CS9840
Learning and Computer Vision
Prof. Olga Veksler

Lecture 7
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

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

- 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

$$\mathbf{J}(\mathbf{x}) = \mathbf{J}(\mathbf{x}_1, \dots, \mathbf{x}_d)$$

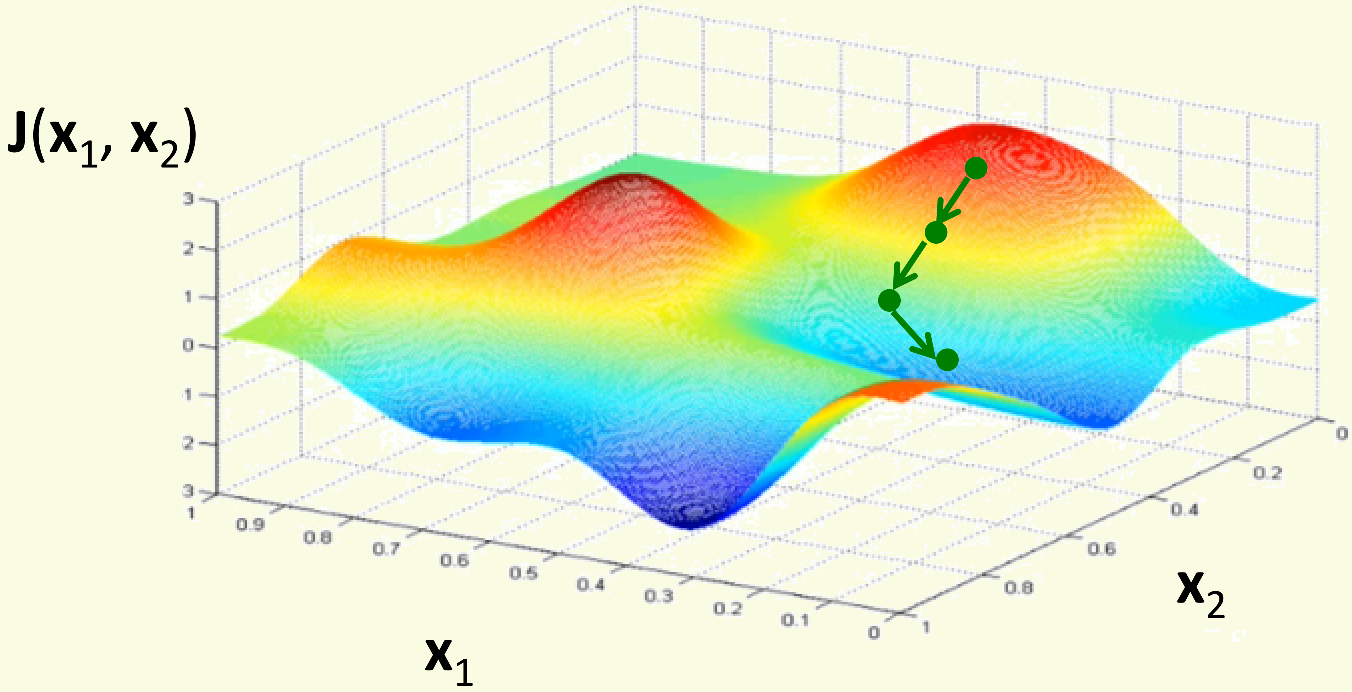
- From calculus, take partial derivatives, set them to 0

gradient

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

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

Optimization: Gradient Direction

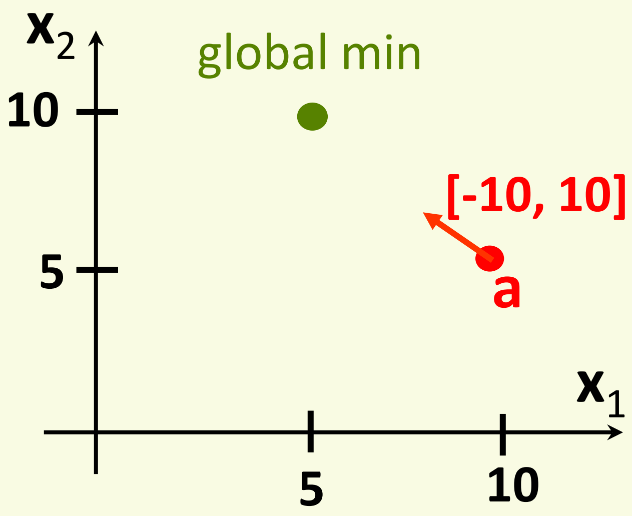


- Gradient $\nabla \mathbf{J}(\mathbf{x})$ points in the direction of steepest increase of function $\mathbf{J}(\mathbf{x})$
- $-\nabla \mathbf{J}(\mathbf{x})$ points in the direction of steepest decrease

Picture from Andrew Ng

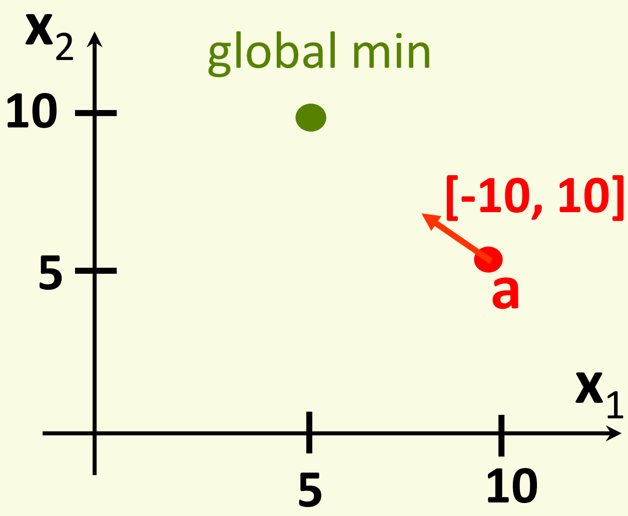
Gradient Direction in 2D

- $J(x_1, x_2) = (x_1-5)^2+(x_2-10)^2$
- $\frac{\partial}{\partial x_1} J(x) = 2(x_1 - 5)$
- $\frac{\partial}{\partial x_2} J(x) = 2(x_2 - 10)$
- Let **a** = [10, 5]
- $-\frac{\partial}{\partial x_1} J(a) = -10$
- $-\frac{\partial}{\partial x_2} J(a) = 10$



