
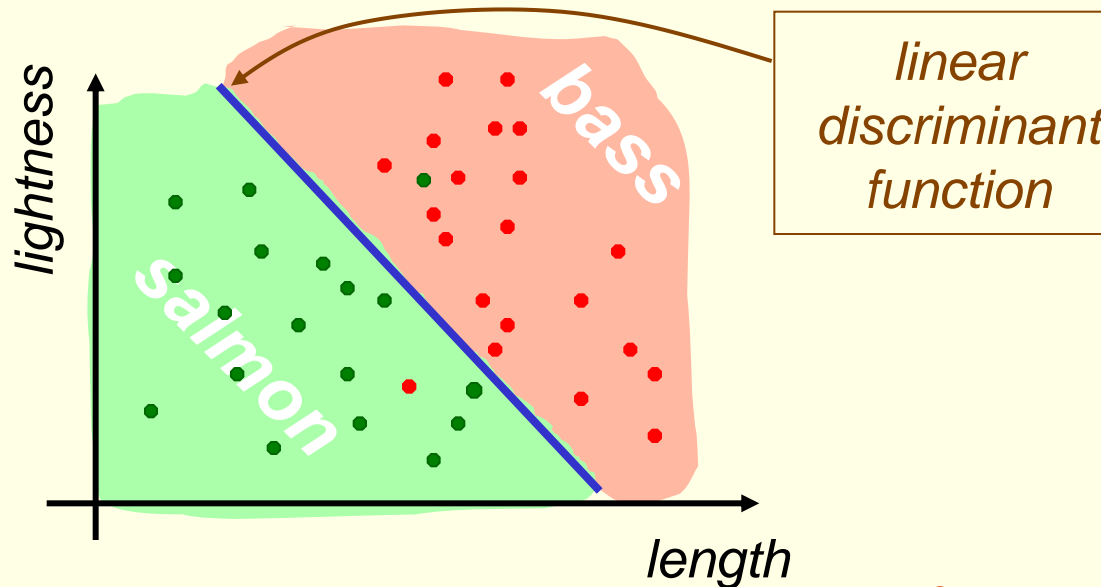


Linear Discriminant Functions

Linear discriminant functions on Road Map

- No probability distribution (no shape or parameters are known)
- Labeled data 
- The shape of discriminant functions is known

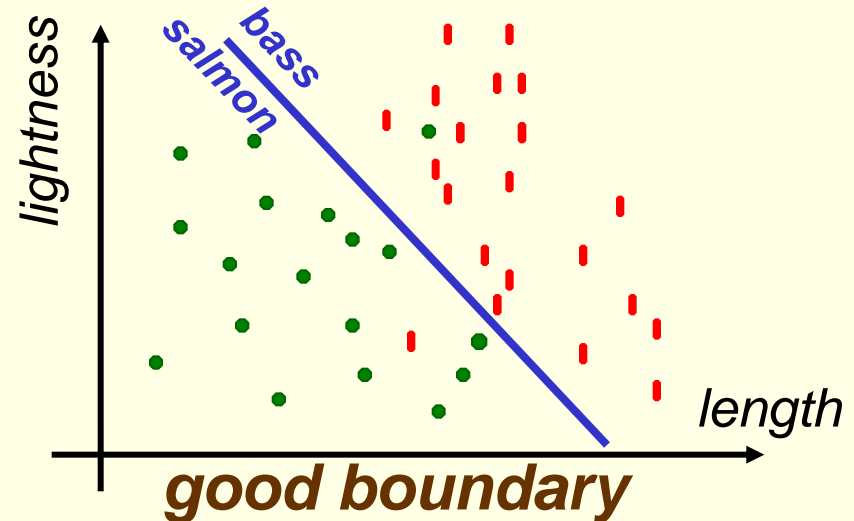
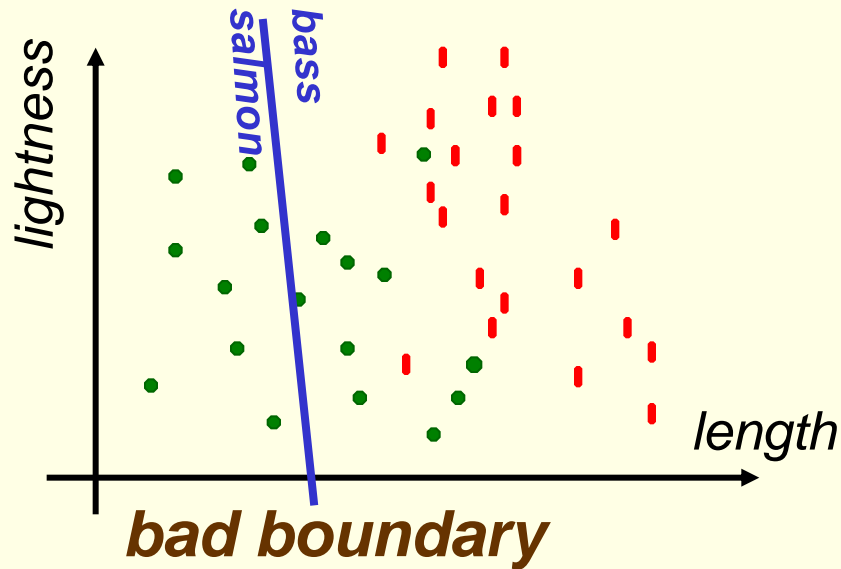


- Need to estimate parameters of the discriminant function (parameters of the line in case of linear discriminant)

a lot is known

little is known

Linear Discriminant Functions: Basic Idea



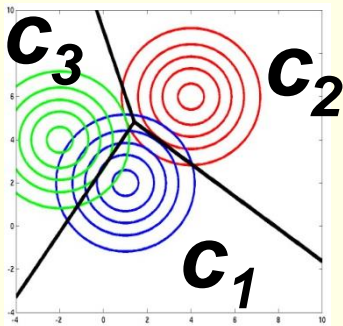
- Have samples from 2 classes $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$
- Assume 2 classes can be separated by a linear boundary $l(\theta)$ with some unknown parameters θ
- Fit the “best” boundary to data by optimizing over parameters θ . **How?**
- Minimize a criterion function.
 - Obvious choice: Minimize classification error on training data. (Does not guarantee small test error)

Parametric Methods vs.

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

Estimate $\theta_1, \theta_2, \dots$ from data

Use a Bayesian classifier to find decision regions

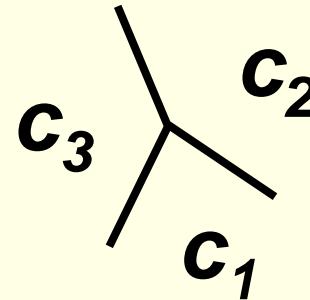


Discriminant Functions

Assume discriminant functions are of known shape $l(\theta_1), l(\theta_2)$, with parameters $\theta_1, \theta_2, \dots$

Estimate $\theta_1, \theta_2, \dots$ from data

Use discriminant functions for classification



Parametric Methods vs. Discriminant Functions

- In theory, Bayesian classifier minimizes the risk, but in practice:
 - do not have confidence in assumed model shapes;
 - 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

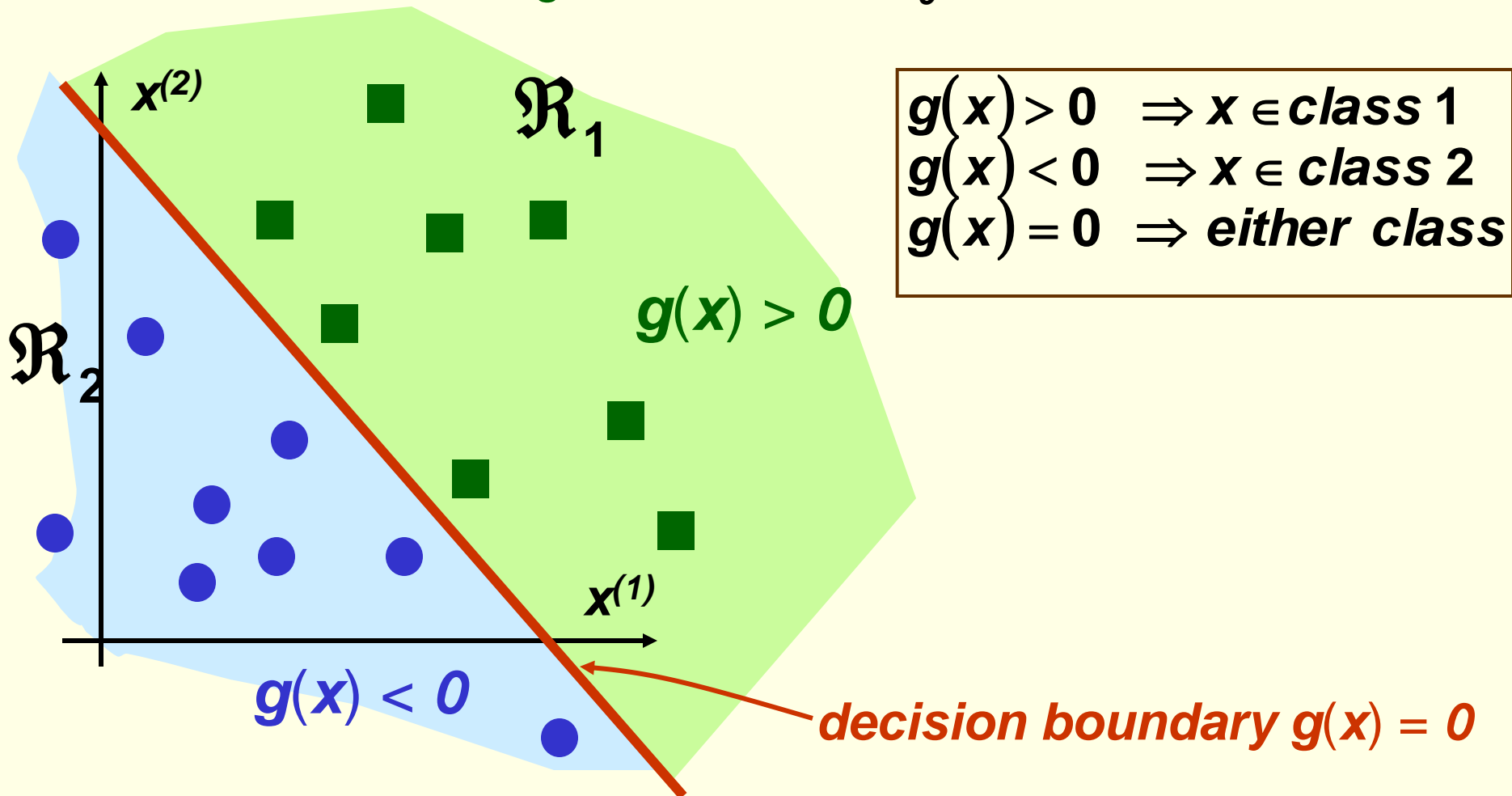
- Discriminant functions can be more general than linear.
- For now, we will study linear discriminant functions
 - Simple model (should try simpler models first)
 - Analytically tractable.
- Linear Discriminant functions are optimal for Gaussian distributions with equal covariance.
- May not be optimal 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**



