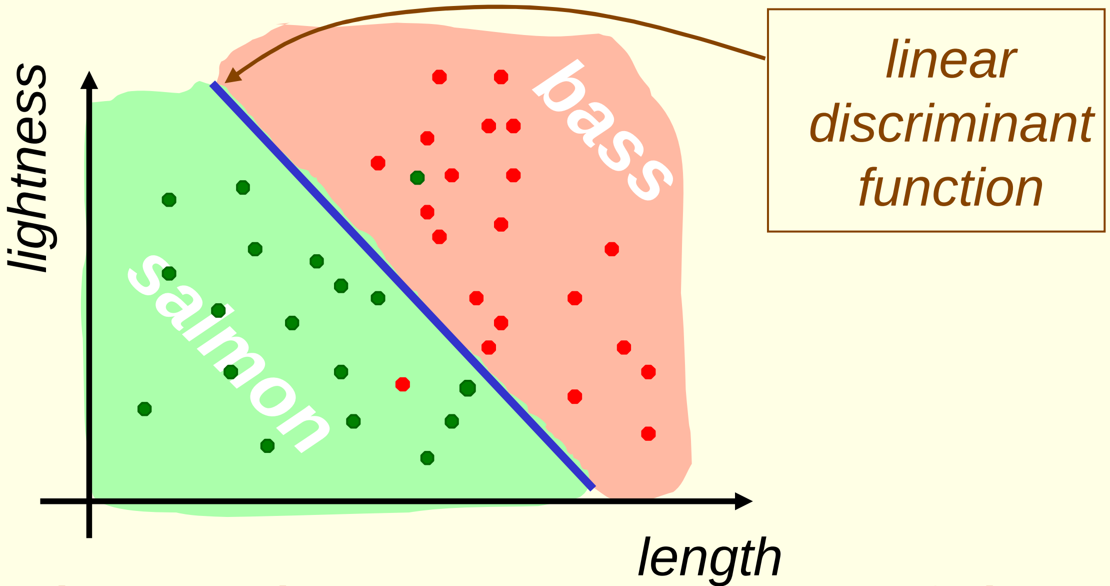
Linear Discriminant Functions

C11

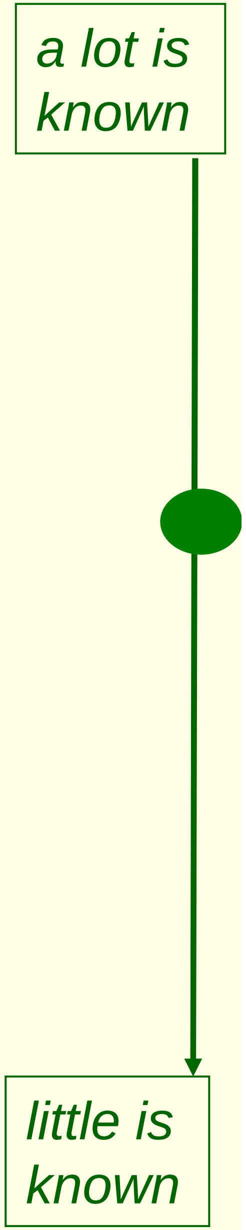
Linear discriminant functions on Road Map

- No probability distribution (no shape or parameters are known)
- Labeled data 

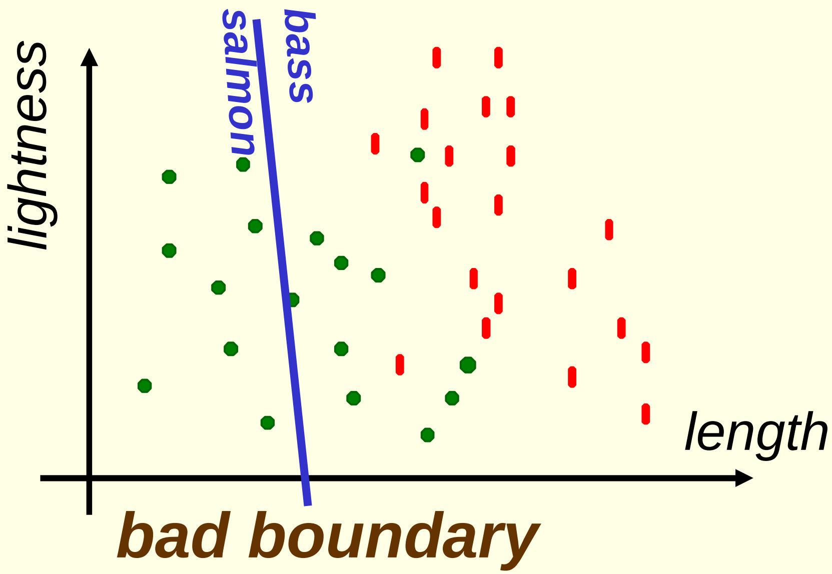
- The shape of discriminant functions is known

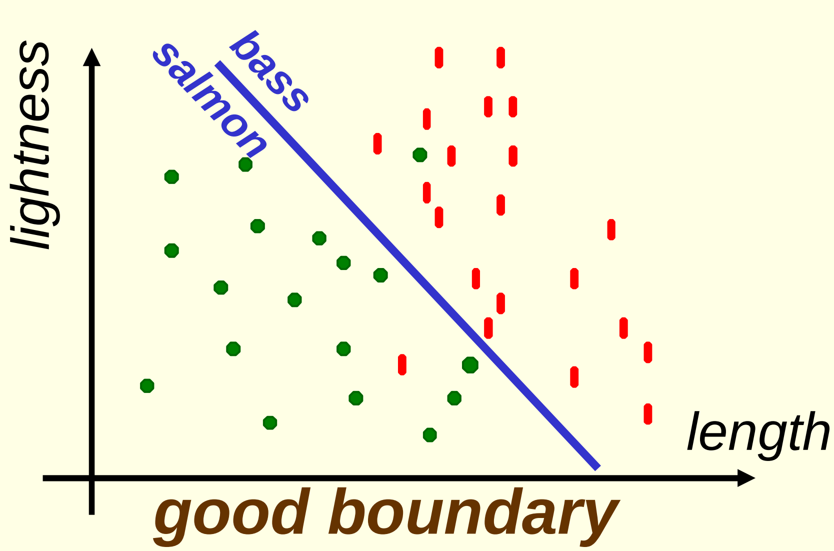


- Need to estimate parameters of the discriminant function (parameters of the line in case of linear discriminant)



Linear Discriminant Functions: Basic Idea





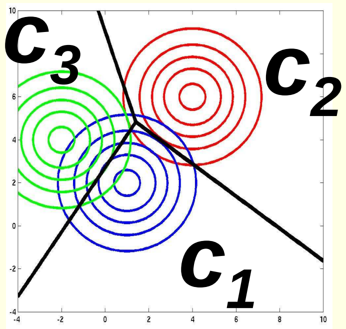
- Have samples from 2 classes $\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}$
- Assume 2 classes can be separated by a linear boundary $\boldsymbol{l}(\theta)$ with some unknown parameters θ
- Fit the "best" boundary to data by optimizing over parameters θ. **How**?
- Minimize a criterion function.
  - Obvious choice: Minimize classification error on training data. (Does not guarantee small test error)

Parametric Methods vs. Discriminant Functions

Assume the shape of density for classes is known $\boldsymbol{p_1}(\boldsymbol{x}|\theta_1)$, $\boldsymbol{p_2}(\boldsymbol{x}|\theta_2)$,…

Estimate θ_1, θ_2,… from data

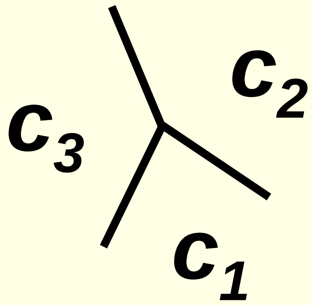
Use a Bayesian classifier to find decision regions



Assume discriminant functions are of known shape $\boldsymbol{l}(\theta_1)$, $\boldsymbol{l}(\theta_2)$, with parameters θ_1, θ_2,…

Estimate θ_1, θ_2,… from data

Use discriminant functions for classification



- In theory, Bayesian classifier minimizes the risk
  - In practice, do not have confidence in assumed model shapes
  - In practice, do not really need the actual density functions in the end
- Estimating accurate density functions is much harder than estimating accurate discriminant functions
- Some argue that estimating densities should be skipped
  - Why solve a harder problem than needed ?

LDF: Introduction

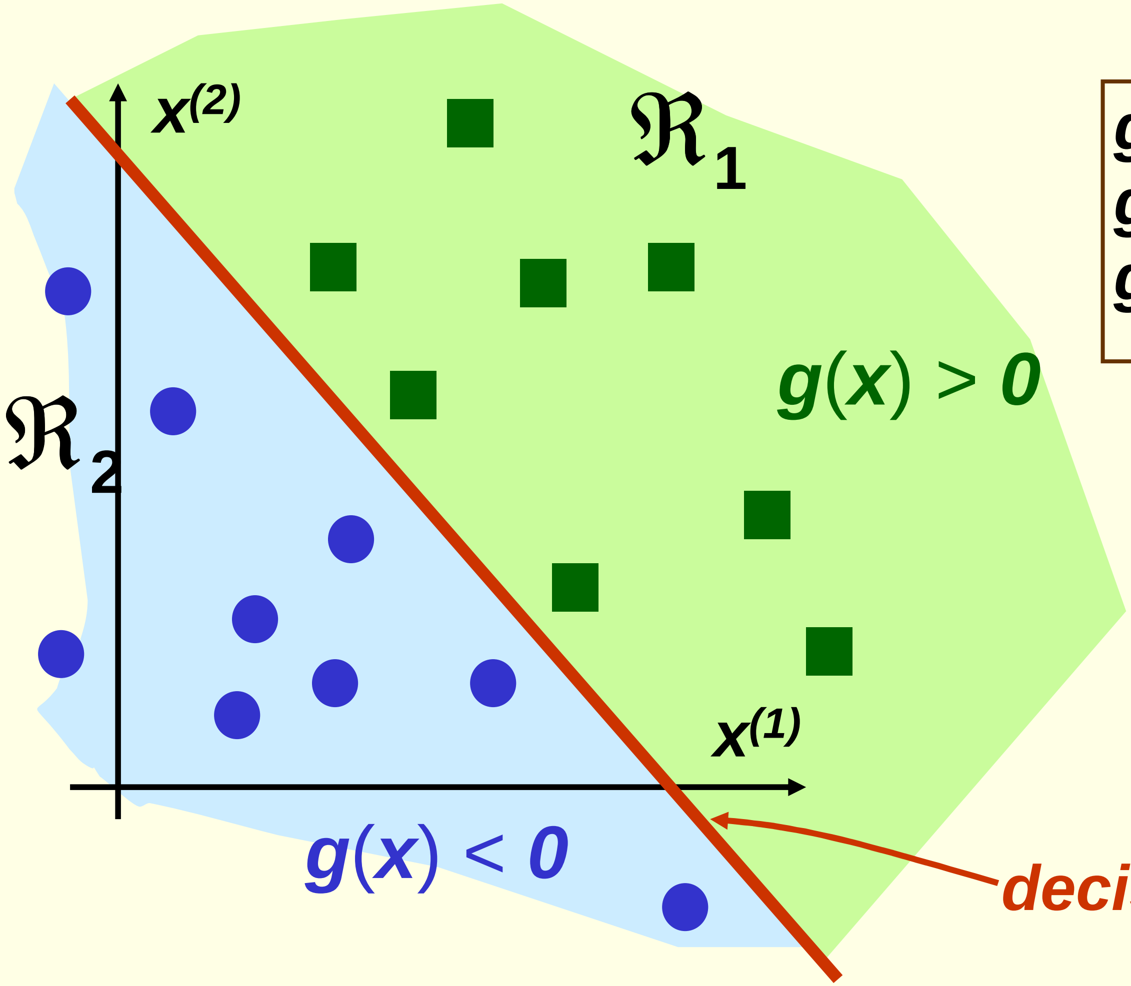
- Discriminant functions can be more general than linear
- For now, we will study linear discriminant functions
  - Simple model (should try simpler models first)
  - Analytically tractable
- Linear Discriminant functions are optimal for Gaussian distributions with equal covariance
- May not be optimal for other data distributions, but they are very simple to use
- Knowledge of class densities is not required when using linear discriminant functions
  - we can say that this is a non-parametric approach

