CS9840
Learning and Computer Vision
Prof. Olga Veksler

Lecture 6
Linear Machines

Today

- Optimization with Gradient descent
- Linear Classifier
    - Two classes
    - Multiple classes
    - Perceptron Criterion Function
        - Batch perceptron rule
        - Single sample perceptron rule
    - Minimum Squared Error (MSE) rule
        - Pseudoinverse
- Generalized Linear Classifier
- Gradient Descent Based learning

Optimization

- How to minimize a function of a single variable

$$J(\mathbf{x}) = (\mathbf{x}\text{-}5)^2$$

- From calculus, take derivative, set it to 0

$$\frac{d}{dx}J(x) = 0$$

- Solve the resulting equation
  - maybe easy or hard to solve

- Example above is easy:

$$\frac{d}{dx}J(x) = 2(x-5) = 0 \Rightarrow x = 5$$

Optimization

- How to minimize a function of many variables
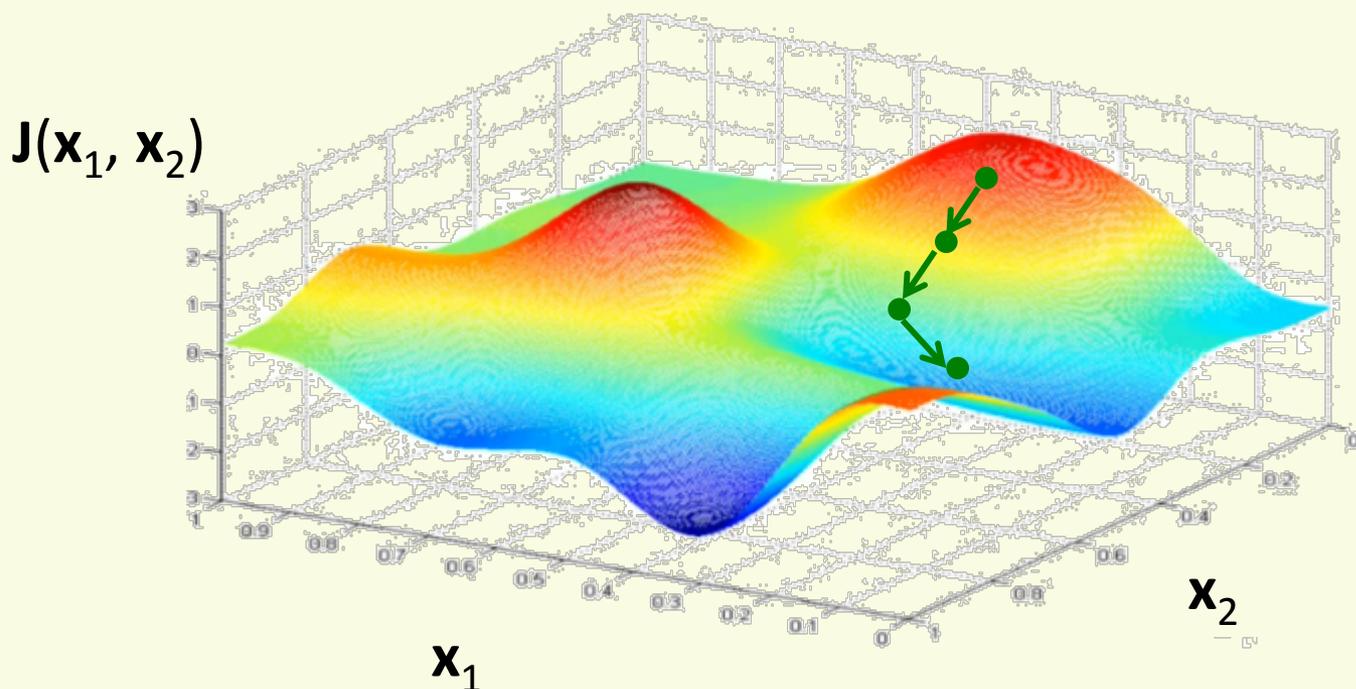
$$\mathbf{J(x)} = \mathbf{J(x}_1,..., \mathbf{x}_d)$$

- From calculus, take partial derivatives, set them to 0

gradient

$$\begin{bmatrix} \dfrac{\partial}{\partial \mathbf{x}_1}\mathbf{J(x)} \\ \vdots \\ \dfrac{\partial}{\partial \mathbf{x}_d}\mathbf{J(x)} \end{bmatrix} = \nabla \mathbf{J(x)} = \mathbf{0}$$

- Solve the resulting system of **d** equations
- It may not be possible to solve the system of equations above analytically

Optimization: Gradient Direction

$J(\mathbf{x}_1, \mathbf{x}_2)$

$\mathbf{x}_2$

$\mathbf{x}_1$

- Gradient $\nabla J(\mathbf{x})$ points in the direction of steepest increase of function $J(\mathbf{x})$

- $-\nabla J(\mathbf{x})$ points in the direction of steepest decrease

Picture from Andrew Ng

Gradient Direction in 2D

- $J(x_1, x_2) = (x_1-5)^2 + (x_2-10)^2$

- $\dfrac{\partial}{\partial x_1} J(x) = 2(x_1 - 5)$

- $\dfrac{\partial}{\partial x_2} J(x) = 2(x_2 - 10)$

- Let $a = [10, 5]$

- $-\dfrac{\partial}{\partial x_1} J(a) = -10$

- $-\dfrac{\partial}{\partial x_2} J(a) = 10$

global min

[-10, 10]

x_2

x_1

a

10

5

5

10
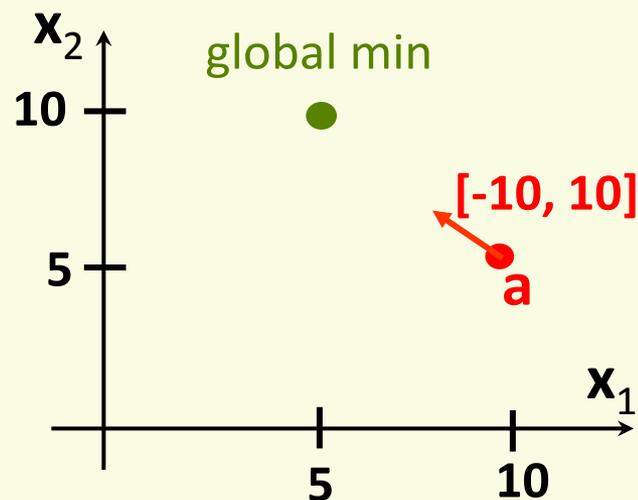
Gradient Descent: Step Size

- $J(x_1, x_2) = (x_1-5)^2+(x_2-10)^2$
- Which step size to take?
- Controlled by parameter α
  - called **learning rate**
- From previous example:
  - $a = [10 \quad 5]$
  - $-\nabla J(a) = [-10 \quad 10]$
- Let $\alpha = 0.2$
- $a - \alpha \nabla J(a) = [10 \quad 5]+0.2 [-10 \quad 10]=[8 \quad 7]$
- $J(10, 5) = 50$
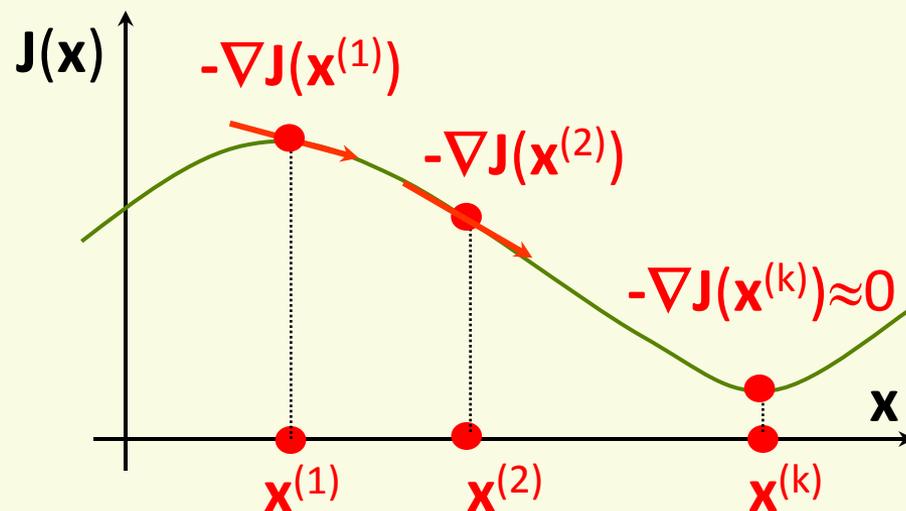- $J(8,7) = 18$
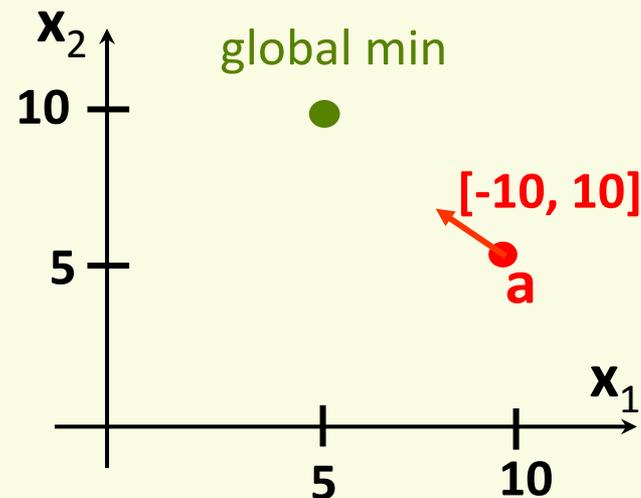
Gradient Descent Algorithm

$k = 1$

$x^{(1)}$ = any initial guess

choose α, ε

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

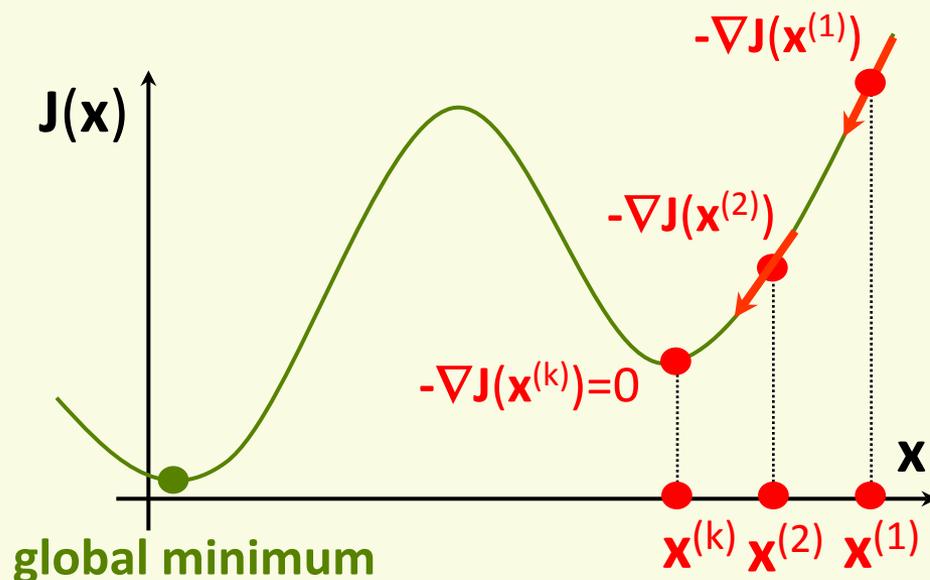$\quad x^{(k+1)} = x^{(k)} - \alpha \nabla J(x^{(k)})$

