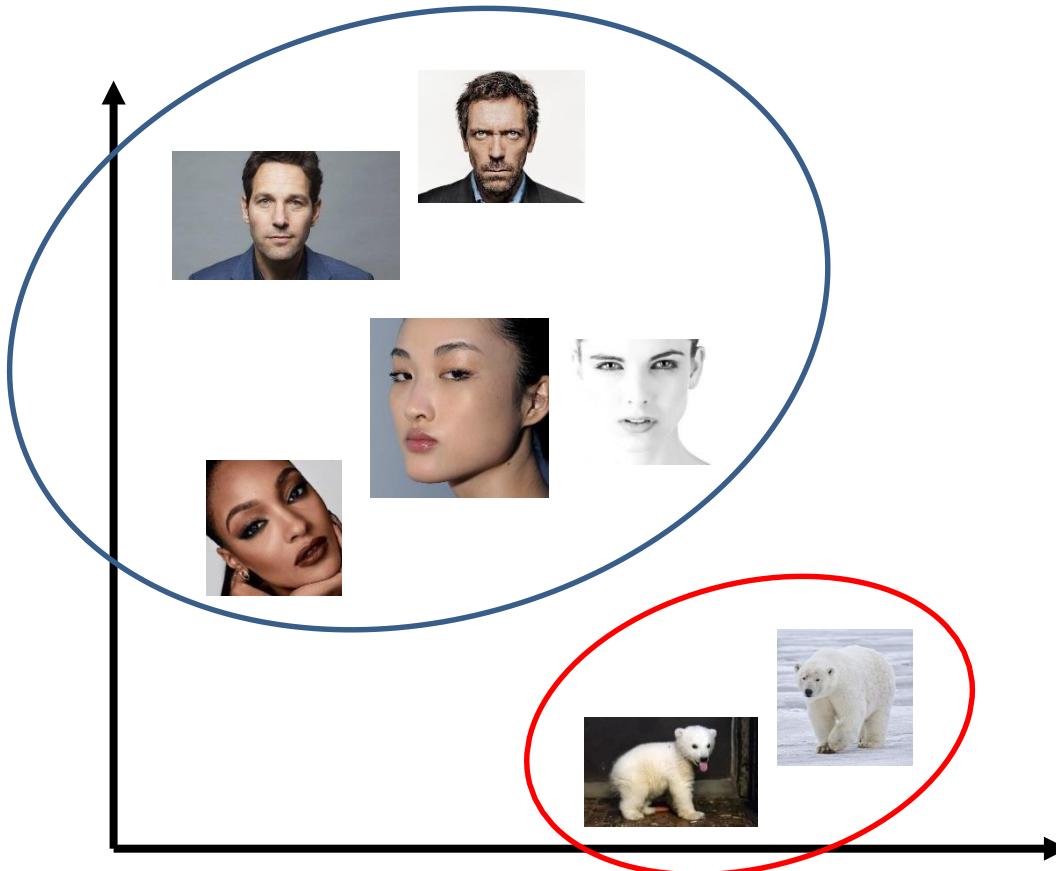


Why do we cluster?

- Two uses of clustering in one application
- Cluster, then ask human to label groups
- Compute a histogram to summarize the data



Unsupervised discovery



Clustering algorithms

- In depth
 - K-means (iterate between finding centers and assigning points)
 - Gaussian Mixture Models (GMMs)

K-means Algorithm

- Goal: represent a data set in terms of K clusters each of which is summarized by a prototype
- Initialize prototypes, then iterate between two phases:
 - E-step: assign each data point to nearest prototype
 - M-step: update prototypes to be the cluster means
- Simplest version is based on Euclidean distance
 - re-scale Old Faithful data

K-means Algorithm

Randomly initialize K cluster centroids $\underline{\mu}_1, \underline{\mu}_2, \dots, \underline{\mu}_K \in \mathbb{R}^n$

Repeat {

Cluster
Assignment
step

for $i = 1$ to m

$\underline{c}^{(i)}$:= index (from 1 to K) of cluster centroid
closest to $x^{(i)}$

$$\min_k \|\underline{x}^{(i)} - \underline{\mu}_k\|^2$$

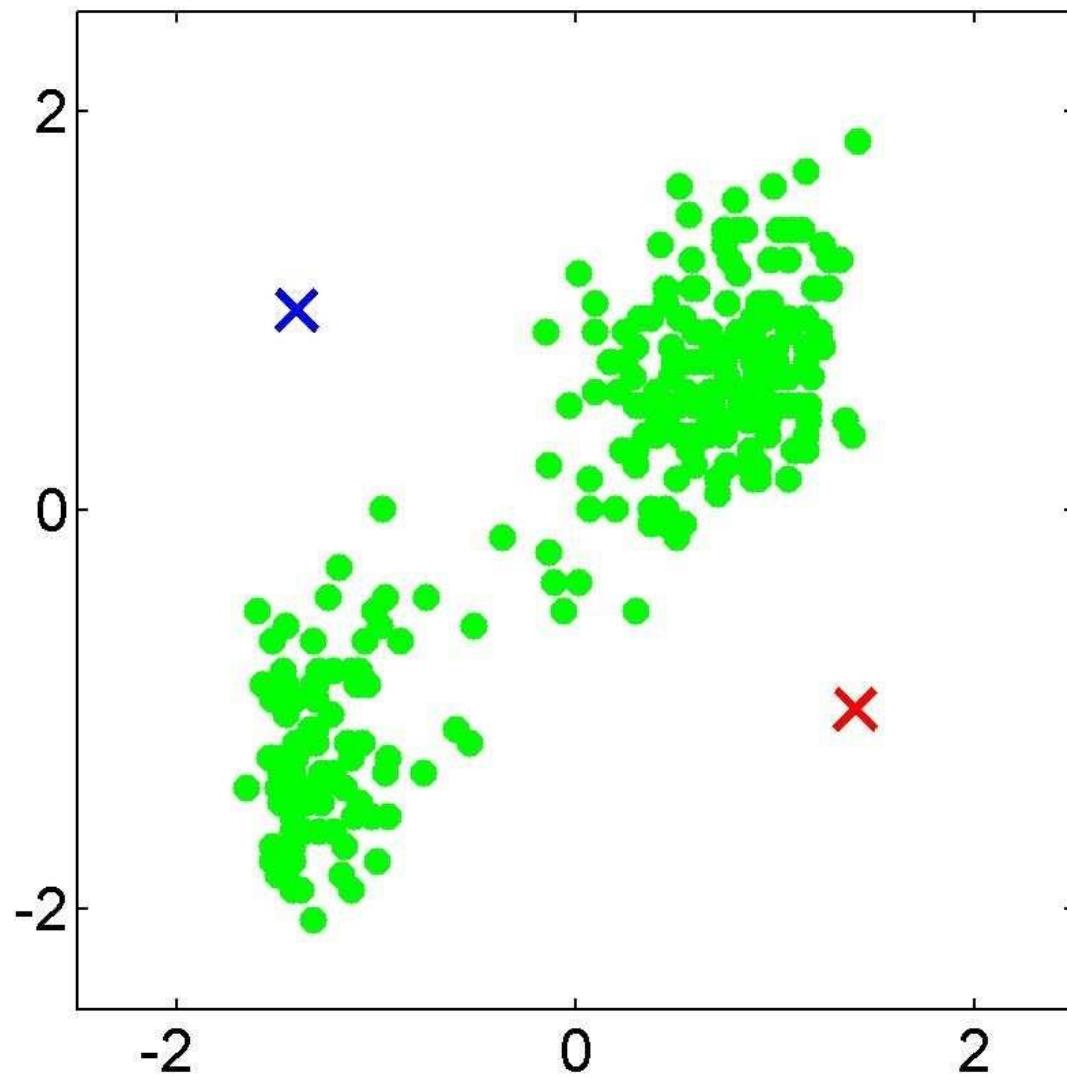
for $k = 1$ to K

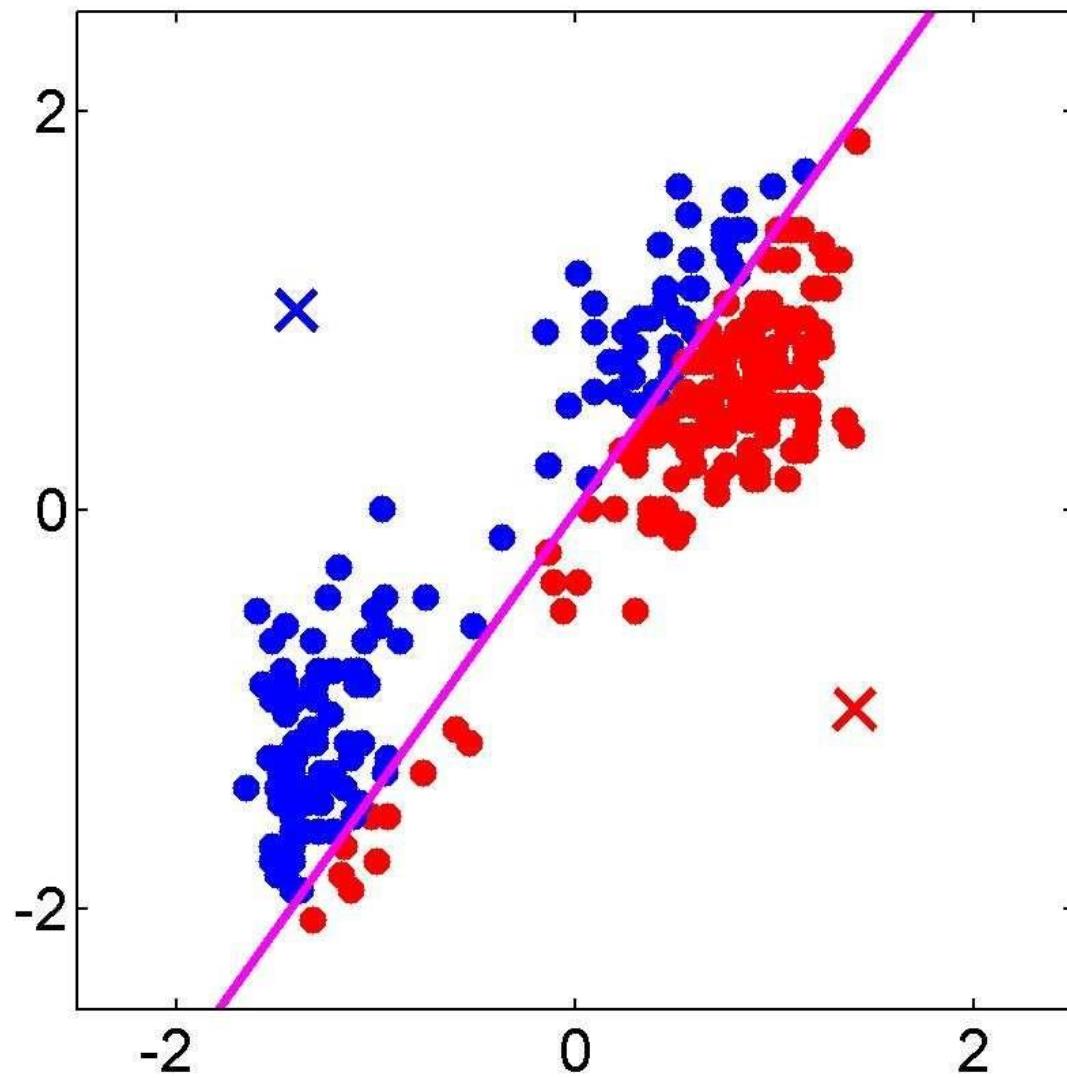
$\rightarrow \underline{\mu}_k$:= average (mean) of points assigned to cluster k

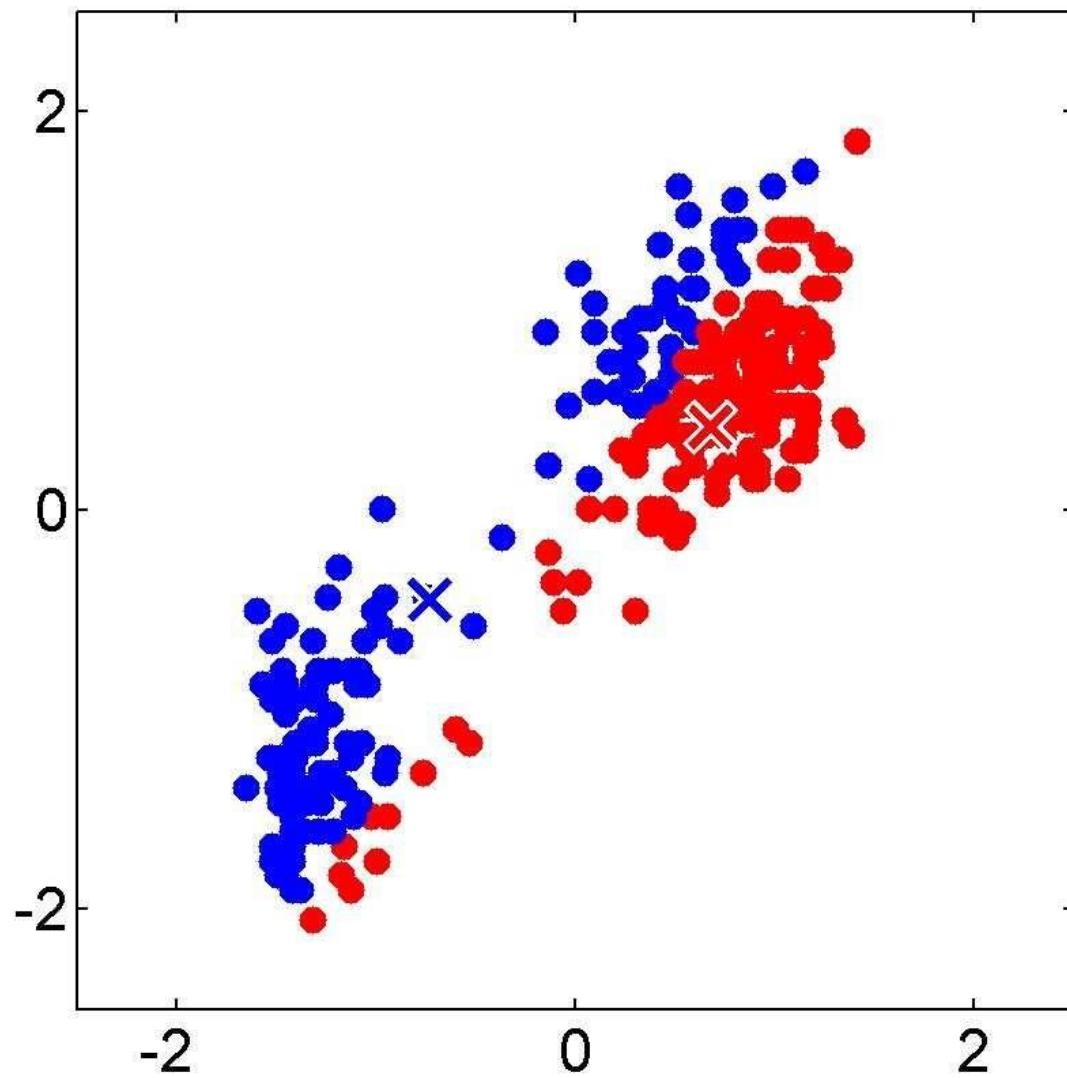
$$x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)} \rightarrow c^{(1)}=2, c^{(2)}=2, c^{(3)}=2, c^{(4)}=2$$