LDF: 2 Classes

- A discriminant function is linear if it can be written as

$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0$$

  - $\mathbf{w}$ is called the weight vector and w_0 called bias or threshold

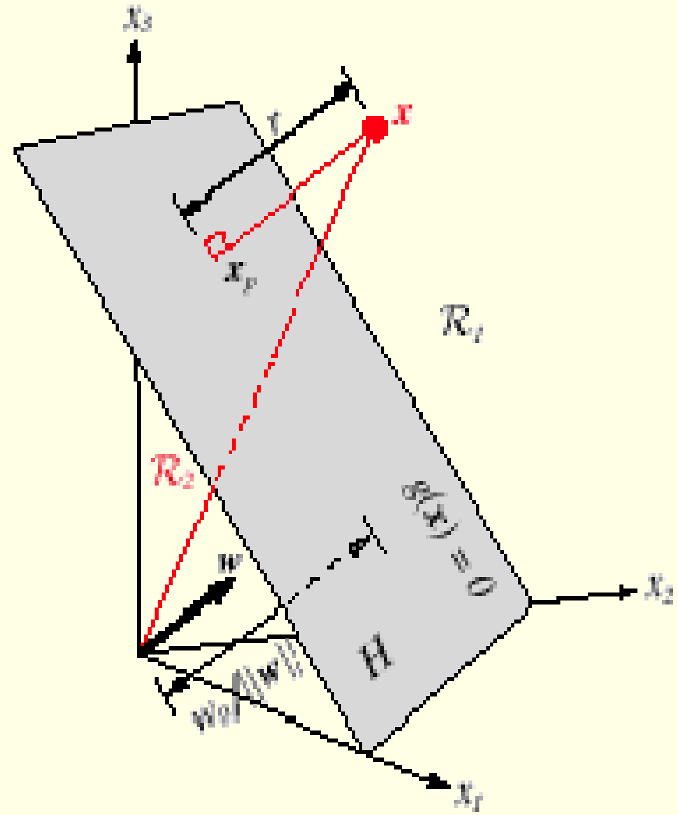


$$\begin{aligned} g(x) > 0 &\Rightarrow x \in \text{class } 1 \\ g(x) < 0 &\Rightarrow x \in \text{class } 2 \\ g(x) = 0 &\Rightarrow \text{either class} \end{aligned}$$

LDF: 2 Classes

- Decision boundary $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{w}^t\boldsymbol{x} + \boldsymbol{w}_0 = 0$ is a ***hyperplane***

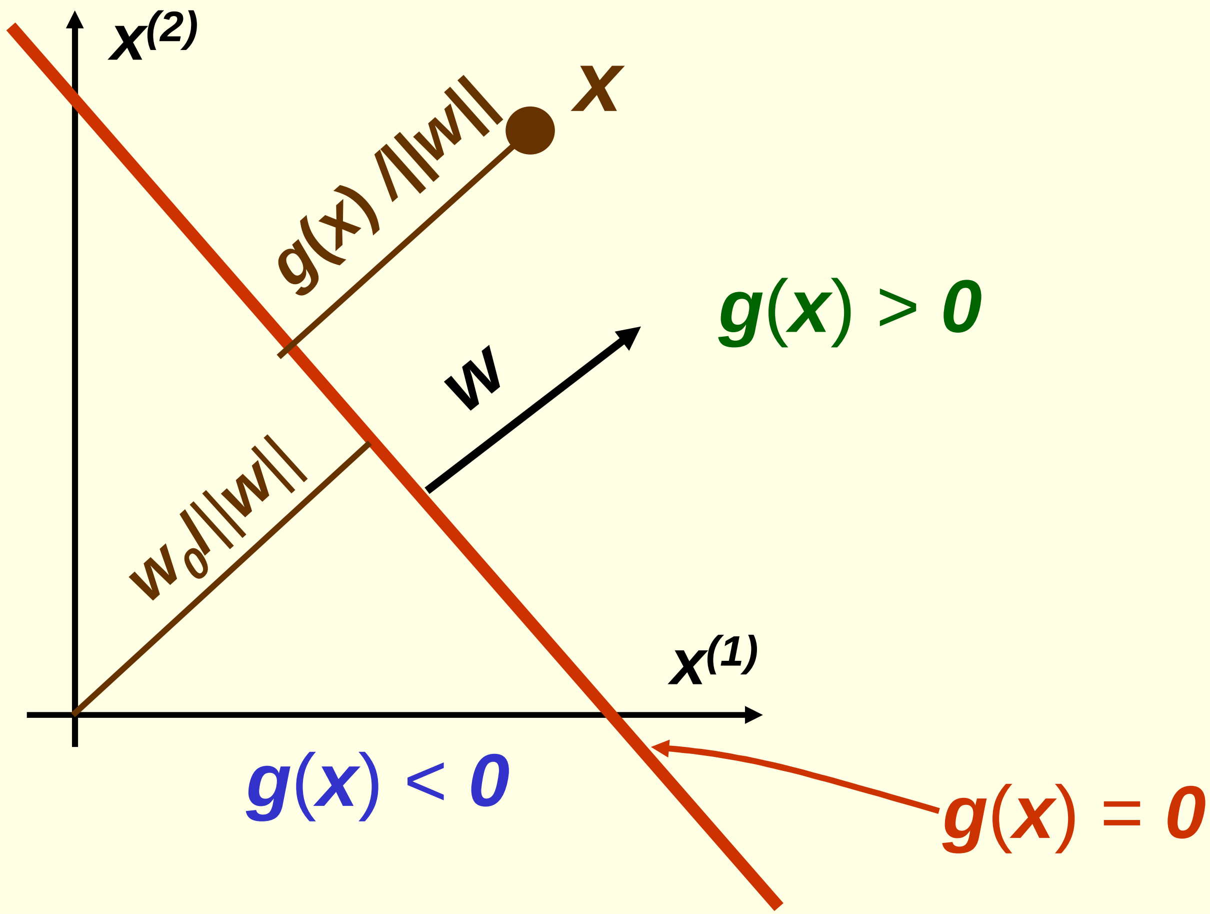
- A hyperplane is
  - a point in 1D
  - a line in 2D
  - a plane in 3D



LDF: 2 Classes

$$g(x) = w^t x + w_0$$

- w determines orientation of the decision hyperplane
- w_0 determines location of the decision surface



LDF: Many Classes

- Suppose we have ***m*** classes
- Define ***m*** linear discriminant functions

$$g_i(x) = w_i^t x + w_{i0} \qquad \mathbf{i = 1,...,m}$$

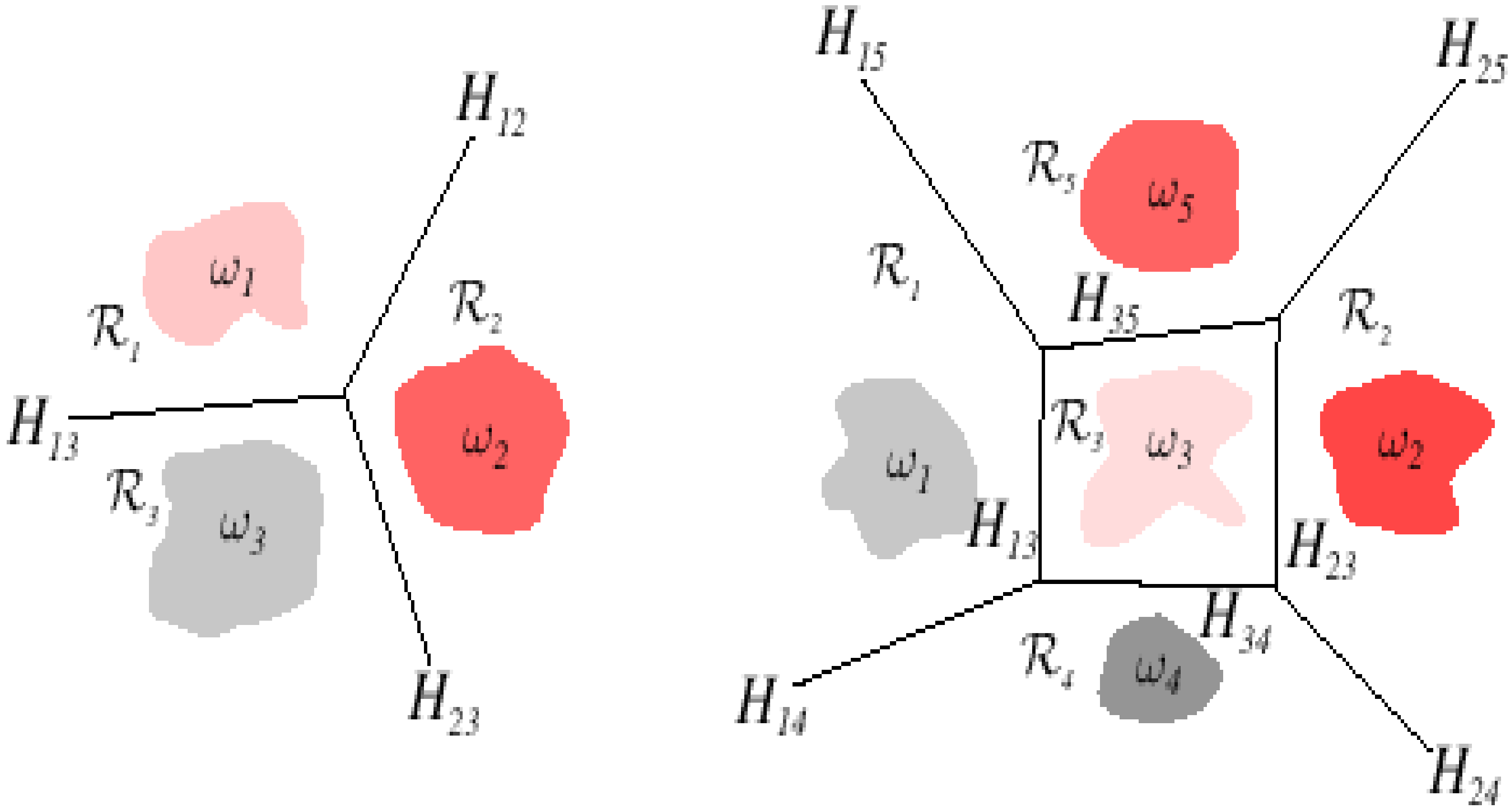
- Given ***x***, assign class c_i if

$$g_i(x) \geq g_j(x) \qquad \forall \mathbf{j \neq i}$$

- Such classifier is called a ***linear machine***

- A linear machine divides the feature space into ***c*** decision regions, with $g_i(x)$ being the largest discriminant if ***x*** is in the region R_i

LDF: Many Classes



LDF: Many Classes

- For a two contiguous regions R_i and R_j; the boundary that separates them is a portion of hyperplane $\boldsymbol{H_{ij}}$ defined by:

$$\boldsymbol{g_i(x) = g_j(x)} \quad \Leftrightarrow \boldsymbol{w_i^t x + w_{i0} = w_j^t x + w_{j0}}$$

$$\Leftrightarrow \boldsymbol{\left(w_i - w_j\right)^t x + \left(w_{i0} - w_{j0}\right) = 0}$$

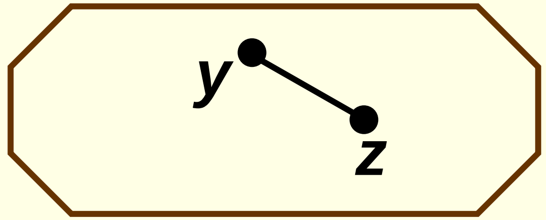
- Thus $\boldsymbol{w_i} - \boldsymbol{w_j}$ is normal to $\boldsymbol{H_{ij}}$
- And distance from $\boldsymbol{x}$ to $\boldsymbol{H_{ij}}$ is given by

$$\boldsymbol{d(x, H_{ij}) = \frac{g_i(x) - g_j(x)}{\left\| w_i - w_j \right\|}}$$