$\quad k = k + 1$

$J(x)$

$-\nabla J(x^{(1)})$

$-\nabla J(x^{(2)})$

$-\nabla J(x^{(k)}) \approx 0$

x

$x^{(1)}$ $x^{(2)}$ $x^{(k)}$

Gradient Descent: Local Minimum

- Not guaranteed to find global minimum
  - gets stuck in local minimum

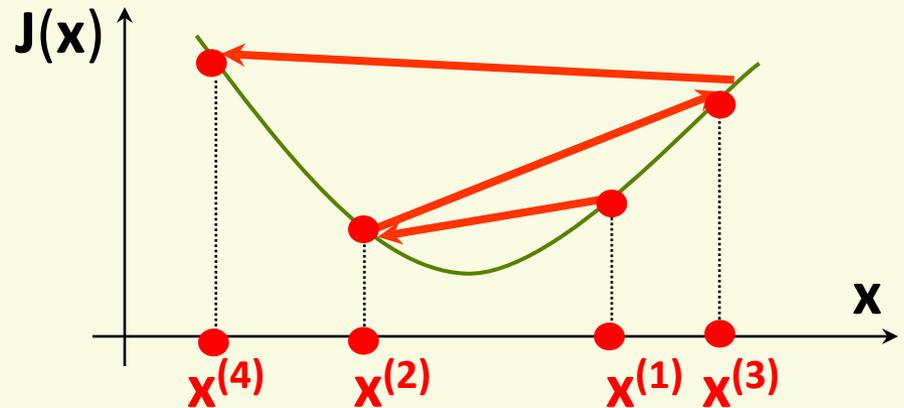- Still gradient descent is very popular because it is simple and applicable to any differentiable function

How to Set Learning Rate α?

- If α too small, too many iterations to converge

- If α too large, may overshoot the local minimum and possibly never even converge

- It helps to compute **J(x)** as a function of iteration number, to make sure we are properly minimizing it

How to Set Learning Rate α?

- As we approach local minimum, often gradient gets smaller

- Step size may get smaller automatically, even if α is fixed

- So it may be unnecessary to decrease α over time in order not to overshoot a local minimum

slope gets smaller

Variable Learning Rate

- If desired, can change learning rate α at each iteration

$k = 1$

$\mathbf{x}^{(1)}$ = any initial guess

choose α, ε

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

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

$\qquad k = k + 1$

$\longrightarrow$

$k = 1$

$\mathbf{x}^{(1)}$ = any initial guess

choose ε

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

$\qquad$ choose $\alpha^{(k)}$

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

$\qquad k = k + 1$

Variable Learning Rate

- Usually don't keep track of all intermediate solutions

$k = 1$

$x^{(1)}$ = any initial guess

choose α, ε

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

$\quad x^{(k+1)} = x^{(k)} - \alpha \nabla J(x^{(k)})$

$\quad k = k + 1$

$\longrightarrow$

x = any initial guess

choose α, ε

while $\alpha\|\nabla J(x)\| > \varepsilon$

$\quad x = x - \alpha \nabla J(x)$

Advanced Optimization Methods

- There are more advanced gradient-based optimization methods

- Such as conjugate gradient

  - automatically pick a good learning rate α

  - usually converge faster

  - however more complex to understand and implement

  - in Matlab, use **fminunc** for various advanced optimization methods

Last Time: Supervised Learning

- Training samples (or examples)

$$\mathbf{x}^1, \mathbf{x}^2, \ldots \mathbf{x}^n$$

- Each example is typically multi-dimensional

  - $\mathbf{x}^i = [\mathbf{x}^i_1, \mathbf{x}^i_2, \ldots, \mathbf{x}^i_d]$
  - $\mathbf{x}^i$ is often called a *feature vector*

- Know desired output for each example

$$\mathbf{y}^1, \mathbf{y}^2, \ldots \mathbf{y}^n$$

  - regression: continuous $\mathbf{y}$
  - classification: finite $\mathbf{y}$

Last Time: Supervised Learning

- Wish to design a *machine* $\mathbf{f}(\mathbf{x},\mathbf{w})$ s.t.

$$\mathbf{f}(\mathbf{x},\mathbf{w}) = \mathbf{y}$$

  - How do we choose $\mathbf{f}$?
    - last lecture studied kNN classifier
    - this lecture in on liner classifier
    - many other choices
  - W is typically multidimensional vector of weights (also called *parameters*)

$$\mathbf{w} = [\mathbf{w}_1,\mathbf{w}_2,...\mathbf{w}_k]$$

  - By modifying $\mathbf{w}$, the machine "learns"

Training and Testing Phases

- Divide all labeled samples $\mathbf{x}^1$, $\mathbf{x}^2$,…, $\mathbf{x}^n$ into *training* and *test* sets

- Training phase

  - Uses training samples

  - goal is to "teach" the machine

  - find weights $\mathbf{w}$ s.t. $\mathbf{f}(\mathbf{x}^i,\mathbf{w}) = \mathbf{y}^i$ "as much as possible"

    - "as much as possible" needs to be defined

- Testing phase

  - Uses only test samples

  - for evaluating how well our machine works on unseen examples

Loss Function

- How to quantify "$f(x^i,w) = y^i$ as much as possible"?

- $f(x,w)$ has to be "close" to the true output y

- Define Loss (or Error, or Criterion) function **L**

- Typically first define per-sample loss $L(x^i,y^i,w)$
  - for classification, $L(x^i,y^i,w) = I[f(x^i,w) \neq y^i]$
    - where $I[true] = 1$, $I[false] = 0$

  - for regression, $L(x^i,y^i,w) = \| f(x^i,w) - y^i \|^2$,
    - how far is the estimated output from the correct one?

- Then loss function $L = \sum_i L(x^i,y^i,w)$

  - classification: counts number of missclassified examples

  - regression: sums distances to the correct output

Linear Machine: Regression

- $f(\mathbf{x},\mathbf{w}) = \mathbf{w}_0 + \sum_{i=1,2,\ldots d} \mathbf{w}_i\mathbf{x}_i$

- In vector notation

  - $\mathbf{x} = [\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_d]$

  - $f(\mathbf{x},\mathbf{w}) = \mathbf{w}_0 + \mathbf{w}^t\mathbf{x}$

- This is standard linear regression

  - line fitting

  - assume $\mathbf{L}(\mathbf{x}^i,\mathbf{y}^i,\mathbf{w}) = \left\|f(\mathbf{x}^i,\mathbf{w}) - \mathbf{y}^i\right\|^2$

- optimal $\mathbf{w}$ can be found by solving a system of linear equations

$$\mathbf{w}^* = [\Sigma\mathbf{x}^i (\mathbf{x}^i)^T]^{-1} \Sigma\mathbf{y}^i\mathbf{x}^i$$
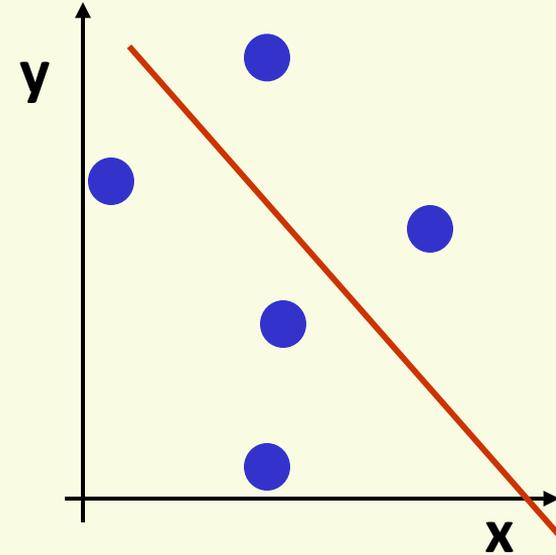
Linear Machine: Classification

- First consider the two-class case
- We choose the following encoding:
  - $y = 1$ for the first class
  - $y = -1$ for the second class
- Linear classifier
  - $-\infty \leq w_0 + x_1 w_1 + \ldots + x_d w_d \leq \infty$
  - we need $f(x,w)$ to be either $+1$ or -1
  - let $g(x,w) = w_0 + x_1 w_1 + \ldots + x_d w_d = w_0 + w^t x$
  - let $f(x,w) = \text{sign}(g(x,w))$
    - 1 if $g(x,w)$ is positive
    - -1 if $g(x,w)$ is negative
    - other choices for $g(x,w)$ are also used
  - $g(x,w)$ is called the **discriminant function**
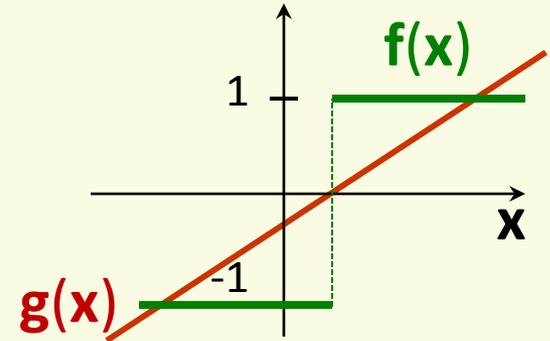
Linear Classifier: Decision Boundary
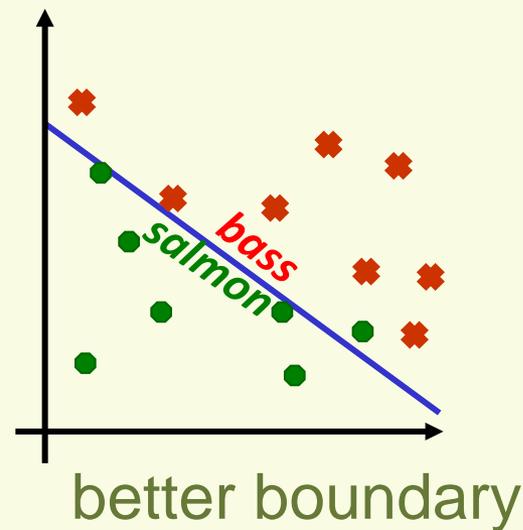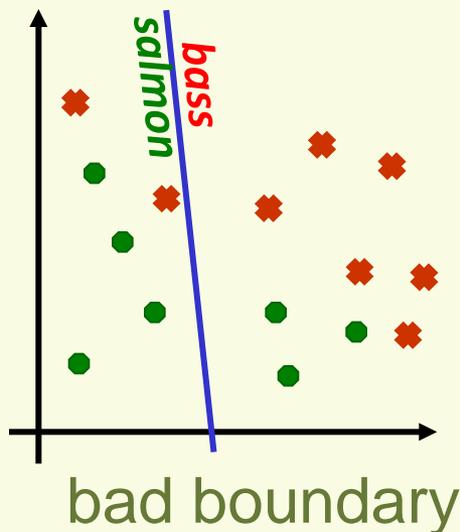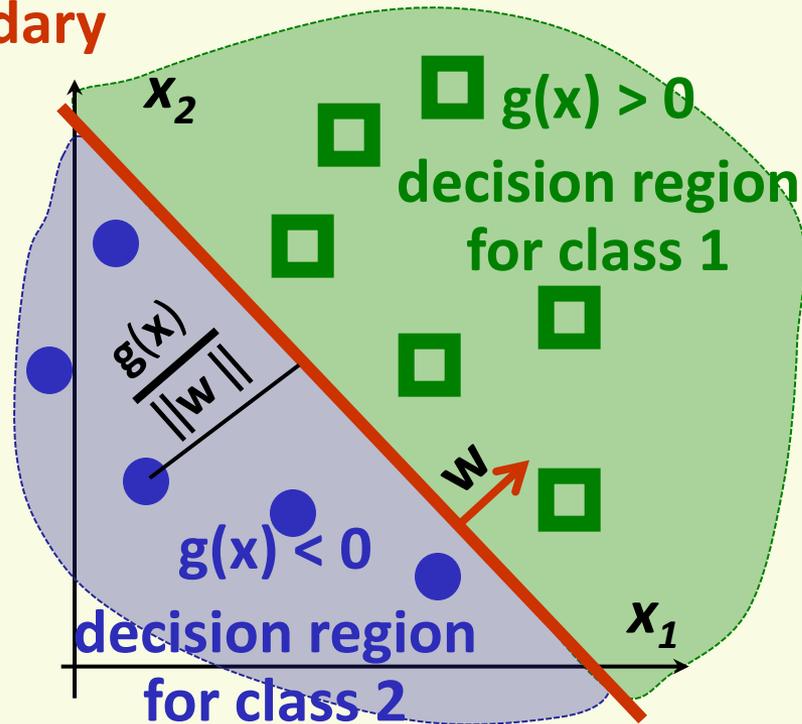