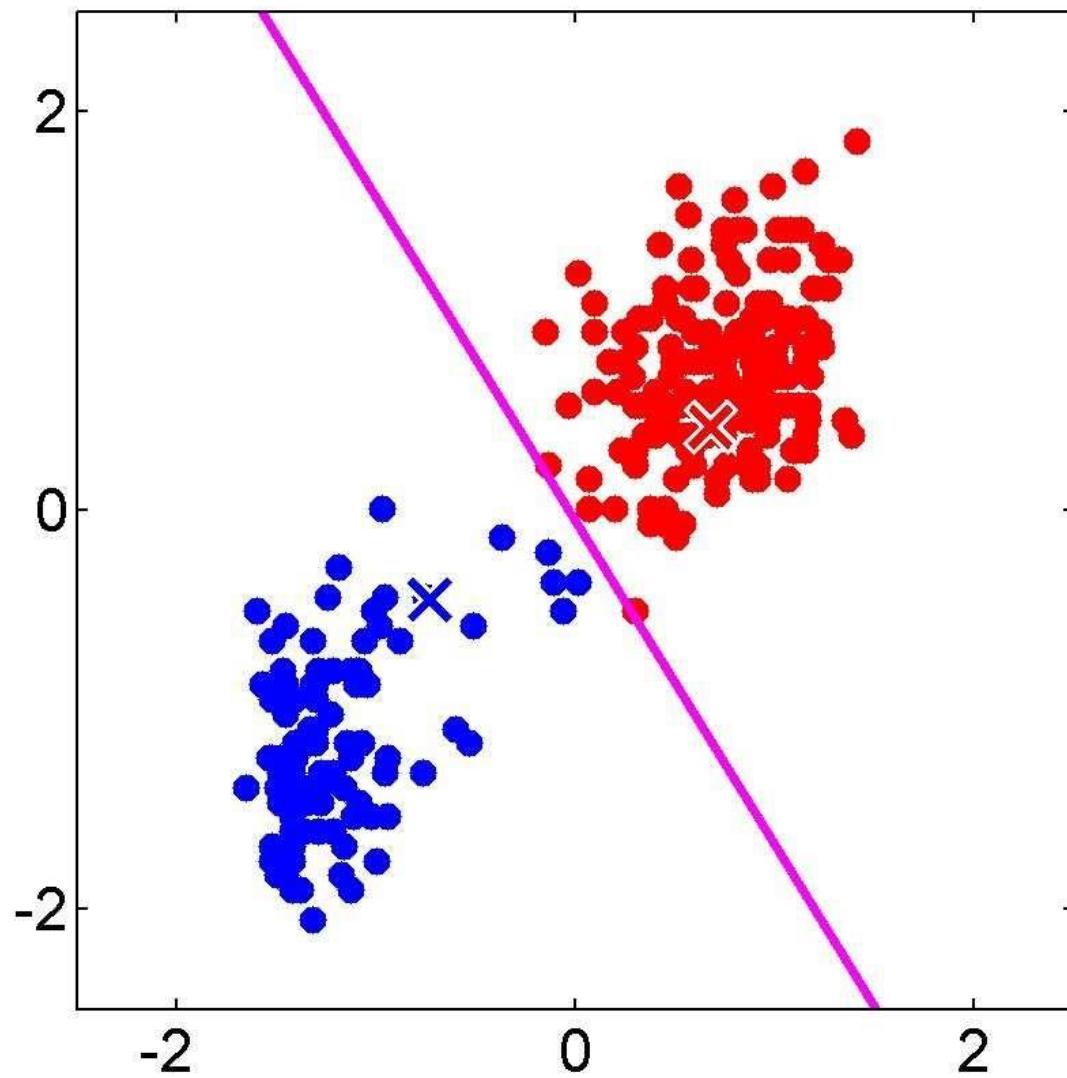
}

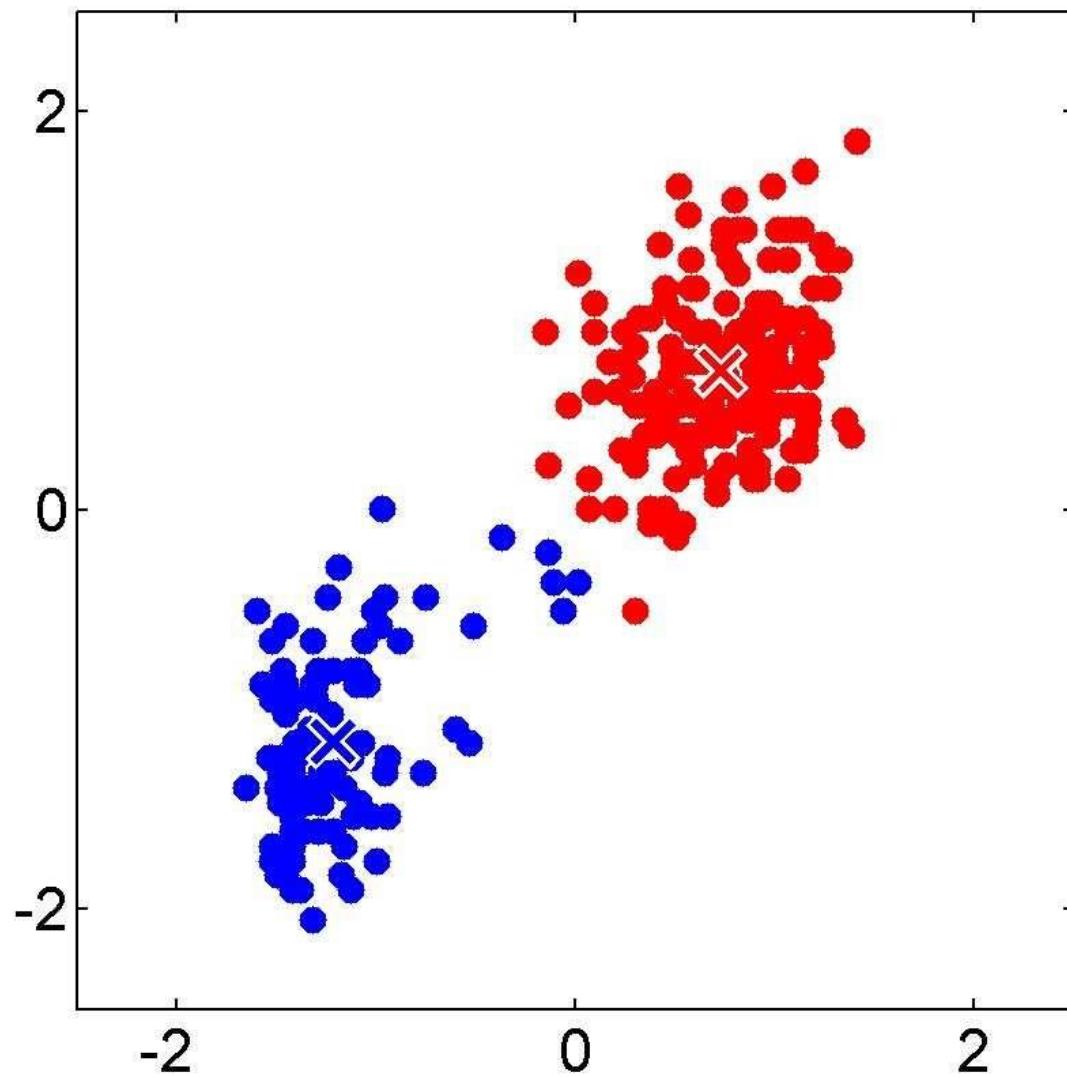
$$\underline{\mu}_2 = \frac{1}{4} \left[\underline{x}^{(1)} + \underline{x}^{(2)} + \underline{x}^{(3)} + \underline{x}^{(4)} \right] \in \mathbb{R}^n$$

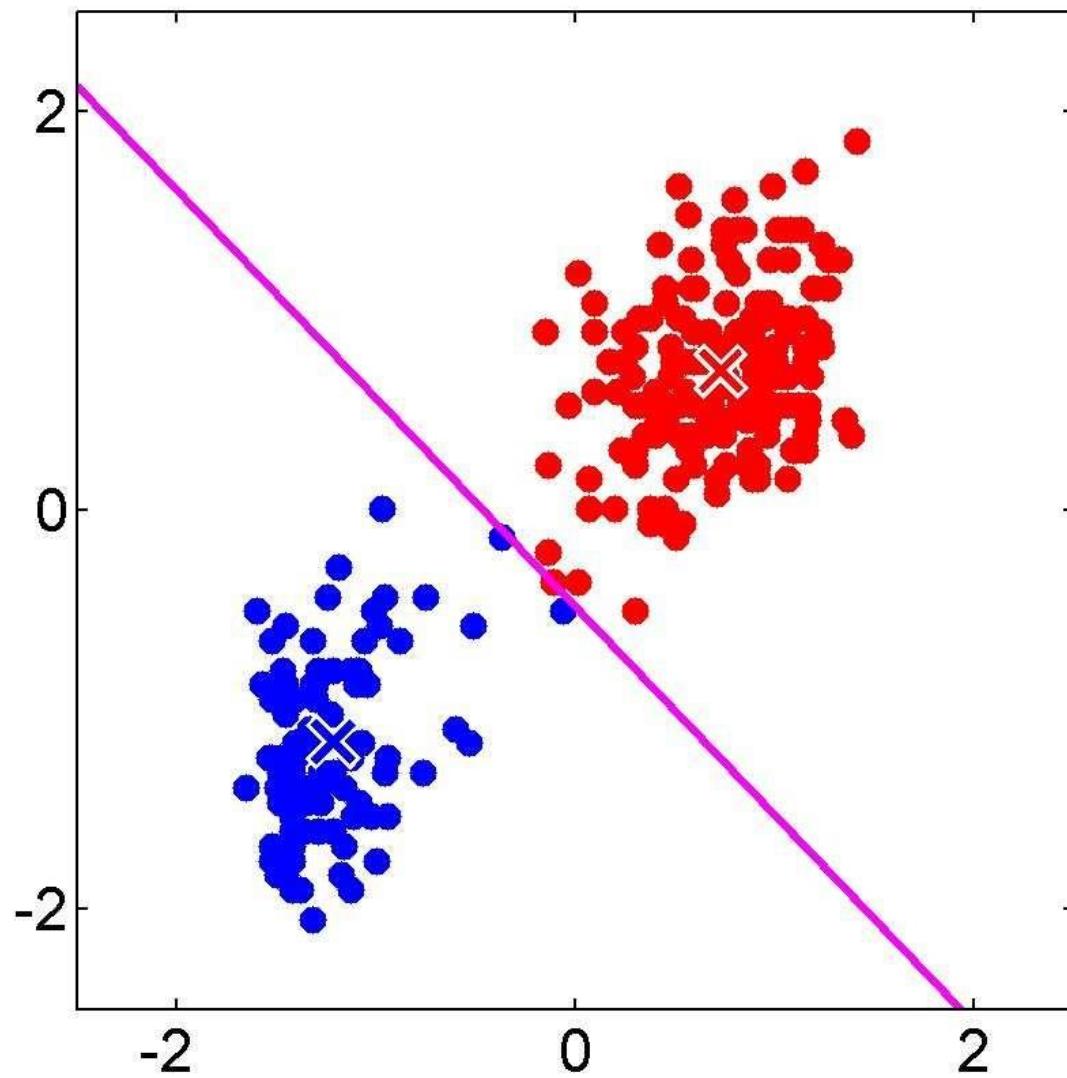


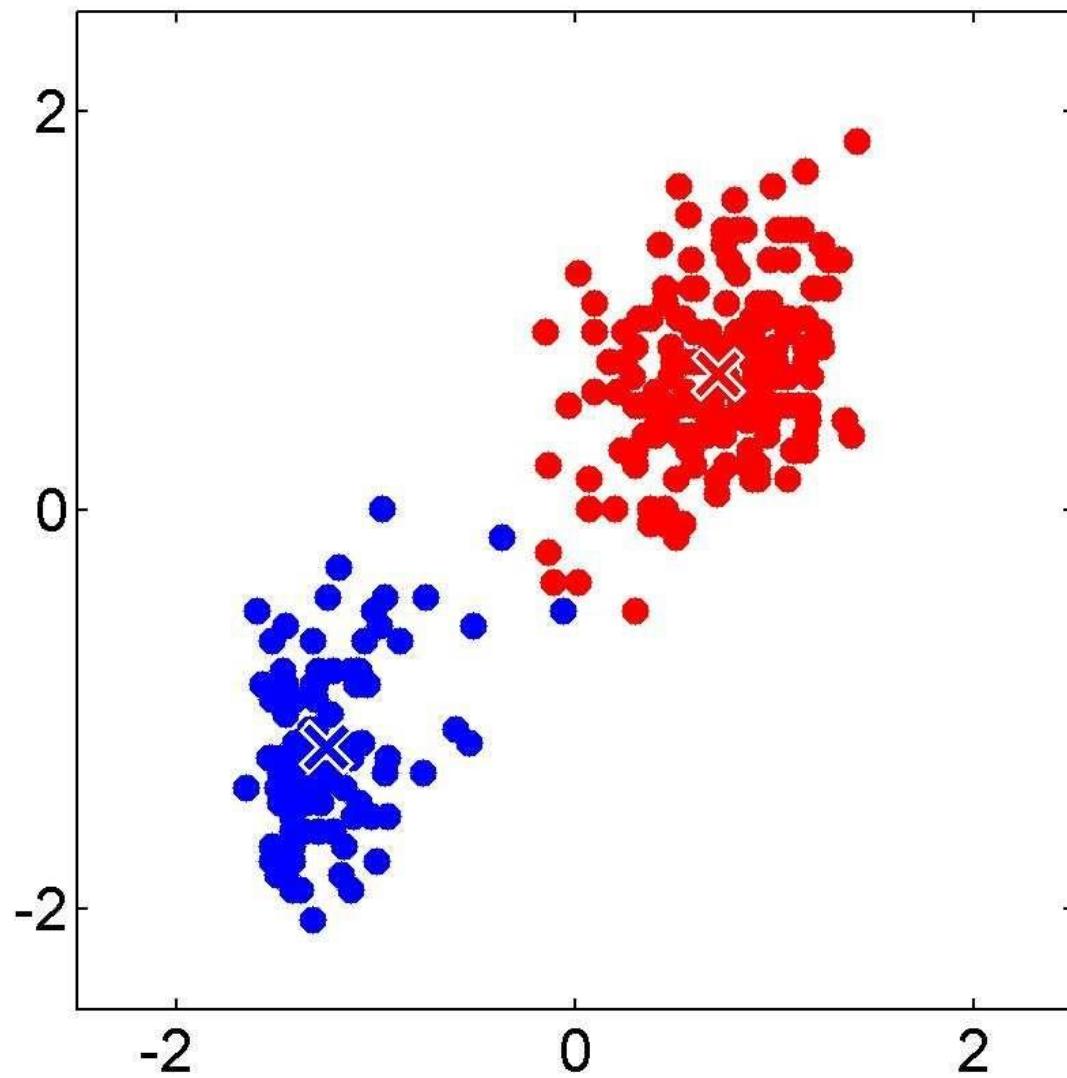


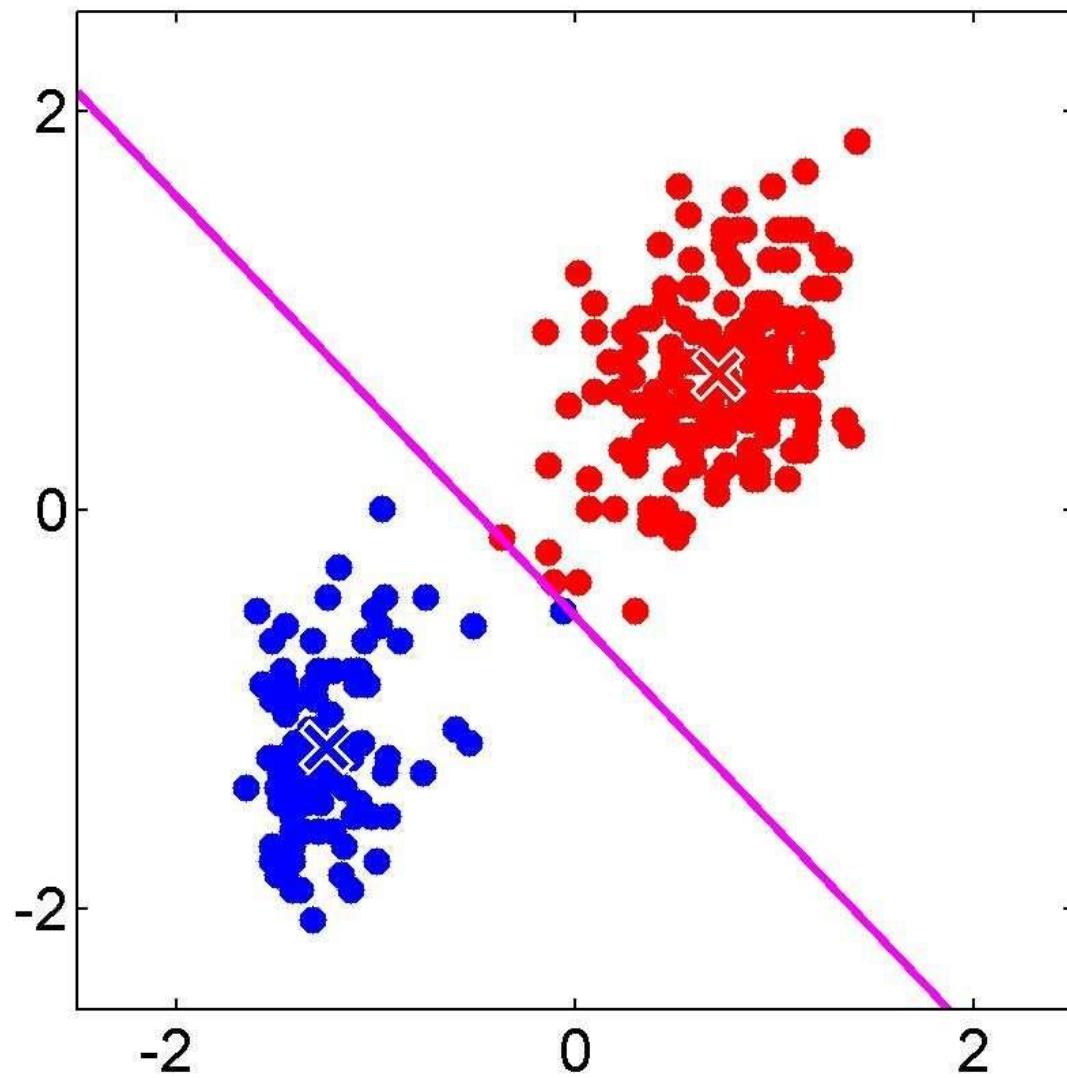


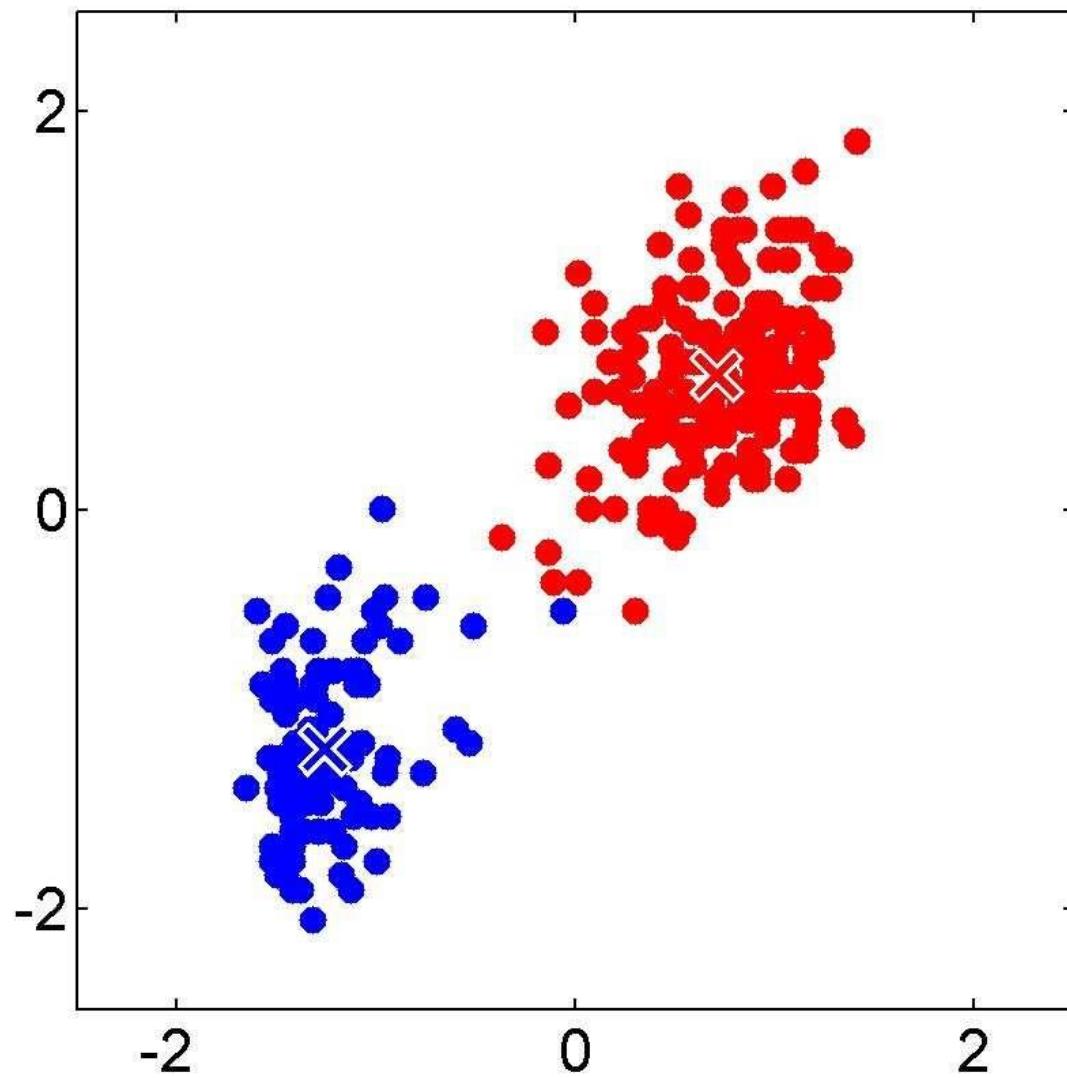












K-means clustering

- Basic idea: randomly initialize the k cluster centers, and iterate between the two steps we just saw.
 1. Randomly initialize the **cluster centers**, c_1, \dots, c_K
 2. Given **cluster centers**, determine **points** in each cluster
 - For each point p , find the closest c_i . Put p into **cluster i**
 3. Given **points** in each **cluster**, solve for c_i
 - Set c_i to be the mean of **points** in **cluster i**
 4. If c_i have changed, repeat Step 2