LDF: Many Classes

- Decision regions for a linear machine are ***convex***

$$\boldsymbol{y},\boldsymbol{z} \in \boldsymbol{R_i} \Rightarrow \alpha\,\boldsymbol{y} + (1-\alpha)\boldsymbol{z} \in \boldsymbol{R_i}$$

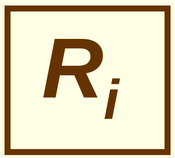


R_i

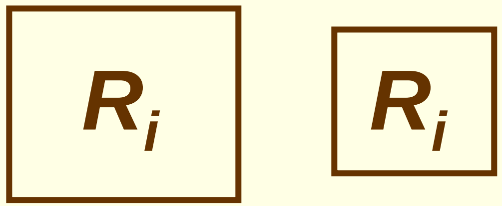
$$\forall j \neq i \quad g_i(\boldsymbol{y}) \geq g_j(\boldsymbol{y}) \text{ and } g_i(\boldsymbol{z}) \geq g_j(\boldsymbol{z}) \Leftrightarrow$$

$$\Leftrightarrow \forall j \neq i \quad g_i(\alpha\,\boldsymbol{y} + (1-\alpha)\boldsymbol{z}) \geq g_j(\alpha\,\boldsymbol{y} + (1-\alpha)\boldsymbol{z})$$

- In particular, decision regions must be spatially contiguous



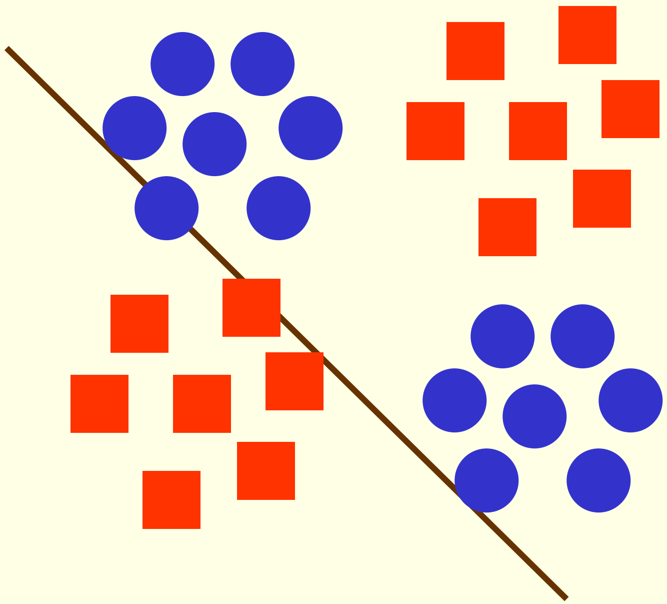
R_j is a valid decision region



R_j is not a valid decision region

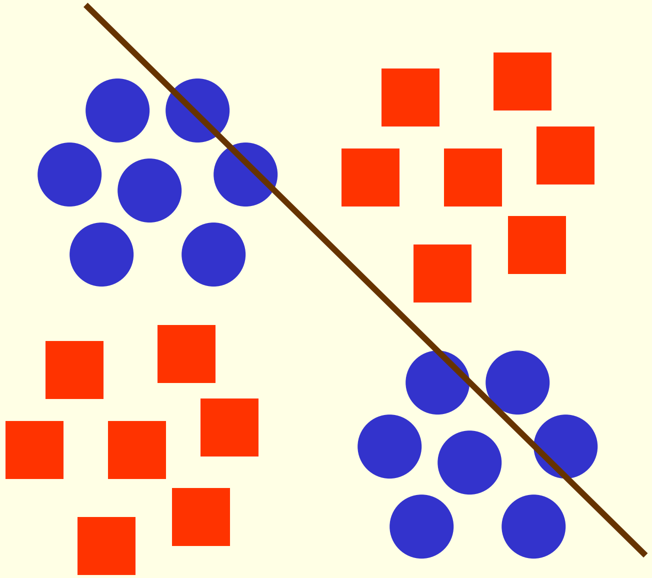
LDF: Many Classes

- Thus applicability of linear machine to mostly limited to unimodal conditional densities $\boldsymbol{p}(\boldsymbol{x}|\theta)$
  - even though we did not assume any parametric models
- Example:



LDF: Many Classes

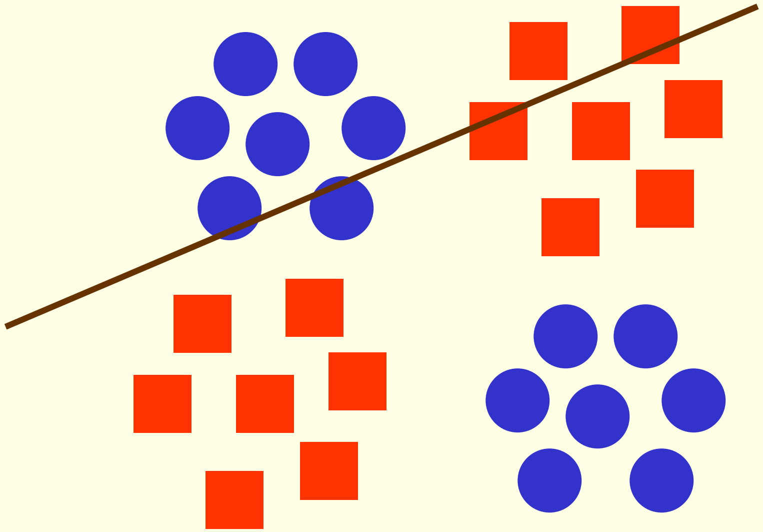
- Thus applicability of linear machine to mostly limited to unimodal conditional densities $\boldsymbol{p}(\boldsymbol{x}|\theta)$
  - even though we did not assume any parametric models
- Example:



LDF: Many Classes

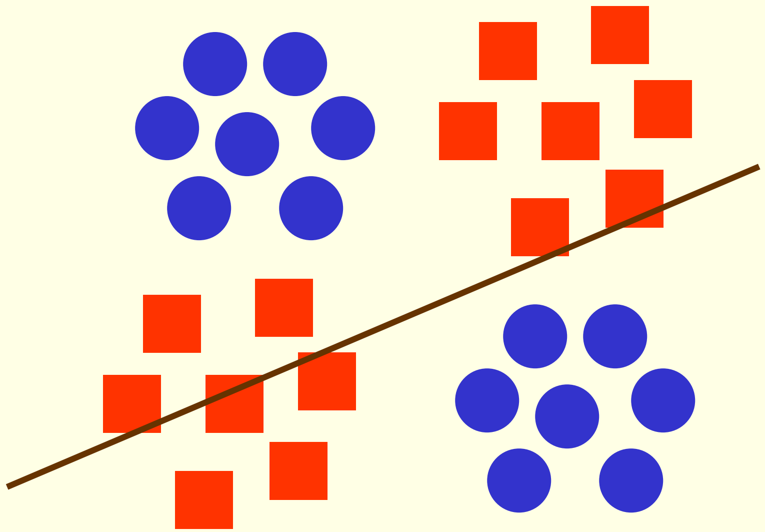
- Thus applicability of linear machine to mostly limited to unimodal conditional densities $\boldsymbol{p}(\boldsymbol{x}|\theta)$
  - even though we did not assume any parametric models

- Example:



LDF: Many Classes

- Thus applicability of linear machine to mostly limited to unimodal conditional densities $\boldsymbol{p}(\boldsymbol{x}|\theta)$
  - even though we did not assume any parametric models
- Example:



  - need non-contiguous decision regions
  - thus linear machine will fail

LDF: Augmented feature vector

- Linear discriminant function: $g(x) = w^t x + w_0$
- Can rewrite it: $$g(x) = \underbrace{\begin{bmatrix} w_0 & w^t \end{bmatrix}}_{\text{new weight vector } a} \underbrace{\begin{bmatrix} 1 \\ x \end{bmatrix}}_{\text{new feature vector } y} = a^t y = g(y)$$
- ***y*** is called the ***augmented feature vector***
- Added a dummy dimension to get a completely equivalent new ***homogeneous*** problem

old problem

$$g(x) = w^t x + w_0$$

$$\begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$$

new problem

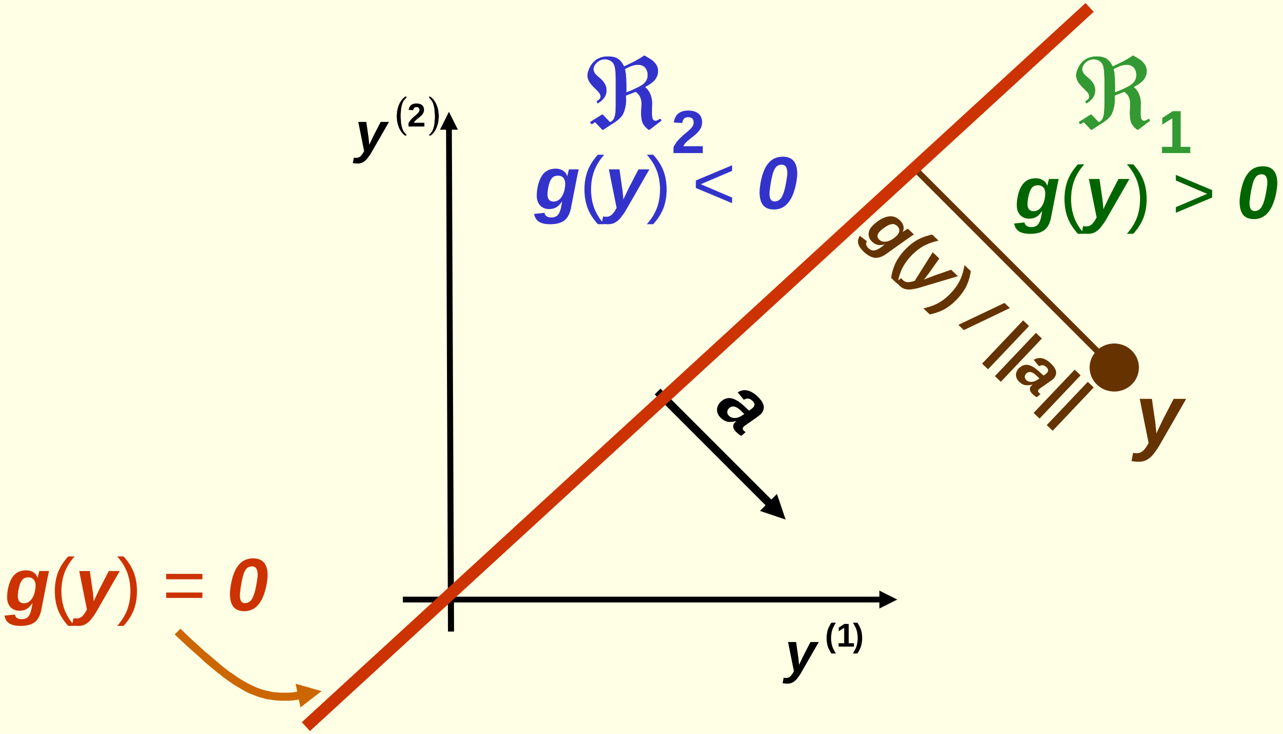
$$g(y) = a^t y$$

$$\begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}$$

LDF: Augmented feature vector

- Feature augmenting is done for simpler notation
- From now on we always assume that we have augmented feature vectors
  - Given samples $\boldsymbol{x_1},\dots,\boldsymbol{x_n}$ convert them to augmented samples $\boldsymbol{y_1},\dots,\boldsymbol{y_n}$ by adding a new dimension of value 1

$$\boldsymbol{y}_i = \begin{bmatrix} \mathbf{1} \\ \boldsymbol{x}_i \end{bmatrix}$$