- $f(\mathbf{x},\mathbf{w}) = \text{sign}(g(\mathbf{x},\mathbf{w})) = \text{sign}(\mathbf{w}_0 + \mathbf{x}_1\mathbf{w}_1 + \ldots + \mathbf{x}_d\mathbf{w}_d)$

- Decision boundary is linear

- Find the best linear boundary to separate two classes

- Search for best $\mathbf{w} = [\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_d]$ to minimize training error

More on Linear Discriminant Function (LDF)

- LDF: $g(\mathbf{x},\mathbf{w}) = \mathbf{w}_0 + \mathbf{x}_1\mathbf{w}_1 + \dots + \mathbf{x}_d\mathbf{w}_d$

- Written using vector notation $g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + \mathbf{w}_0$

weight vector   **bias** or **threshold**

decision boundary

$g(x) = 0$

x_2

$g(x) > 0$

decision region for class 1

$\dfrac{g(x)}{\|\mathbf{w}\|}$

$\mathbf{w}$

$g(x) < 0$

decision region for class 2

x_1

More on Linear Discriminant Function (LDF)

- Decision boundary: $g(\mathbf{x},\mathbf{w}) = \mathbf{w}_0 + \mathbf{x}_1\mathbf{w}_1 + \dots + \mathbf{x}_d\mathbf{w}_d = 0$
- This is a hyperplane, by definition
    - a point in 1D
    - a line in 2D
    - a plane in 3D
   - a hyperplane in higher dimensions

Multiple Classes

- We have **m** classes
- Define **m** linear discriminant functions
$$g_i(x) = w_i^t x + w_{i0} \text{ for } i = 1, 2, \dots m$$

- Assign **x** to class **i** if

$$g_i(x) > g_j(x) \text{ for all } j \neq i$$

- Let R_i be the decision region for class **i**
  - That is all examples in R_i get assigned class **i**

R_1
$g_1(x) > g_2(x)$
$g_1(x) > g_3(x)$

R_2
$g_2(x) > g_1(x)$
$g_2(x) > g_3(x)$

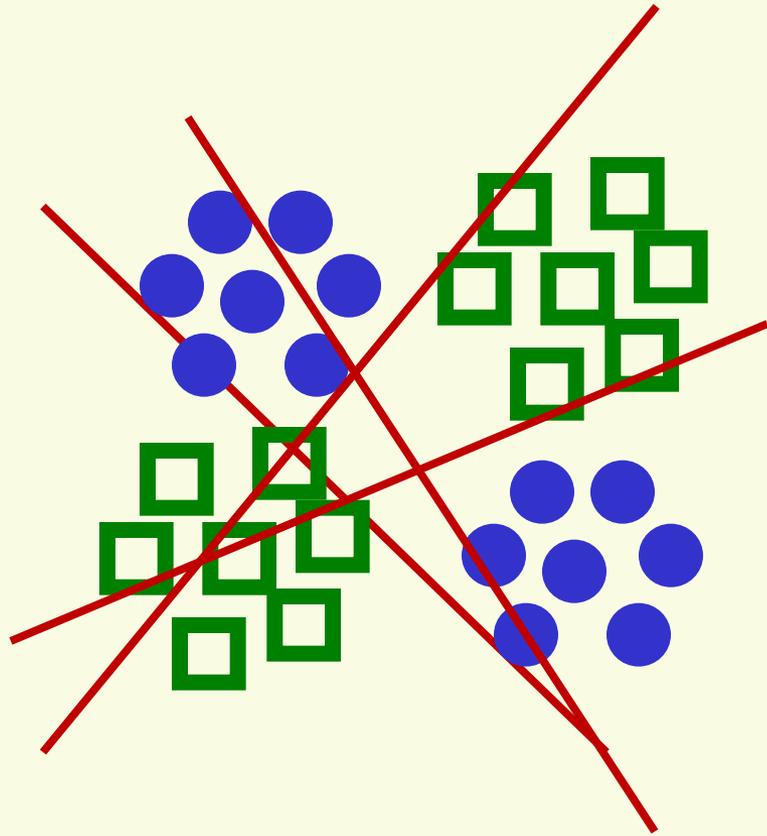R_3
$g_3(x) > g_1(x)$
$g_3(x) > g_2(x)$
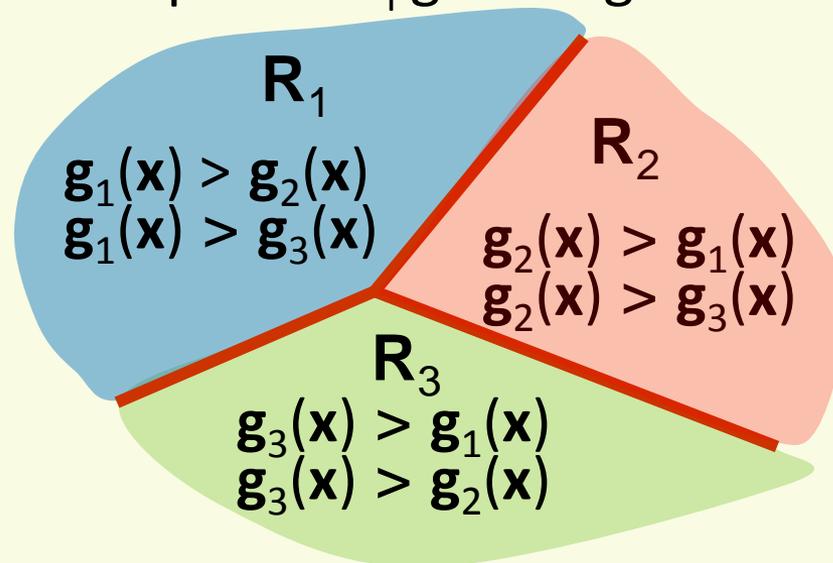
Multiple Classes

- Can be shown that decision regions are convex
- In particular, they must be spatially contiguous
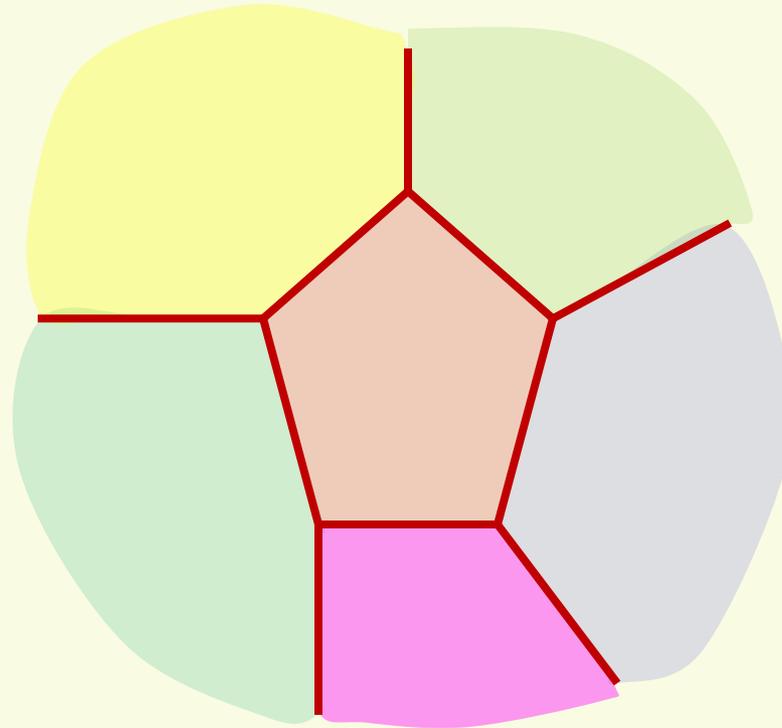
Failure Cases for Linear Classifier

- Thus applicability of linear classifiers is limited to mostly unimodal distributions, such as Gaussian

- Not unimodal data

- Need non-contiguous decision regions

- Linear classifier will fail

Linear Classifiers

- Linear classifiers give simple decision boundary
  - try simpler models first
- Linear classifiers are optimal for certain type of data
  - Gaussian distributions with equal covariance
- May not be optimal for other data distributions, but they are very simple to use