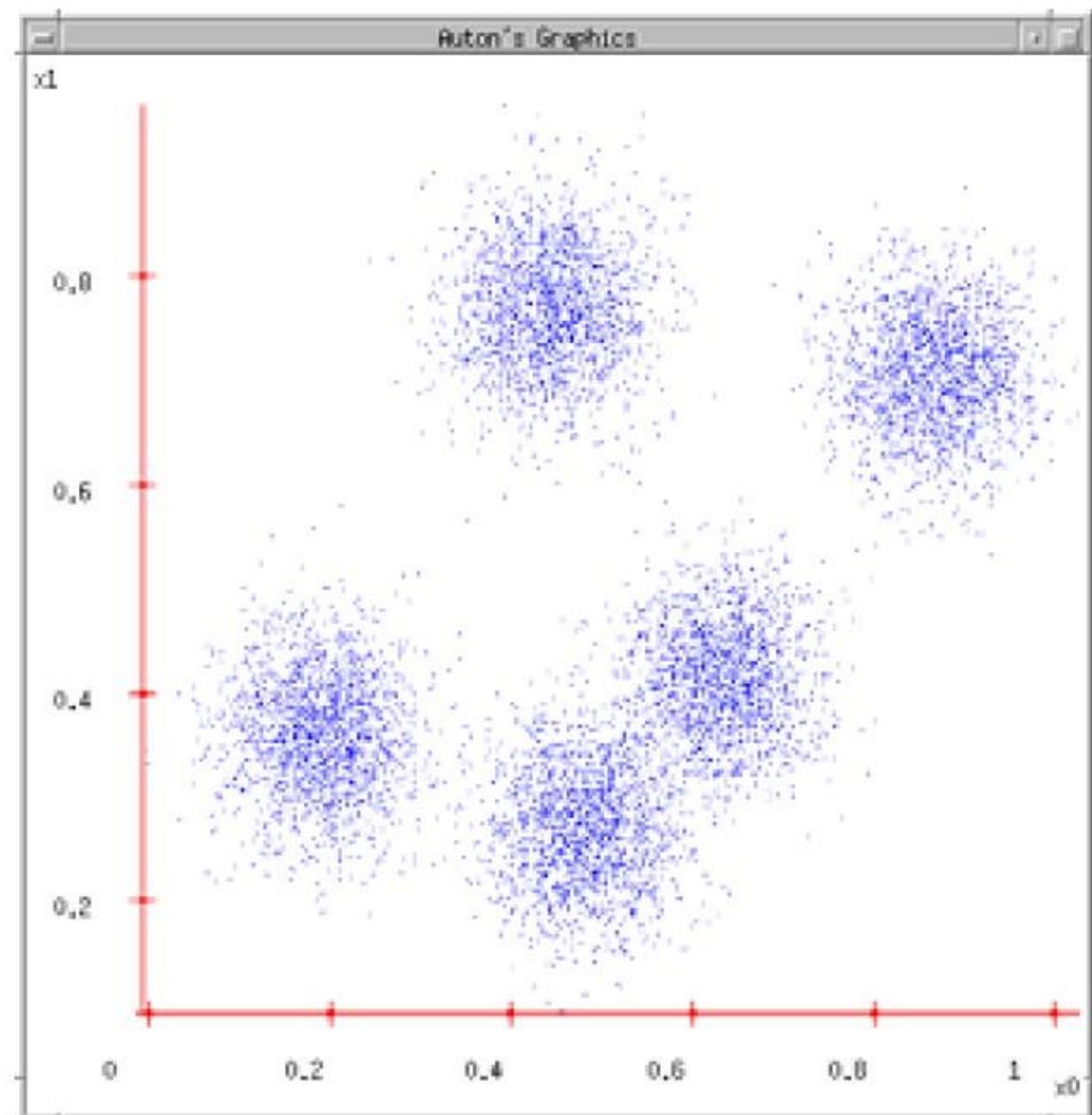


Properties

- Will always converge to some solution
- Can be a “local minimum”