LDF: Training Error

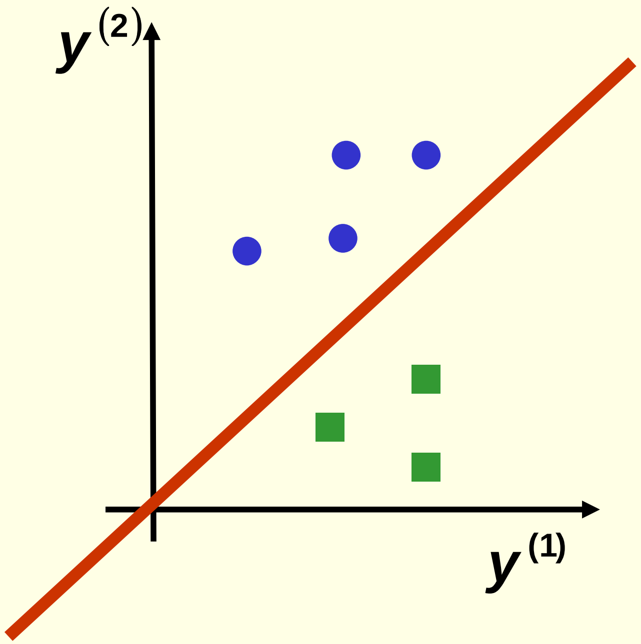
- For the rest of the lecture, assume we have 2 classes
- Samples $\boldsymbol{y_1},\ldots,\boldsymbol{y_n}$ some in class 1, some in class 2
- Use these samples to determine weights $\boldsymbol{a}$ in the discriminant function $g(\boldsymbol{y}) = \boldsymbol{a}^t \boldsymbol{y}$
- What should be our criterion for determining $\boldsymbol{a}$?
  - For now, suppose we want to minimize the training error (that is the number of misclassifed samples $\boldsymbol{y_1},\ldots,\boldsymbol{y_n}$)
- Recall that
$$g(y_i) > 0 \Rightarrow y_i \ \ \textit{classified } c_1$$
$$g(y_i) < 0 \Rightarrow y_i \ \ \textit{classified } c_2$$
- Thus training error is ***0*** if
$$\begin{cases} g(y_i) > 0 & \forall y_i \in c_1 \\ g(y_i) < 0 & \forall y_i \in c_2 \end{cases}$$

LDF: Problem "Normalization"

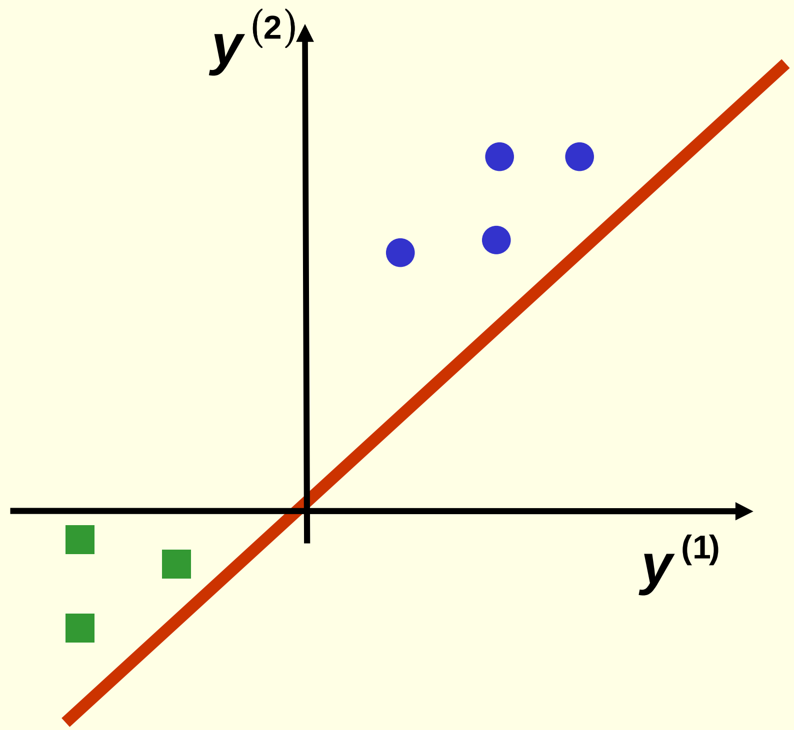
- Thus training error is ***0*** if $$\begin{cases} \boldsymbol{a}^t \boldsymbol{y}_i > \mathbf{0} & \forall \boldsymbol{y}_i \in \boldsymbol{c}_1 \\ \boldsymbol{a}^t \boldsymbol{y}_i < \mathbf{0} & \forall \boldsymbol{y}_i \in \boldsymbol{c}_2 \end{cases}$$
- Equivalently, training error is ***0*** if
$$\begin{cases} \boldsymbol{a}^t \boldsymbol{y}_i > \boldsymbol{0} & \forall \boldsymbol{y}_i \in \boldsymbol{c}_1 \\ \boldsymbol{a}^t (-\boldsymbol{y}_i) > \boldsymbol{0} & \forall \boldsymbol{y}_i \in \boldsymbol{c}_2 \end{cases}$$
- This suggest problem "normalization":
  1. Replace all examples from class $\boldsymbol{c_2}$ by their negative
  $$\boldsymbol{y}_i \to -\boldsymbol{y}_i \qquad \forall \boldsymbol{y}_i \in \boldsymbol{c}_2$$
  2. Seek weight vector ***a*** s.t.
  $$\boldsymbol{a}^t \boldsymbol{y}_i > \mathbf{0} \qquad \forall \boldsymbol{y}_i$$
     - If such ***a*** exists, it is called a *separating* or *solution* vector
     - Original samples $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$ can indeed be separated by a line then

LDF: Problem "Normalization"

before normalization



after "normalization"

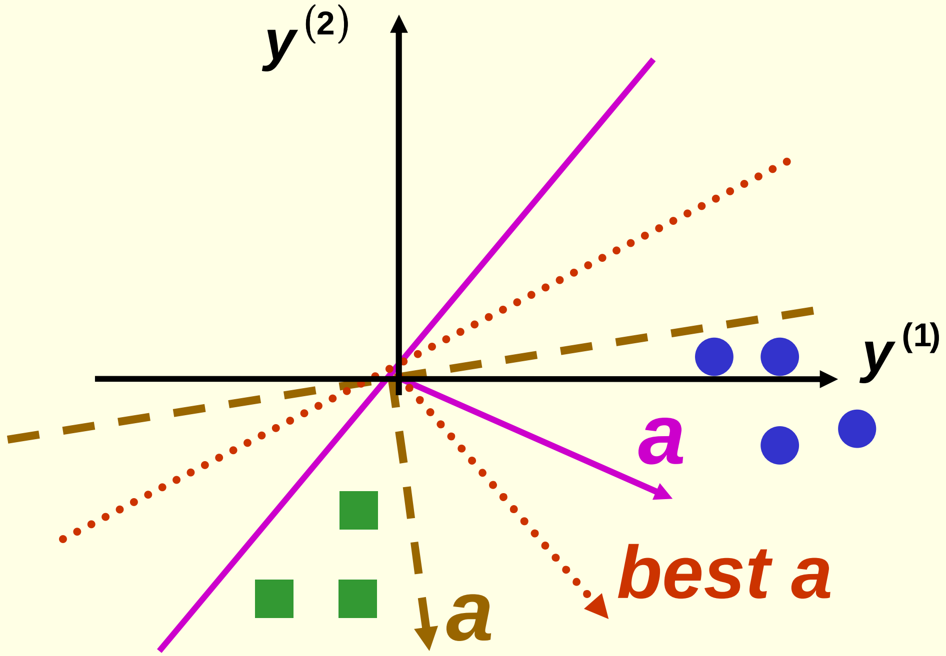


Seek a hyperplane that separates patterns from different categories ⇐ Seek hyperplane that puts *normalized* patterns on the same (positive) side

LDF: Solution Region

- Find weight vector $\boldsymbol{a}$ s.t. for all samples $\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}$

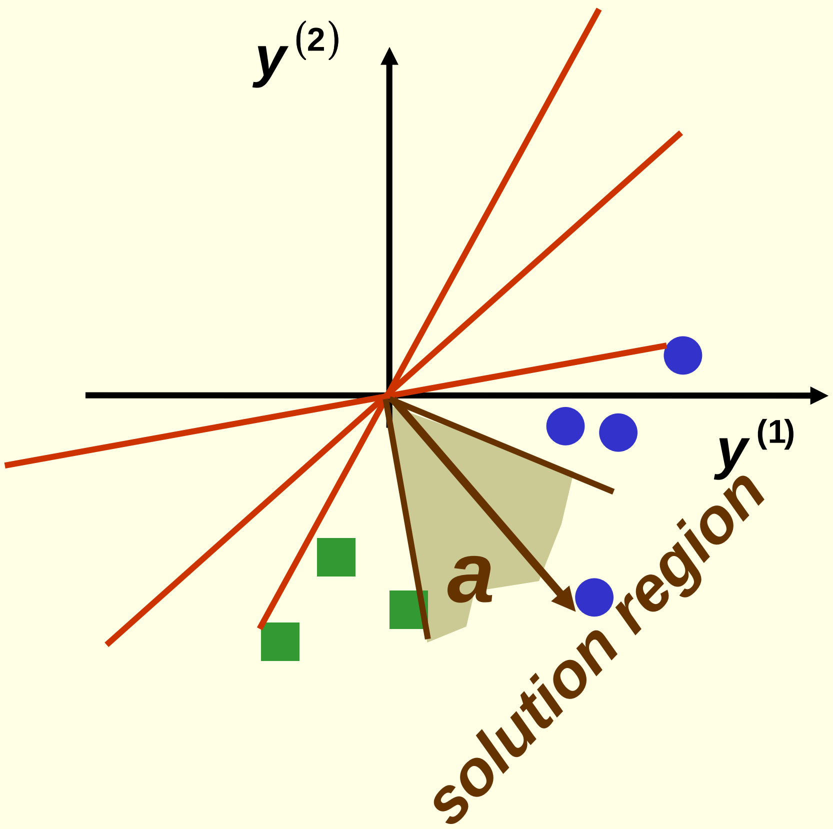
$$\boldsymbol{a}^t \boldsymbol{y}_i = \sum_{k=0}^{d} a_k y_i^{(k)} > 0$$



- In general, there are many such solutions $\boldsymbol{a}$

LDF: Solution Region

- ***Solution region*** for ***a***: set of all possible solutions
  - defined in terms of normal ***a*** to the separating hyperplane



Optimization

- Need to minimize a function of many variables

$$J(\mathbf{x}) = J(x_1, \dots, x_d)$$

- We know how to minimize ***J***(***x***)
  - Take partial derivatives and set them to zero

$$\begin{bmatrix} \frac{\partial}{\partial x_1} J(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial x_d} J(\mathbf{x}) \end{bmatrix} = \nabla J(\mathbf{x}) = 0$$

gradient

- However solving analytically is not always easy
  - Would you like to solve this system of nonlinear equations?