Fitting Parameters w

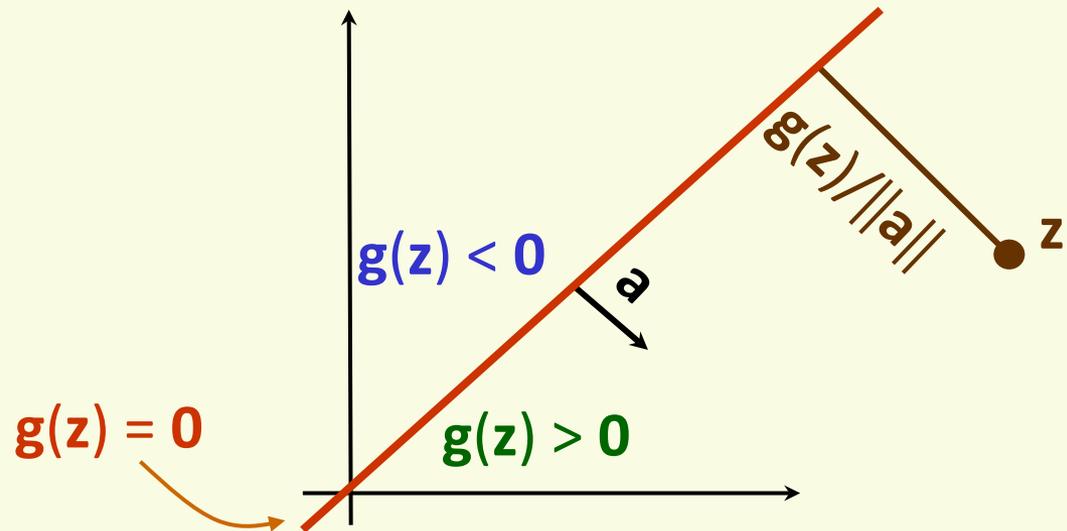- Linear discriminant function $g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + \mathbf{w}_0$

- Can rewrite it $g(\mathbf{x}) = \begin{bmatrix} \mathbf{w}_0 & \mathbf{w}^t \end{bmatrix} \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix} = \mathbf{a}^t\mathbf{z} = g(\mathbf{z})$

  new weight vector **a**

  new feature vector **z**

- **z** is called augmented feature vector

- new problem equivalent to the old $g(\mathbf{z}) = \mathbf{a}^t\mathbf{z}$

$$\begin{bmatrix} \mathbf{w}_0 \\ \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_d \end{bmatrix} \qquad \begin{bmatrix} 1 \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_d \end{bmatrix}$$

Augmented Feature Vector

- Feature augmenting is done to simplify notation
- From now on we assume that we have augmented feature vectors
  - given samples $x^1,\dots, x^n$ convert them to augmented samples $z^1,\dots, z^n$ by adding a new dimension of value 1
- $g(z) = a^t z$

Training Error

- For the rest of the lecture, assume we have 2 classes

- Samples $z^1,..., z^n$ some in class 1, some in class 2

- Use these samples to determine weights **a** in the discriminant function $g(z) = a^t z$

- Want to minimize number of misclassified samples

- Recall that $\begin{cases} g(z^i) > 0 \Rightarrow \text{class 1} \\ g(z^i) < 0 \Rightarrow \text{class 2} \end{cases}$

- Thus training error is 0 if $\begin{cases} g(z^i) > 0 \quad \forall z^i \text{ class 1} \\ g(z^i) < 0 \quad \forall z^i \text{ class 2} \end{cases}$

Simplifying Notation Further

- Thus training error is 0 if
$$\begin{cases} \mathbf{a}^t\mathbf{z}^i > 0 & \forall \mathbf{z}^i \text{ class 1} \\ \mathbf{a}^t\mathbf{z}^i < 0 & \forall \mathbf{z}^i \text{ class 2} \end{cases}$$

- Equivalently, training error is 0 if
$$\begin{cases} \mathbf{a}^t\mathbf{z}^i > 0 & \forall \mathbf{z}^i \text{ class 1} \\ \mathbf{a}^t(-\mathbf{z}^i) > 0 & \forall \mathbf{z}^i \text{ class 2} \end{cases}$$

- Problem "normalization":
  1. replace all examples $\mathbf{z}^i$ from class 2 by $-\mathbf{z}^i$
  2. seek weights $\mathbf{a}$ s.t. $\mathbf{a}^t\mathbf{z}^i > 0$ for $\forall \mathbf{z}^i$

- If exists, such $\mathbf{a}$ is called a ***separating*** or ***solution*** vector

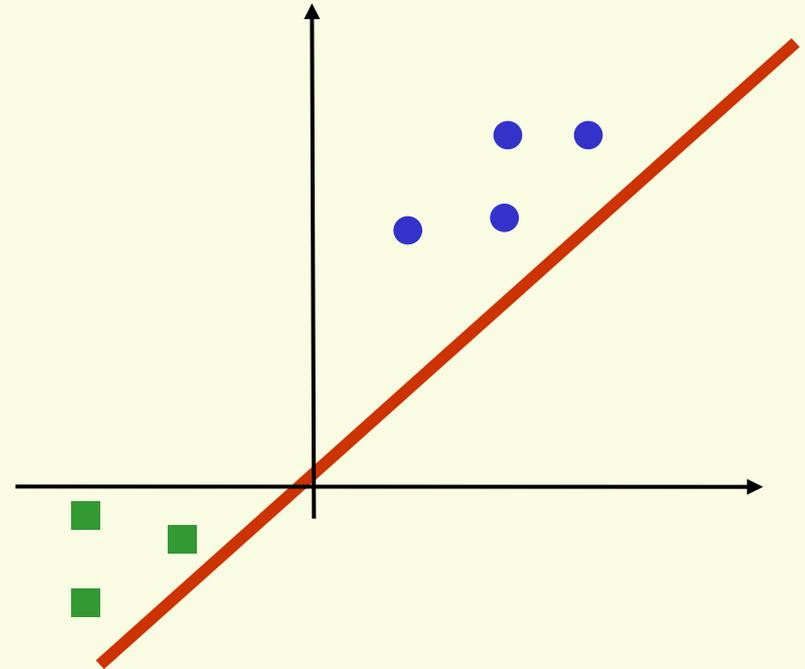- Original samples $\mathbf{x}^1, \dots \mathbf{x}^n$ can also be linearly separated

Effect of Normalization

before normalization
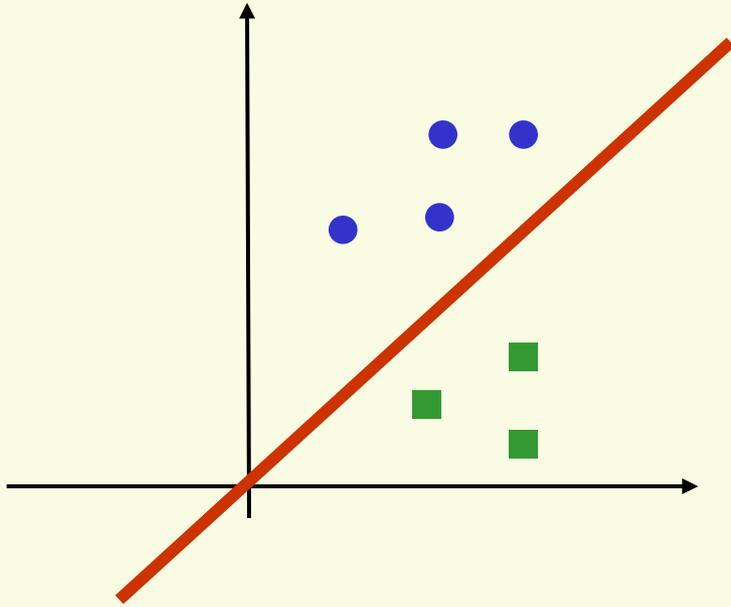
seek a hyperplane that separates samples from different categories
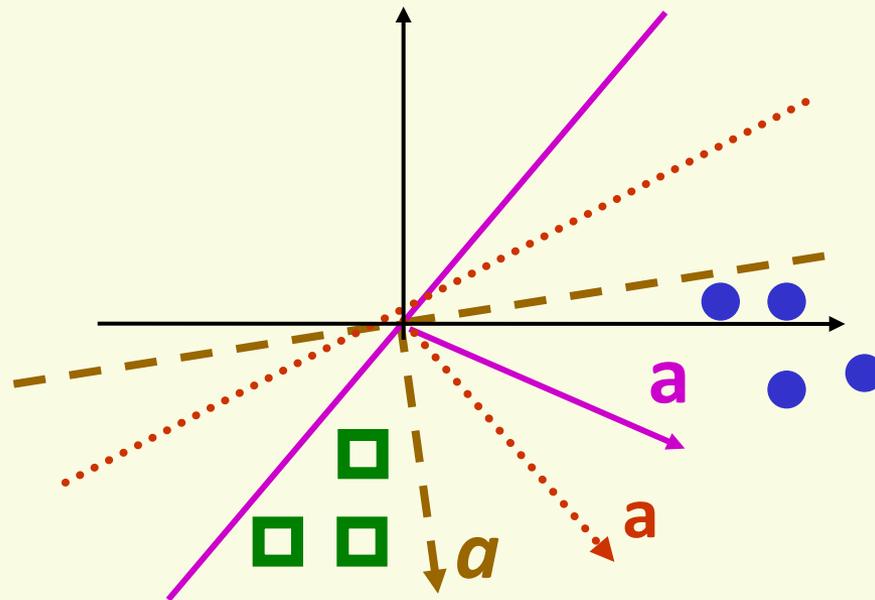
after normalization

seek hyperplane that puts normalized samples on the same (positive) side

Solution Region

- Find weight vector **a** s.t. for all samples $\mathbf{z}^1, \ldots, \mathbf{z}^n$

$$\mathbf{a}^t \mathbf{z}^i = \sum_{k=0}^{d} a_k z_d^i > 0$$
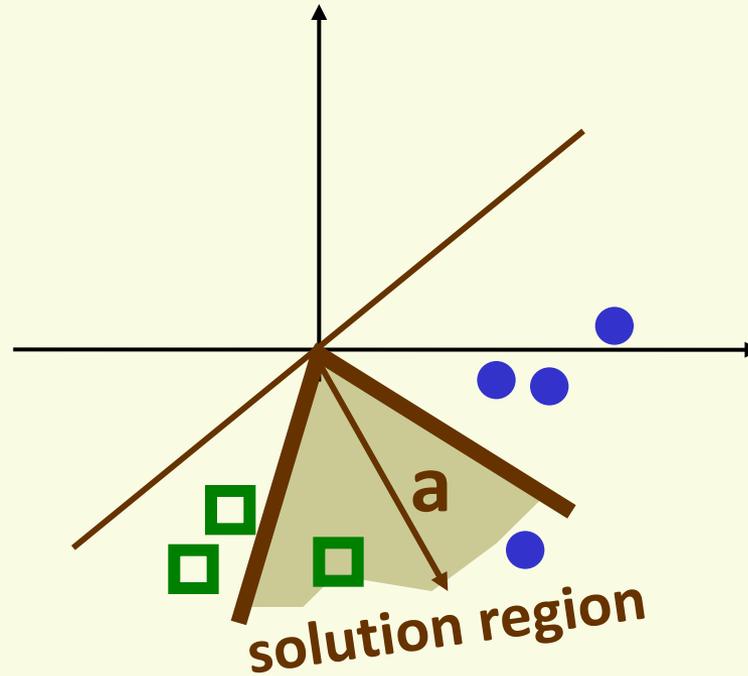