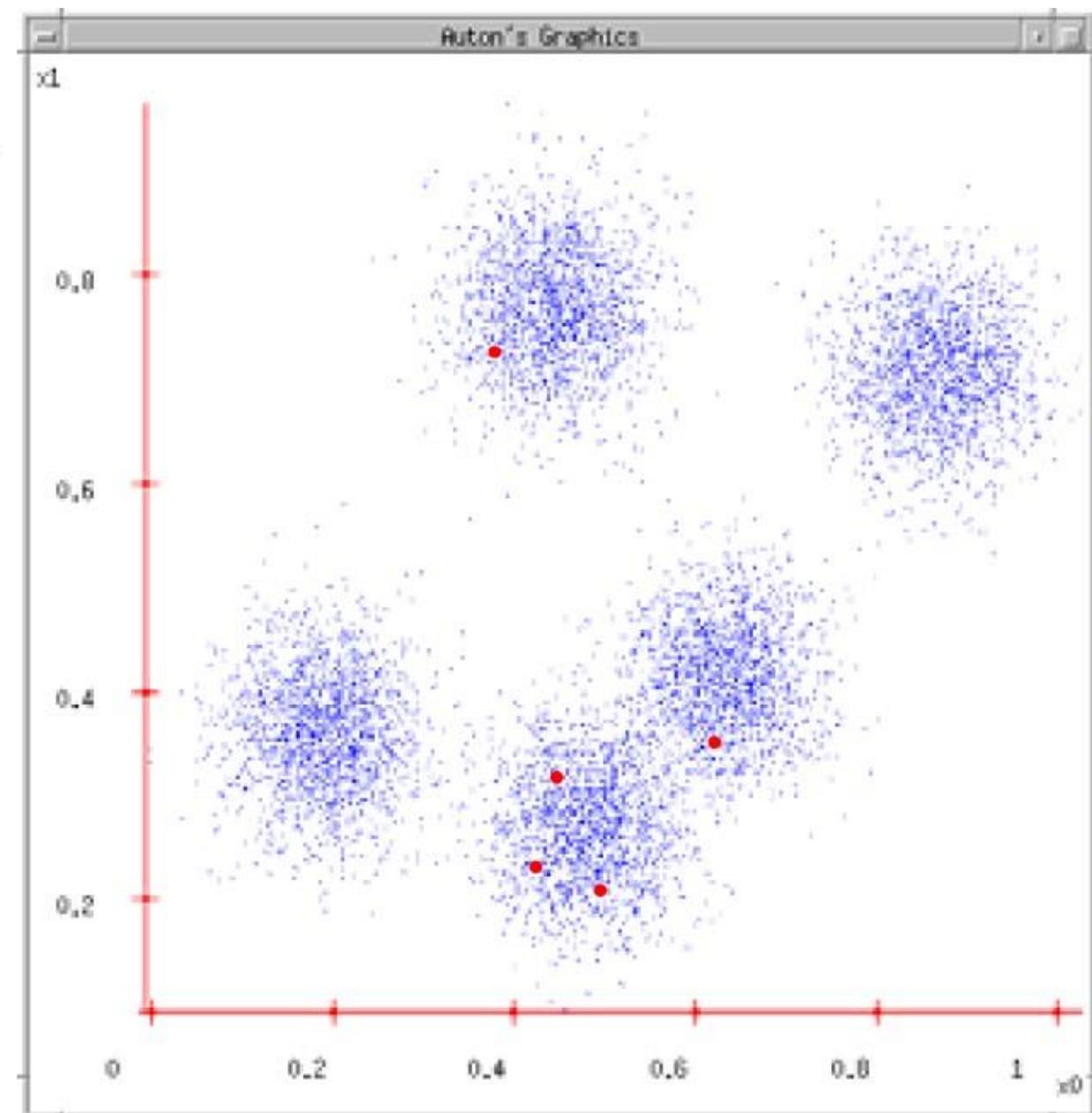
K-means

1. Ask user how many clusters they'd like.
(e.g. k=5)



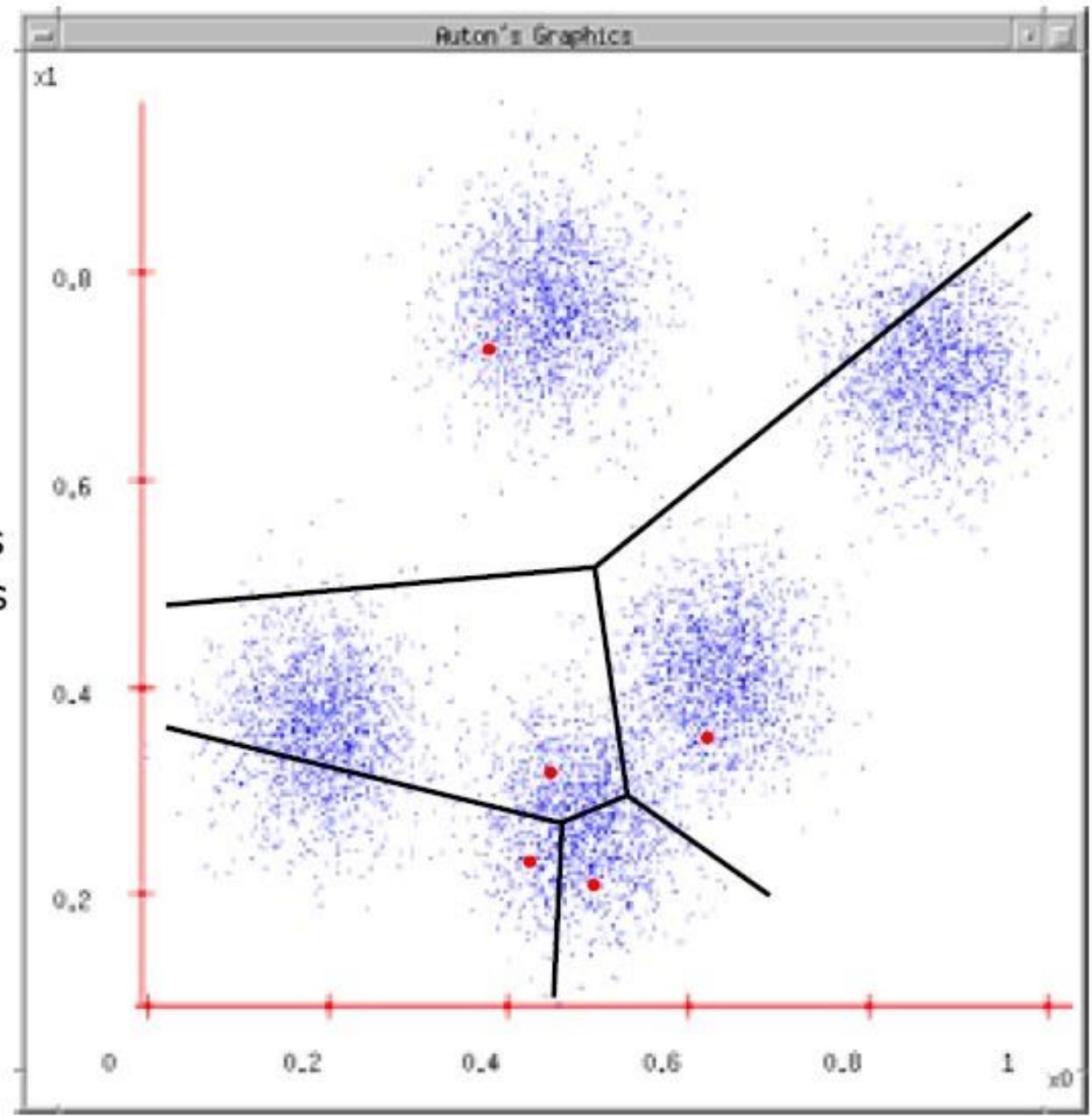
K-means

1. Ask user how many clusters they'd like.
(e.g. k=5)
2. Randomly guess k cluster Center locations



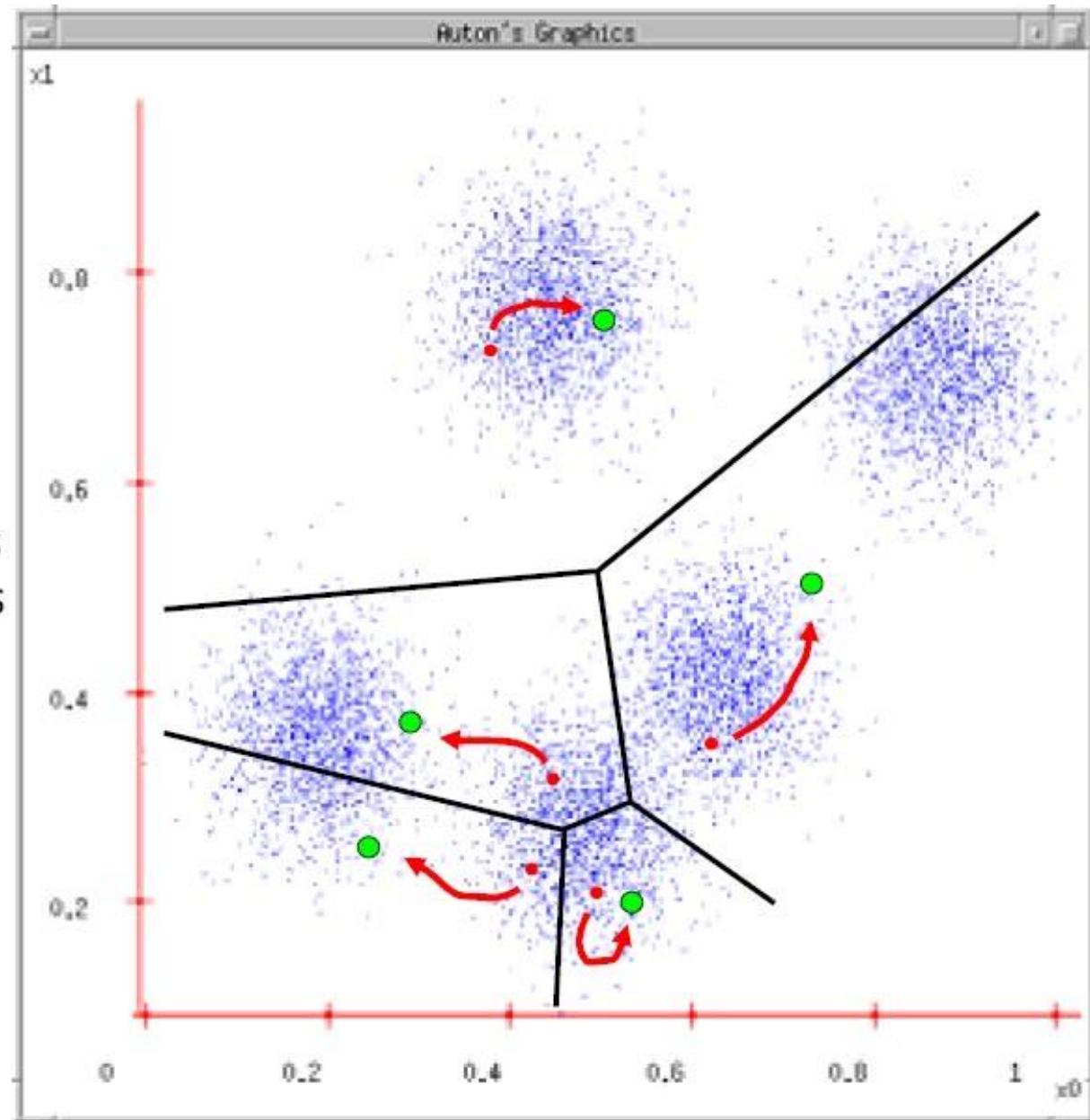
K-means

1. Ask user how many clusters they'd like.
(e.g. k=5)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)



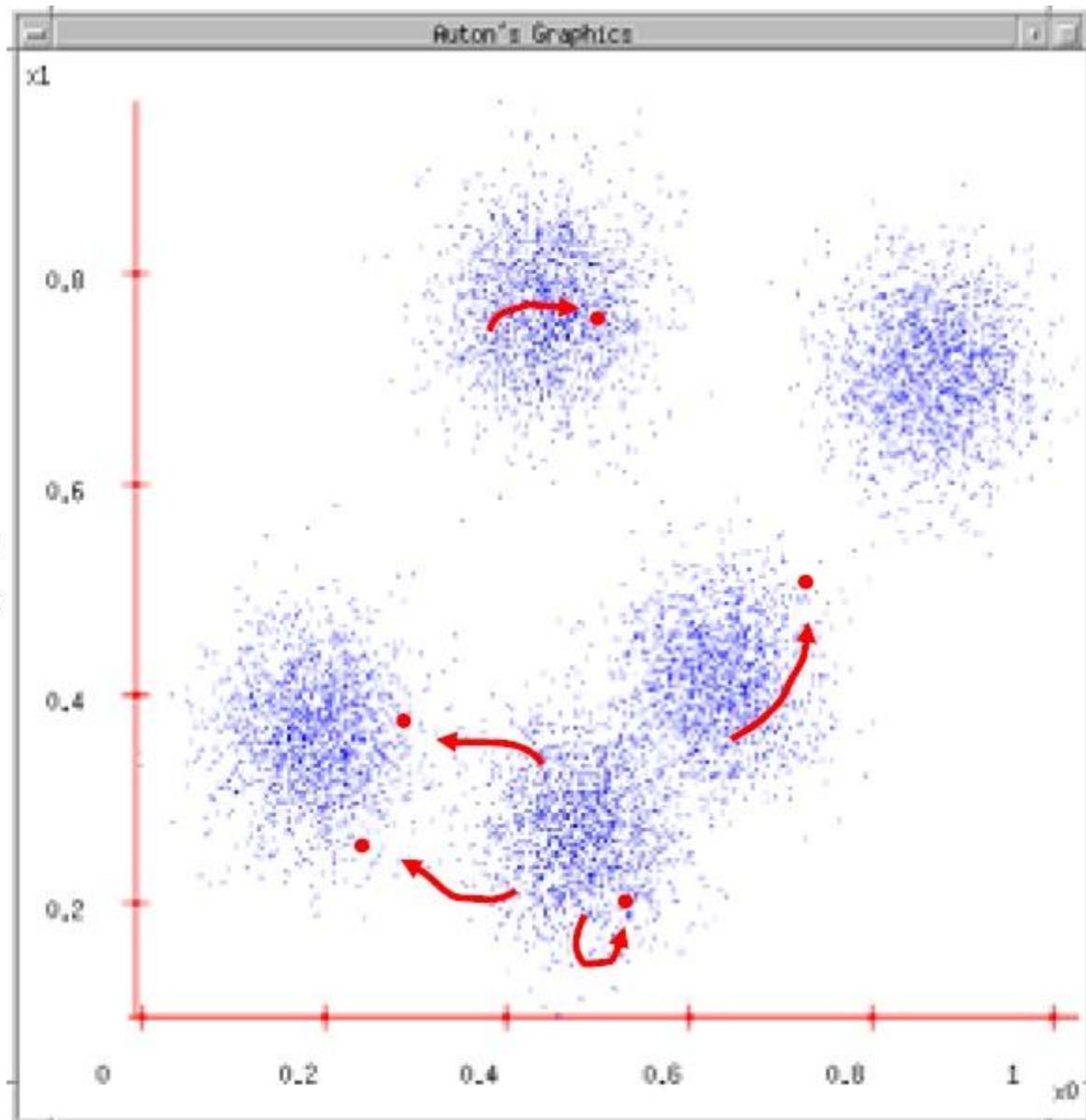
K-means

1. Ask user how many clusters they'd like.
(e.g. k=5)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns



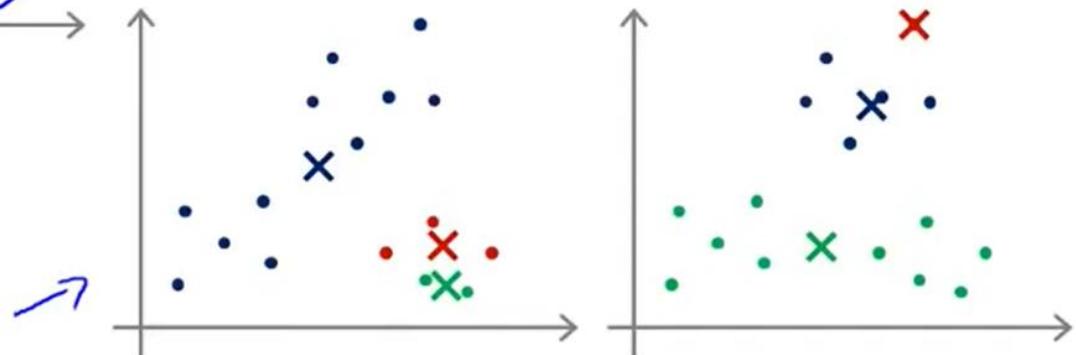
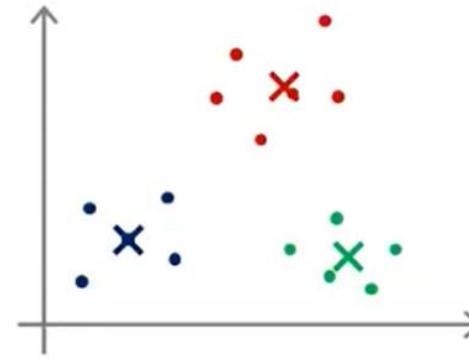
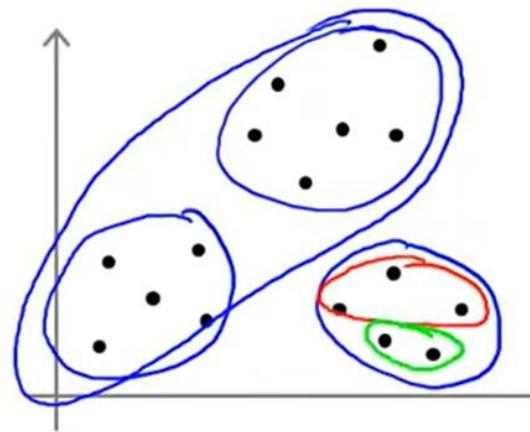
K-means

1. Ask user how many clusters they'd like.
(e.g. k=5)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns...
5. ...and jumps there
6. ...Repeat until terminated!



Local Optima

Local optima



Andrew Ng

Random Initialization

For i = 1 to 100 { *50 - 1000*

→ Randomly initialize K-means.

Run K-means. Get $c^{(1)}, \dots, c^{(m)}$, μ_1, \dots, μ_K .

Compute cost function (distortion)

→ $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$

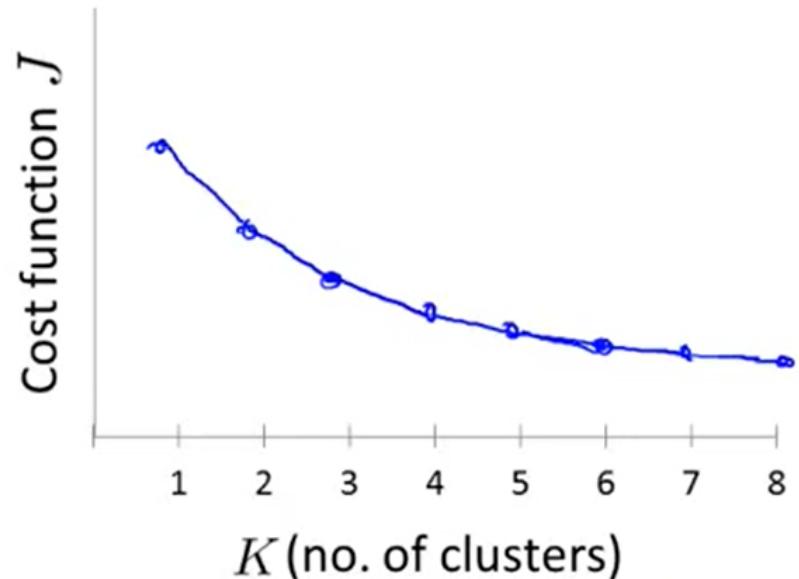
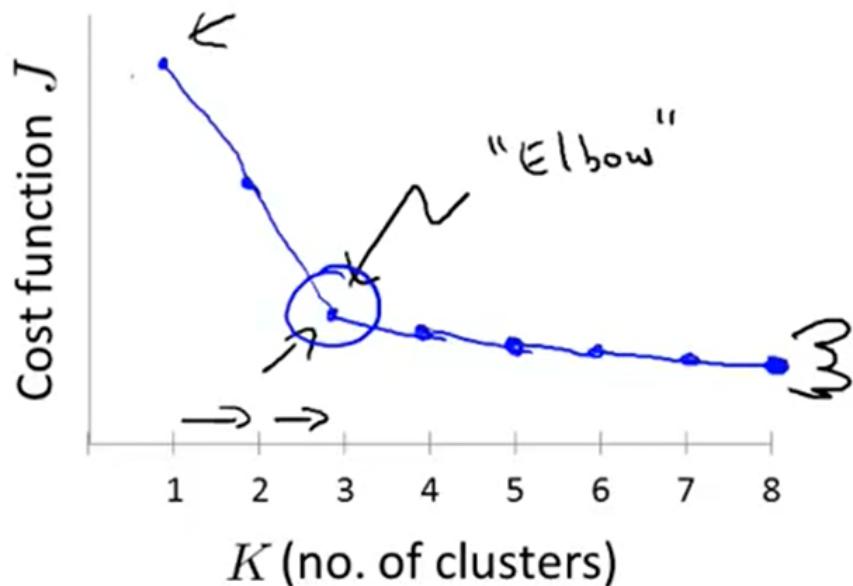
}

Pick clustering that gave lowest cost $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$

k = 2 - 10

Choosing K

Elbow method:



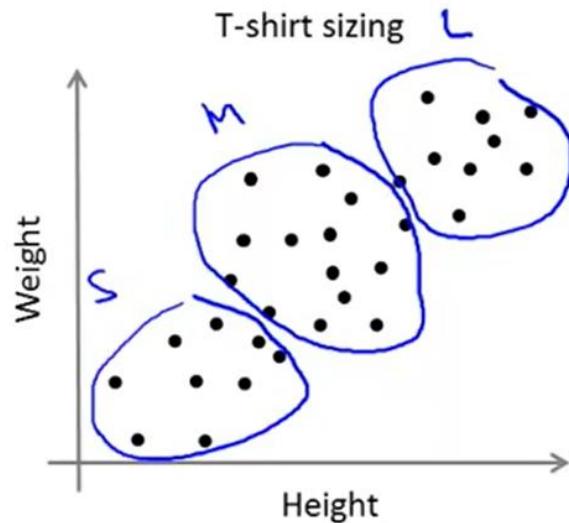
Choosing K

Choosing the value of K

Sometimes, you're running K-means to get clusters to use for some later/downstream purpose. Evaluate K-means based on a metric for how well it performs for that later purpose.

$K=3$ S, M, L

E.g.



$K=5$ XS, S, M, L, XL

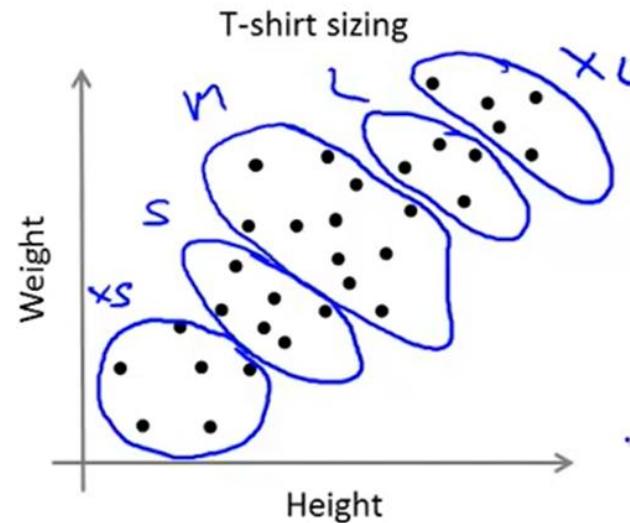
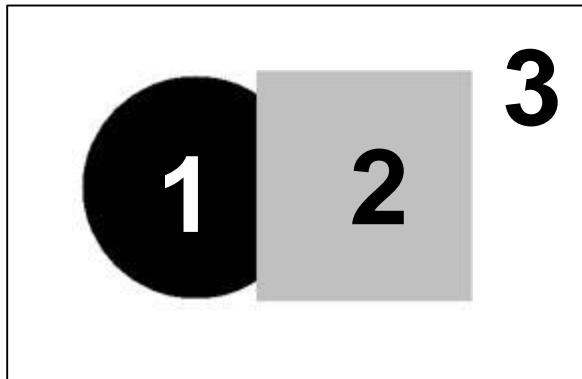
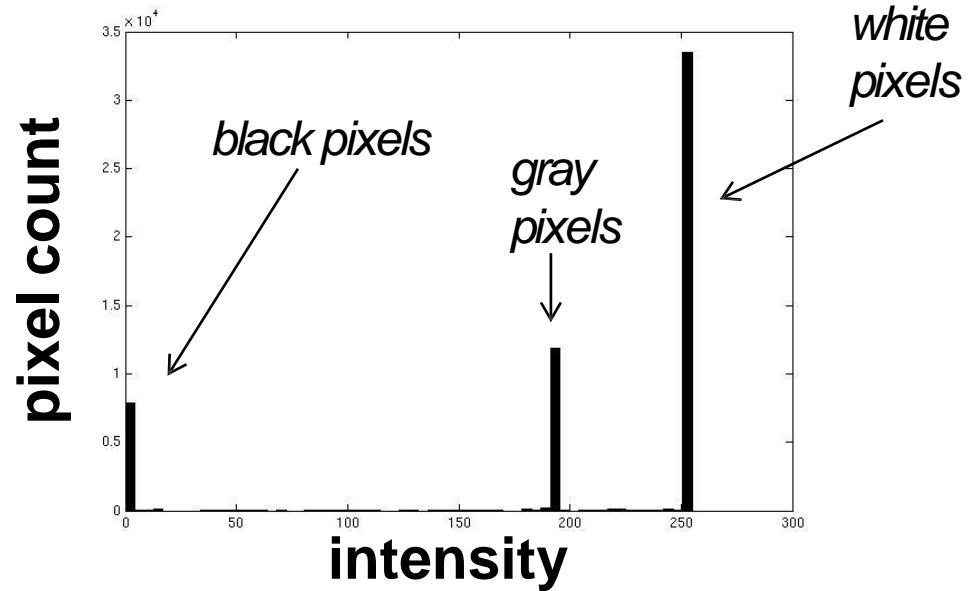


Image segmentation: toy example



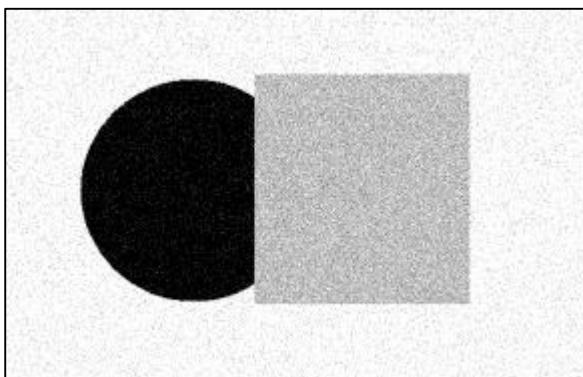
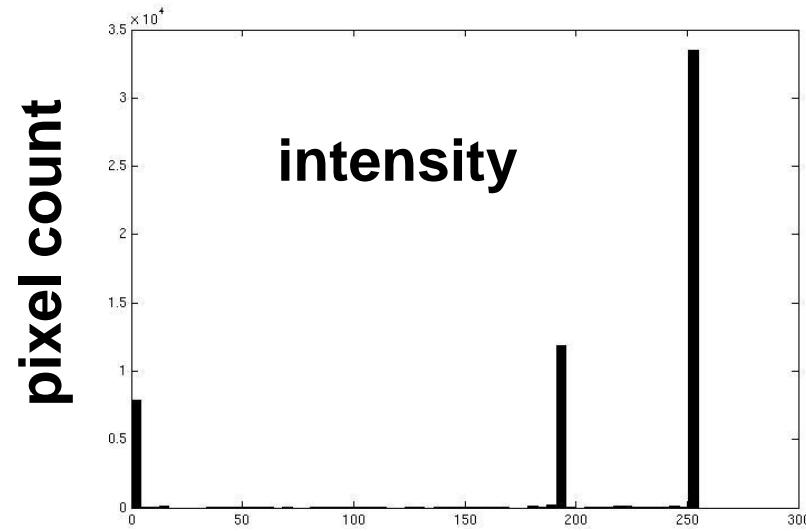
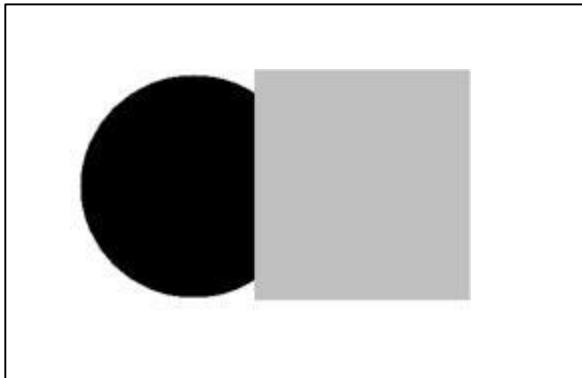
input image



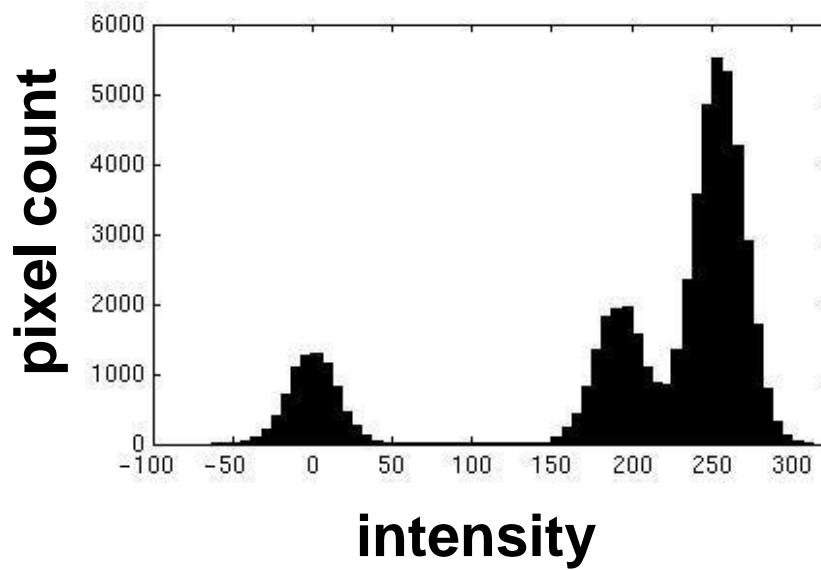
- These intensities define the three groups.
- We could label every pixel in the image according to which of these primary intensities it is.
 - i.e., *segment* the image based on the intensity feature.
- What if the image isn't quite so simple?