Gradient Descent: Step Size

- $J(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 - 5)^2 + (\mathbf{x}_2 - 10)^2$
- Which step size to take?
- Controlled by parameter α
  - called **learning rate**
- From previous example:
  - $\mathbf{a} = [10 \quad 5]$
  - $-\nabla J(\mathbf{a}) = [-10 \quad 10]$
- Let $\alpha = 0.2$
- $\mathbf{a} - \alpha \nabla J(\mathbf{a}) = [10 \quad 5] + 0.2\,[-10 \quad 10] = [8 \quad 7]$
- $J(10, 5) = 50$
- $J(8,7) = 18$



Gradient Descent Algorithm

k = 1

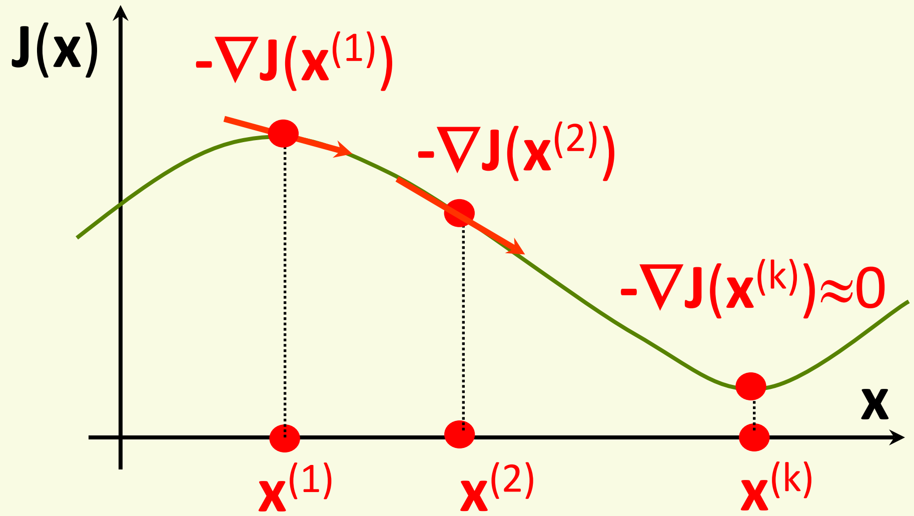
$\mathbf{x}^{(1)}$ = any initial guess

choose α, ε

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

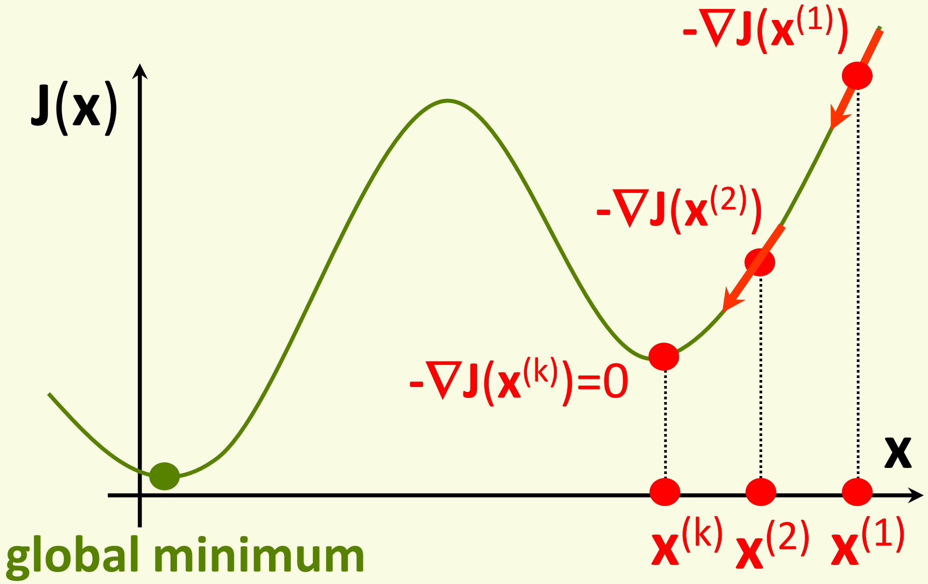
$\quad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha \nabla J(\mathbf{x}^{(k)})$

$\quad \mathbf{k} = \mathbf{k} + 1$



Gradient Descent: Local Minimum

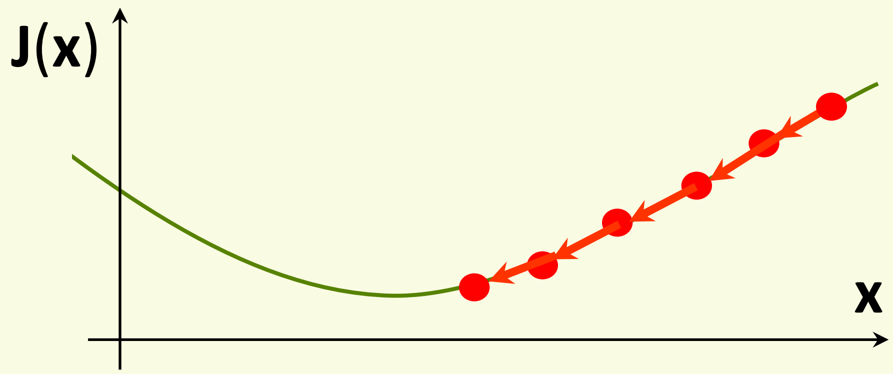
- Not guaranteed to find global minimum
  - gets stuck in local minimum



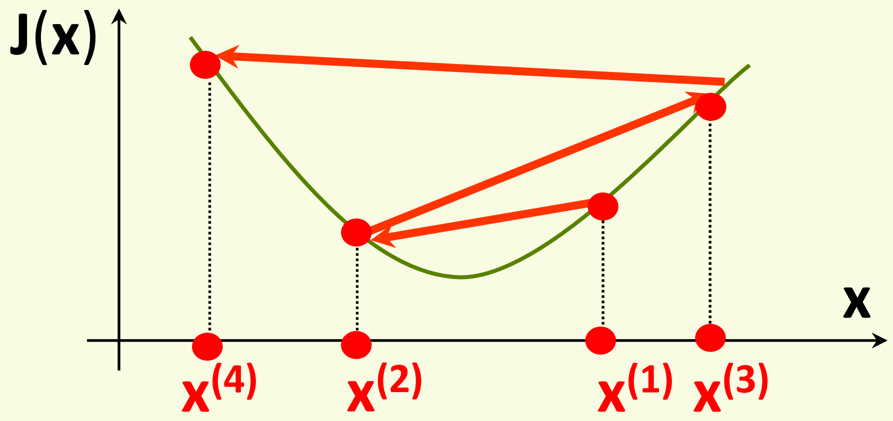
- Still gradient descent is very popular because it is simple and applicable to any differentiable function

How to Set Learning Rate α?