- If there is one such **a**, then there are infinitely many **a**

Solution Region

- Solution region: the set of all possible solutions for **a**

Minimum Squared Error Optimization (MSE)

- Linear Regression is a very well understood problem

- Problem is not regression, but let's convert to regression!

$$\mathbf{a}^t\mathbf{z}^i > 0 \text{ for all samples } \mathbf{z}^i$$

solve system of linear inequalities

$$\mathbf{a}^t\mathbf{z}^i = \boldsymbol{b}_i \text{ for all samples } \mathbf{z}^i$$

solve system of linear equations

- MSE procedure
  - choose positive constants $\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_n$
  - try to find weight vector $\mathbf{a}$ s.t. $\mathbf{a}^t\mathbf{z}^i = \mathbf{b}_i$ for all samples $\mathbf{z}^i$
  - if succeed, then $\mathbf{a}$ is a solution because $\mathbf{b}_i$'s are positive
  - consider all the samples (not just the misclassified ones)

MSE: Margins

- By setting $\mathbf{a}^t\mathbf{z}^i = \mathbf{b}_i$, we expect $\mathbf{z}^i$ to be at a relative distance $\mathbf{b}_i$ from the separating hyperplane

- Thus $\mathbf{b}_1$, $\mathbf{b}_2$,..., $\mathbf{b}_n$ are expected relative distances of examples from the separating hyperplane

- Should make $\mathbf{b}_i$ small if sample $\mathbf{i}$ is expected to be near separating hyperplane, and make $\mathbf{b}_i$ larger otherwise

- In the absence of any such information, there are good reasons to set

$$\mathbf{b}_1 = \mathbf{b}_2 = ... = \mathbf{b}_n = 1$$
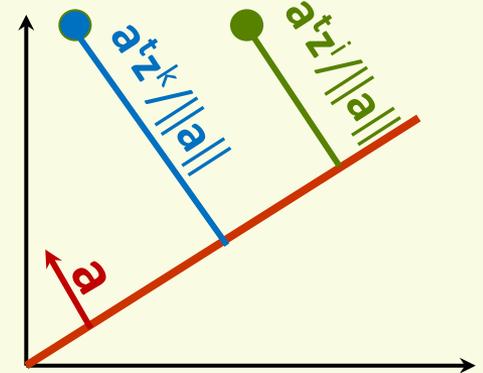
MSE: Matrix Notation

- Solve system of **n** equations $\begin{cases} \mathbf{a^t z}^1 = \mathbf{b}_1 \\ \quad \vdots \\ \mathbf{a^t z^n} = \mathbf{b_n} \end{cases}$

- Using matrix notation:

$$\begin{bmatrix} \mathbf{z}_0^1 & \mathbf{z}_1^1 & \cdots & \mathbf{z}_d^1 \\ \mathbf{z}_0^2 & \mathbf{z}_1^2 & \cdots & \mathbf{z}_d^2 \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ \mathbf{z}_0^n & \mathbf{z}_1^n & \cdots & \mathbf{z}_d^n \end{bmatrix} \begin{bmatrix} \mathbf{a}_0 \\ \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_d \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \vdots \\ \mathbf{b}_n \end{bmatrix}$$

$$\textcolor{red}{\mathbf{Z}} \qquad \textcolor{blue}{\mathbf{a}} \qquad \textcolor{green}{\mathbf{b}}$$

- Solve a linear system **Za** = **b**

MSE:Approximate Solution

- Typically **Z** is overdetermined
  - more rows (examples) than columns (features)

$$\boxed{}\; \boxed{a} \;=\; \boxed{b}$$

- No exact solution for **Za** = **b** in this case

- Find an approximate solution **a**, that is **Za** $\approx$ **b**
  - approximate solution **a** does not necessarily give a separating hyperplane in the separable case
  - but hyperplane corresponding to an approximate **a** may still be a good solution

- Least Squares Solution: **a** = **(Z^{t}Z)$^{-1}$ Z^{t}b**

MSE: Example

- Class 1: (6 9), (5 7)
- Class 2: (5 9), (0 4)
- Add extra feature and "normalize"

$$\mathbf{z}^1 = \begin{bmatrix} 1 \\ 6 \\ 9 \end{bmatrix} \quad \mathbf{z}^2 = \begin{bmatrix} 1 \\ 5 \\ 7 \end{bmatrix} \quad \mathbf{z}^3 = \begin{bmatrix} -1 \\ -5 \\ -9 \end{bmatrix} \quad \mathbf{z}^4 = \begin{bmatrix} -1 \\ 0 \\ -4 \end{bmatrix}$$

- $$\mathbf{Z} = \begin{bmatrix} 1 & 6 & 9 \\ 1 & 5 & 7 \\ -1 & -5 & -9 \\ -1 & 0 & -4 \end{bmatrix}$$
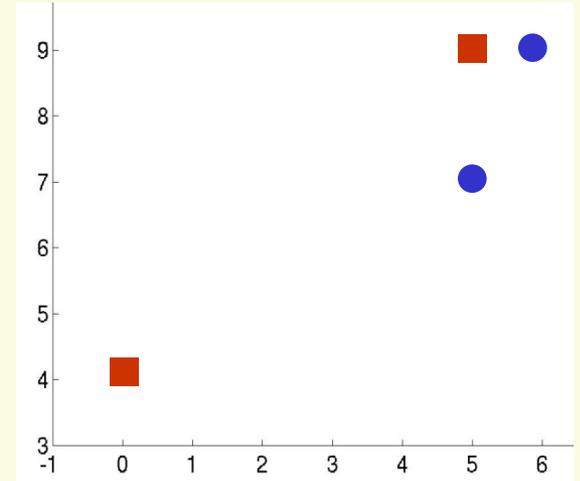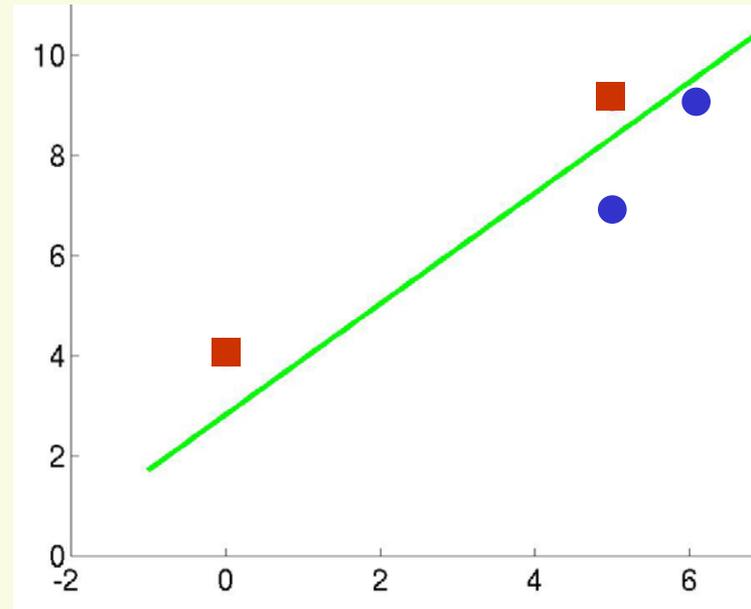
MSE: Example

- Choose **b**=$\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$

- Use **a**=**Z****b** to solve in Matlab

$$\mathbf{a} = \begin{bmatrix} 2.7 \\ 1.0 \\ -0.9 \end{bmatrix}$$

- Note **a** is an approximation since **Za** $= \begin{bmatrix} 0.4 \\ 1.3 \\ 0.6 \\ 1.1 \end{bmatrix} \neq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$
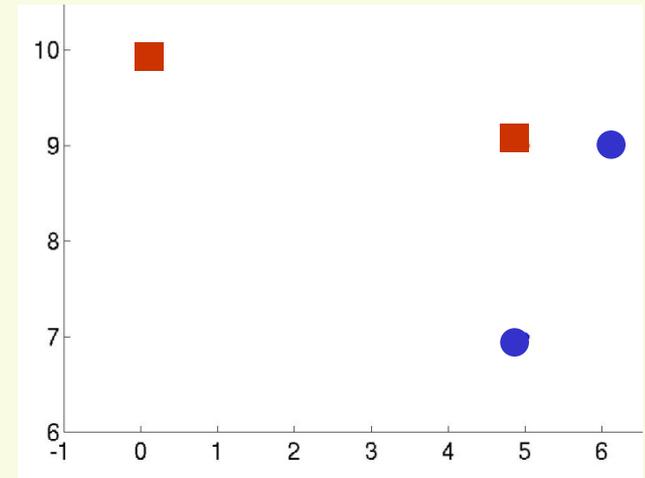
- Gives a separating hyperplane since **Za** > 0

MSE: Example

- Class 1: (6 9), (5 7)

- Class 2: (5 9), (0 10)

- One example is far compared to others from separating hyperplane

$$\mathbf{z}^1 = \begin{bmatrix} 1 \\ 6 \\ 9 \end{bmatrix} \qquad \mathbf{z}^2 = \begin{bmatrix} 1 \\ 5 \\ 7 \end{bmatrix} \qquad \mathbf{z}^3 = \begin{bmatrix} -1 \\ -5 \\ -9 \end{bmatrix} \qquad \mathbf{z}^4 = \begin{bmatrix} -1 \\ 0 \\ -10 \end{bmatrix}$$

- $\mathbf{Z} = \begin{bmatrix} 1 & 6 & 9 \\ 1 & 5 & 7 \\ -1 & -5 & -9 \\ -1 & 0 & -10 \end{bmatrix}$
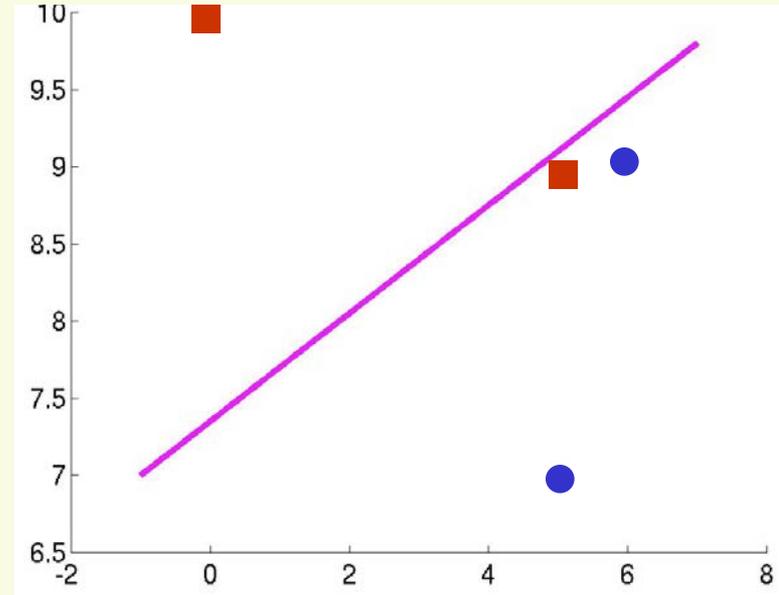
MSE: Example