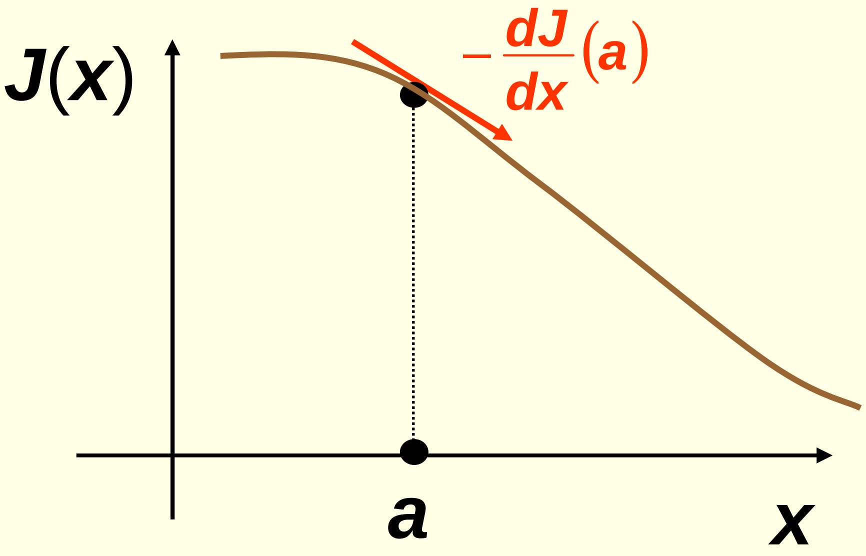
$$\begin{cases} \sin(x_1^2 + x_2^3) + e^{x_4^2} = 0 \\ \cos(x_1^2 + x_2^3) + \log(x_5^3)^{x_4^2} = 0 \end{cases}$$

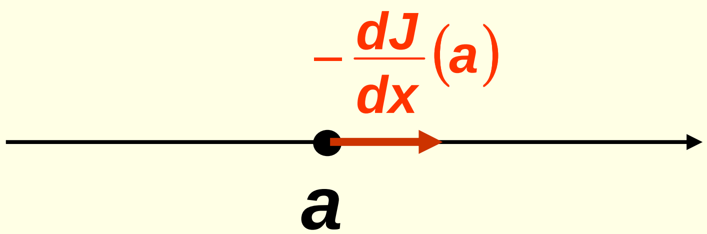
  - Sometimes it is not even possible to write down an analytical expression for the derivative, we will see an example later today

Optimization: Gradient Descent

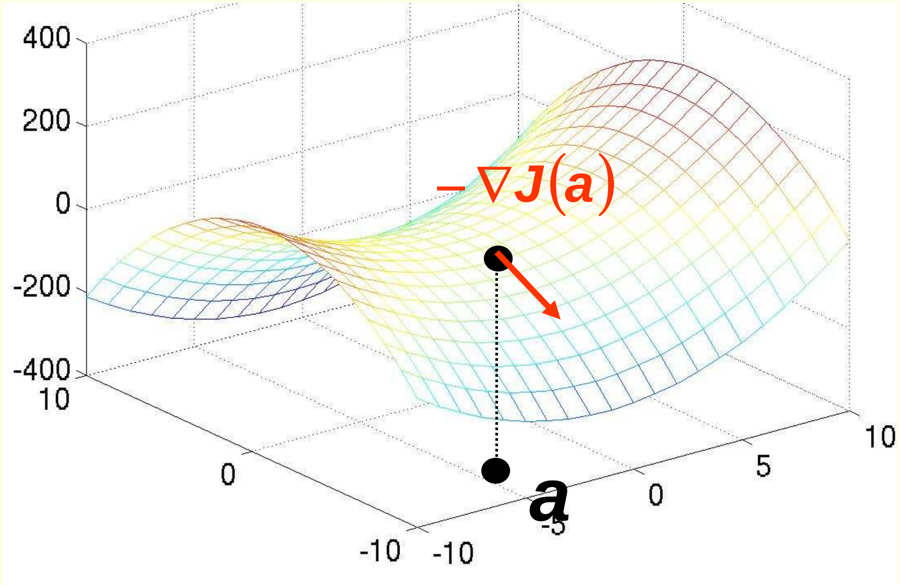
- Gradient $\nabla J(x)$ points in direction of steepest increase of $J(x)$, and $-\nabla J(x)$ in direction of steepest decrease

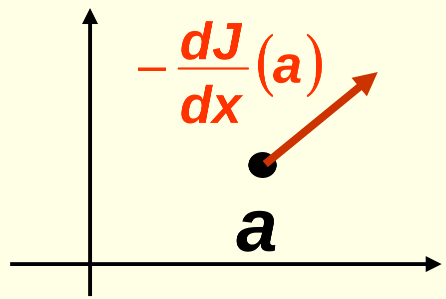
one dimension



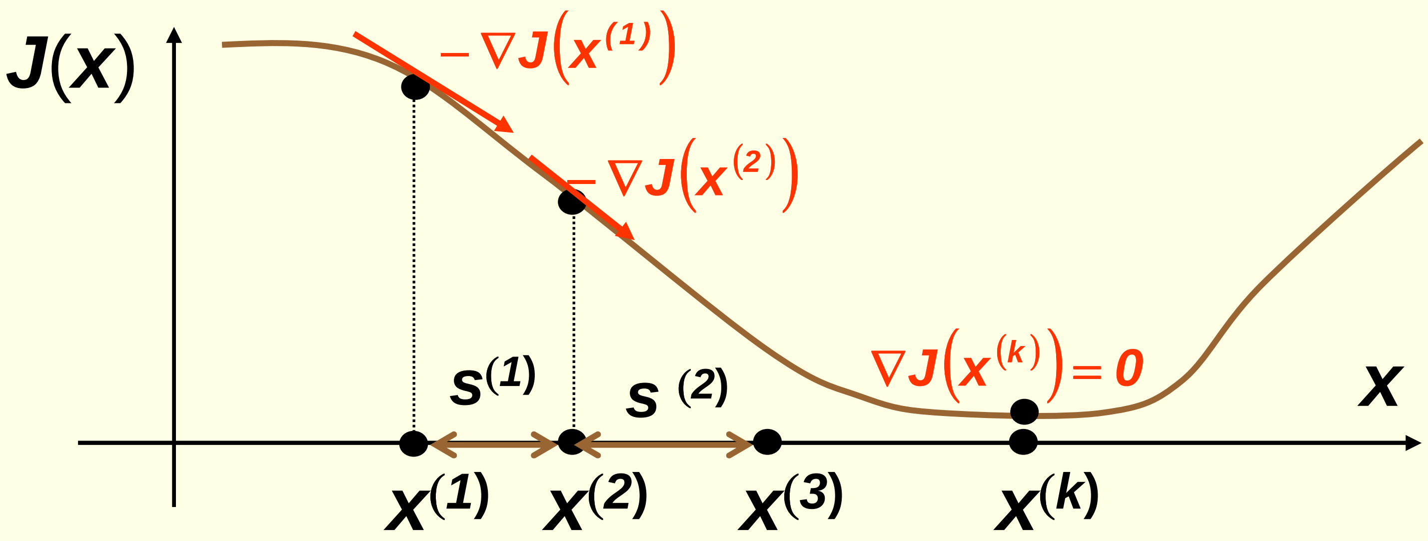


two dimensions





Optimization: Gradient Descent



Gradient Descent for minimizing any function $J(x)$

set $k = 1$ *and* $x^{(1)}$ to some initial guess for the weight vector

while $\eta^{(k)} \left| \nabla J\left(x^{(k)}\right) \right| > \varepsilon$

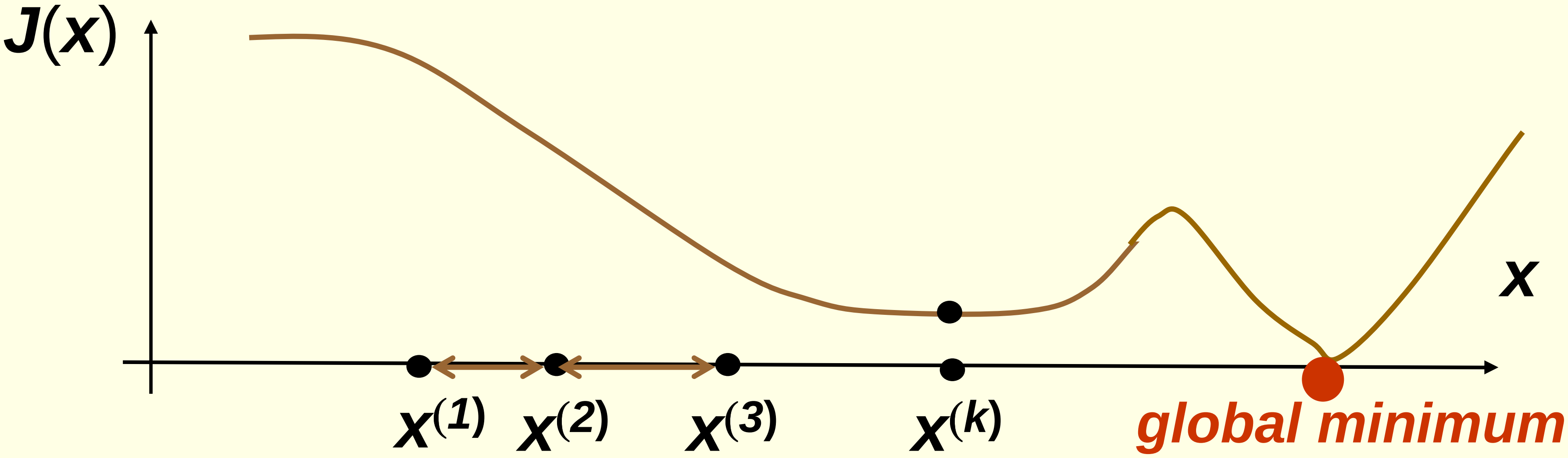
choose ***learning rate*** $\eta^{(k)}$

$x^{(k+1)} = x^{(k)} - \eta^{(k)} \nabla J(x)$ (*update rule*)

$k = k + 1$

Optimization: Gradient Descent

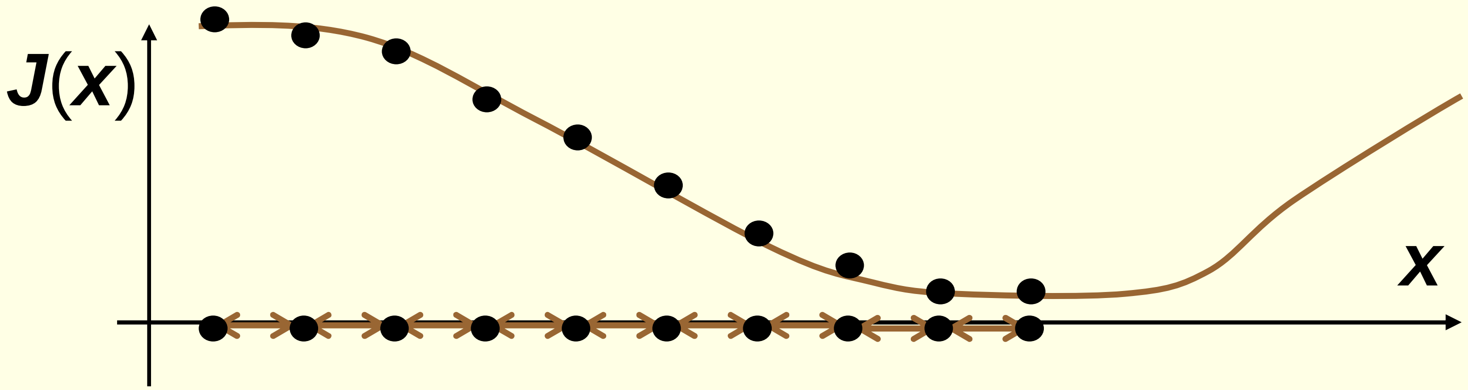
- Gradient descent is guaranteed to find only a local minimum



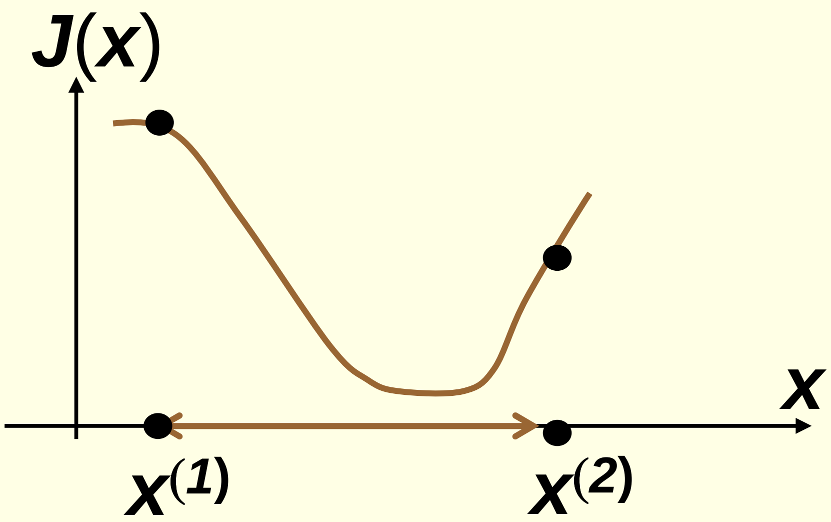
- Nevertheless gradient descent is very popular because it is simple and applicable to any function