- If α too small, too many iterations to converge



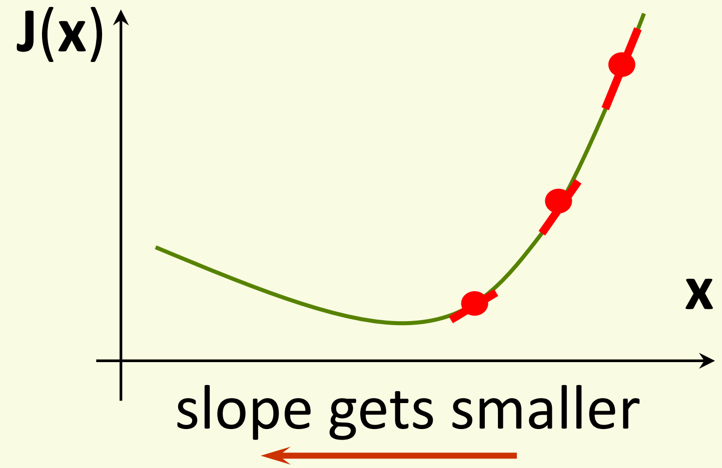
- If α too large, may overshoot the local minimum and possibly never even converge



- 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



Variable Learning Rate

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

$\mathbf{k} = 1$

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

choose α, ε

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

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

$\quad \mathbf{k} = \mathbf{k} + 1$

→

$\mathbf{k} = 1$

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

choose ε

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

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

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

$\quad \mathbf{k} = \mathbf{k} + 1$

Variable Learning Rate

- Usually don't keep track of all intermediate solutions

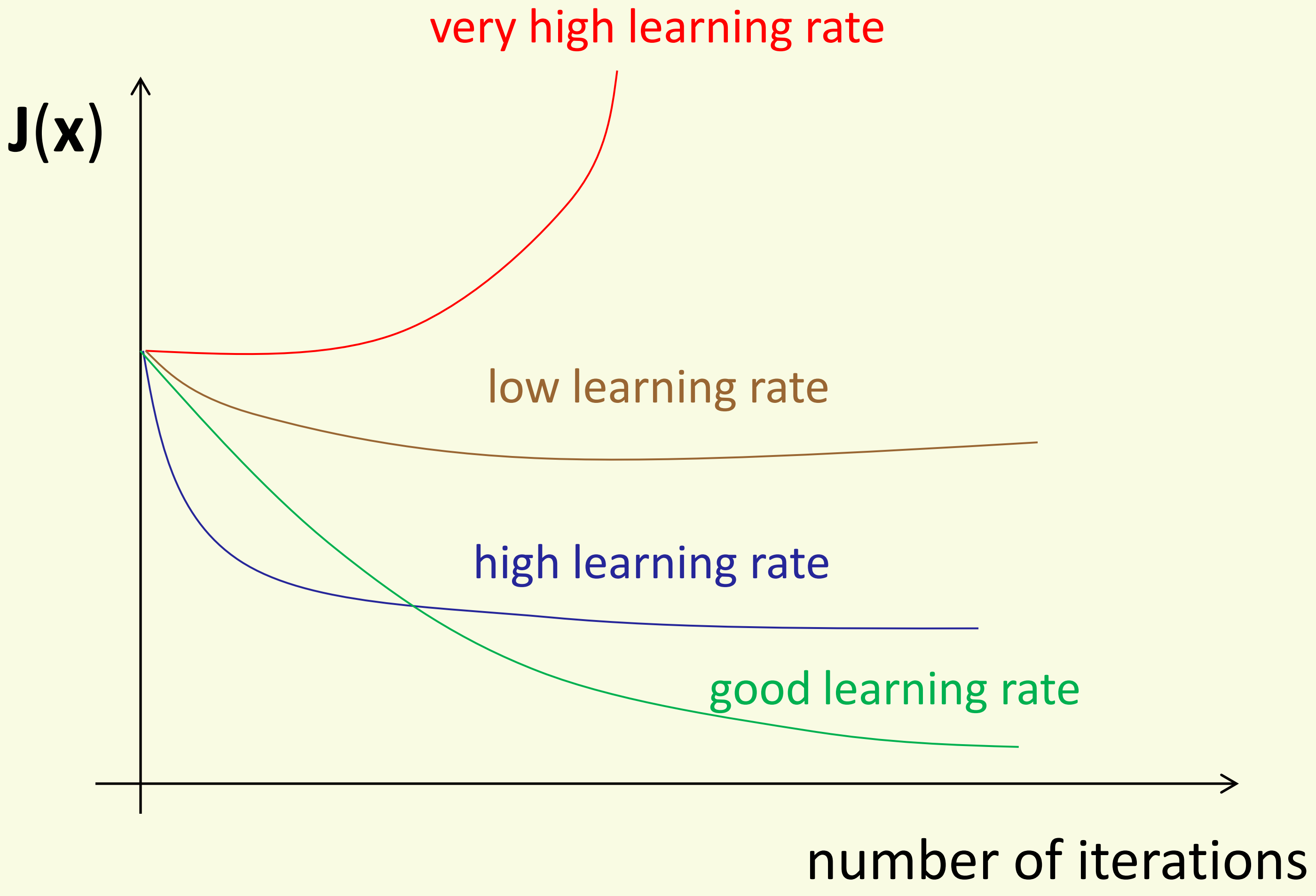
k = 1
$\mathbf{x}^{(1)}$ = any initial guess
choose α, ε
while $\alpha\|\nabla \mathbf{J}(\mathbf{x}^{(k)})\| > \varepsilon$
    $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha \nabla \mathbf{J}(\mathbf{x}^{(k)})$
    k = **k** + 1

→

x = any initial guess
choose α, ε
while $\alpha\|\nabla \mathbf{J}(\mathbf{x})\| > \varepsilon$
    $\mathbf{x} = \mathbf{x} - \alpha \nabla \mathbf{J}(\mathbf{x})$

Learning Rate

- Monitor learning rate by looking at how fast the objective function decreases



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

Supervised Learning Review

- 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 $\mathbf{y}$

Supervised Learning Review

- Wish to design a *machine* **f**(**x**,**w**) s.t.