- Choose **b** = $\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$

- Solve **a** = **Z****b** = $\begin{bmatrix} 3.2 \\ 0.2 \\ -0.4 \end{bmatrix}$

- **Za** = $\begin{bmatrix} 0.2 \\ 0.9 \\ -0.04 \\ 1.16 \end{bmatrix} \neq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$
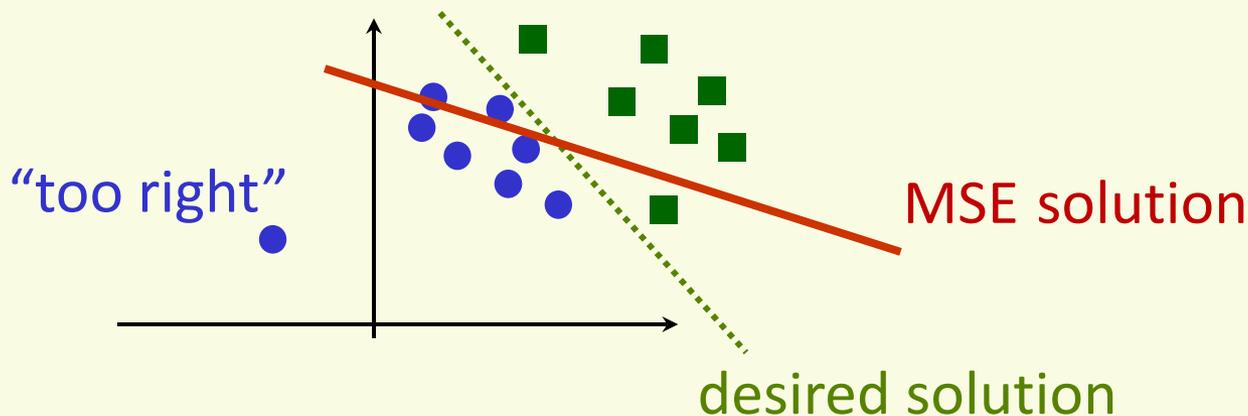

- Does not give a separating hyperplane since $\mathbf{a}^t\mathbf{z}^3 < \boldsymbol{0}$

MSE: Problems

- MSE wants all examples to be at the same distance from the separating hyperplane

- Examples that are "too right", i.e. too far from the boundary cause problems

"too right"

MSE solution

desired solution

- No problems with convergence though, both in separable and non-separable cases

- Can fix it in linearly separable case, i.e find better **b**

Another Approach: Design a Loss Function

- Find weight vector **a** s.t. $\forall \mathbf{z}^1, ..., \mathbf{z}^n$, $\mathbf{a}^t \mathbf{z}^i > 0$

- Design a loss function **J(a)**, which is minimum when **a** is a solution vector

- Let **Z(a)** be the set of examples misclassified by **a**

$$Z(\mathbf{a}) = \{ \mathbf{z}^i \mid \mathbf{a}^t \mathbf{z}^i < 0 \}$$

- Natural choice: number of misclassified examples

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

- Unfortunately, can't be minimized with gradient descent
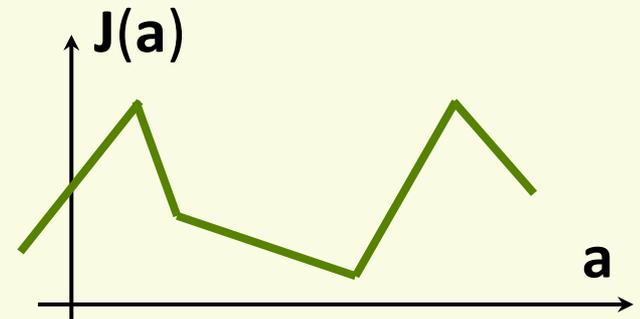
  - piecewise constant, gradient zero or does not exist
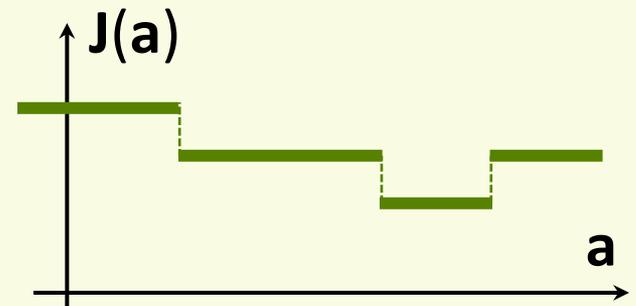
Perceptron Loss Function

- Better choice: Perceptron loss function

$$J_p(a) = \sum_{z \in Z(a)} \left(-a^t z \right)$$

- If z is misclassified, $a^t z < 0$

- Thus $J(a) \geq 0$

- $J_p(a)$ is proportional to the sum of distances of misclassified examples to decision boundary

- $J_p(a)$ is piecewise linear and suitable for gradient descent
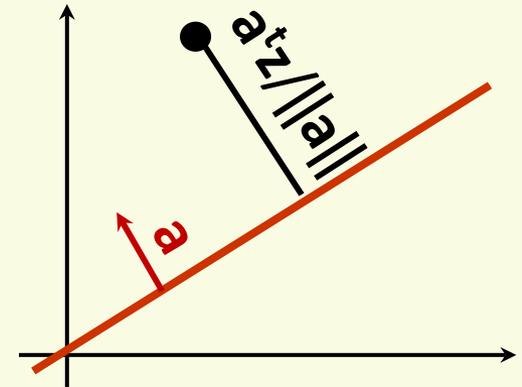
Optimizing with Gradient Descent

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

- Gradient of $J_p(\mathbf{a})$ is $\nabla J_p(\mathbf{a}) = \sum_{\mathbf{z} \in \mathbf{Z}(\mathbf{a})} \left(-\mathbf{z} \right)$

  - cannot solve $\nabla J_p(\mathbf{a}) = 0$ analytically because of $\mathbf{Z}(\mathbf{a})$

- Recall update rule for gradient descent

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

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

$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \alpha \sum_{\mathbf{z} \in \mathbf{Z}(\mathbf{a})} \mathbf{z}$$

  - called **batch rule** because it is based on all examples
  - true gradient descent

Perceptron Single Sample Rule

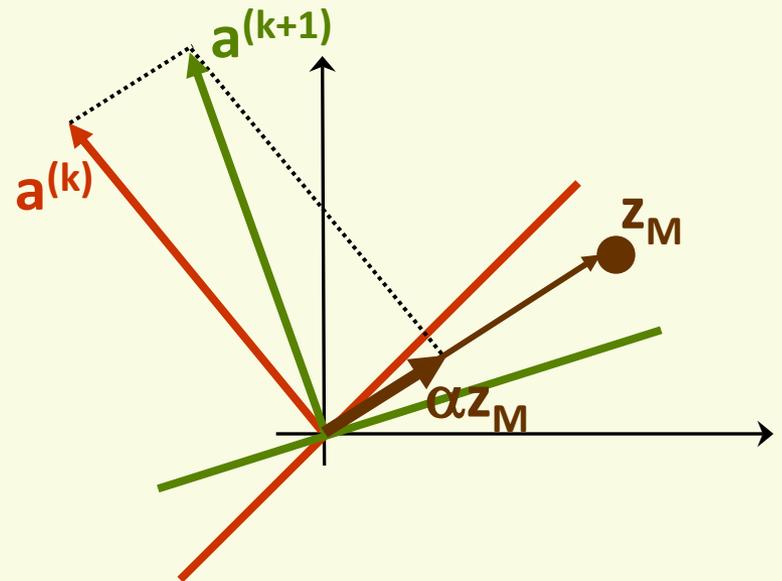- Gradient decent single sample rule for $\mathbf{J}_p(\mathbf{a})$ is

$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \alpha \cdot \mathbf{z}_M$$

  - $\mathbf{z}_M$ is one sample misclassified by $\mathbf{a}^{(k)}$
  - must have a consistent way to visit samples

- Geometric Interpretation:

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

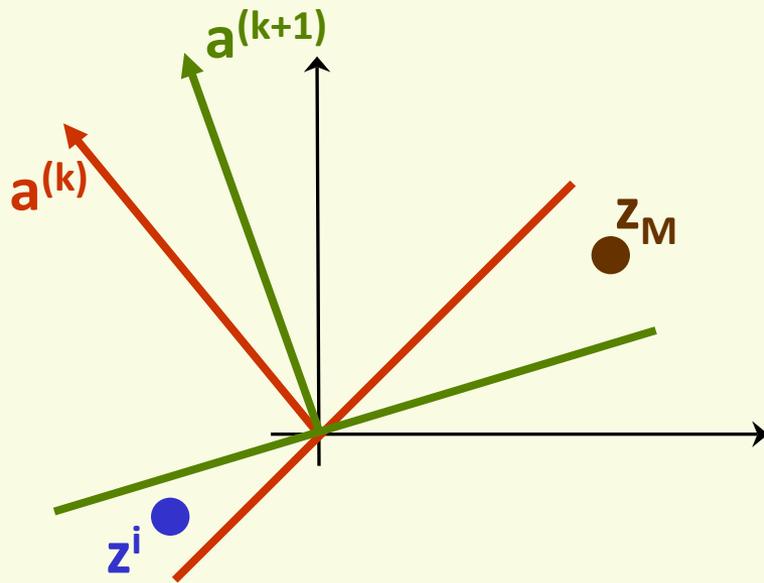$$\left(\mathbf{a}^{(k)}\right)^t \mathbf{z}_M \leq 0$$

- $\mathbf{z}_M$ is on the wrong side of decision boundary

- adding $\alpha \cdot \mathbf{z}_M$ to a moves decision boundary in the right direction