- Now how to determine the three main intensities that define our groups?
- We need to **cluster**.

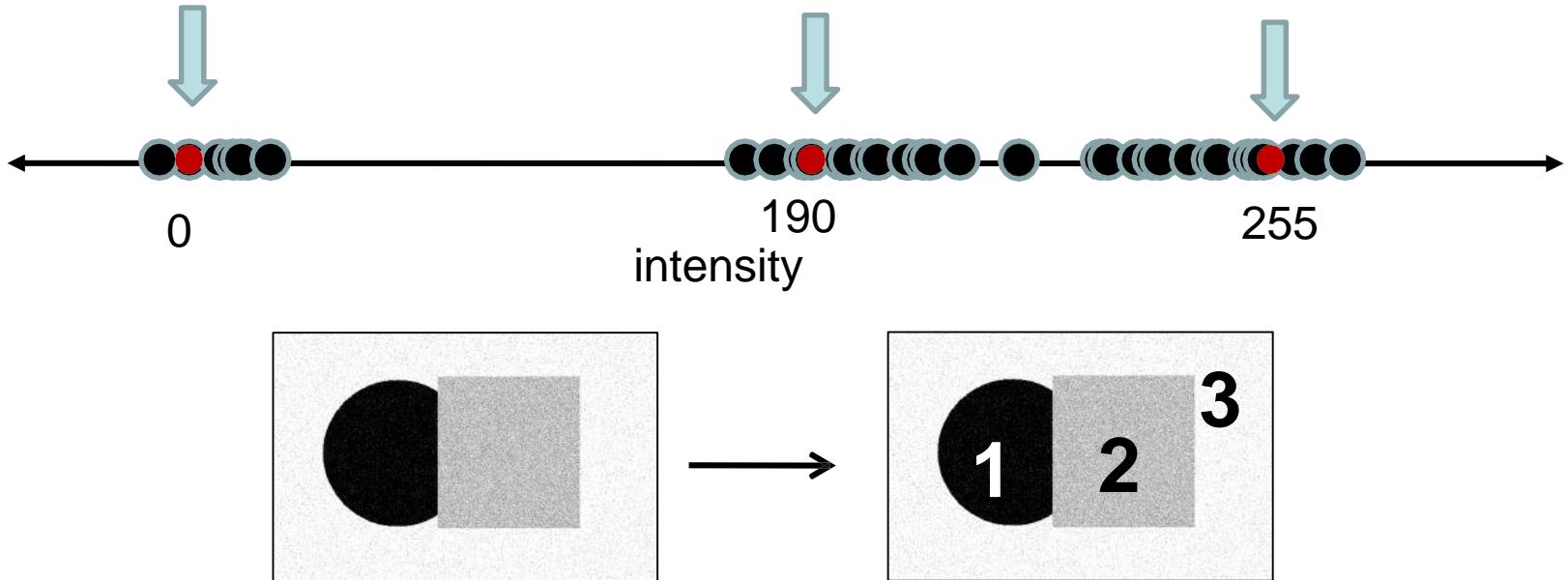
input image



input image



intensity



- Goal: choose three “centers” as the **representative** intensities, and label every pixel according to which of these centers it is nearest to.
- Best cluster centers are those that minimize SSD between all points and their nearest cluster center c_i :

$$\sum_{\text{clusters } i} \sum_{\text{points } p \text{ in cluster } i} \|p - c_i\|^2$$

K-means clustering

- Visualization

<https://www.naftaliharris.com/blog/visualizing-k-means-clustering/>

- Java demo

http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/AppletKM.html

- Matlab demo

http://www.cs.pitt.edu/~kovashka/cs1699_fa15/kmeans_demo.m

Time Complexity

- Let n = number of instances, d = dimensionality of the features, k = number of clusters
- Assume computing distance between two instances is $O(d)$
- Reassigning clusters:
 - $O(kn)$ distance computations, or $O(knd)$
- Computing centroids:
 - Each instance vector gets added once to a centroid: $O(nd)$
- Assume these two steps are each done once for a fixed number of iterations I : $O(Iknd)$
 - Linear in all relevant factors

Another way of writing objective

- **K-means:** Let $r_{nk} = 1$ if instance n belongs to cluster k , 0 otherwise

$$\boldsymbol{\mu}_k = \frac{\sum_n r_{nk} \mathbf{x}_n}{\sum_n r_{nk}}$$

$$J = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

- k-means the centre of a cluster is not necessarily one of the input data points (it is the average between the points in the cluster).

- **K-medoids (more general distances):**

$$\tilde{J} = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \mathcal{V}(\mathbf{x}_n, \boldsymbol{\mu}_k)$$

- k-medoids chooses data points as centers (medoids or exemplars) and can be used with arbitrary distances
- k-medoids more robust to noise and outliers as compared to [k-means](#) because it minimizes a sum of pairwise dissimilarities instead of a sum of squared Euclidean distances.

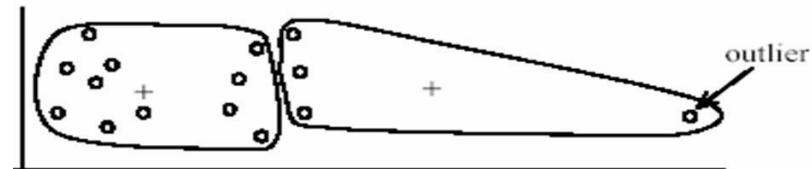
K-means: pros and cons

Pros

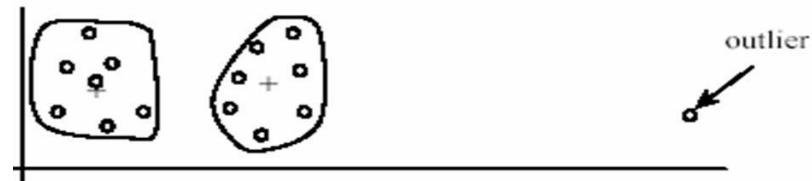
- Simple, fast to compute
- Converges to local minimum of within-cluster squared error

Cons/issues

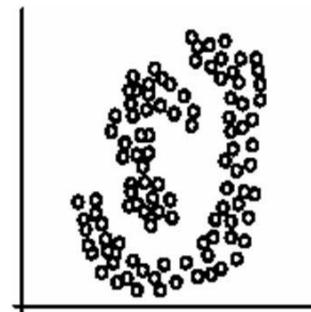
- Setting k?
- Sensitive to initial centers
 - Use heuristics or output of another method
- Sensitive to outliers
- Detects spherical clusters



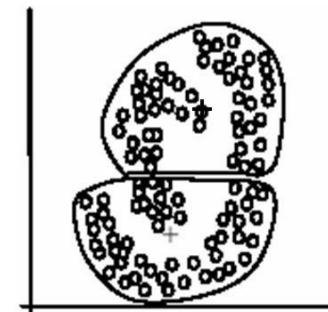
(A): Undesirable clusters



(B): Ideal clusters



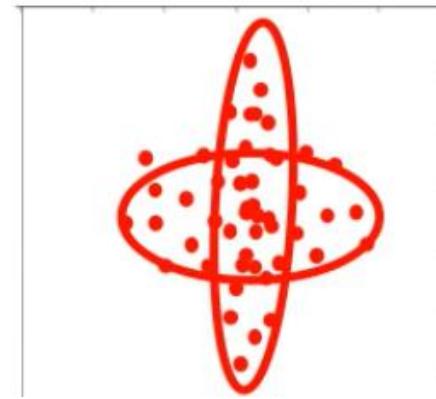
(A): Two natural clusters



(B): k -means clusters

Gaussian Mixture Models (GMM)

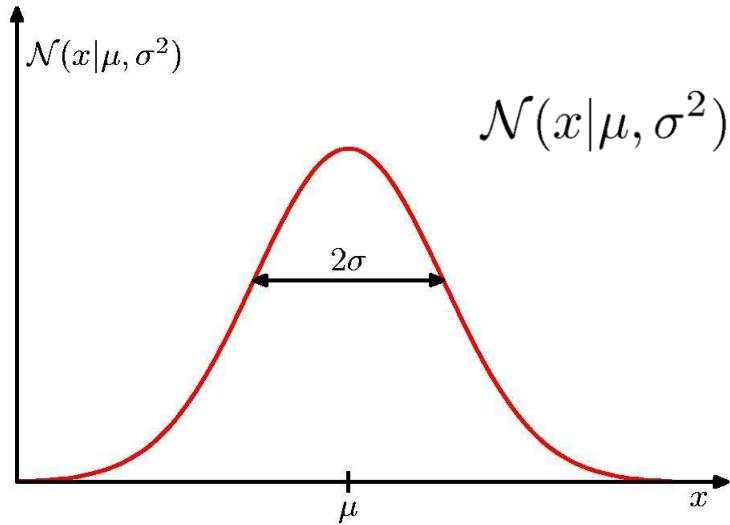
- K-means algorithm
 - Assigned each example to exactly one cluster
 - What if clusters are overlapping?
 - Hard to tell which cluster is right
 - Maybe we should try to remain uncertain
 - Used Euclidean distance
 - What if cluster has a non-circular shape?
- Gaussian mixture models
 - Clusters modeled as Gaussians
 - Not just by their mean
 - EM algorithm: assign data to cluster with some *probability*
 - Gives probability model of x ! (“generative”)



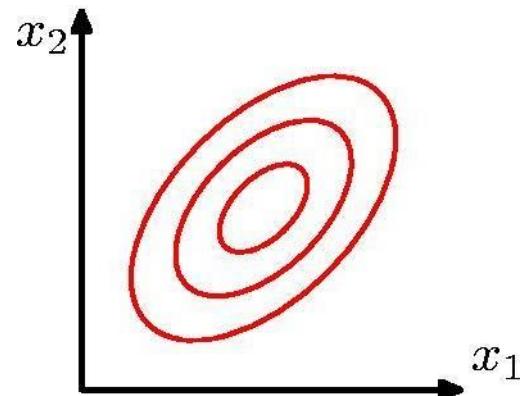
Probabilistic Clustering

- Represent the probability distribution of the data as a *mixture model*
 - captures uncertainty in cluster assignments
 - gives model for data distribution
 - Bayesian* mixture model allows us to determine K
- Consider mixtures of *Gaussians*

Review: Gaussian Distribution



$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp \left\{ -\frac{1}{2\sigma^2}(x - \mu)^2 \right\}$$



$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp \left\{ -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\}$$

Multivariate Gaussian Distribution

Density estimation

Training set: $\{x^{(1)}, \dots, x^{(m)}\}$

Each example is $x \in \mathbb{R}^n$

$$p(x)$$

$$= p(x_1; \mu_1, \sigma_1^2) p(x_2; \mu_2, \sigma_2^2) p(x_3; \mu_3, \sigma_3^2) \cdots p(x_n; \mu_n, \sigma_n^2)$$

$$= \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2)$$

$$x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$$

$$x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$$

$$x_3 \sim \mathcal{N}(\mu_3, \sigma_3^2)$$

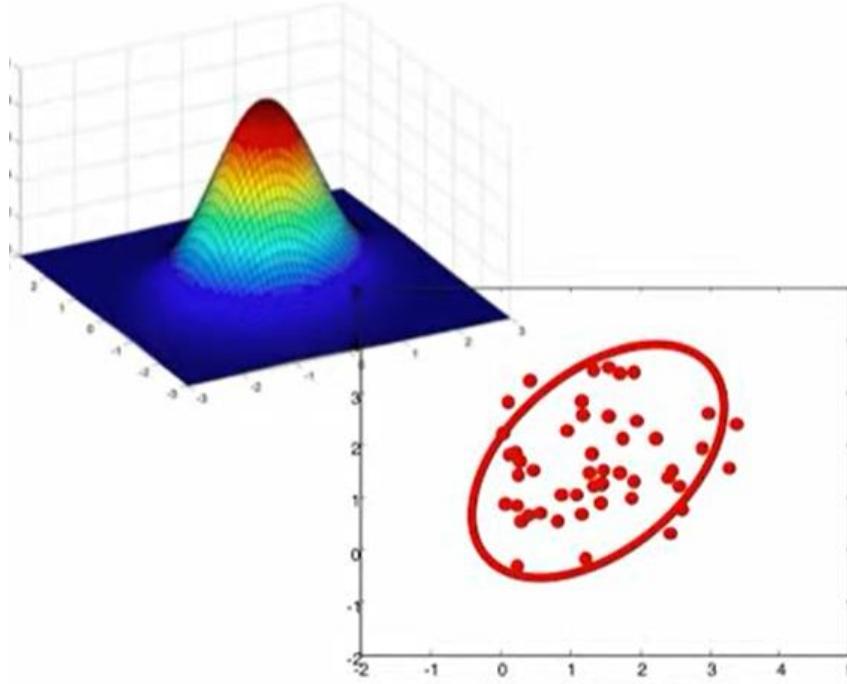
Multivariate Gaussian Distribution

- Similar to univariate case

$$\mathcal{N}(\underline{x} ; \underline{\mu}, \Sigma) = \frac{1}{(2\pi)^{d/2}} |\Sigma|^{-1/2} \exp \left\{ -\frac{1}{2} (\underline{x} - \underline{\mu}) \Sigma^{-1} (\underline{x} - \underline{\mu})^T \right\}$$

$\underline{\mu}$ = length-d row vector
 Σ = d x d matrix

$|\Sigma|$ = matrix determinant



Maximum likelihood estimate:

$$\hat{\mu} = \frac{1}{m} \sum_j \underline{x}^{(j)}$$

$$\hat{\Sigma} = \frac{1}{m} \sum_j (\underline{x}^{(j)} - \hat{\mu})^T (\underline{x}^{(j)} - \hat{\mu})$$

(average of dxd matrices)

Multivariate Gaussian Distribution

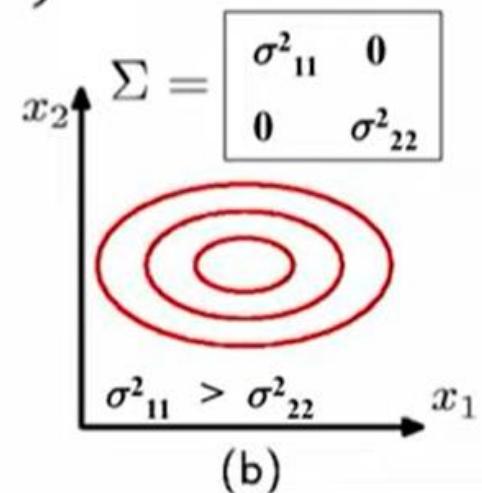
$$p(x_1) = \frac{1}{Z} \exp \left\{ -\frac{1}{2\sigma_1^2} (x_1 - \mu_1)^2 \right\} \quad p(x_2) = \frac{1}{Z_2} \exp \left\{ -\frac{1}{2\sigma_2^2} (x_2 - \mu_2)^2 \right\}$$

$$\underline{x} = [x_1 \ x_2]$$

$$p(x_1)p(x_2) = \frac{1}{Z_1 Z_2} \exp \left\{ -\frac{1}{2} (\underline{x} - \underline{\mu})^T \Sigma^{-1} (\underline{x} - \underline{\mu}) \right\}$$

$$\underline{\mu} = [\mu_1 \ \mu_2]$$

$$\Sigma = \text{diag}(\sigma_1^2, \ \sigma_2^2)$$



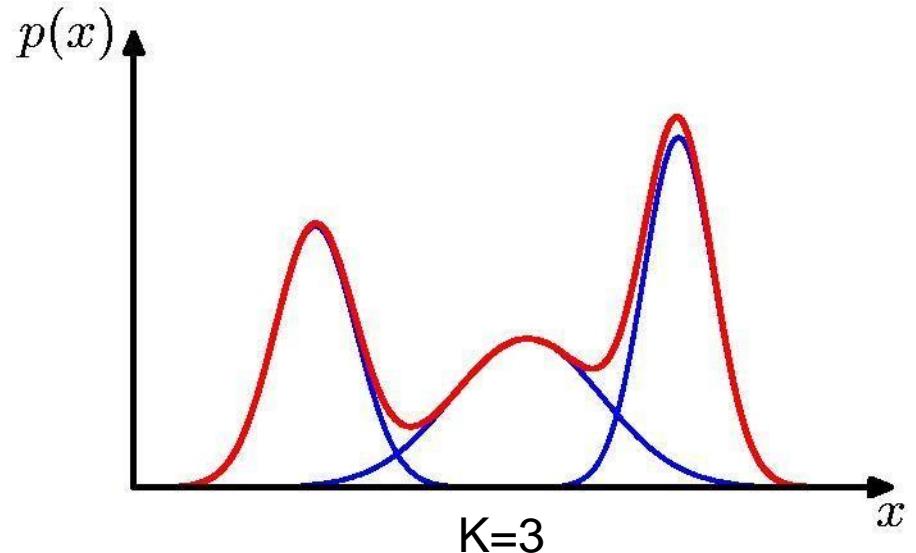
Mixtures of Gaussians

- Combine simple models into a complex model:

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

↑
Component
Mixing coefficient

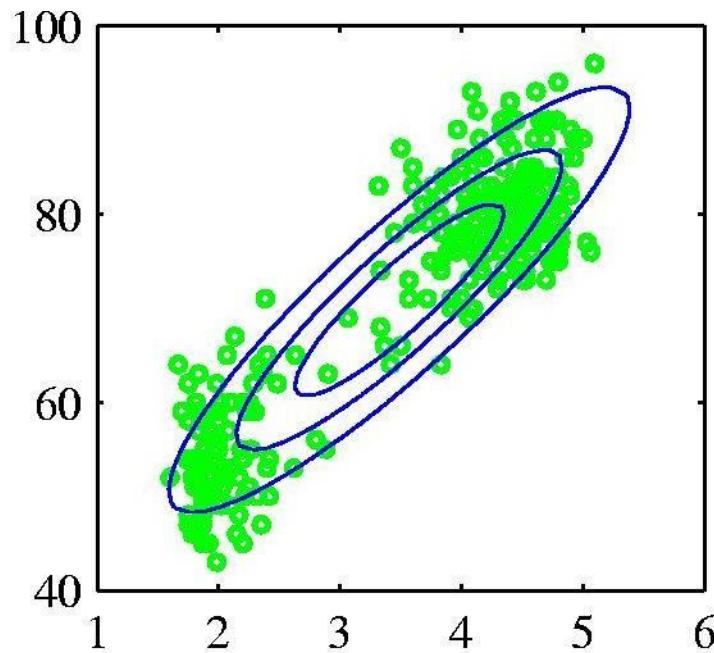
$$\forall k : \pi_k \geq 0 \quad \sum_{k=1}^K \pi_k = 1$$



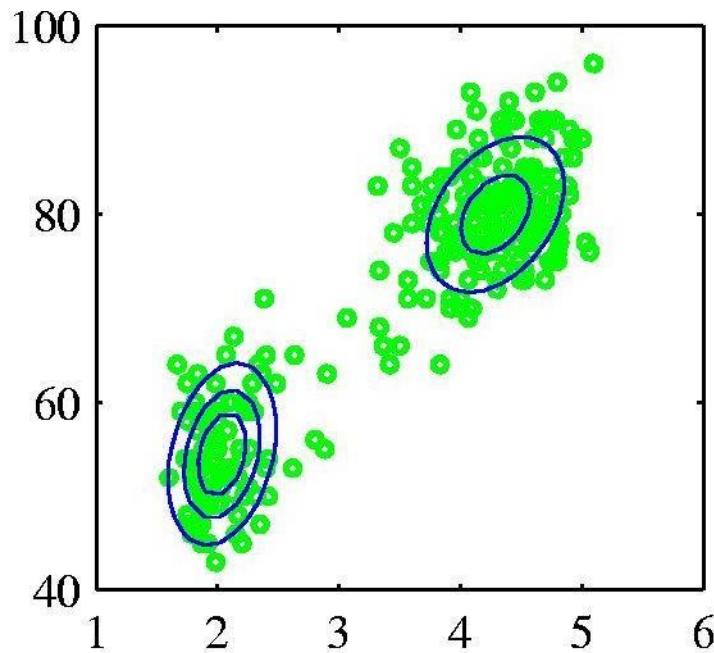
- Find parameters through EM (Expectation Maximization) algorithm

Probabilistic version: Mixtures of Gaussians

- Old Faithful data set



Single Gaussian

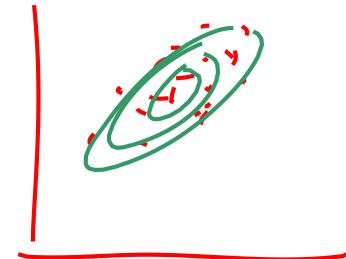


Mixture of two
Gaussians

Gaussian Mixture Model

- Data set

$$D = \{\mathbf{x}_n\} \quad n = 1, \dots, N$$



- Consider first a single Gaussian
- Assume observed data points generated independently

$$p(D|\boldsymbol{\mu}, \Sigma) = \prod_{n=1}^N \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}, \Sigma)$$

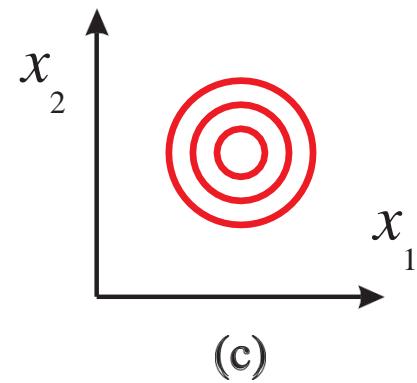
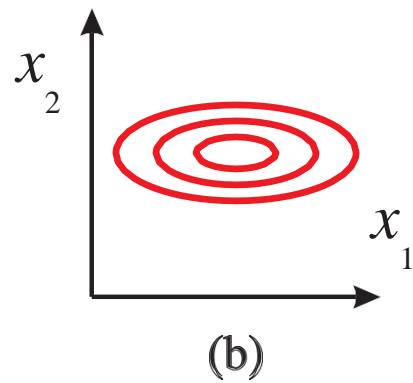
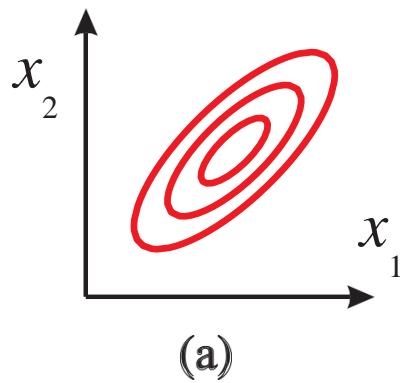
- Viewed as a function of the parameters, this is known as the *likelihood function*

The Gaussian Distribution

- Multivariate Gaussian

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}|^{1/2}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\}$$

mean covariance



Gaussian Mixture Model

- K-dimensional binary random variable z having a 1-of-K representation in which a particular element z_k is equal to 1 and all other elements are equal to 0.
- The values of z_k therefore satisfy $z_k \in \{0,1\}$
- K possible states for the vector z according to which element is nonzero.
- Joint distribution $p(x,z)$ in terms of a marginal distribution $p(z)$ and a conditional distribution $p(x|z)$,
- Marginal distribution over z is specified in terms of the mixing coefficients π_k , such that $p(z_k = 1) = \pi_k$

Gaussian Mixture Model

- Start with parameters describing each cluster
 - Mean μ_c , variance σ_c , “size” π_c
 - Probability distribution: $p(x) = \sum_c \pi_c \mathcal{N}(x ; \mu_c, \sigma_c)$
 - Equivalent “latent variable” form:

$$p(z = c) = \pi_c$$

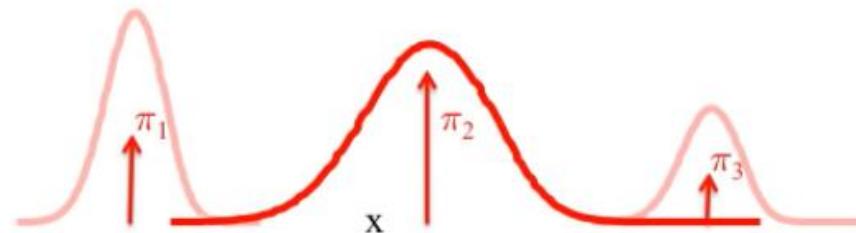
Select a mixture component with probability π

$$p(x|z = c) = \mathcal{N}(x ; \mu_c, \sigma_c)$$

Sample from that component’s Gaussian

“Latent assignment” z:
we observe x, but z is hidden

$p(x)$ = marginal over x



Sampling from the Gaussian

- To generate a data point:
 - first pick one of the components with probability π_k
 - then draw a sample \mathbf{x}_n from that component
- Repeat these two steps for each new data point

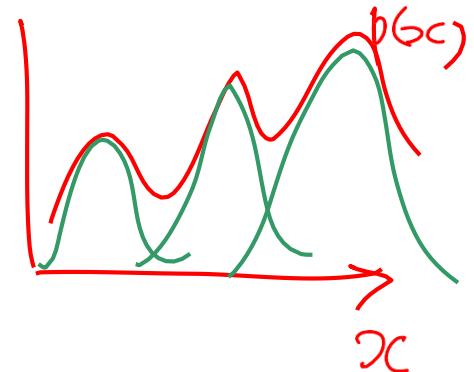
Fitting the Gaussian Mixture

- We wish to invert this process – given the data set, find the corresponding parameters:
 - mixing coefficients
 - means
 - covariances
- If we knew which component generated each data point, the maximum likelihood solution would involve fitting each component to the corresponding cluster
- Problem: the data set is unlabelled
- We shall refer to the labels as *latent* (= hidden) variables

Gaussian Mixture Model

- Linear super-position of Gaussians

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$



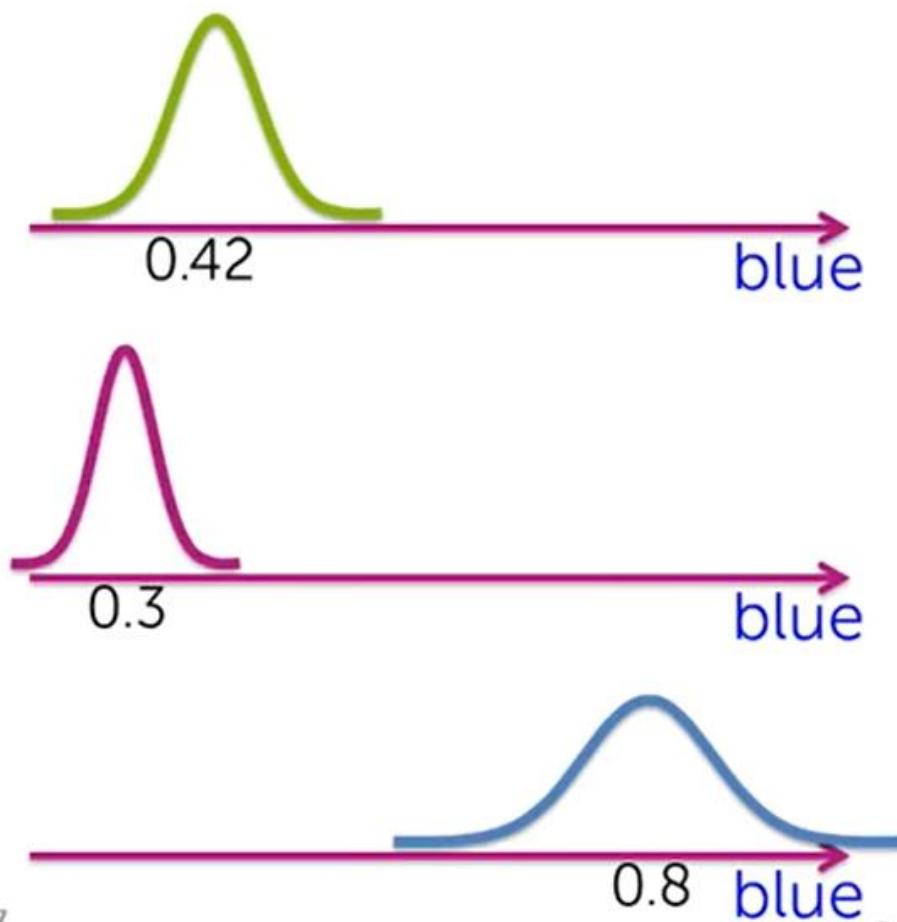
- Normalization and positivity require

$$\sum_{k=1}^K \pi_k = 1 \quad 0 \leq \pi_k \leq 1$$

- Can interpret the mixing coefficients as prior probabilities

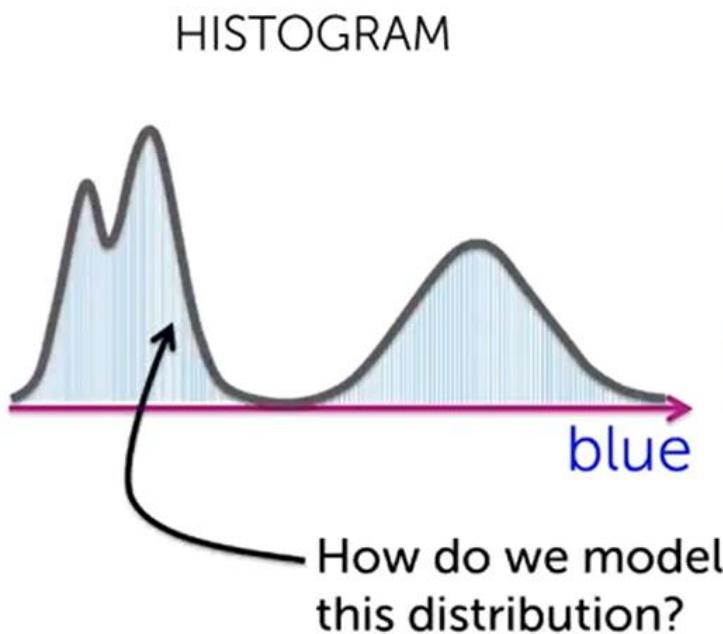
$$p(\mathbf{x}) = \sum_{k=1}^K p(k)p(\mathbf{x}|k)$$

Example



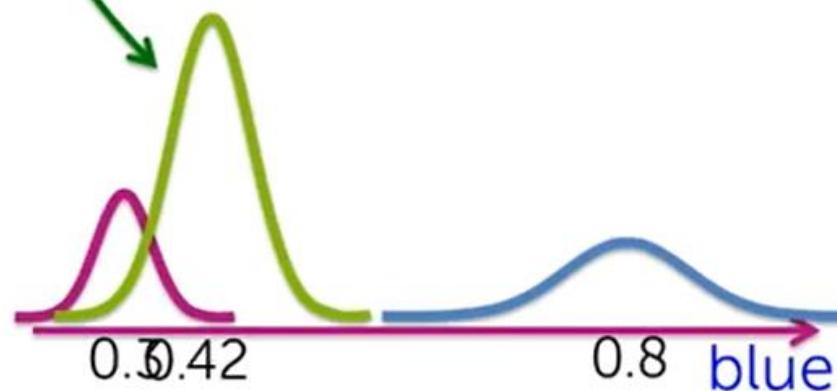
Example

Jumble of unlabeled images

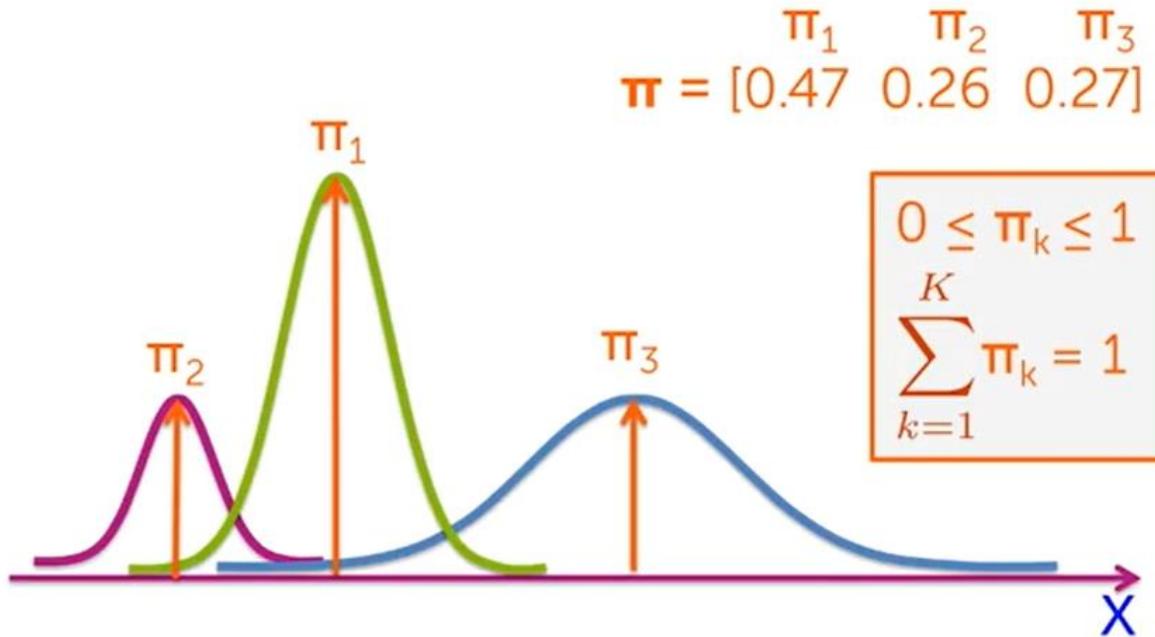


Example

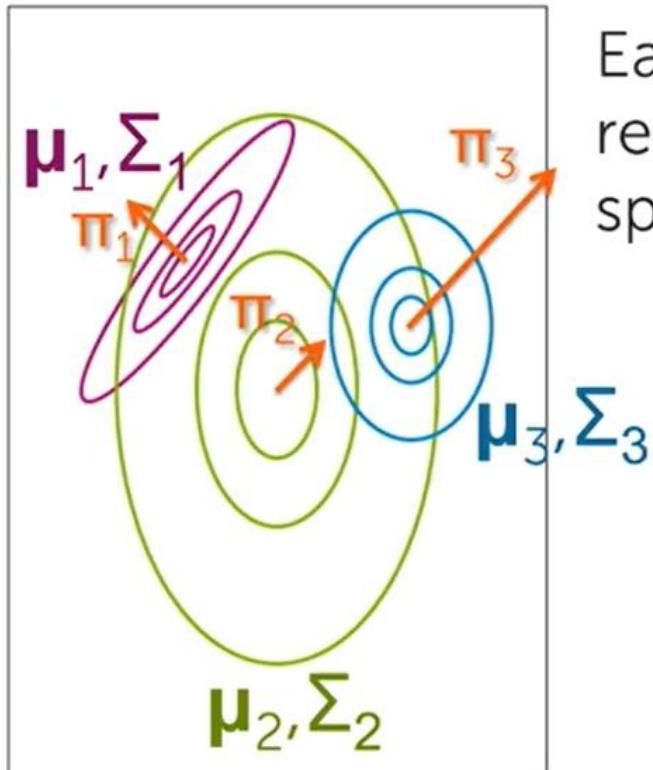
e.g., forest images are very likely in the collection