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

  - How do we choose **f**?
    - last time studied kNN classifier
    - this lecture in on liner classifier
    - many other choices
  - **w** is multidimensional vector of weights (also called *parameters*)

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

  - By modifying **w**, the machine "learns"

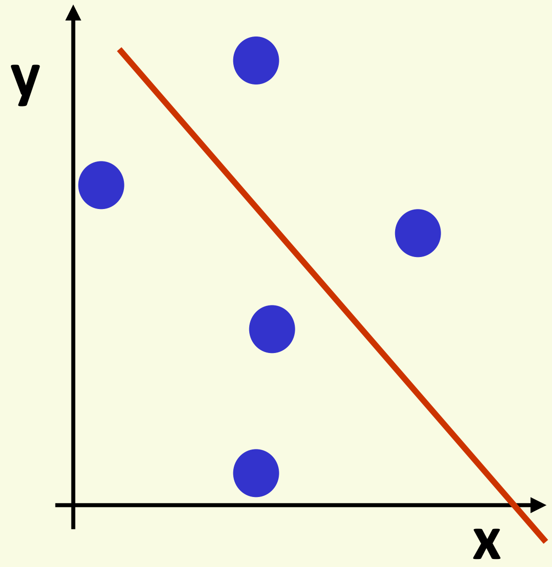
Training and Testing Phases

- Divide all labeled samples $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^n$ into *training* and *test* sets
- Training phase
  - Uses training samples
  - goal is to "teach" the machine
  - find weights **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 "$\mathbf{f}(\mathbf{x}^i,\mathbf{w}) = \mathbf{y}^i$ as much as possible"?
- $\mathbf{f}(\mathbf{x},\mathbf{w})$ has to be "close" to the true output $\mathbf{y}$
- Define Loss (or Error, or Criterion) function $\mathbf{L}$
- First define per-sample loss $\mathbf{L}(\mathbf{x}^i,\mathbf{y}^i,\mathbf{w})$
- Examples of loss function
  - for classification, $\mathbf{L}(\mathbf{x}^i,\mathbf{y}^i,\mathbf{w}) = \mathbf{I}[\mathbf{f}(\mathbf{x}^i,\mathbf{w}) \neq \mathbf{y}^i]$
    - $\mathbf{I}$[true] = 1, $\mathbf{I}$[false] = 0
  - for regression, $\mathbf{L}(\mathbf{x}^i,\mathbf{y}^i,\mathbf{w}) = \| \mathbf{f}(\mathbf{x}^i,\mathbf{w}) - \mathbf{y}^i \|^2$,
    - how far is the estimated output from the correct one?
- Then loss function $\mathbf{L} = \sum_i \mathbf{L}(\mathbf{x}^i,\mathbf{y}^i,\mathbf{w})$
  - classification: counts number of misclassified examples
  - regression: sums distances to the correct output

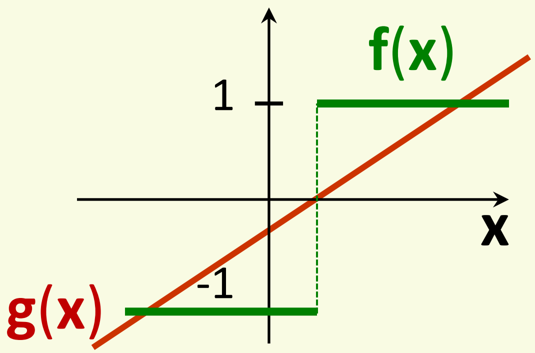
Linear Machine: Regression



- $\mathbf{f}(\mathbf{x},\mathbf{w}) = \mathbf{w}_0 + \sum_{i=1,2,\ldots d} \mathbf{w}_i \mathbf{x}_i$
- In vector notation
  - $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d]$
  - $\mathbf{f}(\mathbf{x},\mathbf{w}) = \mathbf{w}_0 + \mathbf{w}^t \mathbf{x}$
- This is standard linear regression
  - line fitting
  - assume $\mathbf{L}(\mathbf{x}^i,\mathbf{y}^i,\mathbf{w}) = ||\mathbf{f}(\mathbf{x}^i,\mathbf{w}) - \mathbf{y}^i||^2$
- optimal **w** can be found by solving a system of linear equations

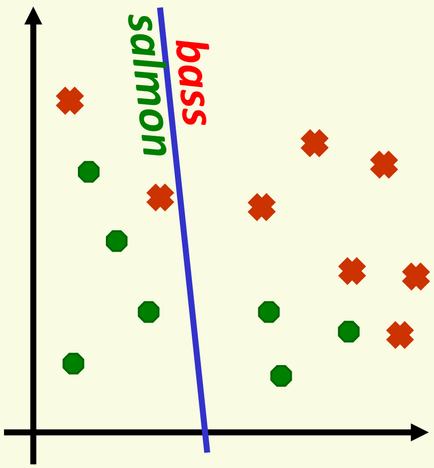
$$\mathbf{w}^* = [\Sigma \mathbf{x}^i (\mathbf{x}^i)^T]^{-1} \Sigma \mathbf{y}^i \mathbf{x}^i$$

Linear Machine: Classification

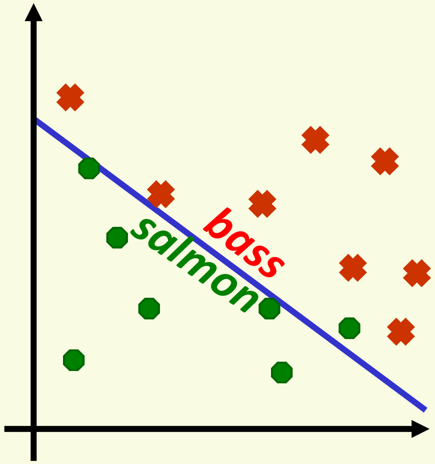


- First consider the two-class case
- Choose encoding
  - $\mathbf{y} = 1$ for the first class
  - $\mathbf{y} = -1$ for the second class
- Linear classifier
  - $-\infty \leq \mathbf{w}_0 + \mathbf{x}_1\mathbf{w}_1 + \ldots + \mathbf{x}_d\mathbf{w}_d \leq \infty$
  - we need $\mathbf{f}(\mathbf{x},\mathbf{w})$ to be either +1 or -1
  - let $\mathbf{g}(\mathbf{x},\mathbf{w}) = \mathbf{w}_0 + \mathbf{x}_1\mathbf{w}_1 + \ldots + \mathbf{x}_d\mathbf{w}_d = \mathbf{w}_0 + \mathbf{w}^t\mathbf{x}$
  - let $\mathbf{f}(\mathbf{x},\mathbf{w}) = \text{sign}(\mathbf{g}(\mathbf{x},\mathbf{w}))$
    - 1 if $\mathbf{g}(\mathbf{x},\mathbf{w})$ is positive
    - -1 if $\mathbf{g}(\mathbf{x},\mathbf{w})$ is negative
    - other choices for $\mathbf{g}(\mathbf{x},\mathbf{w})$ are also used
  - $\mathbf{g}(\mathbf{x},\mathbf{w})$ is called the **discriminant function**