LDF: 2 Classes

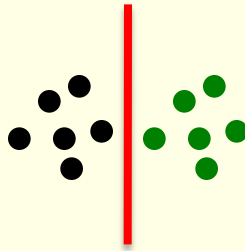
- Decision boundary $g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0 = 0$ is a *hyperplane*

- A hyperplane is

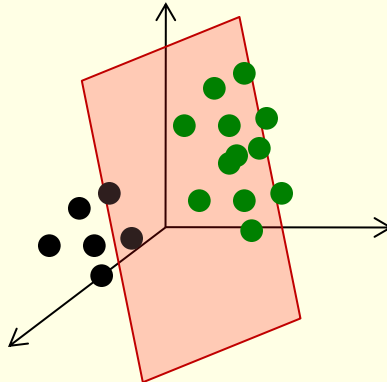
- a point in 1D



- a line in 2D



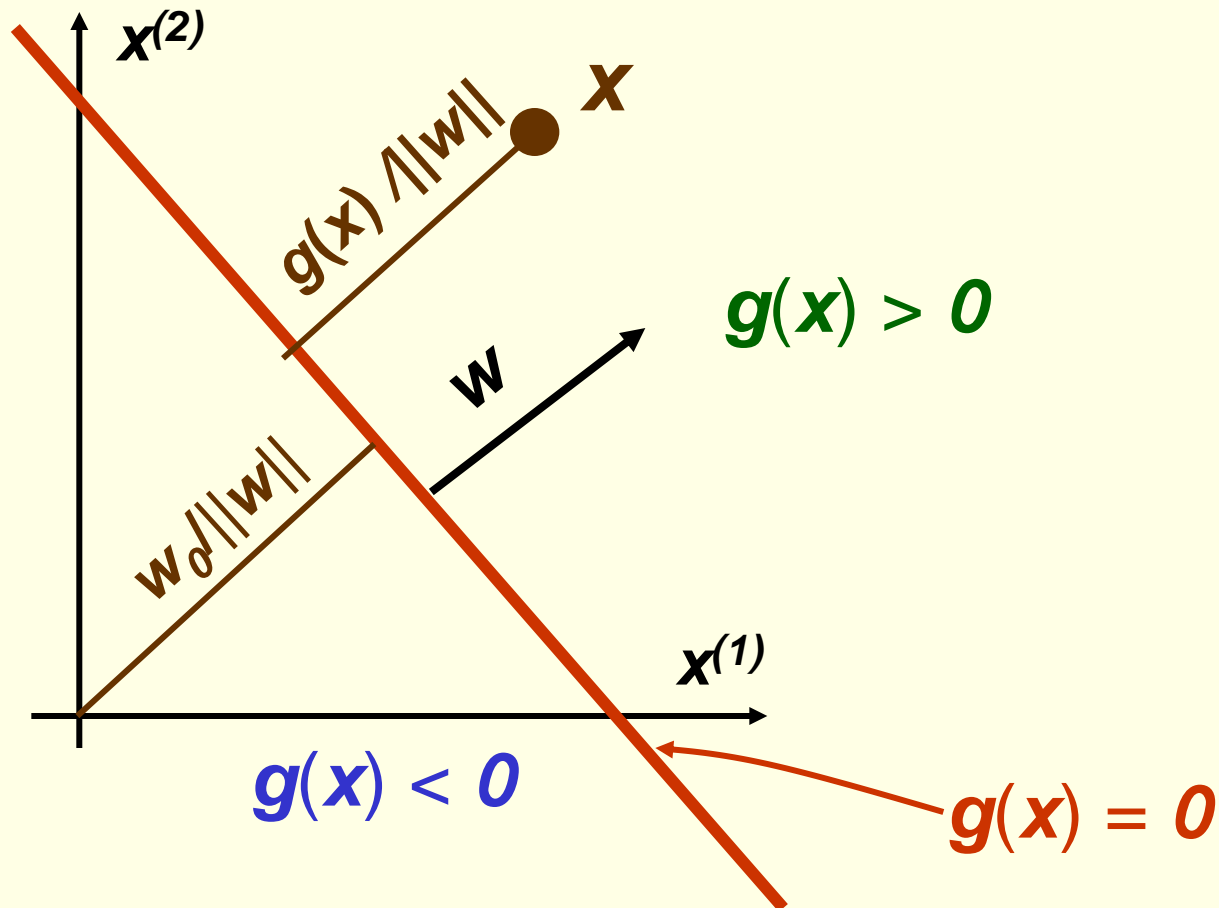
- a plane in 3D



LDF: 2 Classes

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

- \mathbf{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

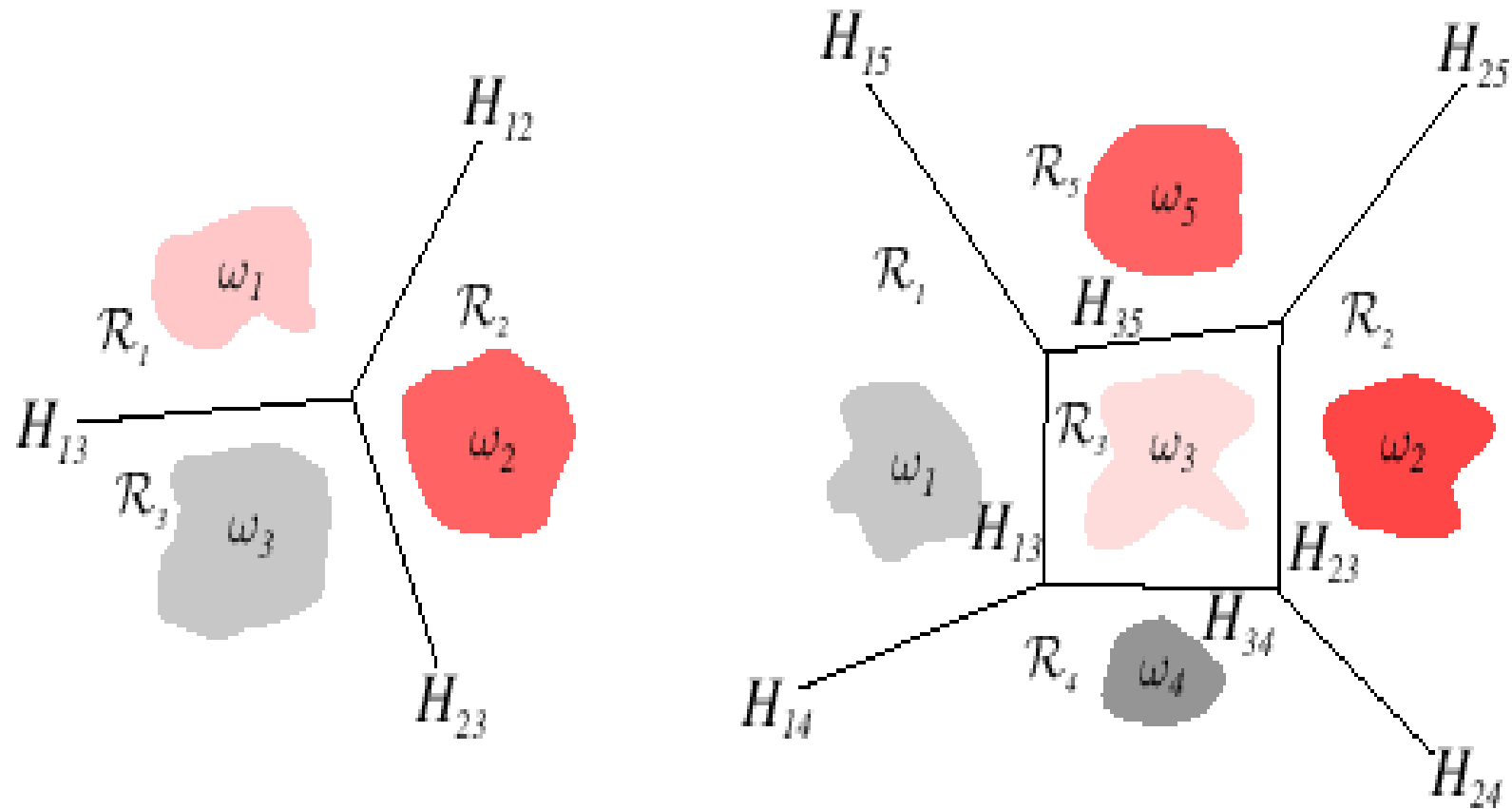
$$\mathbf{g}_i(\mathbf{x}) = \mathbf{w}_i^t \mathbf{x} + w_{i0} \quad \mathbf{i} = 1, \dots, m$$

- Given \mathbf{x} , assign class \mathbf{c}_i if

$$\mathbf{g}_i(\mathbf{x}) \geq \mathbf{g}_j(\mathbf{x}) \quad \forall j \neq i$$

- Such classifier is called a *linear machine*
- A linear machine divides the feature space into \mathbf{c} decision regions, with $\mathbf{g}_i(\mathbf{x})$ being the largest discriminant if \mathbf{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 H_{ij} defined by:

$$\begin{aligned} \mathbf{g}_i(\mathbf{x}) = \mathbf{g}_j(\mathbf{x}) &\Leftrightarrow \mathbf{w}_i^t \mathbf{x} + w_{i0} = \mathbf{w}_j^t \mathbf{x} + w_{j0} \\ &\Leftrightarrow (\mathbf{w}_i - \mathbf{w}_j)^t \mathbf{x} + (w_{i0} - w_{j0}) = 0 \end{aligned}$$