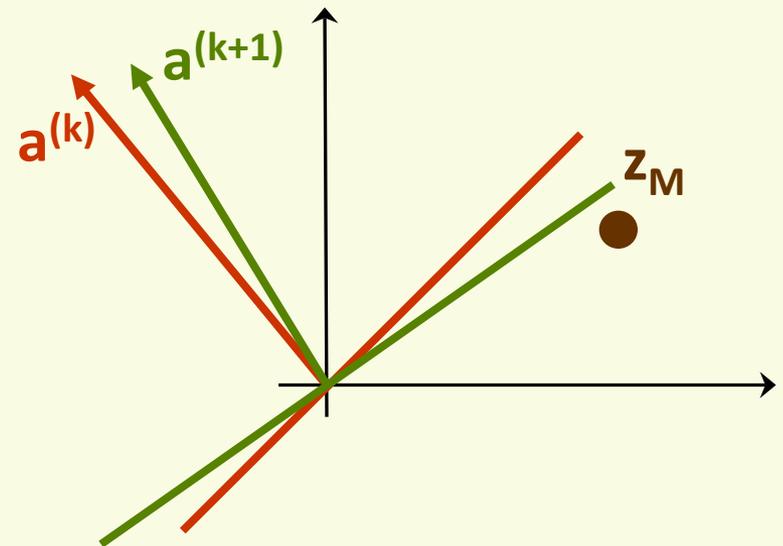# Perceptron Single Sample Rule

if α is too large, previously correctly classified sample $\mathbf{z^i}$ is now misclassified
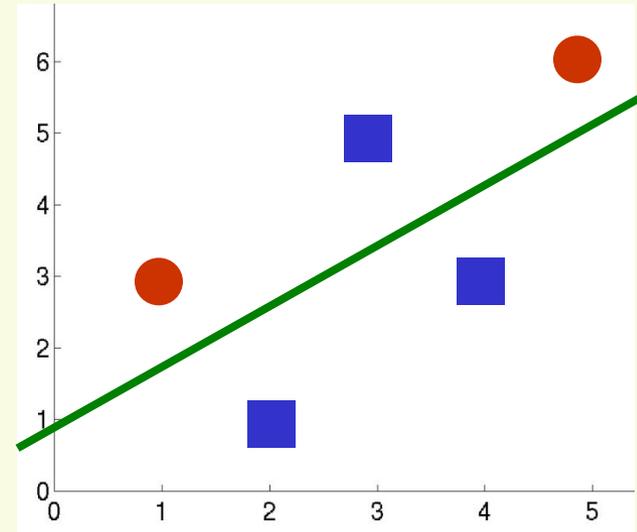
if α is too small, $\mathbf{z}_M$ is still misclassified

Non-Linearly Separable Case

- Suppose we have examples:
  - class 1: [2,1], [4,3], [3,5]
  - class 2: [1,3] , [5,6]
  - not linearly separable
- Still would like to get approximate separation
- Good line choice is shown in green
- Let us run gradient descent
  - Add extra feature and "normalize"

$$\mathbf{z}^1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \qquad \mathbf{z}^2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \qquad \mathbf{z}^3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \qquad \mathbf{z}^4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \qquad \mathbf{z}^5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$

Non-Linearly Separable Case

- single sample perceptron rule
- Initial weights $a^{(1)} = [1\ 1\ 1]$
- This is line $x_1 + x_2 + 1 = 0$
- Use fixed learning rate $\alpha = 1$
- Rule is: $a^{(k+1)} = a^{(k)} + z_M$
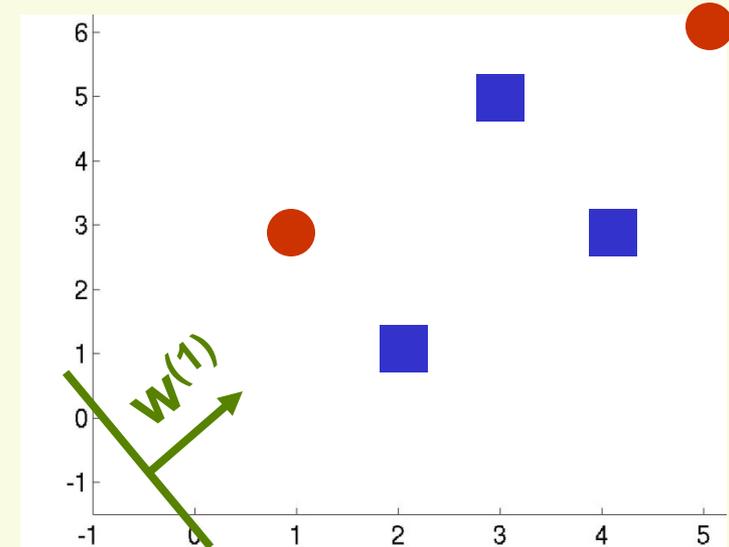
$$z^1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad z^2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad z^3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad z^4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad z^5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$

- $a^t z^1 = [1\ 1\ 1] \cdot [1\ 2\ 1]^t > 0$
- $a^t z^2 = [1\ 1\ 1] \cdot [1\ 4\ 3]^t > 0$
- $a^t z^3 = [1\ 1\ 1] \cdot [1\ 3\ 5]^t > 0$

Non-Linearly Separable Case

- $\mathbf{a^{(1)}} = [1 \quad 1 \quad 1]$

- rule is: $\mathbf{a^{(k+1)}} = \mathbf{a^{(k)}} + \mathbf{z_M}$

$$\mathbf{z}^1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad \mathbf{z}^2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad \mathbf{z}^3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad \mathbf{z}^4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad \mathbf{z}^5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$
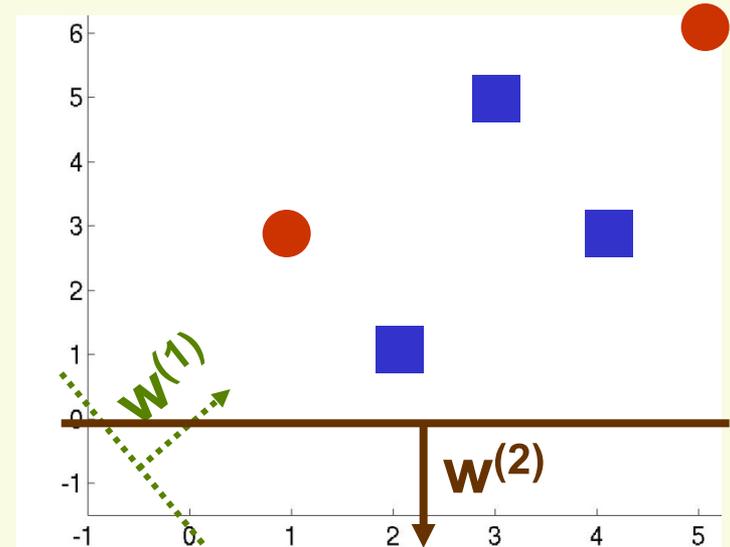

- $\mathbf{a^t z}^4 = [1 \ 1 \ 1] \cdot [-1 \ -1 \ -3]^t = -5 < 0$

- *Update:* $\mathbf{a^{(2)}} = \mathbf{a^{(1)}} + \mathbf{z_M} = [1 \quad 1 \quad 1] + [-1 \quad -1 \quad -3] = [0 \quad 0 \quad -2]$

- $\mathbf{a^t z}^5 = [0 \ 0 \ -2] \cdot [-1 \ -5 \ -6]^t = 12 > 0$

- $\mathbf{a^t z}^1 = [0 \ 0 \ -2] \cdot [1 \ 2 \ 1]^t < 0$

- *Update:* $\mathbf{a^{(3)}} = \mathbf{a^{(2)}} + \mathbf{z_M} = [0 \quad 0 \quad -2] + [1 \quad 2 \quad 1] = [1 \quad 2 \quad -1]$

Non-Linearly Separable Case

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

- rule is: $\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \mathbf{z}_M$

$$\mathbf{z}^1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad \mathbf{z}^2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad \mathbf{z}^3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad \mathbf{z}^4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad \mathbf{z}^5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$

- $\mathbf{a}^t\mathbf{z}^2 = [1 \ 4 \ 3] \cdot [1 \ 2 \ -1]^t = 6 > 0$

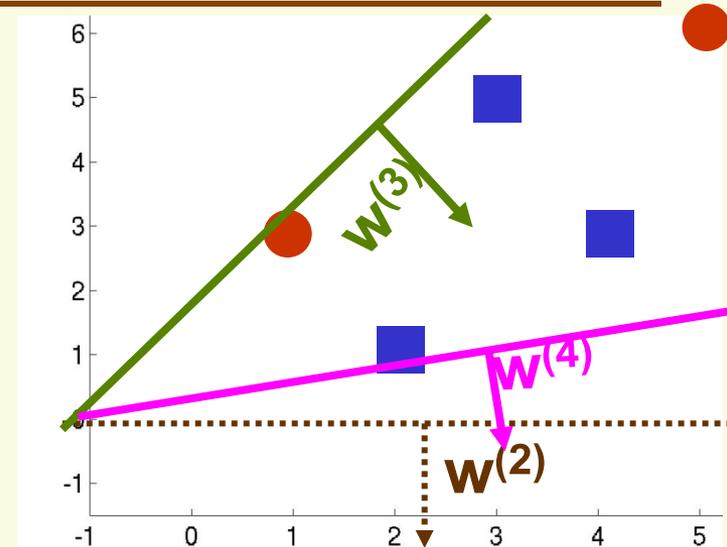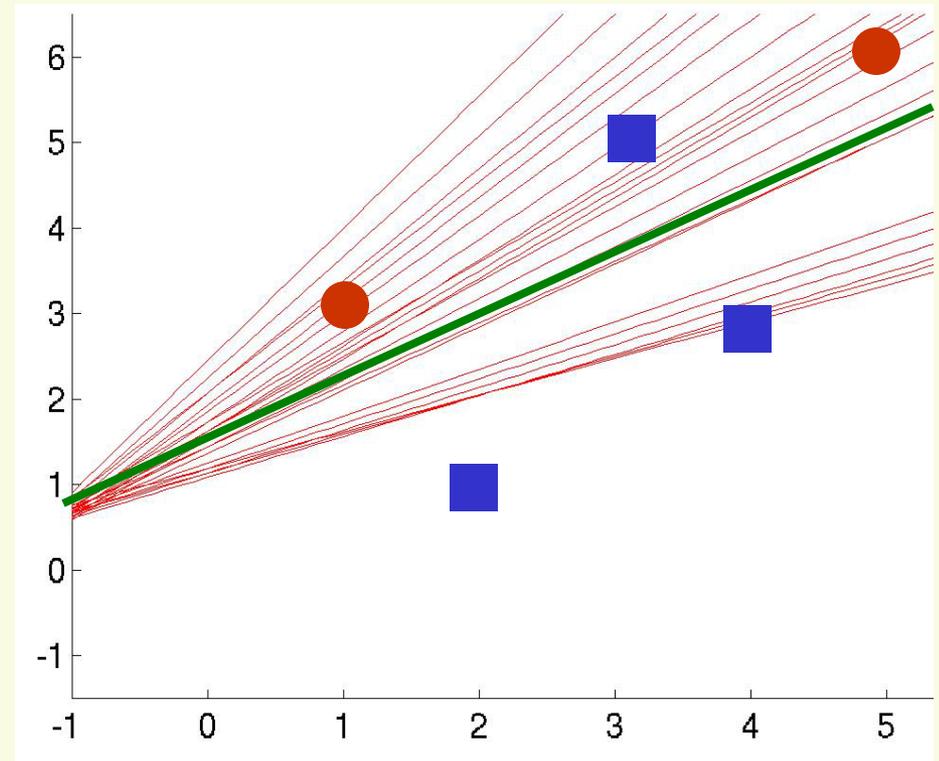- $\mathbf{a}^t\mathbf{z}^3 = [1 \ 3 \ 5] \cdot [1 \ 2 \ -1]^t = 2 > 0$

- $\mathbf{a}^t\mathbf{z}^4 = [-1 \ -1 \ -3] \cdot [1 \ 2 \ -1]^t = 0$

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

Non-Linearly Separable Case

- We can continue this forever
  - there is no solution vector **a** satisfying for all $\mathbf{a}^t\mathbf{z}_i > 0$ for all **i**

- Need to stop at a good point

- Solutions at iterations 900 through 915

- Some are good some are not

- How do we stop at a good solution?