Linear Classifier: Decision Boundary



bad boundary



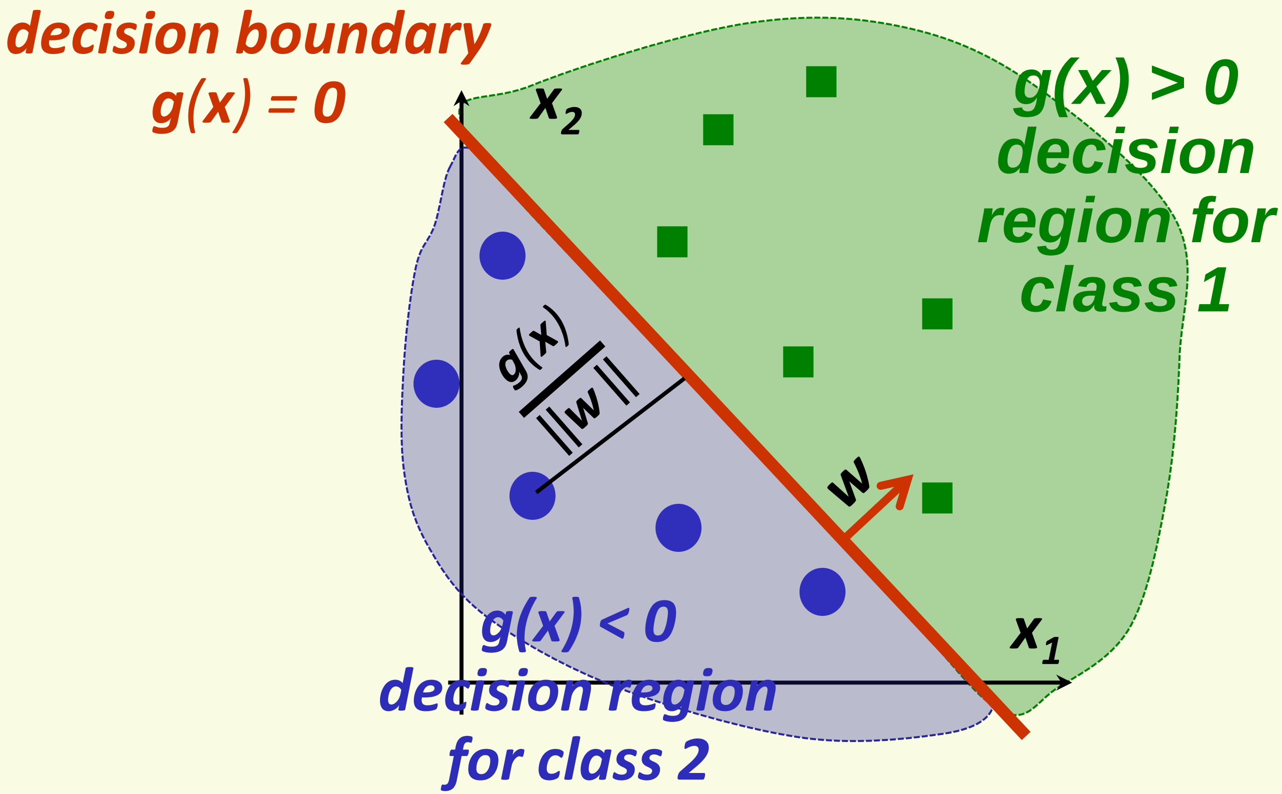
better boundary

- $\mathbf{f}(\mathbf{x},\mathbf{w}) = \text{sign}(\mathbf{g}(\mathbf{x},\mathbf{w})) = \text{sign}(\mathbf{w}_0+\mathbf{x}_1\mathbf{w}_1 + \ldots + \mathbf{x}_d\mathbf{w}_d)$
- 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: $\mathbf{g}(\mathbf{x},\mathbf{w}) = \mathbf{w}_0+\mathbf{x}_1\mathbf{w}_1 + \ldots + \mathbf{x}_d\mathbf{w}_d$
- Written using vector notation $\mathbf{g}(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + \mathbf{w}_0$

weight vector *bias* or *threshold*



More on Linear Discriminant Function (LDF)

- Decision boundary: $\mathbf{g}(\mathbf{x},\mathbf{w}) = \mathbf{w}_0+\mathbf{x}_1\mathbf{w}_1 + \ldots + \mathbf{x}_d\mathbf{w}_d = 0$
- 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

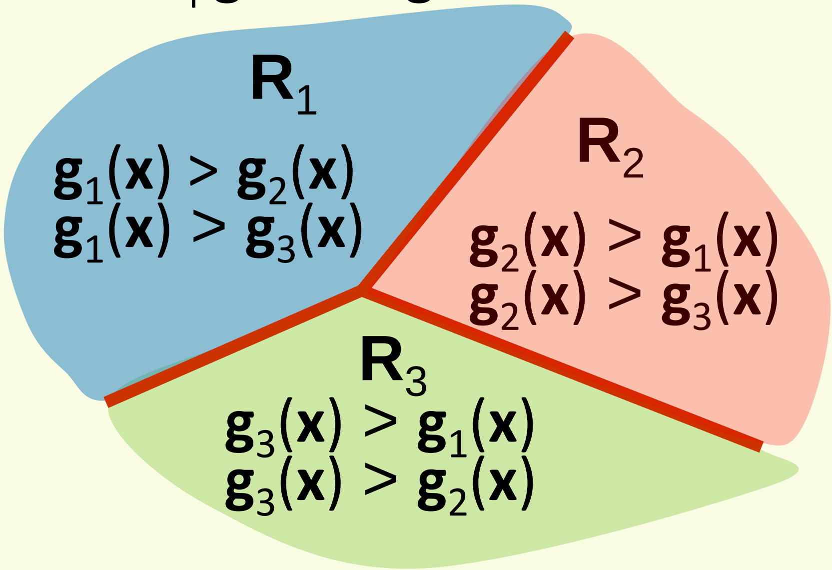
- Have **m** classes
- Define **m** linear discriminant functions

$$\mathbf{g}_i(\mathbf{x}) = \mathbf{w}_i^t\mathbf{x} + \mathbf{w}_{i0} \quad \text{for } \mathbf{i} = 1, 2, \ldots \mathbf{m}$$