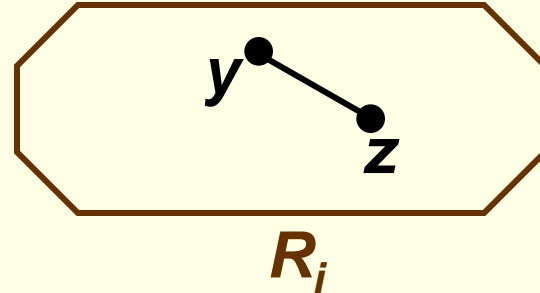
- Thus $\mathbf{w}_i - \mathbf{w}_j$ is normal to H_{ij}
- And distance from \mathbf{x} to H_{ij} is given by

$$d(\mathbf{x}, H_{ij}) = \frac{\mathbf{g}_i(\mathbf{x}) - \mathbf{g}_j(\mathbf{x})}{\|\mathbf{w}_i - \mathbf{w}_j\|}$$

LDF: Many Classes

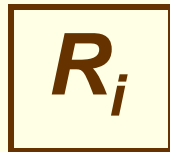
- Decision regions for a linear machine are **convex**

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

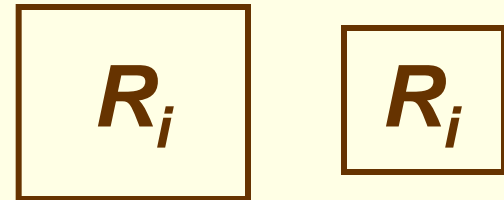


$$\begin{aligned} \forall j \neq i \quad g_i(y) \geq g_j(y) \text{ and } g_i(z) \geq g_j(z) &\Leftrightarrow \\ \Leftrightarrow \forall j \neq i \quad g_i(\alpha y + (1 - \alpha)z) \geq g_j(\alpha y + (1 - \alpha)z) \end{aligned}$$

- In particular, decision regions must be spatially contiguous



R_i is a valid decision region

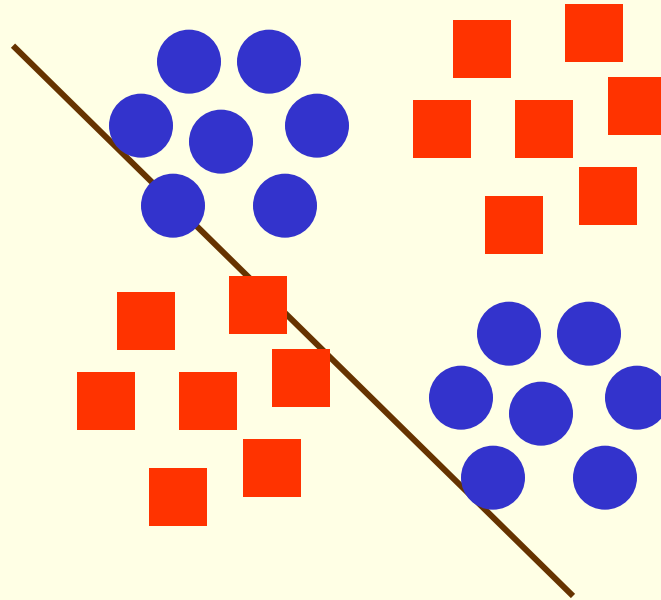


R_j is not a valid decision region

LDF: Many Classes

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

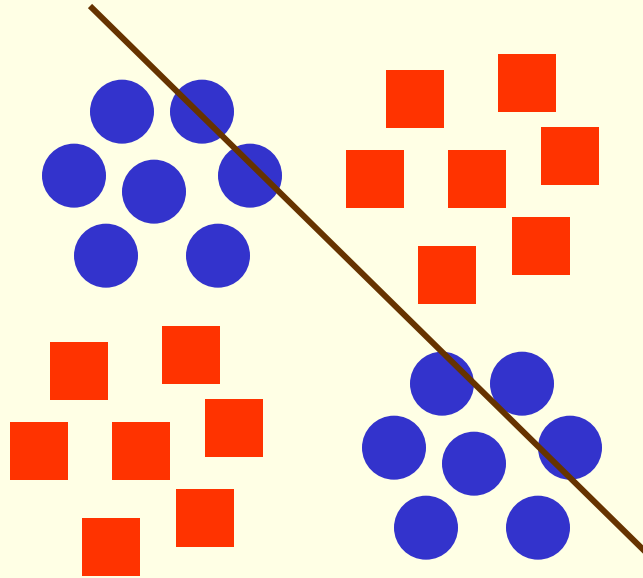
- Example:



LDF: Many Classes

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

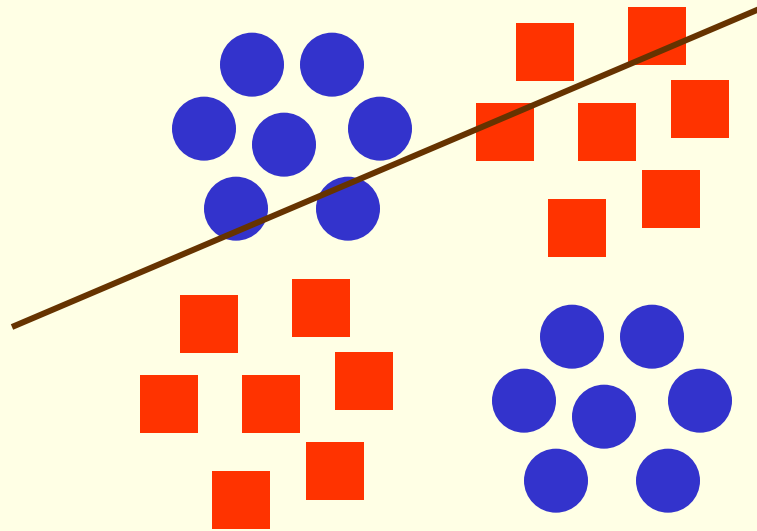
- Example:



LDF: Many Classes

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

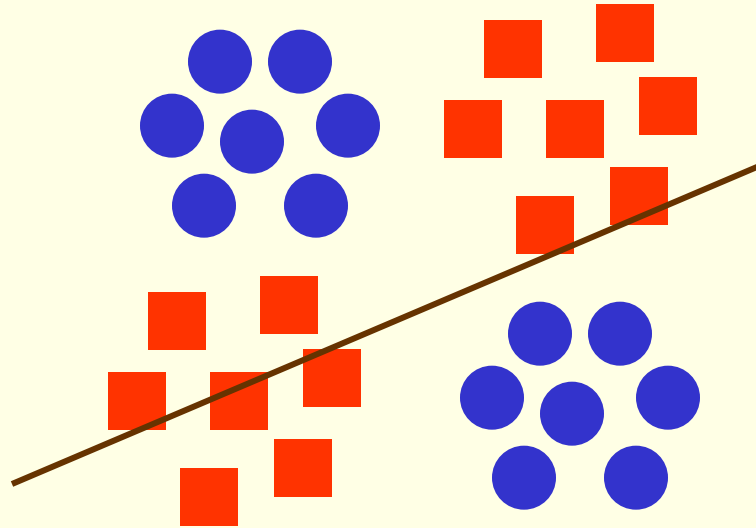
- Example:



LDF: Many Classes

- Thus applicability of linear machine to mostly limited to unimodal conditional densities $p(\mathbf{x}|\theta)$
 - 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: $g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0$
- Can rewrite it: $g(\mathbf{x}) = \underbrace{\begin{bmatrix} w_0 & \mathbf{w}^t \end{bmatrix}}_{\text{new weight vector } \mathbf{a}} \underbrace{\begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}}_{\text{new feature vector } \mathbf{y}} = \mathbf{a}^t \mathbf{y} = g(\mathbf{y})$
- \mathbf{y} is called the *augmented feature vector*
- Added a dummy dimension to get a completely equivalent new *homogeneous* problem

old problem

$$g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0$$
$$\begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$$

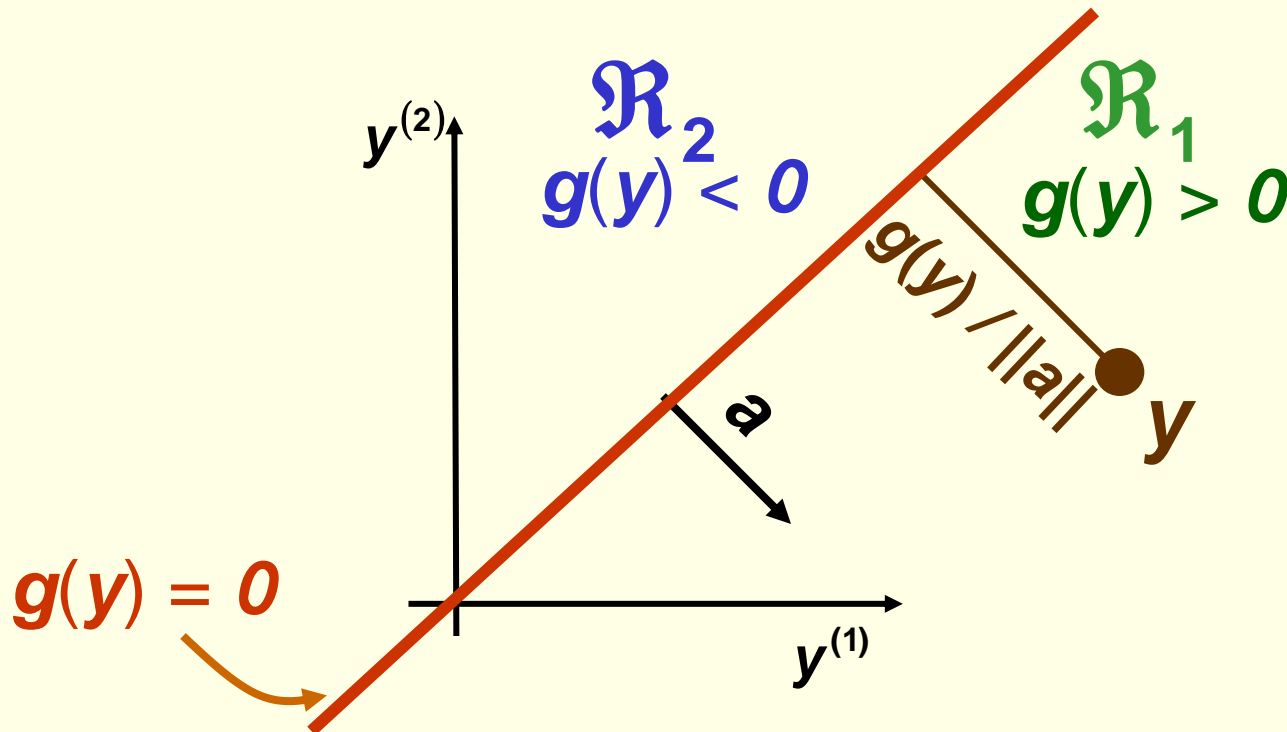
new problem

$$g(\mathbf{y}) = \mathbf{a}^t \mathbf{y}$$
$$\begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}$$