Optimization: Gradient Descent

- Main issue: how to set parameter η (***learning rate***)
- If η is too small, need too many iterations



- If η is too large may overshoot the minimum and possibly never find it (if we keep overshooting)



LDF: Criterion Function

- Find weight vector $\boldsymbol{a}$ s.t. for all samples $\boldsymbol{y_1},\ldots,\boldsymbol{y_n}$

$$\boldsymbol{a}^t\boldsymbol{y}_i = \sum_{k=0}^{d} a_k y_i^{(k)} > 0$$

- Need criterion function $\boldsymbol{J}(\boldsymbol{a})$ which is minimized when $\boldsymbol{a}$ is a solution vector

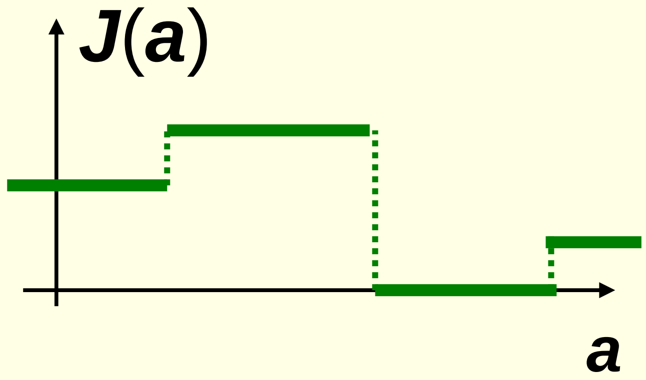
- Let $\boldsymbol{Y_M}$ be the set of examples misclassified by $\boldsymbol{a}$

$$Y_M(\boldsymbol{a}) = \left\{ \textit{sample } \boldsymbol{y}_i \textit{ s.t. } \boldsymbol{a}^t\boldsymbol{y}_i < 0 \right\}$$

- First natural choice: number of misclassified examples

$$J(\boldsymbol{a}) = \left|Y_M(\boldsymbol{a})\right|$$

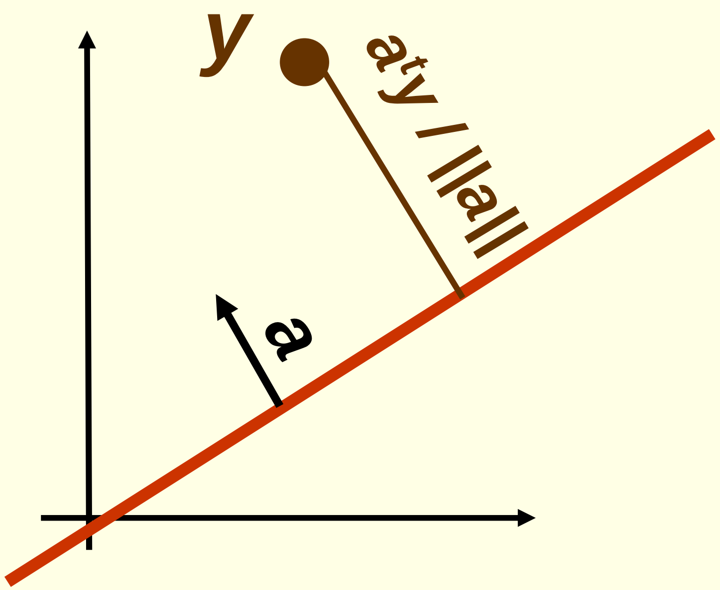
  - piecewise constant, gradient descent is useless



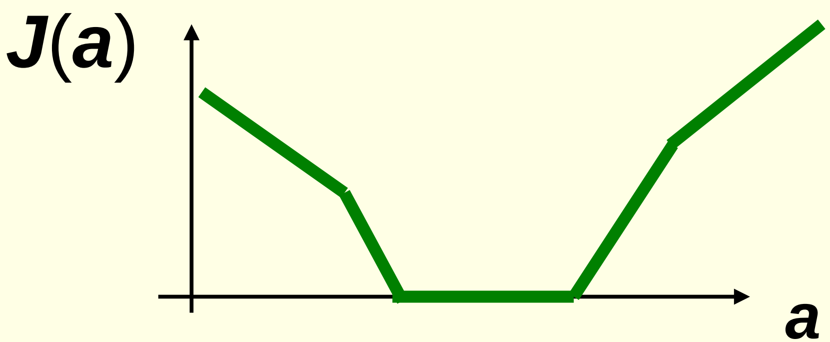
LDF: Perceptron Criterion Function

- Better choice: ***Perceptron*** criterion function

$$J_p(\mathbf{a}) = \sum_{y \in Y_M} \left(-\mathbf{a}^t \mathbf{y}\right)$$



- If ***y*** is misclassified, $\mathbf{a}^t \mathbf{y} \leq 0$
- Thus $J_p(\mathbf{a}) \geq 0$
- J_p(a) is -||a|| times sum of distances of misclassified examples to decision boundary



- $J_p(\mathbf{a})$ is piecewise linear and thus suitable for gradient descent

LDF: Perceptron Batch Rule

$$J_p(\boldsymbol{a}) = \sum_{y \in Y_M} \left(-\boldsymbol{a}^t \boldsymbol{y}\right)$$

- Gradient of $J_p(\boldsymbol{a})$ is $\nabla J_p(\boldsymbol{a}) = \sum_{y \in Y_M} \left(-\boldsymbol{y}\right)$
  - Y_M are samples misclassified by $\boldsymbol{a}^{(k)}$
  - It is not possible to solve $\nabla J_p(\boldsymbol{a}) = \boldsymbol{0}$ analytically because of Y_M
- Update rule for gradient descent: $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \eta^{(k)} \nabla J(\boldsymbol{x})$
- Thus *gradient decent batch update rule* for $J_p(\boldsymbol{a})$ is:

$$\boldsymbol{a}^{(k+1)} = \boldsymbol{a}^{(k)} + \eta^{(k)} \sum_{y \in Y_M} \boldsymbol{y}$$

- It is called batch rule because it is based on all misclassified examples

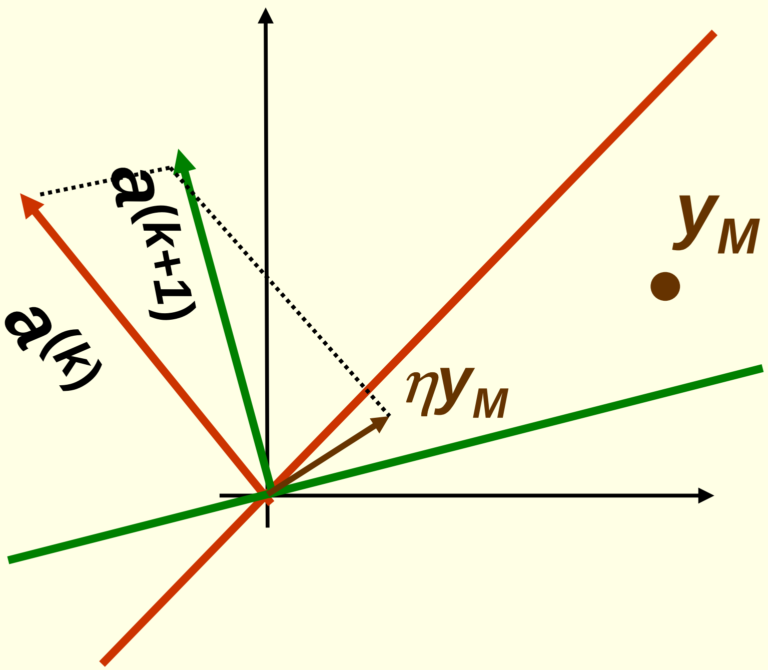
LDF: Perceptron Single Sample Rule

- Thus *gradient decent single sample rule* for $\boldsymbol{J_p}(\boldsymbol{a})$ is:

$$\boldsymbol{a}^{(k+1)} = \boldsymbol{a}^{(k)} + \eta^{(k)} \boldsymbol{y}_M$$

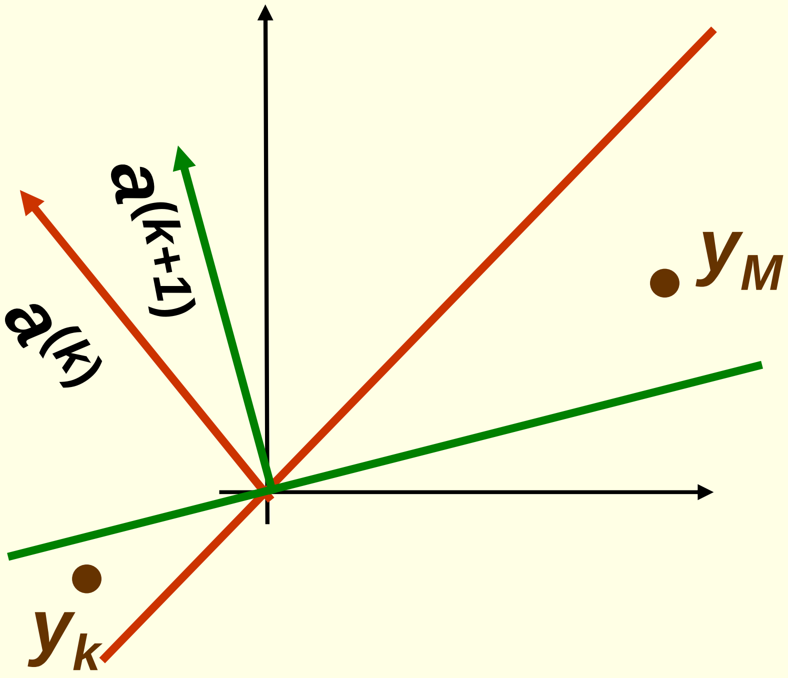
  - note that y_M is one sample misclassified by $\boldsymbol{a}^{(k)}$
  - must have a consistent way of visiting samples

- Geometric Interpretation:
  - y_M misclassified by $\boldsymbol{a}^{(k)}$
    $$\left(\boldsymbol{a}^{(k)}\right)^t \boldsymbol{y}_M \le \mathbf{0}$$
  - y_M is on the wrong side of decision hyperplane
  - adding $\eta \boldsymbol{y}_M$ to $\boldsymbol{a}$ moves new decision hyperplane in the right direction with respect to $\boldsymbol{y}_M$

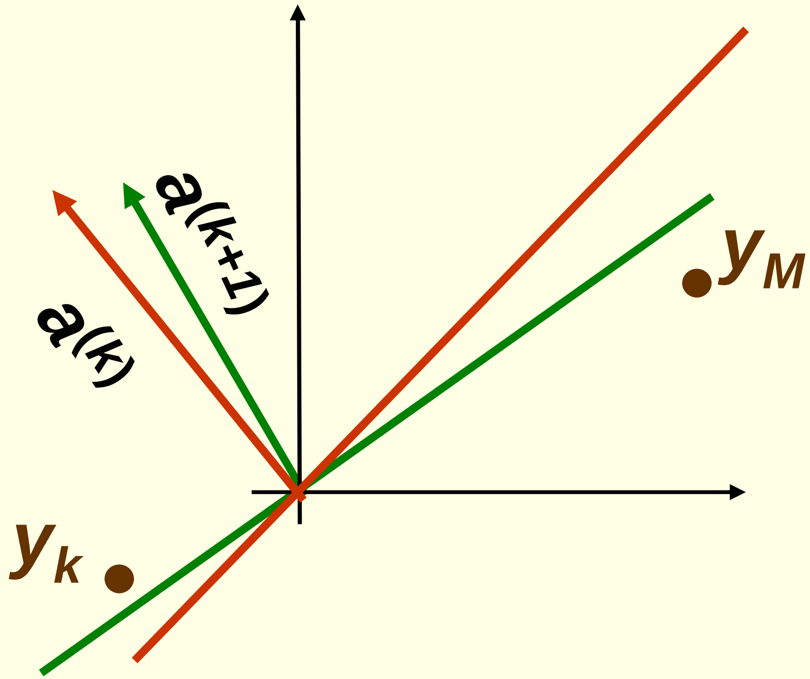


LDF: Perceptron Single Sample Rule

$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \eta^{(k)}\mathbf{y}_M$$



η is too large, previously correctly classified sample $\mathbf{y}_k$ is now misclassified