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

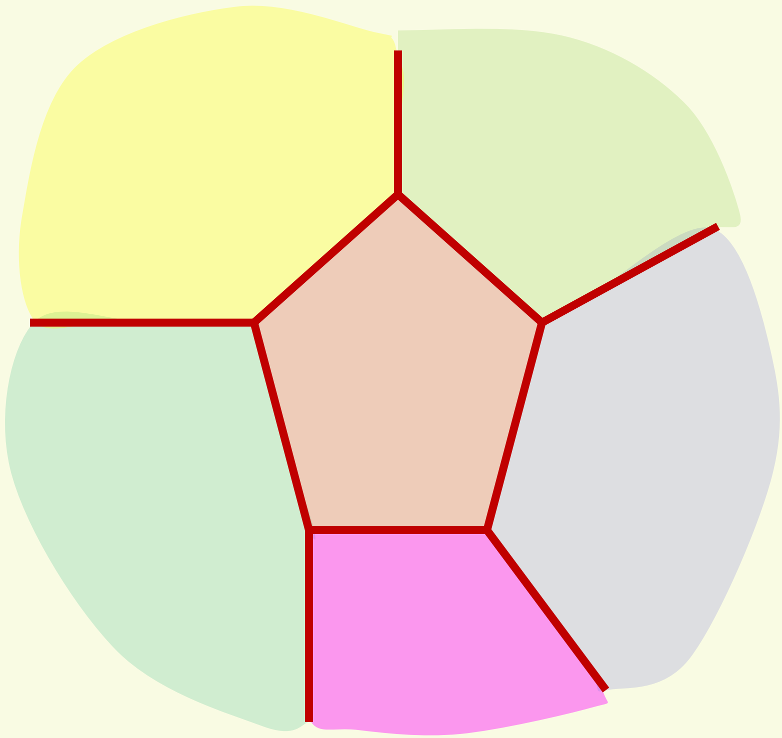
$$\mathbf{g}_i(\mathbf{x}) > \mathbf{g}_j(\mathbf{x}) \text{ for all } \mathbf{j} \neq \mathbf{i}$$

- Let $\mathbf{R}_i$ be the decision region for class **i**
  - all examples in $\mathbf{R}_i$ get assigned class **i**



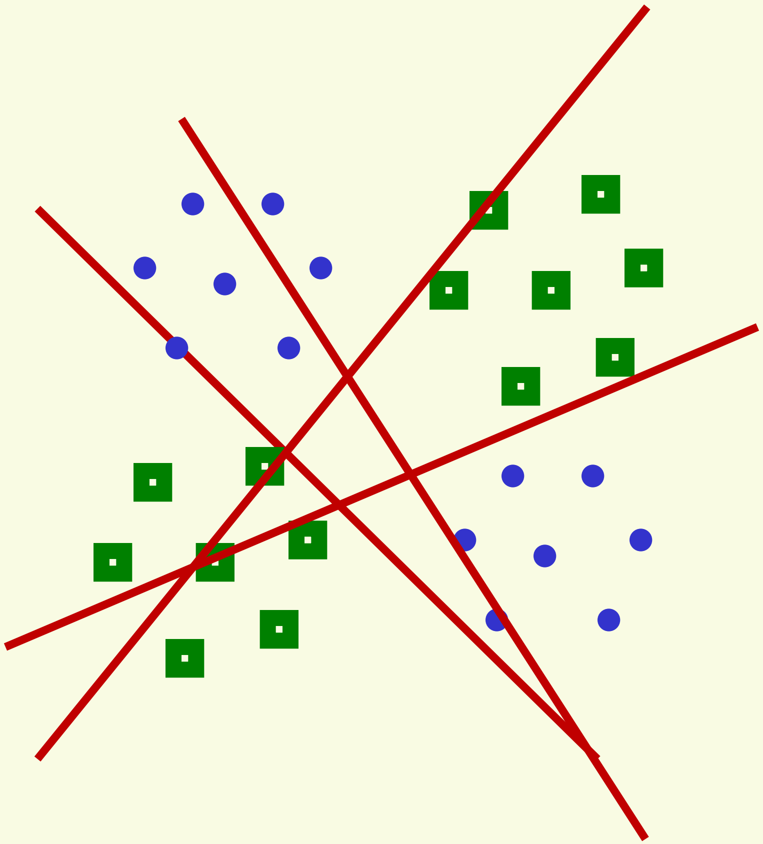
Multiple Classes

- 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

- Give simple decision boundary
  - try simpler models first
  - can still overfit in very high dimensions
- 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 $\mathbf{g(x) = w^t x + w_0}$
- Can rewrite it $\mathbf{g(x)} = [\mathbf{w}_0 \quad \mathbf{w}^t] \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix} = \mathbf{a^t z} = \mathbf{g(z)}$

  new weight vector ***a***

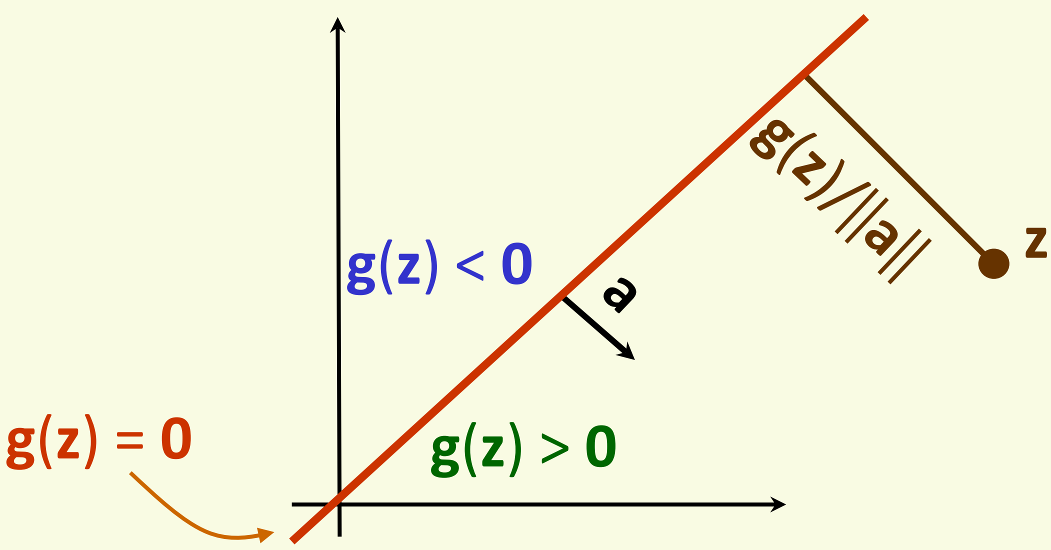
  new feature vector ***z***

- **z** is called augmented feature vector
- new problem equivalent to the old $\mathbf{g(z) = a^t z}$