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 $\mathbf{x}_1, \dots, \mathbf{x}_n$ convert them to augmented samples $\mathbf{y}_1, \dots, \mathbf{y}_n$ by adding a new dimension of value 1

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



LDF: Training Error

- For the rest of the lecture, assume we have 2 classes
- Samples $\mathbf{y}_1, \dots, \mathbf{y}_n$ some in class 1, some in class 2
- Use these samples to determine weights \mathbf{a} in the discriminant function $\mathbf{g}(\mathbf{y}) = \mathbf{a}^t \mathbf{y}$
- What should be our criterion for determining \mathbf{a} ?
 - For now, suppose we want to minimize the training error (that is the number of misclassified samples $\mathbf{y}_1, \dots, \mathbf{y}_n$)
- Recall that
$$\mathbf{g}(\mathbf{y}_i) > 0 \Rightarrow \mathbf{y}_i \text{ classified } \mathbf{c}_1$$
$$\mathbf{g}(\mathbf{y}_i) < 0 \Rightarrow \mathbf{y}_i \text{ classified } \mathbf{c}_2$$
- Thus training error is 0 if
$$\begin{cases} \mathbf{g}(\mathbf{y}_i) > 0 & \forall \mathbf{y}_i \in \mathbf{c}_1 \\ \mathbf{g}(\mathbf{y}_i) < 0 & \forall \mathbf{y}_i \in \mathbf{c}_2 \end{cases}$$

LDF: Problem “Normalization”

- Thus training error is 0 if
$$\begin{cases} \mathbf{a}^t \mathbf{y}_i > 0 & \forall \mathbf{y}_i \in \mathbf{c}_1 \\ \mathbf{a}^t \mathbf{y}_i < 0 & \forall \mathbf{y}_i \in \mathbf{c}_2 \end{cases}$$

- Equivalently, training error is 0 if

$$\begin{cases} \mathbf{a}^t \mathbf{y}_i > 0 & \forall \mathbf{y}_i \in \mathbf{c}_1 \\ \mathbf{a}^t (-\mathbf{y}_i) > 0 & \forall \mathbf{y}_i \in \mathbf{c}_2 \end{cases}$$

- This suggests problem “normalization”:

1. Replace all examples from class \mathbf{c}_2 by their negative

$$\mathbf{y}_i \rightarrow -\mathbf{y}_i \quad \forall \mathbf{y}_i \in \mathbf{c}_2$$

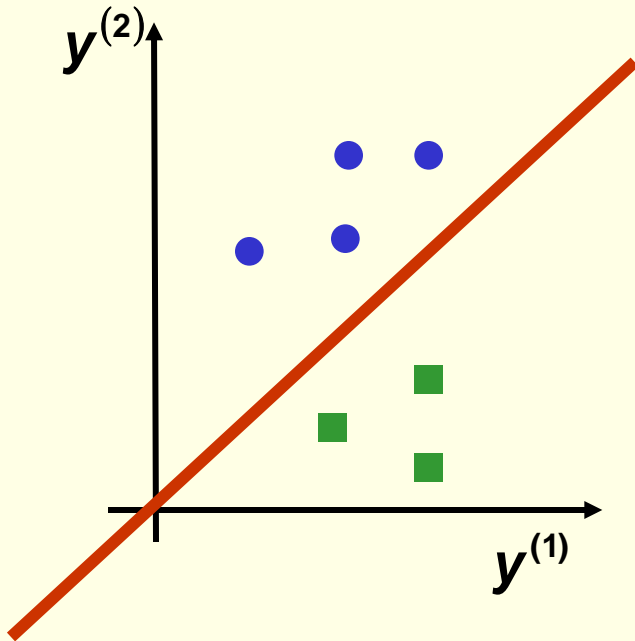
2. Seek weight vector \mathbf{a} s.t.

$$\mathbf{a}^t \mathbf{y}_i > 0 \quad \forall \mathbf{y}_i$$

- If such \mathbf{a} exists, it is called a *separating* or *solution* vector
- Original samples $\mathbf{x}_1, \dots, \mathbf{x}_n$ can indeed be separated by a line then

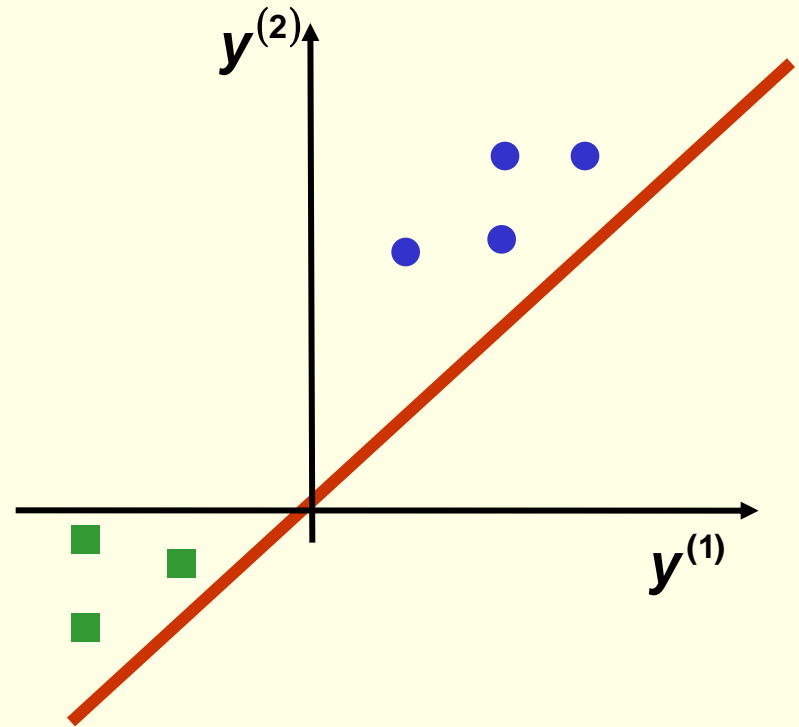
LDF: Problem “Normalization”

before normalization

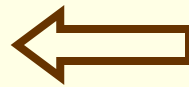


Seek a hyperplane that separates patterns from different categories

after “normalization”



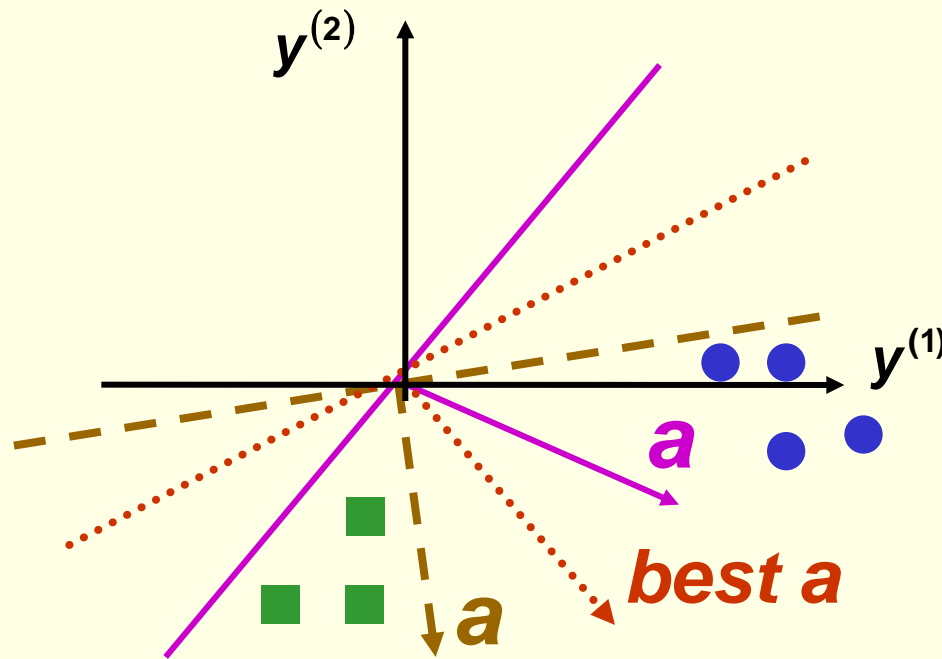
Seek hyperplane that puts *normalized* patterns on the same (positive) side



LDF: Solution Region

- Find weight vector \mathbf{a} s.t. for all samples $\mathbf{y}_1, \dots, \mathbf{y}_n$