Associate weight with each Gaussian



Mixture of Gaussian



Each mixture component represents a unique cluster specified by:

$$\{\pi_k, \mu_k, \Sigma_k\}$$

Example

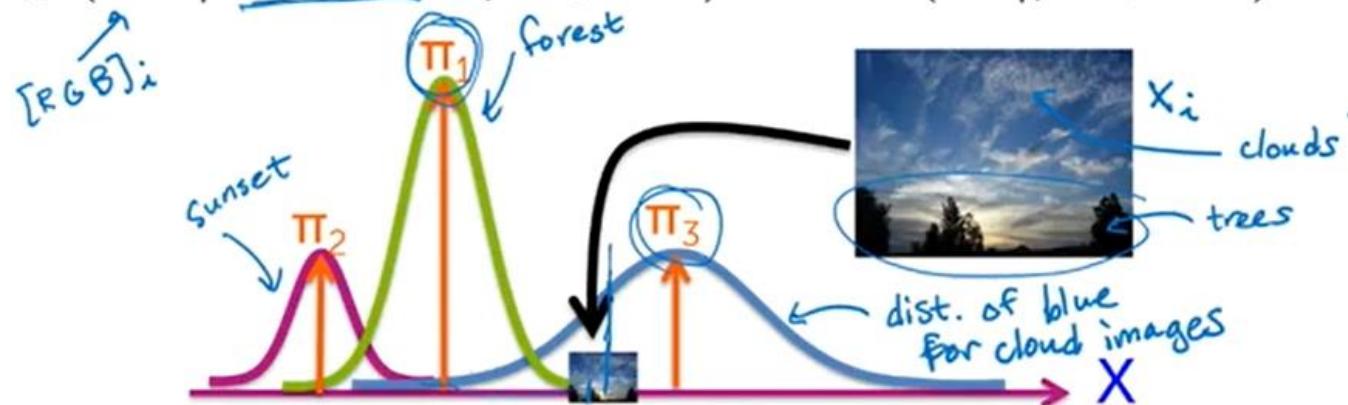
Without observing the image content, what's the probability it's from cluster k? (e.g., prob. of seeing "clouds" image)

$$p(z_i = k) = \underline{\pi_k} \quad \text{prior}$$

cluster assignment for obs. x_i

Given observation x_i is from cluster k, what's the likelihood of seeing x_i ? (e.g., just look at distribution for "clouds")

$$p(x_i | z_i = k, \mu_k, \Sigma_k) = N(x_i | \mu_k, \Sigma_k) \quad \text{likelihood}$$



Gaussian Mixture Model

- \mathbf{z} uses a 1-of- K representation, we can also write this distribution in the form

$$p(\mathbf{z}) = \prod_{k=1}^K \pi_k^{z_k}$$

- Conditional distribution of \mathbf{x} given a particular value for \mathbf{z} is a Gaussian

$$p(\mathbf{x}|\mathbf{z}) = \prod_{k=1}^K \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}$$

- Joint distribution is given by $p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$, and the marginal distribution of \mathbf{x} is then obtained by summing the joint distribution over all possible states of \mathbf{z} to give

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{z})p(\mathbf{x}|\mathbf{z}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Gaussian Mixture Model

- Conditional probability of z given x
- use $\gamma(z_k)$ to denote $p(z_k = 1 | x)$, whose value can be found using Bayes' theorem

$$\begin{aligned}\gamma(z_k) \equiv p(z_k = 1 | x) &= \frac{p(z_k = 1)p(x|z_k = 1)}{\sum_{j=1}^K p(z_j = 1)p(x|z_j = 1)} \\ &= \frac{\pi_k \mathcal{N}(x|\mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(x|\mu_j, \Sigma_j)}.\end{aligned}$$

- π_k as the prior probability of $z_k = 1$, and the quantity $\gamma(z_k)$ as the corresponding posterior probability once we have observed x .
-

Maximum Likelihood

Log of likelihood function:

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

- Maximizing the log likelihood function for a Gaussian mixture model turns out to be a more complex problem than for the case of a single Gaussian.
- The difficulty arises from the presence of the summation over k that appears inside the logarithm, so that the logarithm function no longer acts directly on the Gaussian.

GMM Problems and Solutions

- How to maximize the log likelihood
 - solved by expectation-maximization (EM) algorithm
- How to avoid singularities in the likelihood function
 - solved by a Bayesian treatment
- How to choose number K of components
 - also solved by a Bayesian treatment

Expectation Maximization (EM) Algorithm

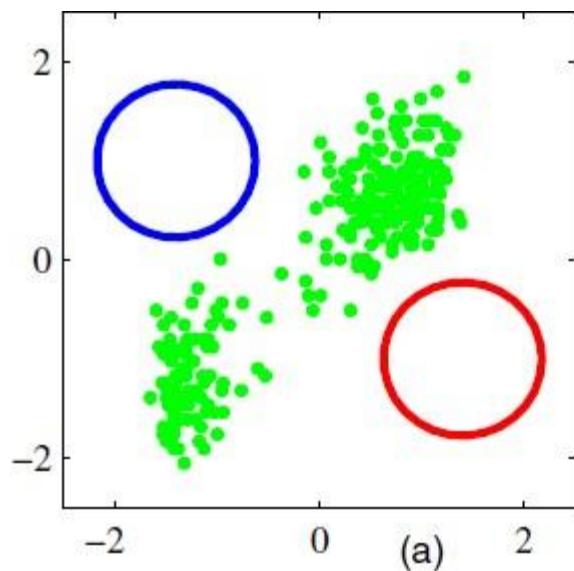


- We first choose some initial values for the means, covariances, and mixing coefficients.
- Then we alternate between the following two updates that we shall call the E step and the M step
- In the expectation step, or E step, we use the current values for the parameters to evaluate the posterior probabilities,
- We then use these probabilities in the maximization step, or M step, to re-estimate the means, covariances, and mixing
- In practice, the algorithm is deemed to have converged when the change in the log likelihood function, or alternatively in the parameters, falls below some threshold

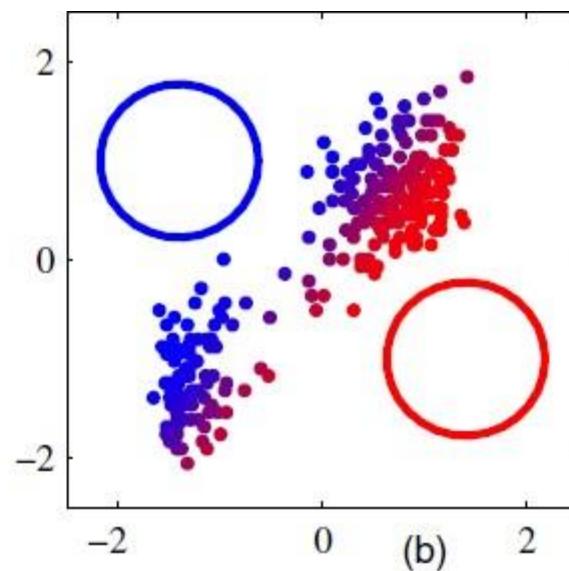
EM Algorithm – Informal Derivation

- The solutions are not closed form since they are coupled
- Suggests an iterative scheme for solving them:
 - make initial guesses for the parameters
 - alternate between the following two stages:
 1. E-step: evaluate responsibilities
 2. M-step: update parameters using ML results
- Each EM cycle guaranteed not to decrease the likelihood