Convergence of Perceptron Rules

1. Classes are linearly separable:

   - with fixed learning rate, both single sample and batch rules converge to a correct solution **a**

   - can be any **a** in the solution space

2. Classes are not linearly separable:

   - with fixed learning rate, both single sample and batch do not converge

   - can ensure convergence with appropriate variable learning rate

     - $\alpha \to 0$ as $k \to \infty$

     - example, inverse linear: $\alpha = c/k$, where **c** is any constant

       - also converges in the linearly separable case

     - no guarantee that we stop at a good point, but there are good reasons to choose inverse linear learning rate

- Practical Issue: both single sample and batch algorithms converge faster if features are roughly on the same scale

  - see kNN lecture on feature normalization

Batch vs. Single Sample Rules

Batch

- True gradient descent, full gradient computed
- Smoother gradient because all samples are used
- Takes longer to converge

Single Sample

- Only partial gradient is computed
- Noisier gradient, therefore may concentrates more than necessary on any isolated training examples (those could be noise)
- Converges faster
- Easier to analyze

Linear Machine: Logistic Regression

- Despite the name, used for classification, not regression

- Instead of putting **g(x)** through a sign function, can put it through a smooth function

  - smooth function is better for gradient descent

- Logistic sigmoid function

- **g(x,w) = w$_0$+x$_1$w$_1$ + … + x$_d$w$_d$**

- let   **f(x,w) = σ(g(x,w))**
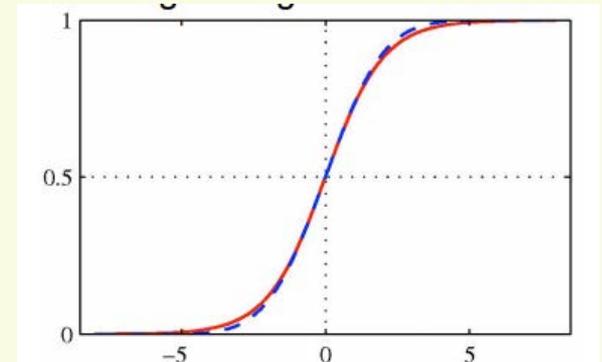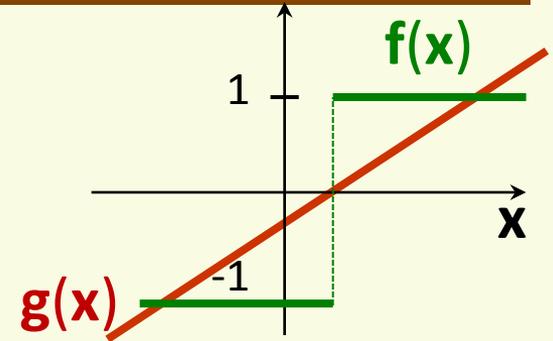
$$\sigma(a) = \frac{1}{1 + exp(-a)}$$

Linear Machine: Logistic Regression

- **f**(**x**,**w**) = σ(**g**(**x**,**w**))
  - bigger 0.5 if **g**(**x**,**w**) is positive
    - decide class 1
  - less 0.5 if **g**(**x**,**w**) is negative
    - decide class 2
- Has an interesting probabilistic interpretation
- P(class 1|**x**) = σ(**g**(**x**,**w**))
- Under a certain loss function, can be optimized exactly with gradient decent
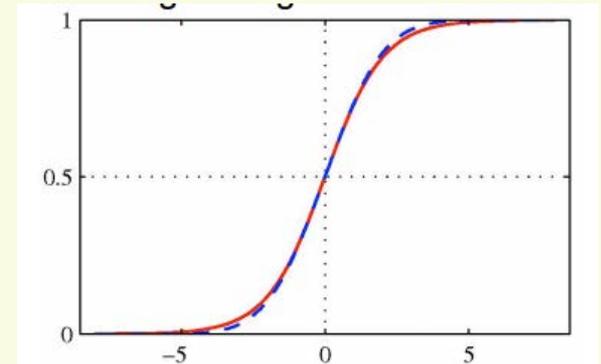
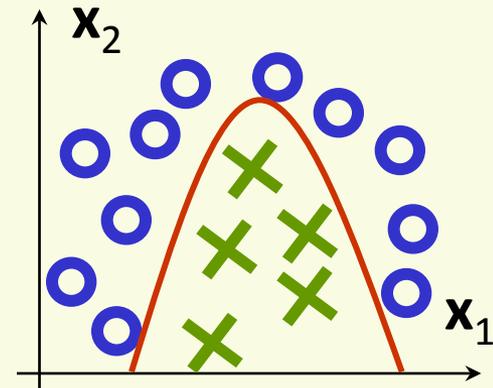$$\sigma(a) = \frac{1}{1 + exp(-a)}$$

Generalized Linear Classifier

- Can use other discriminant functions, like quadratics

  $g(\mathbf{x}) = \mathbf{w}_0 + \mathbf{w}_1\mathbf{x}_1 + \mathbf{w}_2\mathbf{x}_2 + \mathbf{w}_{12}\mathbf{x}_1\mathbf{x}_2 + \mathbf{w}_{11}\mathbf{x}_1^2 + \mathbf{w}_{22}\mathbf{x}_2^2$

- Methodology is almost the same as in the linear case:

  - $\mathbf{f}(\mathbf{x}) = \text{sign}(\mathbf{w}_0 + \mathbf{w}_1\mathbf{x}_1 + \mathbf{w}_2\mathbf{x}_2 + \mathbf{w}_{12}\mathbf{x}_1\mathbf{x}_2 + \mathbf{w}_{11}\mathbf{x}_1^2 + \mathbf{w}_{22}\mathbf{x}_2^2)$

  - $\mathbf{z} = [\ 1 \quad \mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_1\mathbf{x}_2 \quad \mathbf{x}_1^2 \quad \mathbf{x}_2^2]$

  - $\mathbf{a} = [\ \mathbf{w}_0 \quad \mathbf{w}_1 \quad \mathbf{w}_2 \quad \mathbf{w}_{12} \quad \mathbf{w}_{11} \quad \mathbf{w}_{22}]$

  - "normalization": multiply negative class samples by -1

  - all the other procedures remain the same, i.e. gradient descent to minimize Perceptron loss function, or MSE procedure, etc.

Generalized Linear Classifier

- In general, to the liner function:

$$g(\mathbf{x},\mathbf{w}) = \mathbf{w}_0 + \sum_{i=1\ldots d} \mathbf{w}_i \mathbf{x}_i$$

- can add quadratic terms:

$$g(\mathbf{x},\mathbf{w}) = \mathbf{w}_0 + \sum_{i=1\ldots d} \mathbf{w}_i \mathbf{x}_i + \sum_{i=1\ldots d} \sum_{j=1,\ldots d} \mathbf{w}_{ij} \mathbf{x}_i \mathbf{x}_j$$

- This is still a linear function in its parameters $\mathbf{w}$

- $g(\mathbf{y},\mathbf{v}) = \mathbf{v}_0 + \mathbf{v}^t \mathbf{y}$

$$\mathbf{v}_0 = \mathbf{w}_0$$

$$\mathbf{y} = [\mathbf{x}_1 \quad \mathbf{x}_2 \ldots \quad \mathbf{x}_d \quad \mathbf{x}_1\mathbf{x}_1 \quad \mathbf{x}_1\mathbf{x}_2 \quad \ldots \quad \mathbf{x}_d\mathbf{x}_d]$$

$$\mathbf{v} = [\mathbf{w}_1 \quad \mathbf{w}_2 \ldots \mathbf{w}_d \quad \mathbf{w}_{11} \quad \mathbf{w}_{12} \quad \ldots \quad \mathbf{w}_{dd}]$$

- Can use all the same training methods as before

Generalized Linear Classifier

- Generalized linear classifier

$$g(\mathbf{x},\mathbf{w}) = \mathbf{w}_0 + \Sigma_{i=1\ldots m} \, \mathbf{w}_i \mathbf{h}_i(\mathbf{x})$$

- $\mathbf{h}(\mathbf{x})$ are called basis function, can be arbitrary functions

    - in strictly linear case, $\mathbf{h}_i(\mathbf{x}) = \mathbf{x}_i$

- Linear function in its parameters $\mathbf{w}$

$$g(\mathbf{x},\mathbf{w}) = \mathbf{w}_0 + \mathbf{w}^t\mathbf{h}$$

$$\mathbf{h} = [\mathbf{h}_1(\mathbf{x}) \ \ \mathbf{h}_2(\mathbf{x}) \ \ \ldots \ \ \mathbf{h}_m(\mathbf{x})]$$

$$[\mathbf{w}_1 \ \ \ldots \ \ \mathbf{w}_m]$$

- Can use all the same training methods as before

Generalized Linear Classifier

- Usually face severe overfitting
  - too many degrees of freedom
  - Boundary can "curve" to fit to the noise in the data
- Helps to regularize by keeping **w** small
  - small **w** means the boundary is not as curvy
- Usually add $\lambda\|\mathbf{w}\|^2$ to the loss function
- Recall quadratic loss function

$$\mathbf{L}(\mathbf{x}^i,\mathbf{y}^i,\mathbf{w}) = \|\, \mathbf{f}(\mathbf{x}^i,\mathbf{w}) - \mathbf{y}^i\, \|^2$$

- Regularized version

$$\mathbf{L}(\mathbf{x}^i,\mathbf{y}^i,\mathbf{w}) = \|\, \mathbf{f}(\mathbf{x}^i,\mathbf{w}) - \mathbf{y}^i\, \|^2 + \lambda\|\mathbf{w}\|^2$$

- How to set λ?
- With cross-validation

Learning by Gradient Descent

- Can have classifiers even more general
- More general than generalized linear ☺
- Suppose we suspect that the machine has to have functional form **f**(**x**,**w**), not necessarily linear
- Pick differentiable per-sample loss function **L**(**x**^{i},**y**^{i},**w**)
- Need to find **w** that minimizes **L** = Σ_i **L**(**x**^{i},**y**^{i},**w**)
- Use gradient-based minimization:
  - Batch rule: **w** = **w** - $\alpha \nabla$**L**(**w**)
  - Or single sample rule: W = W - $\alpha \nabla$**L** (**x**^{i},**y**^{i},**w**)