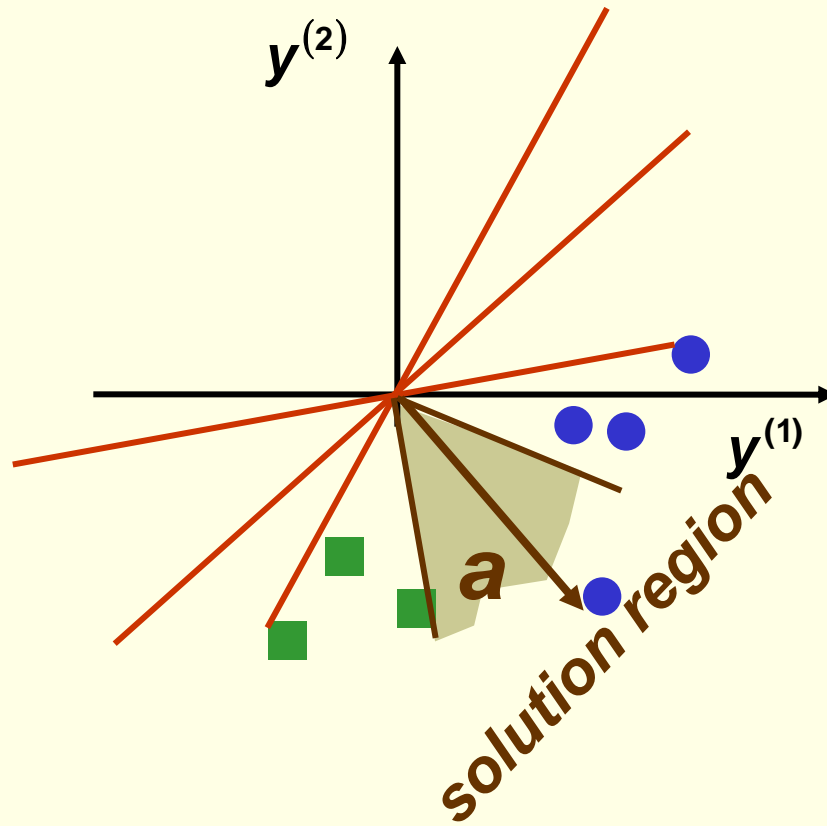
$$\mathbf{a}^t \mathbf{y}_i = \sum_{k=0}^d \mathbf{a}_k \mathbf{y}_i^{(k)} > 0$$



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

LDF: Solution Region

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



Optimization

- Need to minimize a function of many variables

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

- We know how to minimize $\mathbf{J}(\mathbf{x})$
 - Take partial derivatives and set them to zero

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

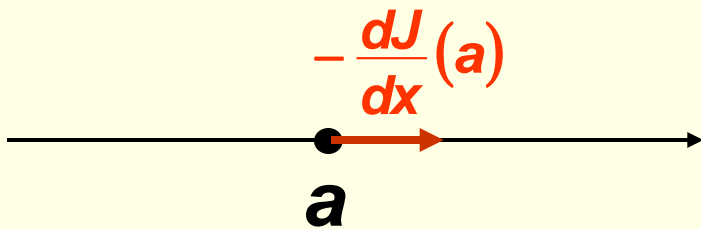
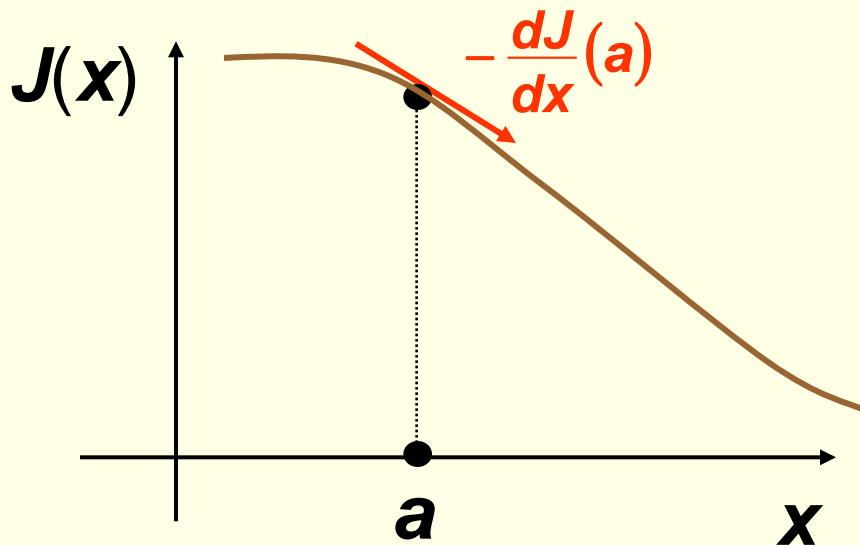
gradient

- However solving analytically is not always easy
- Sometimes it is not even possible to write down an analytical expression for the derivative, we will see an example later today.

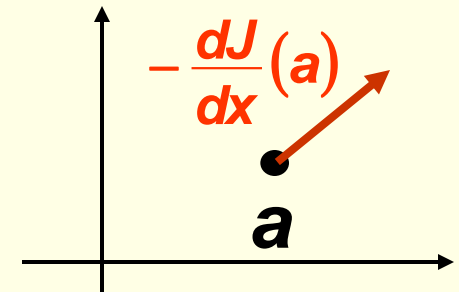
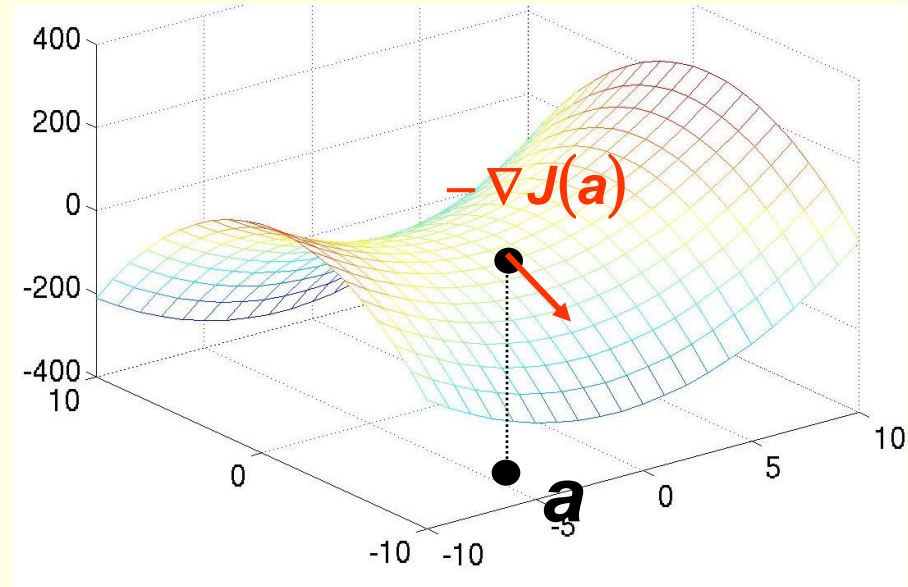
Optimization: Gradient Descent

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

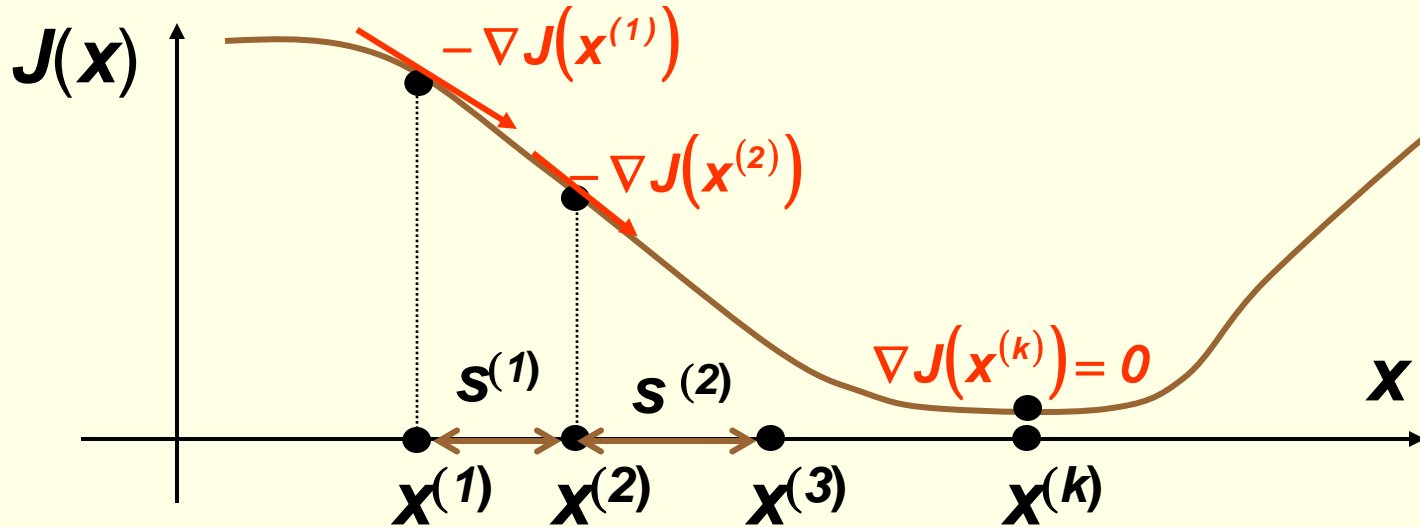
one dimension



two dimensions



Optimization: Gradient Descent



Gradient Descent for minimizing any function $J(\mathbf{x})$

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

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

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

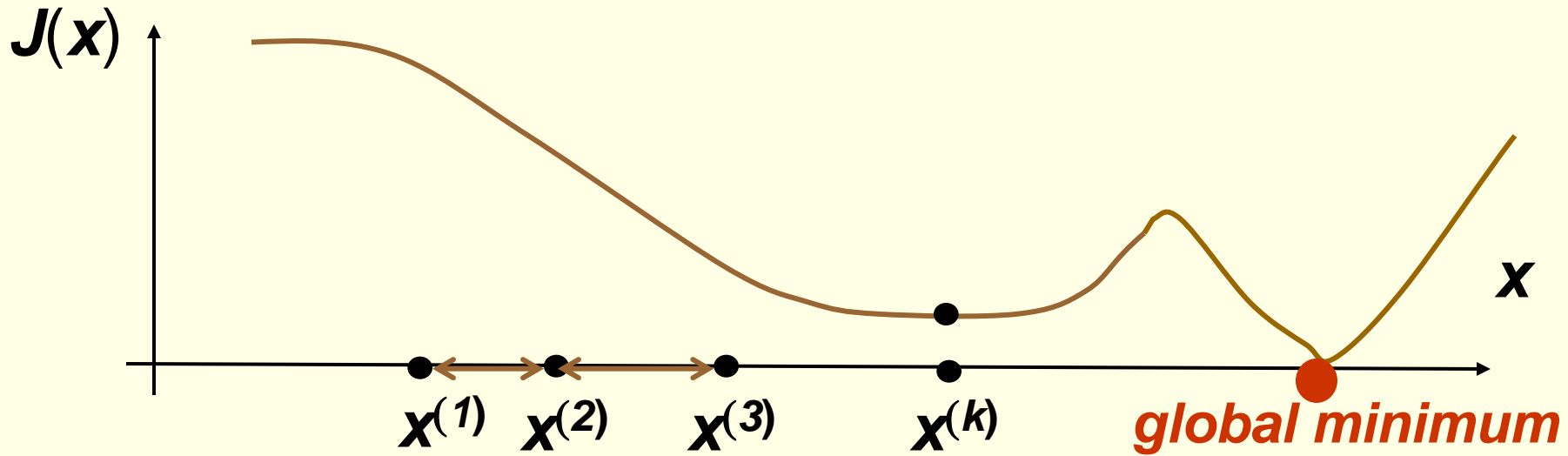
$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \eta^{(k)} \nabla J(\mathbf{x})$$