$$\mathbf{a} = \begin{bmatrix} \mathbf{w}_0 \\ \mathbf{w}_1 \\ \vdots \\ \mathbf{w_d} \end{bmatrix} \qquad \mathbf{z} = \begin{bmatrix} 1 \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x_d} \end{bmatrix}$$

Augmented Feature Vector

- Feature augmenting is done to simplify notation
- The rest of this lecture assumes augmented features
  - given samples $\mathbf{x}^1,\dots,\mathbf{x}^n$ convert them to augmented samples $\mathbf{z}^1,\dots,\mathbf{z}^n$ by adding a new dimension of value 1
- $\mathbf{g(z) = a^t z}$



Training Error

- Assume we have 2 classes
- Samples $\mathbf{z}^1,\dots,\mathbf{z}^n$ some in class 1, some in class 2
- Use samples to determine weights $\mathbf{a}$ in $\mathbf{g}(\mathbf{z}) = \mathbf{a}^t\mathbf{z}$
- Want to minimize number of misclassified samples

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

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

Simplifying Notation Further

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

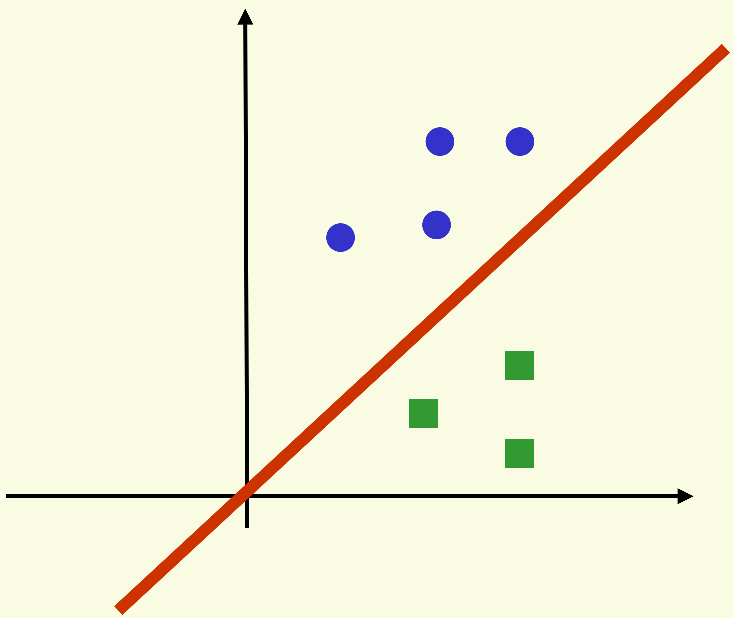
- Equivalently, training error is 0 if $\begin{cases} \mathbf{a}^t\mathbf{z}^i > 0 & \forall \mathbf{z}^i \text{ class } 1 \\ \mathbf{a}^t(-\mathbf{z}^i) > 0 & \forall \mathbf{z}^i \text{ class } 2 \end{cases}$

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

- If exists, such **a** is called a ***separating*** or ***solution*** vector
- Original samples $\mathbf{x}^1, \ldots \mathbf{x}^n$ can also be linearly separated