η is too small, $\mathbf{y}_M$ is still misclassified

LDF: Perceptron Example

	features				grade
name	***good attendance?***	***tall?***	***sleeps in class?***	***chews gum?***	
Jane	*yes (1)*	*yes (1)*	*no (-1)*	*no (-1)*	*A*
Steve	*yes (1)*	*yes (1)*	*yes (1)*	*yes (1)*	*F*
Mary	*no (-1)*	*no (-1)*	*no (-1)*	*yes (1)*	*F*
Peter	*yes (1)*	*no (-1)*	*no (-1)*	*yes (1)*	*A*

- ***class 1***: students who get grade *A*
- ***class 2***: students who get grade *F*

LDF Example: Augment feature vector

	features					grade
name	***extra***	***good attendance?***	***tall?***	***sleeps in class?***	***chews gum?***	
Jane	*1*	*yes (1)*	*yes (1)*	*no (-1)*	*no (-1)*	*A*
Steve	*1*	*yes (1)*	*yes (1)*	*yes (1)*	*yes (1)*	*F*
Mary	*1*	*no (-1)*	*no (-1)*	*no (-1)*	*yes (1)*	*F*
Peter	*1*	*yes (1)*	*no (-1)*	*no (-1)*	*yes (1)*	*A*

- convert samples $\boldsymbol{x_1},\ldots,\boldsymbol{x_n}$ to augmented samples $\boldsymbol{y_1},\ldots,\boldsymbol{y_n}$ by adding a new dimension of value 1

LDF: Perform "Normalization"

	features					grade
name	***extra***	***good attendance?***	***tall?***	***sleeps in class?***	***chews gum?***	
Jane	*1*	*yes (1)*	*yes (1)*	*no (-1)*	*no (-1)*	*A*
Steve	*-1*	*yes (-1)*	*yes (-1)*	*yes (-1)*	*yes (-1)*	*F*
Mary	*-1*	*no (1)*	*no (1)*	*no (1)*	*yes (-1)*	*F*
Peter	*1*	*yes (1)*	*no (-1)*	*no (-1)*	*yes (1)*	*A*

- Replace all examples from class $\boldsymbol{c_2}$ by their negative

$$\boldsymbol{y}_i \rightarrow -\boldsymbol{y}_i \qquad \forall \boldsymbol{y}_i \in \boldsymbol{c}_2$$

- Seek weight vector ***a*** s.t. $\boldsymbol{a}^t \boldsymbol{y}_i > \mathbf{0} \qquad \forall \boldsymbol{y}_i$

LDF: Use Single Sample Rule

	features					grade
name	***extra***	***good attendance?***	***tall?***	***sleeps in class?***	***chews gum?***	
Jane	*1*	*yes (1)*	*yes (1)*	*no (-1)*	*no (-1)*	*A*
Steve	*-1*	*yes (-1)*	*yes (-1)*	*yes (-1)*	*yes (-1)*	*F*
Mary	*-1*	*no (1)*	*no (1)*	*no (1)*	*yes (-1)*	*F*
Peter	*1*	*yes (1)*	*no (-1)*	*no (-1)*	*yes (1)*	*A*

- Sample is misclassified if $\mathbf{a}^t\mathbf{y}_i = \sum_{k=0}^{4} \mathbf{a}_k \mathbf{y}_i^{(k)} < \mathbf{0}$
- gradient descent single sample rule: $\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \eta^{(k)}\mathbf{y}_M$
- Set ***fixed*** learning rate to $\eta^{(k)} = 1$: $\boxed{\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \mathbf{y}_M}$

LDF: Gradient decent Example

- set equal initial weights $\mathbf{a}^{(1)}$=[0.25, 0.25, 0.25, 0.25]
- visit all samples sequentially, modifying the weights for after finding a misclassified example

name	$\mathbf{a}^t\mathbf{y}$	***misclassified?***
Jane	*0.25*1+0.25*1+0.25*1+0.25*(-1)+0.25*(-1) >0*	*no*
Steve	*0.25*(-1)+0.25*(-1)+0.25*(-1)+0.25*(-1)+0.25*(-1)<0*	*yes*

- new weights

$$\mathbf{a}^{(2)} = \mathbf{a}^{(1)} + \mathbf{y}_M = [\mathbf{0.25}\ \ \mathbf{0.25}\ \ \mathbf{0.25}\ \ \mathbf{0.25}\ \ \mathbf{0.25}] + $$
$$+ [-\mathbf{1}\ \ -\mathbf{1}\ \ -\mathbf{1}\ \ -\mathbf{1}\ \ -\mathbf{1}] =$$
$$= [-\mathbf{0.75}\ \ -\mathbf{0.75}\ \ -\mathbf{0.75}\ \ -\mathbf{0.75}\ \ -\mathbf{0.75}]$$

LDF: Gradient decent Example

$$\mathbf{a}^{(2)} = [-0.75 \;\; -0.75 \;\; -0.75 \;\; -0.75 \;\; -0.75]$$

name	$\mathbf{a}^t\mathbf{y}$	***misclassified?***
Mary	*-0.75*(-1)-0.75*1 -0.75 *1 -0.75 *1 -0.75*(-1) <0*	*yes*

- new weights

$$\mathbf{a}^{(3)} = \mathbf{a}^{(2)} + \mathbf{y}_M = [-0.75 \;\; -0.75 \;\; -0.75 \;\; -0.75 \;\; -0.75] +$$

$$+ [-1 \quad 1 \quad 1 \;\; 1 \;\; -1] =$$

$$= [-1.75 \quad 0.25 \quad 0.25 \;\; 0.25 \;\; -1.75]$$

LDF: Gradient decent Example

$$\mathbf{a}^{(3)} = [-1.75 \quad 0.25 \quad 0.25 \quad 0.25 \quad -1.75]$$

name	$\mathbf{a}^t\mathbf{y}$	***misclassified?***
Peter	*-1.75 *1 +0.25* 1+0.25* (-1) +0.25 *(-1)-1.75*1 <0*	*yes*

- new weights

$$\mathbf{a}^{(4)} = \mathbf{a}^{(3)} + \mathbf{y}_M = [-1.75 \quad 0.25 \quad 0.25 \quad 0.25 \quad -1.75] +$$

$$+ [1 \quad 1 \quad -1 \quad -1 \quad 1] =$$

$$= [-0.75 \quad 1.25 \quad -0.75 \quad -0.75 \quad -0.75]$$

LDF: Gradient decent Example

$$\mathbf{a}^{(4)} = \left[-0.75 \;\; 1.25 \;\; -0.75 \;\; -0.75 \;\; -0.75\right]$$

name	$\mathbf{a}^t\mathbf{y}$	***misclassified?***
Jane	*-0.75 *1 +1.25*1 -0.75*1 -0.75 *(-1) -0.75 *(-1)+0*	*no*
Steve	*-0.75*(-1)+1.25*(-1) -0.75*(-1) -0.75*(-1)-0.75*(-1)>0*	*no*
Mary	*-0.75 *(-1)+1.25*1-0.75*1 -0.75 *1 –0.75*(-1) >0*	*no*
Peter	*-0.75 *1+ 1.25*1-0.75* (-1)-0.75* (-1) -0.75 *1 >0*	*no*

- Thus the discriminant function is

$$g(y) = -0.75 * y^{(0)} + 1.25 * y^{(1)} - 0.75 * y^{(2)} - 0.75 * y^{(3)} - 0.75 * y^{(4)}$$

- Converting back to the original features ***x***:

$$g(x) = 1.25 * x^{(1)} - 0.75 * x^{(2)} - 0.75 * x^{(3)} - 0.75 * x^{(4)} - 0.75$$

LDF: Gradient decent Example

- Converting back to the original features ***x***:

$$\mathbf{1.25} * x^{(1)} - \mathbf{0.75} * x^{(2)} - \mathbf{0.75} * x^{(3)} - \mathbf{0.75} * x^{(4)} > \mathbf{0.75} \Rightarrow \textit{grade A}$$

$$\mathbf{1.25} * x^{(1)} - \mathbf{0.75} * x^{(2)} - \mathbf{0.75} * x^{(3)} - \mathbf{0.75} * x^{(4)} < \mathbf{0.75} \Rightarrow \textit{grade F}$$

good attendance **tall** **sleeps in class** **chews gum**

- This is just one possible solution vector

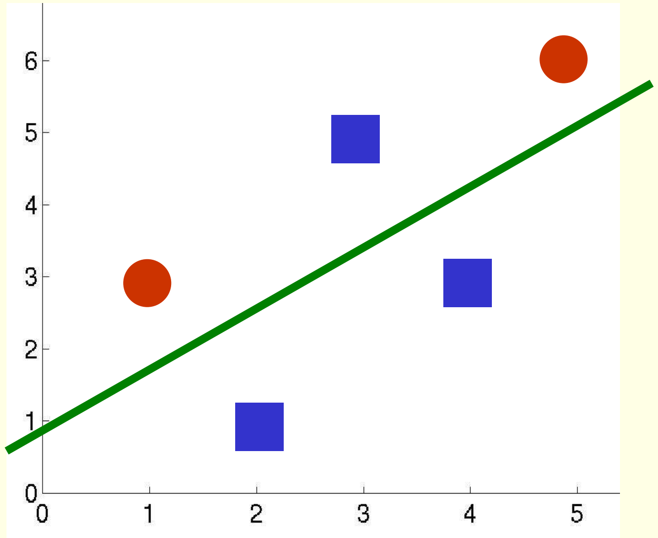
- If we started with weights $\boldsymbol{a}^{(1)}$=[0,0.5, 0.5, 0, 0], solution would be [-1,1.5, -0.5, -1, -1]

$$1.5 * x^{(1)} - 0.5 * x^{(2)} - x^{(3)} - x^{(4)} > 1 \Rightarrow \textit{grade A}$$

$$1.5 * x^{(1)} - 0.5 * x^{(2)} - x^{(3)} - x^{(4)} < 1 \Rightarrow \textit{grade F}$$

  - In this solution, being tall is the least important feature

LDF: Nonseparable Example

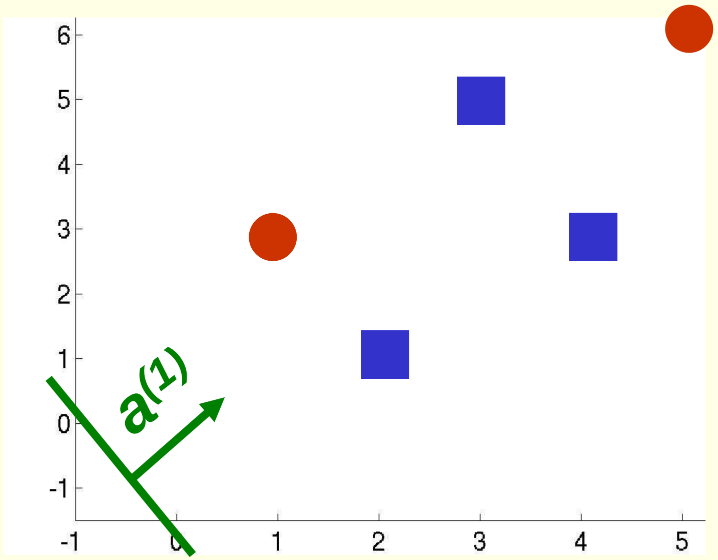


- Suppose we have 2 features and samples are:
  - Class 1: [2,1], [4,3], [3,5]
  - Class 2: [1,3] and [5,6]
- These samples are not separable by a line
- Still would like to get approximate separation by a line, good choice is shown in green
  - some samples may be "noisy", and it's ok if they are on the wrong side of the line
- Get $\boldsymbol{y_1}$, $\boldsymbol{y_2}$, $\boldsymbol{y_3}$, $\boldsymbol{y_4}$ by adding extra feature and "normalizing"

$$y_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad y_3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad y_5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$