(update rule)

$$k = k + 1$$

Optimization: Gradient Descent

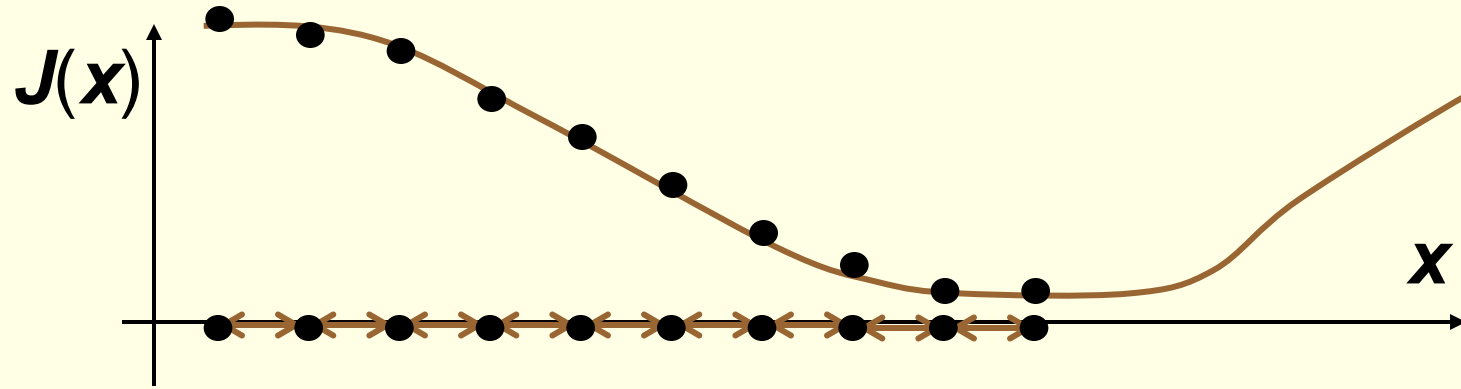
- Gradient descent is guaranteed to find only a local minimum



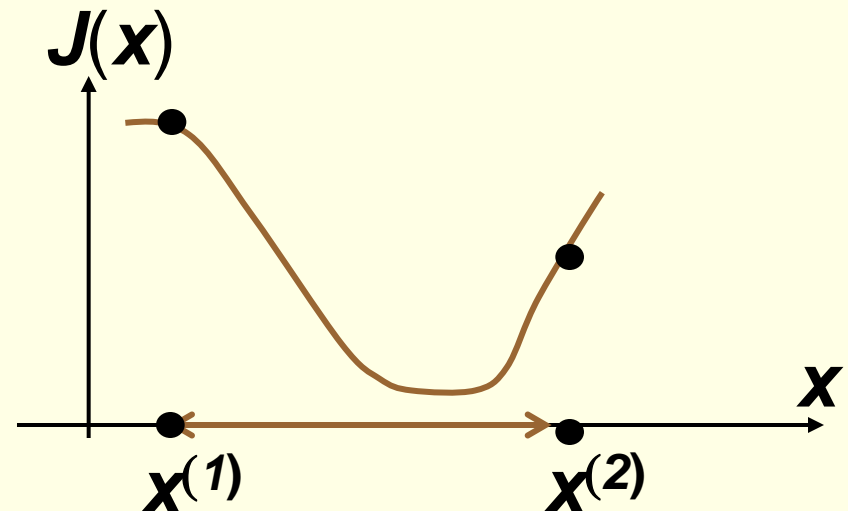
- Nevertheless gradient descent is very popular because it is simple and applicable to any function

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 \mathbf{a} s.t. for all samples $\mathbf{y}_1, \dots, \mathbf{y}_n$

$$\mathbf{a}^t \mathbf{y}_i = \sum_{k=0}^d \mathbf{a}_k \mathbf{y}_i^{(k)} > 0$$

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

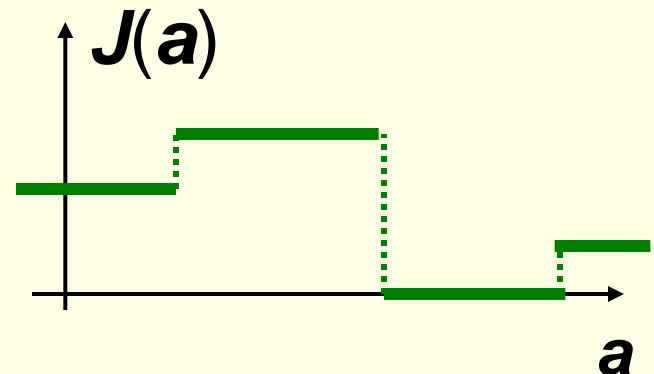
- Let Y_M be the set of examples misclassified by \mathbf{a}

$$Y_M(\mathbf{a}) = \{ \text{sample } \mathbf{y}_i \text{ s.t. } \mathbf{a}^t \mathbf{y}_i < 0 \}$$

- First natural choice: number of misclassified examples

$$\mathbf{J}(\mathbf{a}) = |Y_M(\mathbf{a})|$$

- piecewise constant, gradient descent is useless



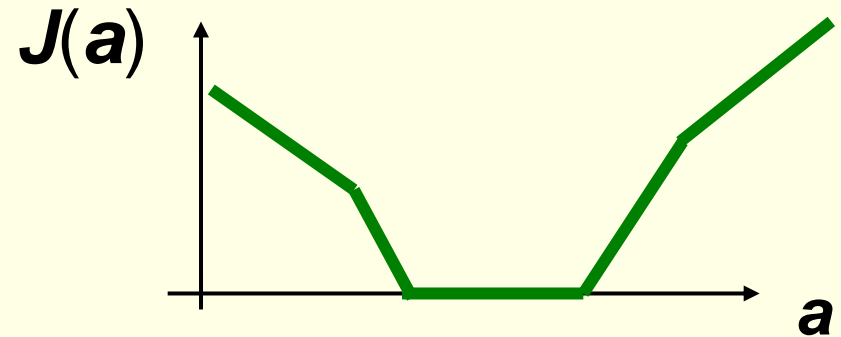
LDF: Perceptron Criterion Function

- Better choice: **Perceptron** criterion function

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

- If \mathbf{y} is misclassified, $\mathbf{a}^t \mathbf{y} \leq 0$
- Thus $J_p(\mathbf{a}) \geq 0$

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



LDF: Perceptron Batch Rule

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

- Gradient of $J_p(\mathbf{a})$ is $\nabla J_p(\mathbf{a}) = \sum_{y \in Y_M} (-\mathbf{y})$
 - Y_M are samples misclassified by $\mathbf{a}^{(k)}$
 - It is not possible to solve $\nabla J_p(\mathbf{a}) = \mathbf{0}$ analytically because of Y_M
- **Gradient decent batch update rule** for $J_p(\mathbf{a})$ is:

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

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

LDF: Perceptron Single Sample Rule

- **Gradient decent single sample rule** for $J_p(\mathbf{a})$ is:

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

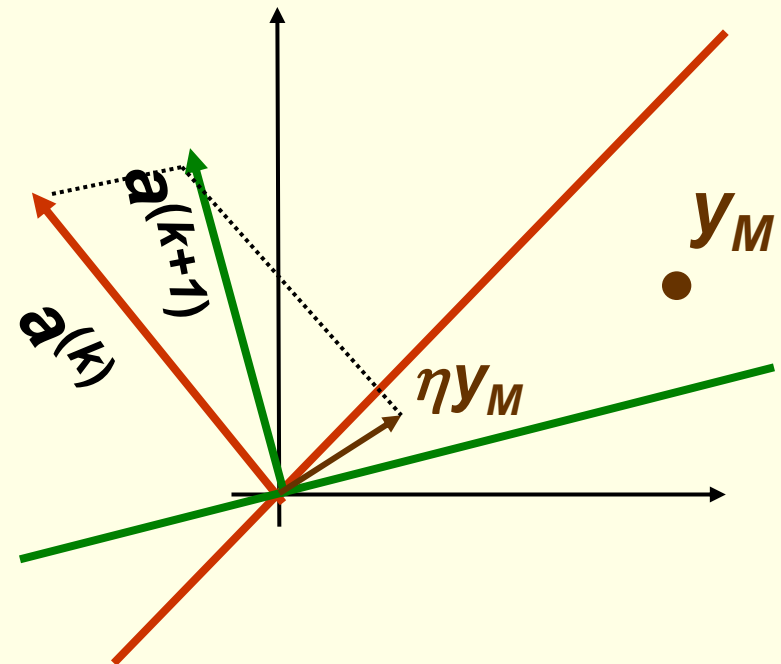
- note that \mathbf{y}_M is one sample misclassified by $\mathbf{a}^{(k)}$
- must have a consistent way of visiting samples

- Geometric Interpretation:

- \mathbf{y}_M misclassified by $\mathbf{a}^{(k)}$

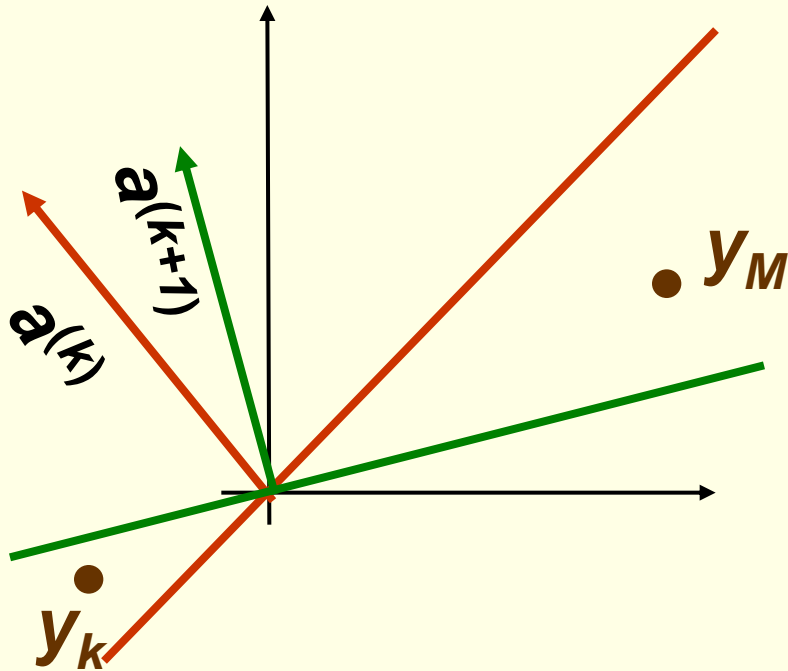
$$(\mathbf{a}^{(k)})^t \mathbf{y}_M \leq 0$$

- \mathbf{y}_M is on the wrong side of decision hyperplane
- adding $\eta \mathbf{y}_M$ to \mathbf{a} moves new decision hyperplane in the right direction with respect to \mathbf{y}_M

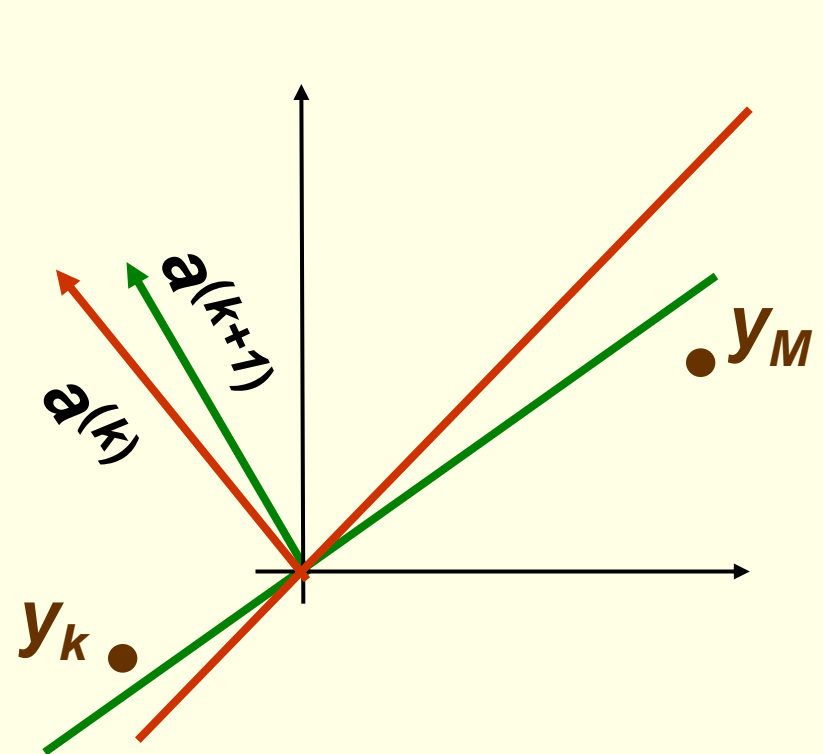


LDF: Perceptron Single Sample Rule

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



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
<i>name</i>	<i>good attendance?</i>	<i>tall?</i>	<i>sleeps in class?</i>	<i>chews gum?</i>	
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
<i>name</i>	<i>extra</i>	<i>good attendance?</i>	<i>tall?</i>	<i>sleeps in class?</i>	<i>chews gum?</i>	
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 $\mathbf{x}_1, \dots, \mathbf{x}_n$ to augmented samples $\mathbf{y}_1, \dots, \mathbf{y}_n$ by adding a new dimension of value 1

LDF: Perform “Normalization”

	features					grade
<i>name</i>	<i>extra</i>	<i>good attendance?</i>	<i>tall?</i>	<i>sleeps in class?</i>	<i>chews gum?</i>	
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 \mathbf{c}_2 by their negative

$$\mathbf{y}_i \rightarrow -\mathbf{y}_i \quad \forall \mathbf{y}_i \in \mathbf{c}_2$$

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

LDF: Use Single Sample Rule

	features					grade
<i>name</i>	<i>extra</i>	<i>good attendance?</i>	<i>tall?</i>	<i>sleeps in class?</i>	<i>chews gum?</i>	
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 $\mathbf{a}^t \mathbf{y}_i = \sum_{k=0}^4 \mathbf{a}_k \mathbf{y}_i^{(k)} < 0$
- gradient descent single sample rule: $\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \eta^{(k)} \mathbf{y}_M$
- Set **fixed** learning rate to $\eta^{(k)} = 1$: $\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \mathbf{y}_M$

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

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

- new weights

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

LDF: Gradient decent Example

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

<i>name</i>	<i>a^ty</i>	<i>misclassified?</i>
Mary	$-0.75*(-1) - 0.75*1 - 0.75*1 - 0.75*1 - 0.75*(-1) < 0$	yes

- new weights

$$\begin{aligned}\mathbf{a}^{(3)} &= \mathbf{a}^{(2)} + \mathbf{y}_M = [-0.75 \ -0.75 \ -0.75 \ -0.75 \ -0.75] + \\ &\quad + [-1 \ 1 \ 1 \ 1 \ -1] = \\ &= [-1.75 \ 0.25 \ 0.25 \ 0.25 \ -1.75]\end{aligned}$$

LDF: Gradient decent Example

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

<i>name</i>	<i>a^ty</i>	<i>misclassified?</i>
Peter	$-1.75 * 1 + 0.25 * 1 + 0.25 * (-1) + 0.25 * (-1) - 1.75 * 1 < 0$	yes

- new weights

$$\begin{aligned} \mathbf{a}^{(4)} &= \mathbf{a}^{(3)} + \mathbf{y}_M = [-1.75 \quad 0.25 \quad 0.25 \quad 0.25 \quad -1.75] + \\ &\quad + [1 \quad 1 \quad -1 \quad -1 \quad 1] = \\ &= [-0.75 \quad 1.25 \quad -0.75 \quad -0.75 \quad -0.75] \end{aligned}$$

LDF: Gradient decent Example

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

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

- Thus the discriminant function is

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

- Converting back to the original features \mathbf{x} :

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

LDF: Gradient decent Example

- Converting back to the original features \mathbf{x} :
$$1.25 * \mathbf{x}^{(1)} - 0.75 * \mathbf{x}^{(2)} - 0.75 * \mathbf{x}^{(3)} - 0.75 * \mathbf{x}^{(4)} > 0.75 \Rightarrow \text{grade A}$$
$$1.25 * \mathbf{x}^{(1)} - 0.75 * \mathbf{x}^{(2)} - 0.75 * \mathbf{x}^{(3)} - 0.75 * \mathbf{x}^{(4)} < 0.75 \Rightarrow \text{grade F}$$

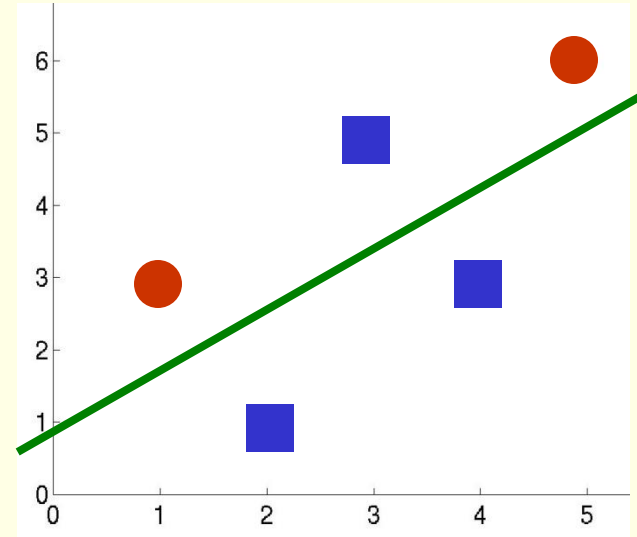
good attendance tall sleeps in class chews gum
- This is just one possible solution vector
- If we started with weights $\mathbf{a}^{(1)} = [0, 0.5, 0.5, 0, 0]$, solution would be $[-1, 1.5, -0.5, -1, -1]$
$$1.5 * \mathbf{x}^{(1)} - 0.5 * \mathbf{x}^{(2)} - \mathbf{x}^{(3)} - \mathbf{x}^{(4)} > 1 \Rightarrow \text{grade A}$$
$$1.5 * \mathbf{x}^{(1)} - 0.5 * \mathbf{x}^{(2)} - \mathbf{x}^{(3)} - \mathbf{x}^{(4)} < 1 \Rightarrow \text{grade F}$$
 - 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 $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4$ by adding extra feature and “normalizing”

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

LDF: Nonseparable Example

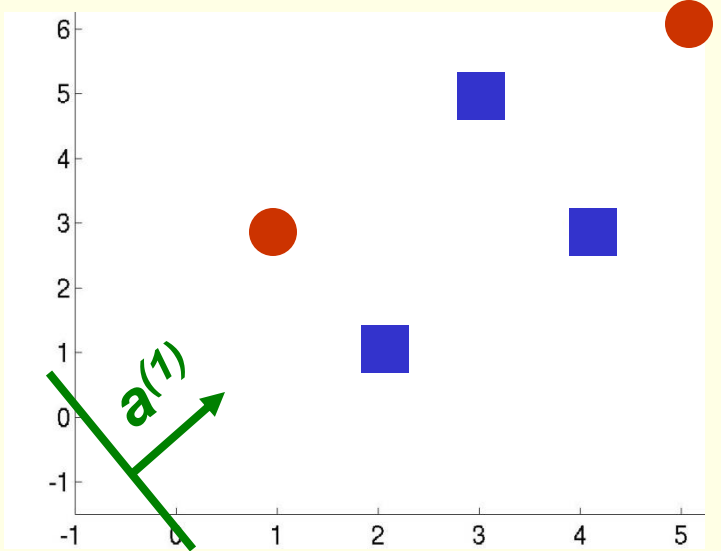
- Let's apply Perceptron single sample algorithm