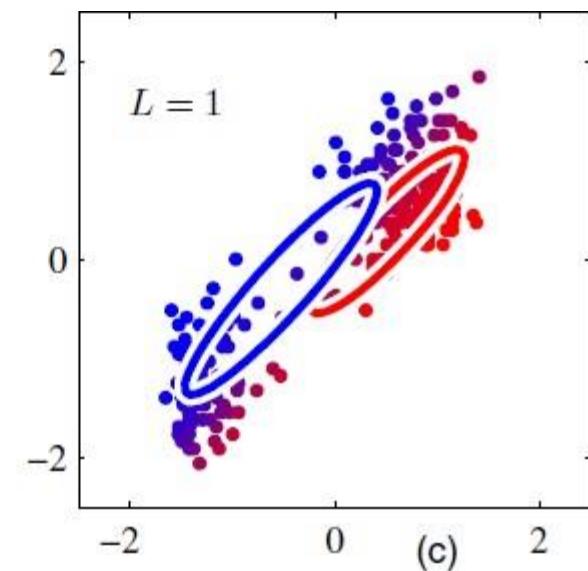
Initialization



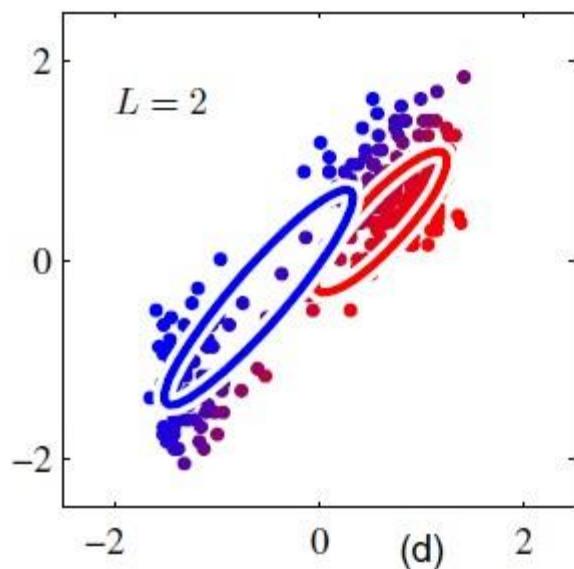
E step



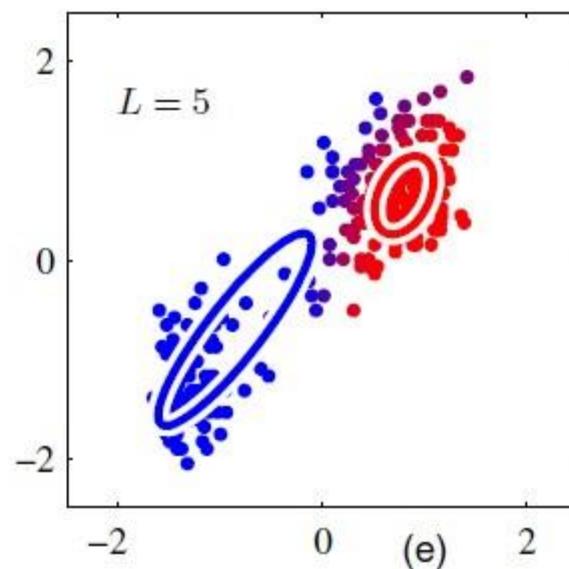
M step



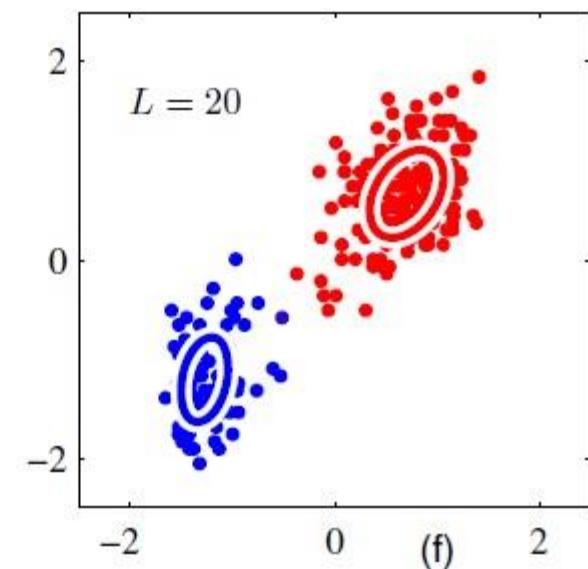
$L = 2$



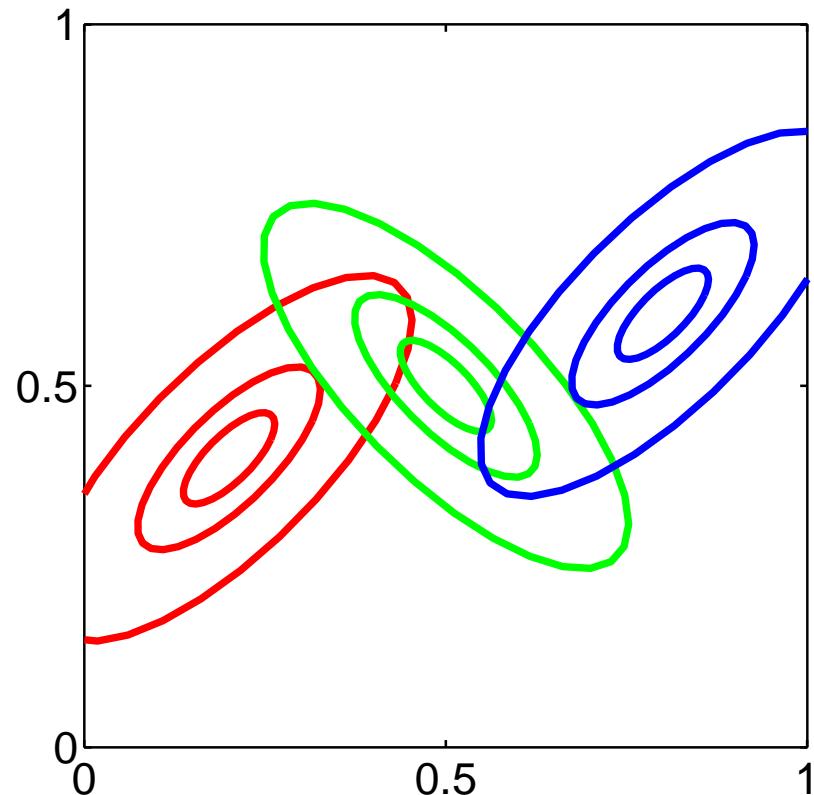
$L = 5$



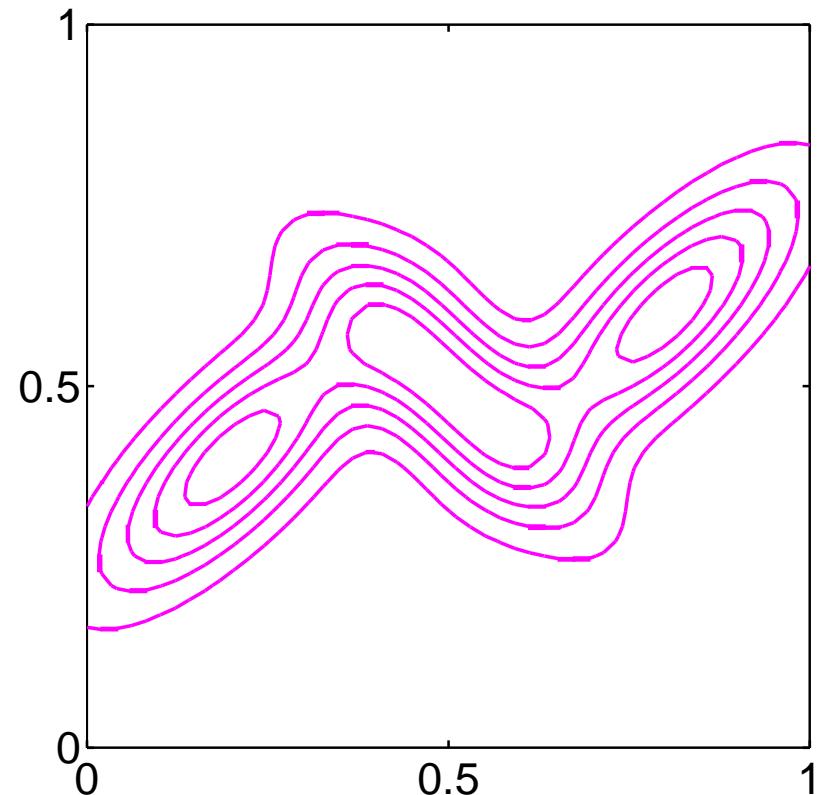
$L = 20$



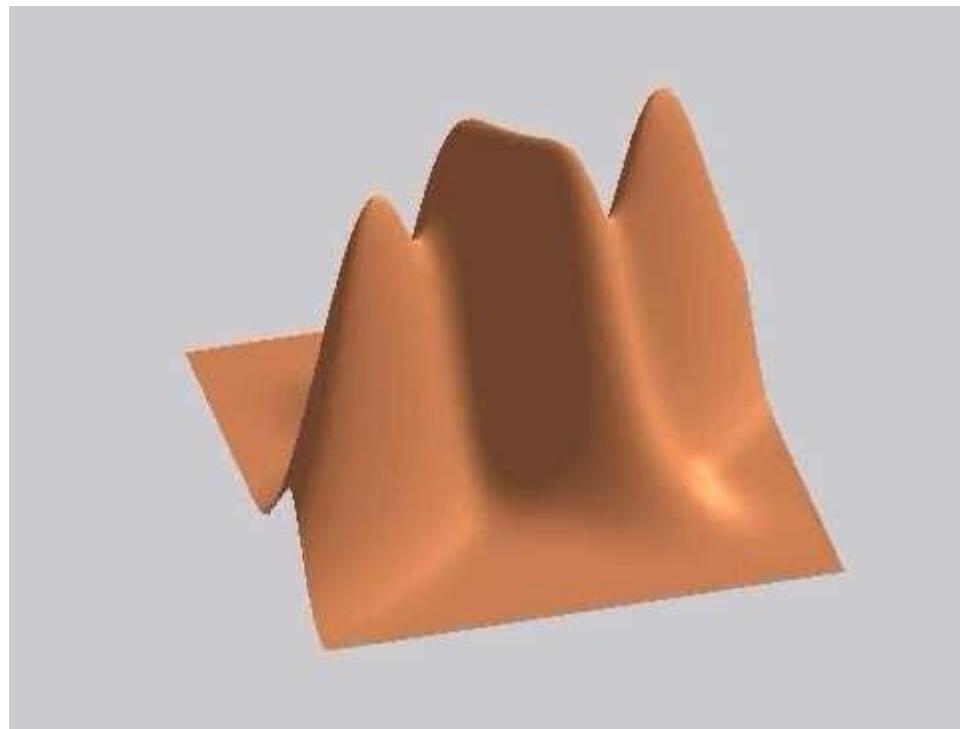
Example: Mixture of 3 Gaussians



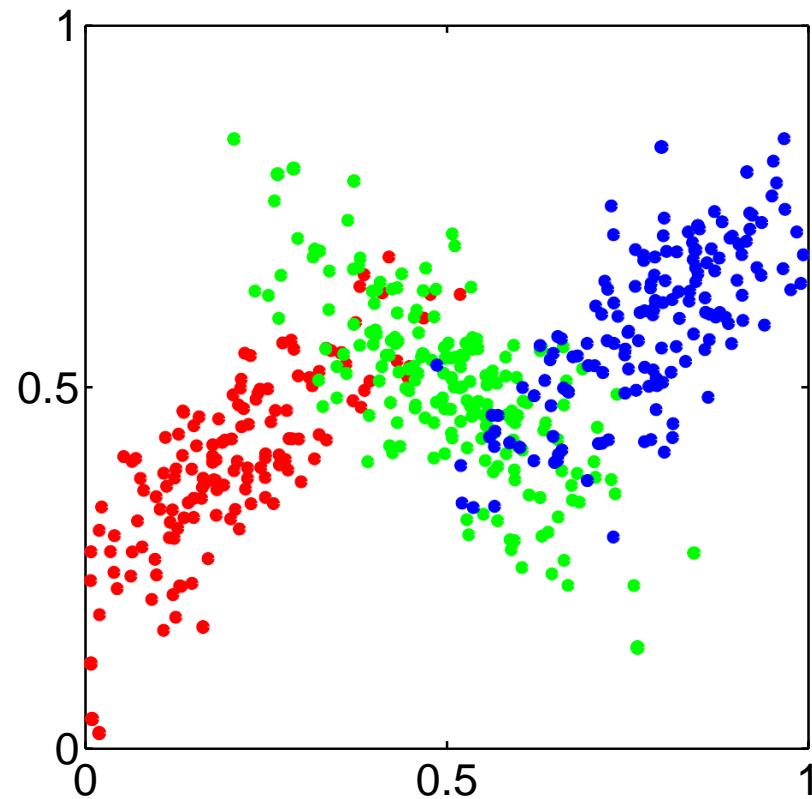
Contours of Probability Distribution



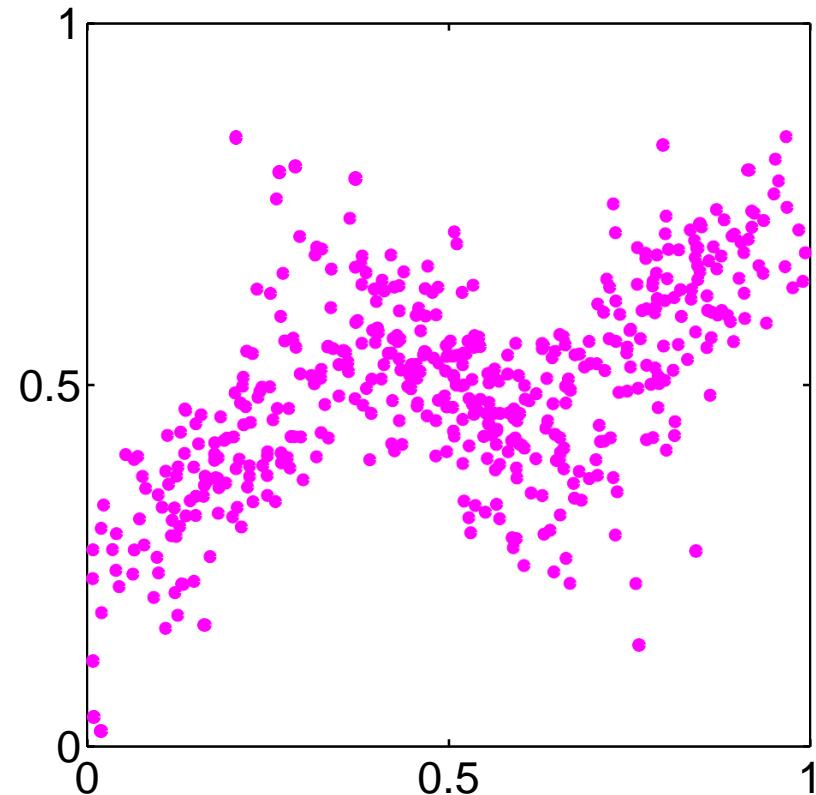
Surface Plot



Synthetic Data Set



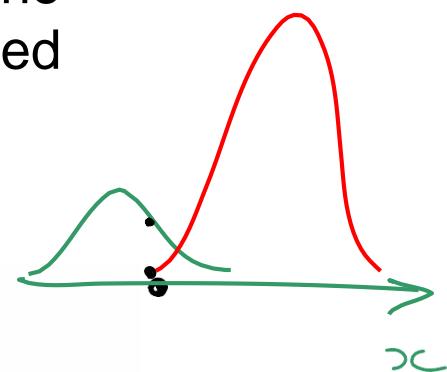
Synthetic Data Set Without Labels



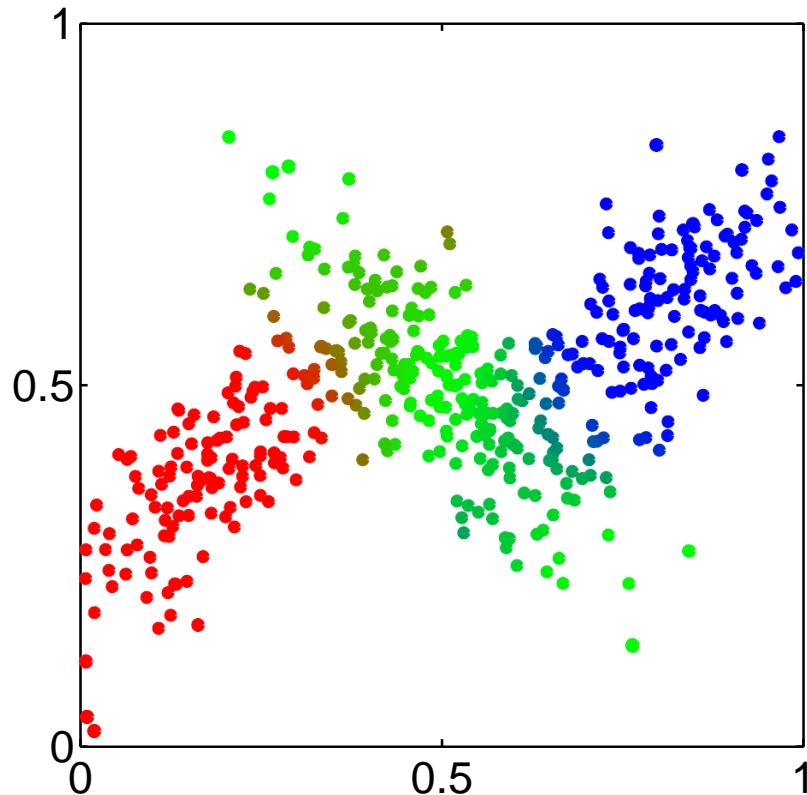
Posterior Probabilities

- We can think of the mixing coefficients as prior probabilities for the components
- For a given value of \mathbf{x} we can evaluate the corresponding posterior probabilities, called *responsibilities*
- These are given from Bayes' theorem by

$$\begin{aligned}\gamma_k(\mathbf{x}) \equiv p(k|\mathbf{x}) &= \frac{p(k)p(\mathbf{x}|k)}{p(\mathbf{x})} \\ &= \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}\end{aligned}$$



Posterior Probabilities (colour coded)



EM algorithm for GMM

1. Initialize the means μ_k , covariances Σ_k and mixing coefficients π_k , and evaluate the initial value of the log likelihood.
2. **E step:** Evaluate the responsibilities using the current parameter values

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

EM algorithm for GMM

3. **M step:** Re-estimate the parameters using the current responsibilities

$$\begin{aligned}
 \boldsymbol{\mu}_k^{\text{new}} &= \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) \mathbf{x}_n \\
 \boldsymbol{\Sigma}_k^{\text{new}} &= \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) (\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}}) (\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}})^T \\
 \pi_k^{\text{new}} &= \frac{N_k}{N} \quad \text{where } N_k = \sum_{n=1}^N \gamma(z_{nk})
 \end{aligned}$$

4. Evaluate the log likelihood

$$\ln p(\mathbf{X} | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

Segmentation as clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

Grouping pixels based
on **intensity** similarity



Feature space: intensity value (1-d)



K=2



*quantization of the feature space;
segmentation label map*

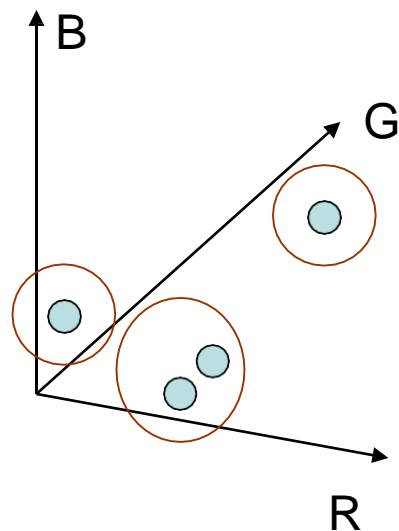
K=3



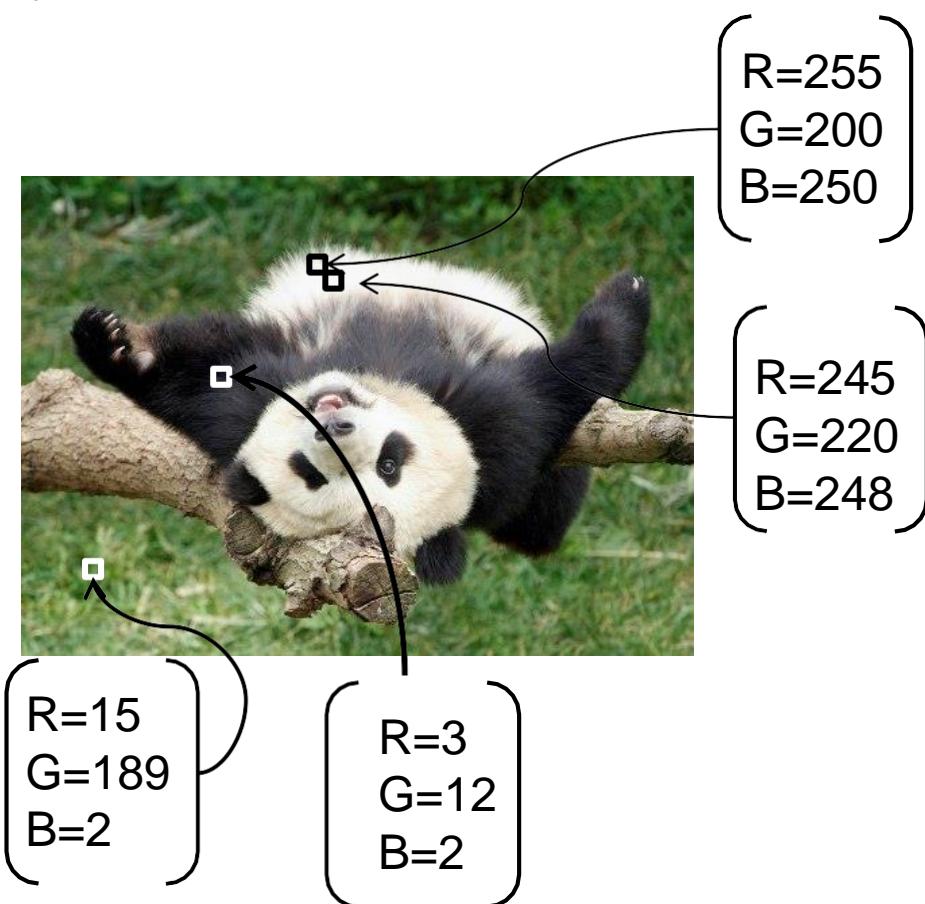
Segmentation as clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

Grouping pixels based on **color** similarity



Feature space: color value (3-d)

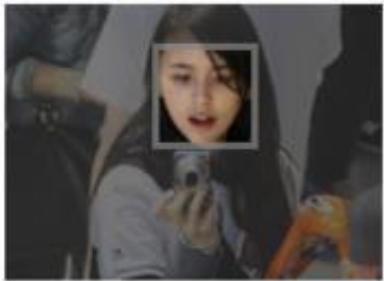


Application: Face Recognition

facebook  3 Search Home

Who's in These Photos?

The photos you uploaded were grouped automatically so you can quickly label and notify friends in these pictures. (Friends can always untag themselves.)



Who is this?



Who is this?



Who is this?



Who is this?



Who is this?



Who is this?

References

Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition

[Gaussian Mixture Models for Clustering – YouTube](#)

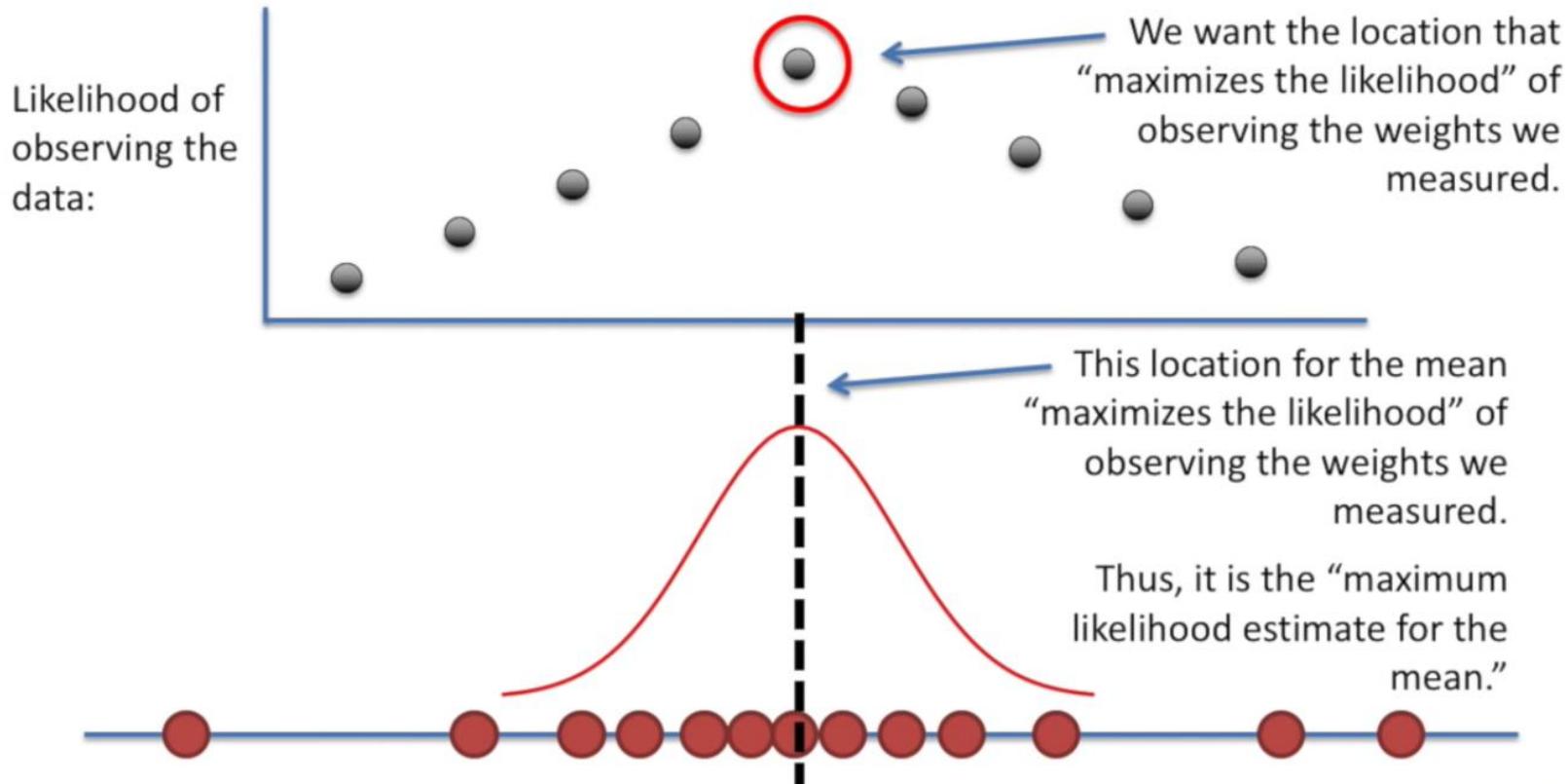
[Clustering \(4\): Gaussian Mixture Models and EM – YouTube](#)

<https://www.youtube.com/watch?v=TG6Bh-NFhA0>

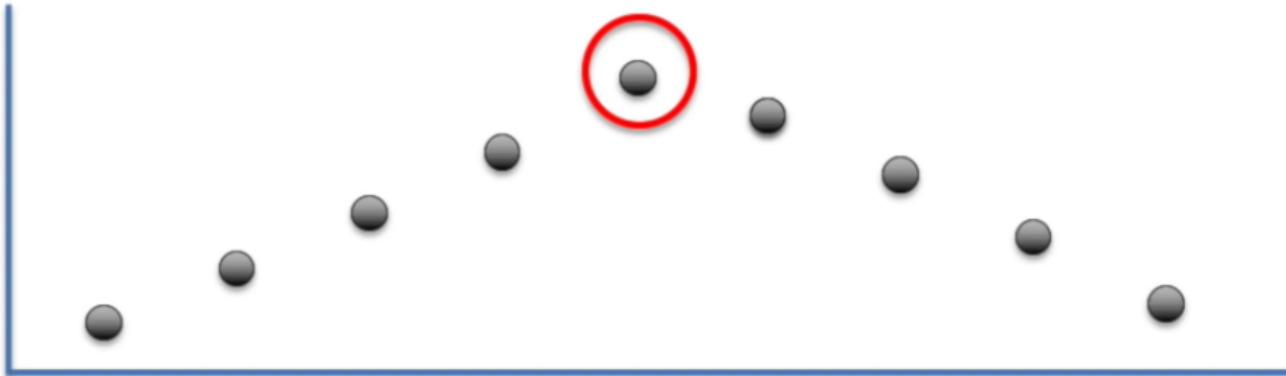
Multivariate Gaussian

<https://www.youtube.com/watch?v=eho8xH3E6mE>

Maximum likelihood Estimate



Likelihood of observing the data:



Now when someone says that they have the maximum likelihood estimates for the mean or the standard deviation, or for something else...

... you know that they found the value for the mean or the standard deviation (or for whatever) that maximizes the likelihood that you observed the things you observed.