LDF: Nonseparable Example

- Let's apply Perceptron single sample algorithm
- initial equal weights $\mathbf{a}^{(1)} = [1\ 1\ 1]$
  - this is line $x^{(1)}+x^{(2)}+1=0$
- fixed learning rate $\eta = 1$



$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \mathbf{y}_M$$

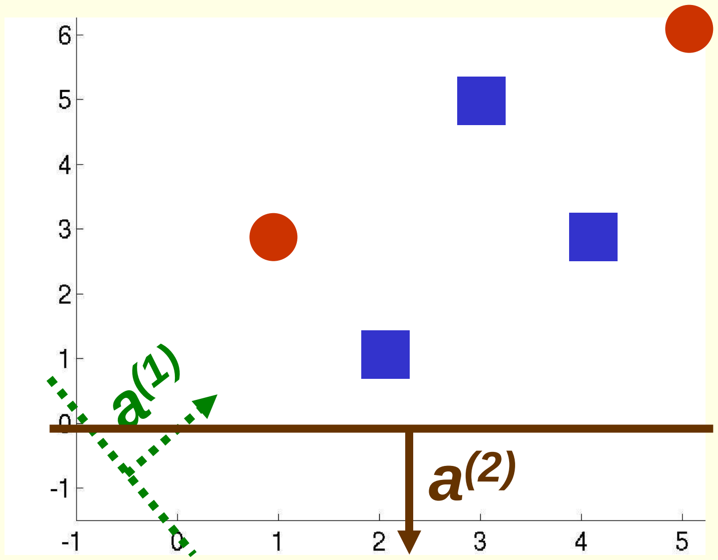
$$\mathbf{y}_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad \mathbf{y}_2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad \mathbf{y}_3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad \mathbf{y}_4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad \mathbf{y}_5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$

- $\mathbf{y}^t_1\mathbf{a}^{(1)} = [1\ 1\ 1]*[1\ 2\ 1]^t > 0$ ✓
- $\mathbf{y}^t_2\mathbf{a}^{(1)} = [1\ 1\ 1]*[1\ 4\ 3]^t > 0$ ✓
- $\mathbf{y}^t_3\mathbf{a}^{(1)} = [1\ 1\ 1]*[1\ 3\ 5]^t > 0$ ✓

LDF: Nonseparable Example

$$a^{(1)} = [1\ 1\ 1] \qquad a^{(k+1)} = a^{(k)} + y_M$$

$$y_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad y_3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad y_5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$



- $y^t_4 a^{(1)}$=[1 1 1]*[-1 -1 -3]^t = -5< 0

$$a^{(2)} = a^{(1)} + y_M = [1\ 1\ 1] + [-1\ -1\ -3] = [0\ \ 0\ -2]$$

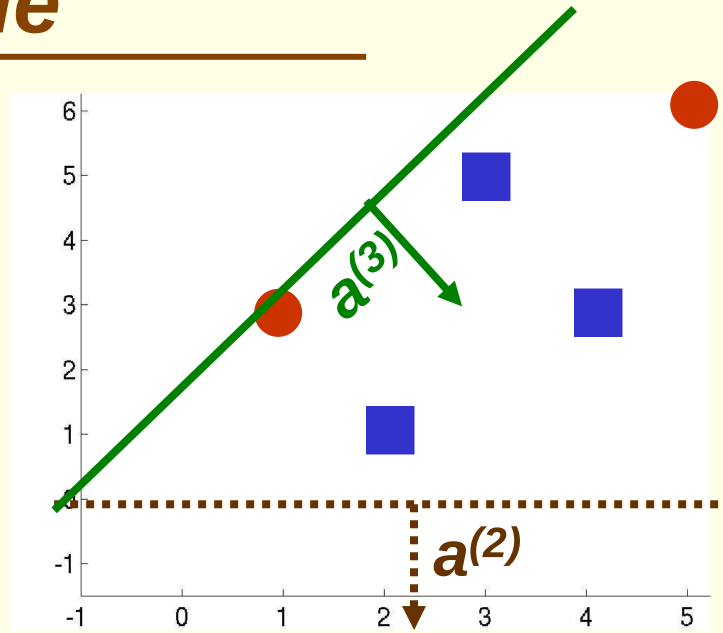
- $y^t_5\, a^{(2)}$=[0 0 -2]*[-1 -5 -6]^t = 12 > 0 ✔
- $y^t_1\, a^{(2)}$=[0 0 -2]*[1 2 1]^t < 0

$$a^{(3)} = a^{(2)} + y_M = [0\ \ 0\ -2] + [1\ 2\ 1] = [1\ 2\ -1]$$

LDF: Nonseparable Example

$$a^{(3)} = [1\ 2\ -1] \qquad a^{(k+1)} = a^{(k)} + y_M$$

$$y_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad y_3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad y_5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$



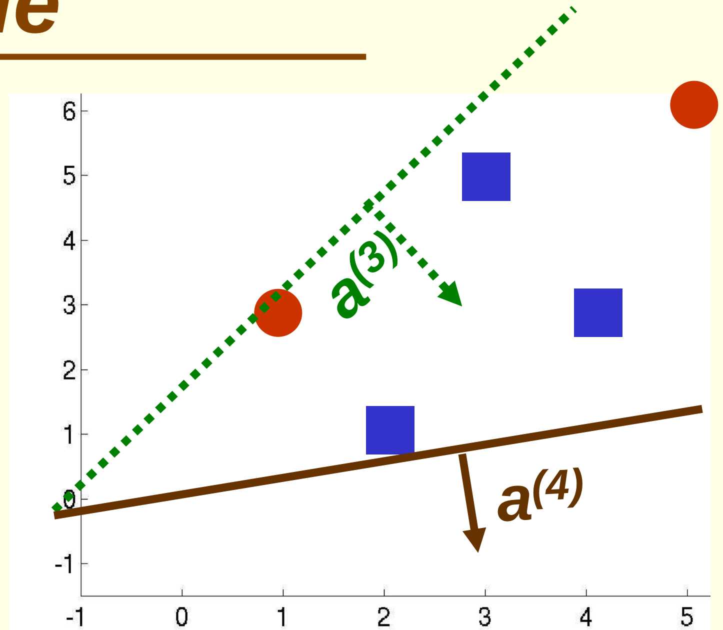
- $y^t_2 a^{(3)}$=[1 4 3]*[1 2 -1]^t =6 > 0 ✔
- $y^t_3 a^{(3)}$=[1 3 5]*[1 2 -1]^t > 0 ✔
- $y^t_4 a^{(3)}$=[-1 -1 -3]*[1 2 -1]^t = 0

$$a^{(4)} = a^{(3)} + y_M = [1\ 2\ -1] + [-1\ -1\ -3] = [0\ 1\ -4]$$

LDF: Nonseparable Example

$$a^{(4)} = [0\ \ 1\ -4] \qquad a^{(k+1)} = a^{(k)} + y_M$$

$$y_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad y_3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad y_5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$



- $y^t_2 a^{(3)}$=[1 4 3]*[1 2 -1]^t =6 > 0 ✓
- $y^t_3 a^{(3)}$=[1 3 5]*[1 2 -1]^t > 0 ✓
- $y^t_4 a^{(3)}$=[-1 -1 -3]*[1 2 -1]^t = 0

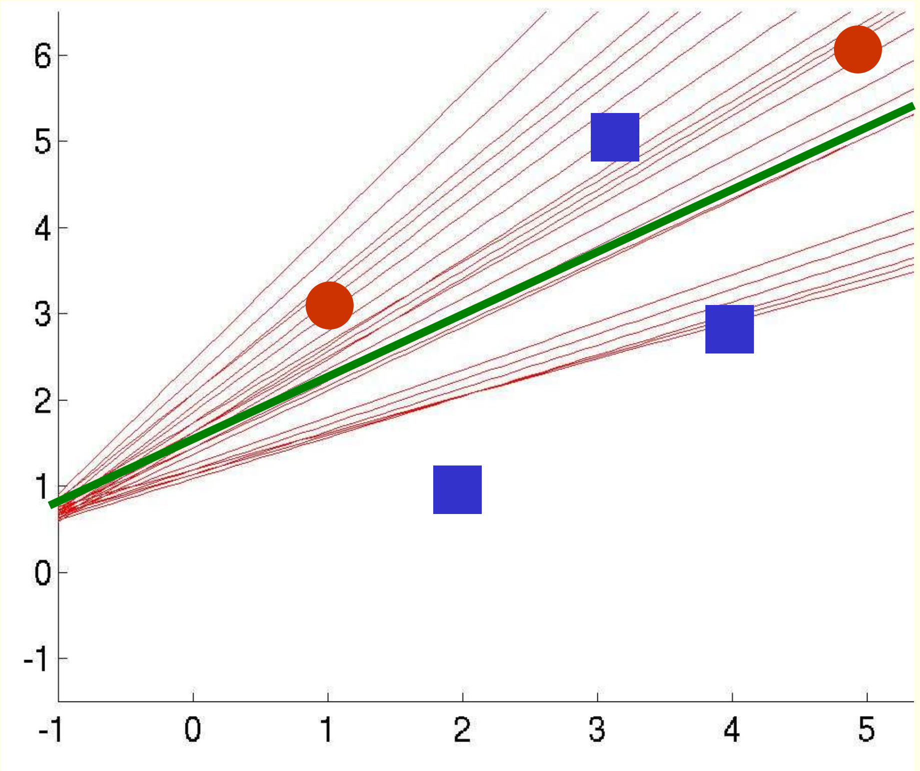
$$a^{(4)} = a^{(3)} + y_M = [1\ \ 2\ -1] + [-1\ -1\ -3] = [0\ \ 1\ -4]$$

LDF: Nonseparable Example

- we can continue this forever
  - there is no solution vector ***a*** satisfying for all ***i***

$$\boldsymbol{a}^t \boldsymbol{y}_i = \sum_{k=0}^{5} a_k y_i^{(k)} > 0$$

- need to stop but at a good point:



- solutions at iterations 900 through 915. Some are good some are not.
- How do we stop at a good solution?

LDF: Convergence of Perceptron rules

- If classes are linearly separable, and use fixed learning rate, that is for some constant ***c***, $\eta^{(k)} = c$
  - *both single sample and batch perceptron rules converge to a correct solution* (could be any ***a*** in the solution space)
- If classes are not linearly separable:
  - algorithm does not stop, it keeps looking for solution which does not exist
  - by choosing appropriate learning rate, can always ensure convergence: $\eta^{(k)} \rightarrow 0 \ \ as \ \ k \rightarrow \infty$
  - for example inverse linear learning rate: $\eta^{(k)} = \frac{\eta^{(1)}}{k}$
  - for inverse linear learning rate convergence in the linearly separable case can also be proven
  - no guarantee that we stopped at a good point, but is popular in practice.

LDF: Perceptron Rule and Gradient decent

- Linearly separable data
  - perceptron rule with gradient decent works well
- Linearly non-separable data
  - need to stop perceptron rule algorithm at a good point, this maybe tricky

Batch Rule	***Single Sample Rule***
▪ Smoother gradient because all samples are used	▪ easier to analyze ▪ Concentrates more than necessary on any isolated “noisy” training examples