- initial equal weights $\mathbf{a}^{(1)} = [1 \ 1 \ 1]$

- this is line $\mathbf{x}^{(1)} + \mathbf{x}^{(2)} + 1 = 0$

- fixed learning rate $\eta = 1$

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



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

- $\mathbf{y}_1^t \mathbf{a}^{(1)} = [1 \ 1 \ 1] * [1 \ 2 \ 1]^t > 0$ ✓
- $\mathbf{y}_2^t \mathbf{a}^{(1)} = [1 \ 1 \ 1] * [1 \ 4 \ 3]^t > 0$ ✓ □
- $\mathbf{y}_3^t \mathbf{a}^{(1)} = [1 \ 1 \ 1] * [1 \ 3 \ 5]^t > 0$ ✓ □

LDF: Nonseparable Example

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

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

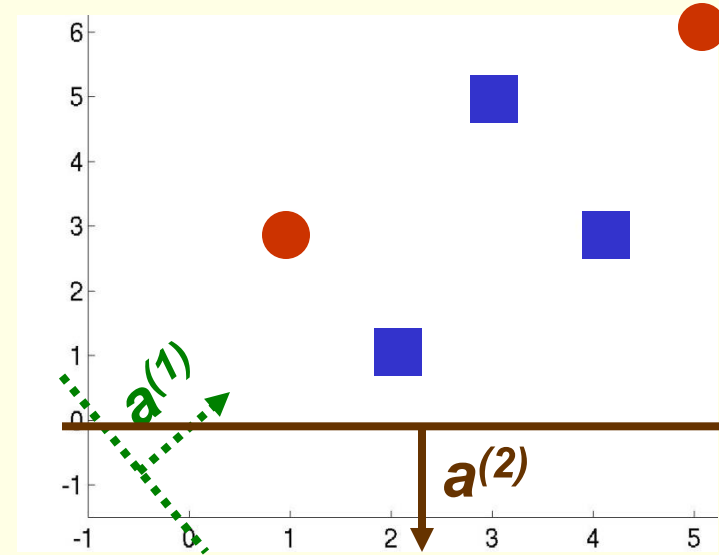
- $\mathbf{y}_4^t \mathbf{a}^{(1)} = [1 \ 1 \ 1]^t [-1 \ -1 \ -3]^t = -5 < 0$

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

- $\mathbf{y}_5^t \mathbf{a}^{(2)} = [0 \ 0 \ -2]^t [-1 \ -5 \ -6]^t = 12 > 0 \quad \checkmark \square$

- $\mathbf{y}_1^t \mathbf{a}^{(2)} = [0 \ 0 \ -2]^t [1 \ 2 \ 1]^t < 0 \square$

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



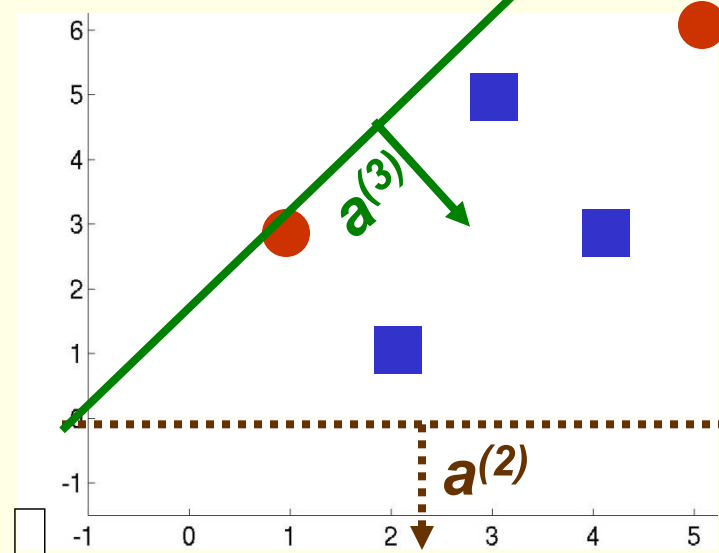
LDF: Nonseparable Example

$$\mathbf{a}^{(3)} = [1 \ 2 \ -1] \quad \mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \mathbf{y}_M$$

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

- $\mathbf{y}_2^t \mathbf{a}^{(3)} = [1 \ 4 \ 3]^* [1 \ 2 \ -1]^t = 6 > 0$ ✓ □
- $\mathbf{y}_3^t \mathbf{a}^{(3)} = [1 \ 3 \ 5]^* [1 \ 2 \ -1]^t > 0$ ✓ □
- $\mathbf{y}_4^t \mathbf{a}^{(3)} = [-1 \ -1 \ -3]^* [1 \ 2 \ -1]^t = 0$ □

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



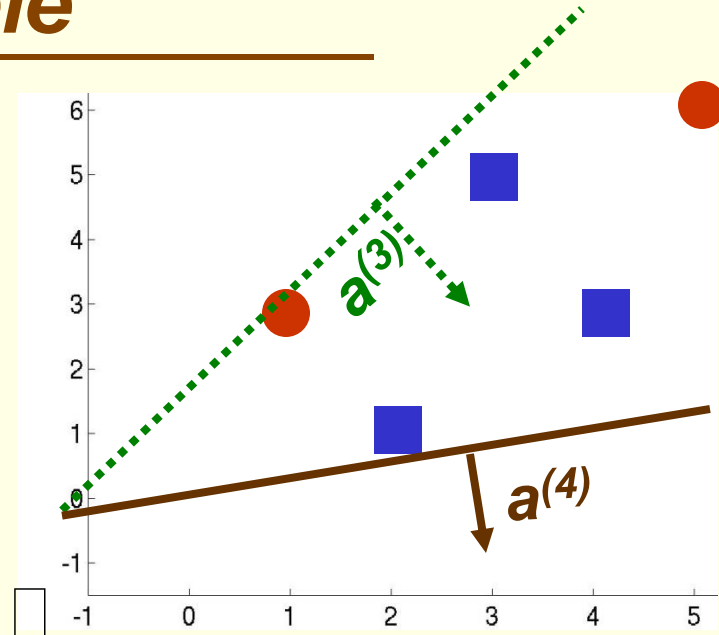
LDF: Nonseparable Example

$$\mathbf{a}^{(4)} = [0 \ 1 \ -4] \quad \mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \mathbf{y}_M$$

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

- $\mathbf{y}_2^t \mathbf{a}^{(3)} = [1 \ 4 \ 3]^* [1 \ 2 \ -1]^t = 6 > 0$ ✓ □
- $\mathbf{y}_3^t \mathbf{a}^{(3)} = [1 \ 3 \ 5]^* [1 \ 2 \ -1]^t > 0$ ✓ □
- $\mathbf{y}_4^t \mathbf{a}^{(3)} = [-1 \ -1 \ -3]^* [1 \ 2 \ -1]^t = 0$ □

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



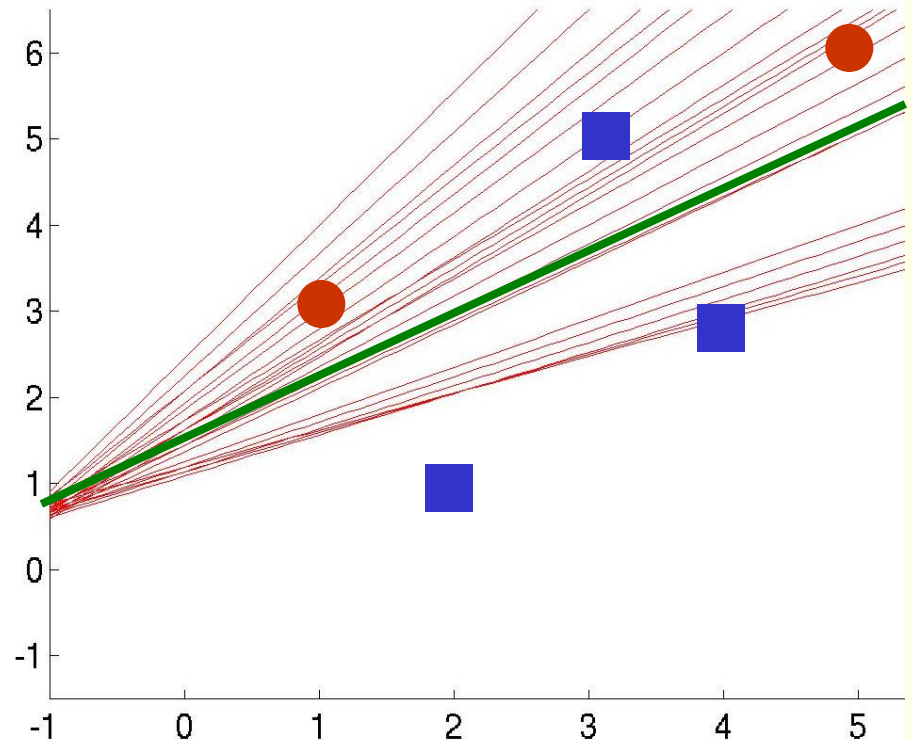
LDF: Nonseparable Example

- we can continue this forever
 - there is no solution vector \mathbf{a} satisfying for all i □

$$\mathbf{a}^t \mathbf{y}_i = \sum_{k=0}^5 \mathbf{a}_k \mathbf{y}_i^{(k)} > 0$$

- 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 \mathbf{c} , $\eta^{(k)} = \mathbf{c}$
 - *both single sample and batch perceptron rules converge to a correct solution* (could be any \mathbf{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: $\eta^{(k)} \rightarrow \mathbf{0}$ 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

- Smoother gradient because all samples are used

Single Sample Rule

- easier to analyze
- Concentrates more than necessary on any isolated “noisy” training examples