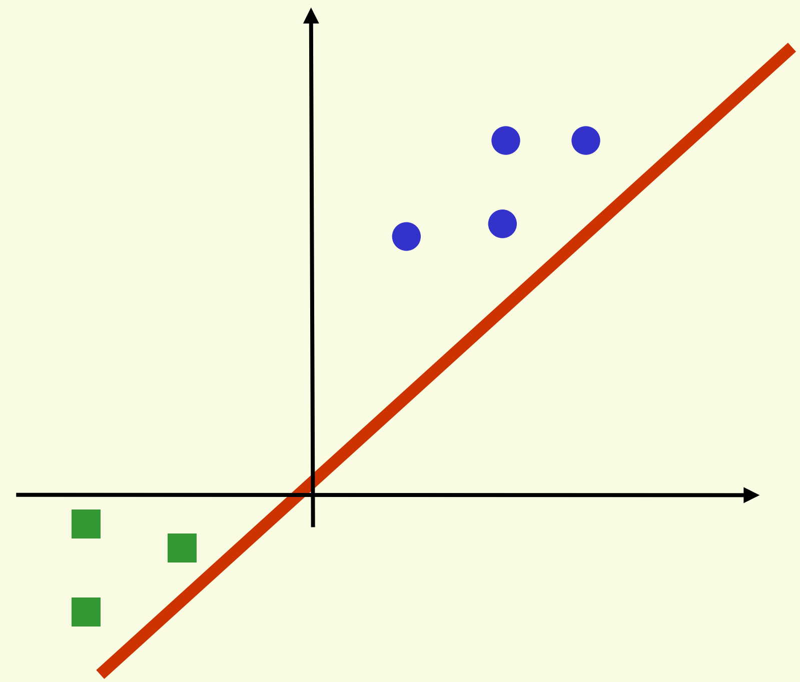
Effect of Normalization

before normalization



seek a hyperplane that separates samples from different categories

after normalization

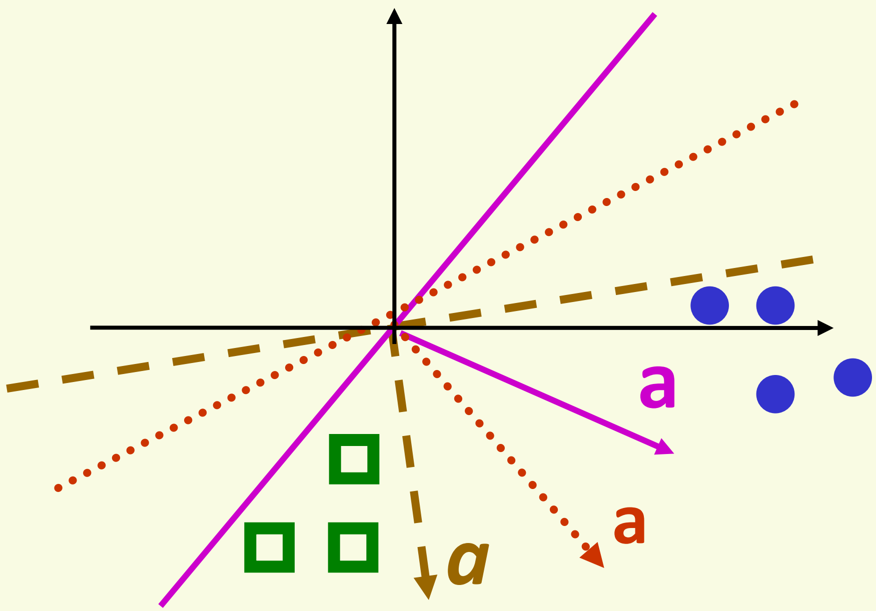


seek hyperplane that puts normalized samples on the same (positive) side

Solution Region

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

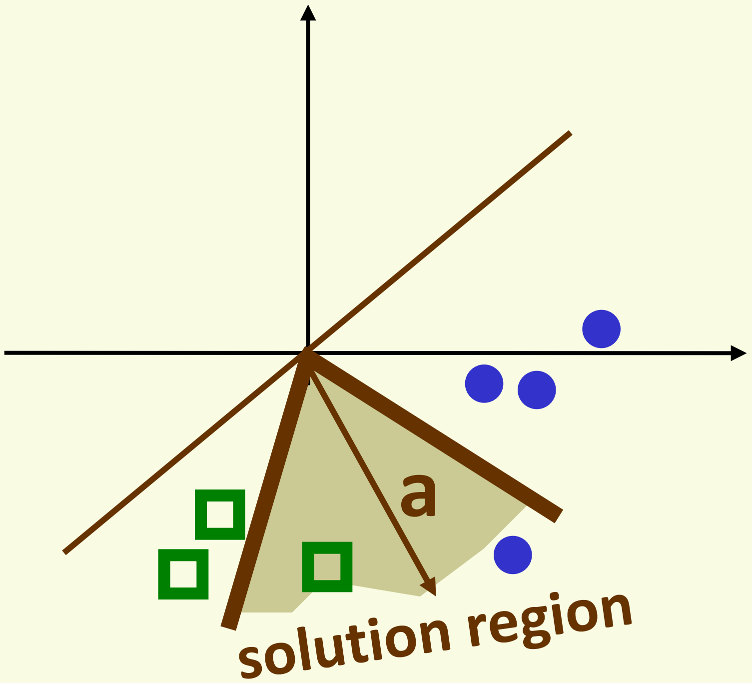
$$\mathbf{a}^t\mathbf{z}^i = \sum_{k=0}^{d} \mathbf{a}_k \mathbf{z}_d^i > \mathbf{0}$$



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

Solution Region

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



Design a Loss Function

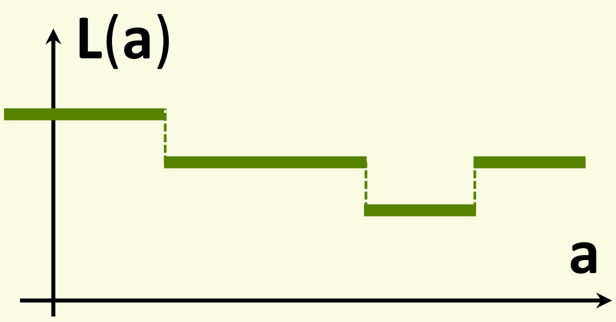
- Find weight vector **a** s.t. $\forall \mathbf{z}^1, \ldots, \mathbf{z}^n$, $\mathbf{a}^t \mathbf{z}^i > 0$
- Design a loss function **L**(**a**), which is minimum when **a** is a solution vector
- Let **Z**(**a**) be the set of examples misclassified by **a**

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

- Natural choice: number of misclassified examples

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

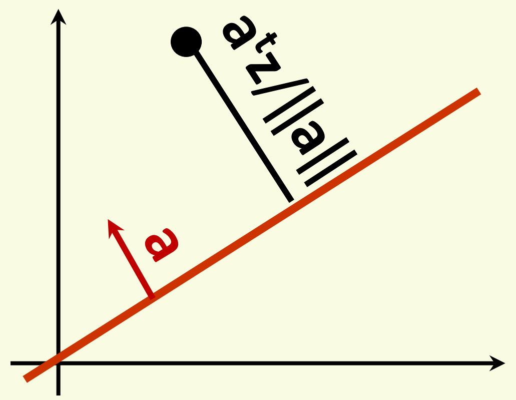
- Unfortunately, cannot minimize with gradient descent
  - piecewise constant, gradient zero or does not exist



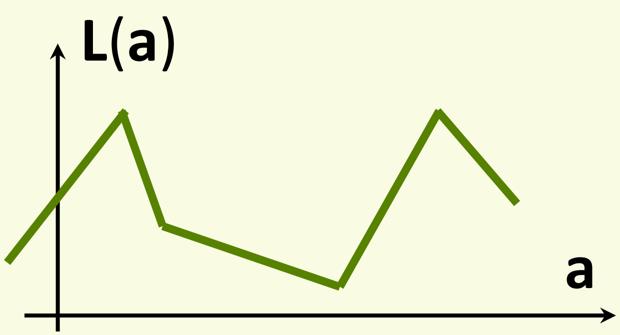
Perceptron Loss Function

- Better choice: Perceptron loss function

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



- If $\mathbf{z}$ is misclassified, $\mathbf{a}^t\mathbf{z} < 0$
- Thus $\mathbf{L}(\mathbf{a}) \geq 0$
- $\mathbf{L}_p(\mathbf{a})$ is proportional to the sum of distances of misclassified examples to decision boundary



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

Optimizing with Gradient Descent

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

- Gradient of $\mathbf{L}_p(\mathbf{a})$ is $\nabla \mathbf{L_p}(\mathbf{a}) = \sum_{\mathbf{z} \in \mathbf{Z}(\mathbf{a})} \left(-\mathbf{z}\right)$
  - cannot solve $\nabla \mathbf{L}_p(\mathbf{a}) = 0$ analytically because of $\mathbf{Z}(\mathbf{a})$
- Recall update rule for gradient descent

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

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

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

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

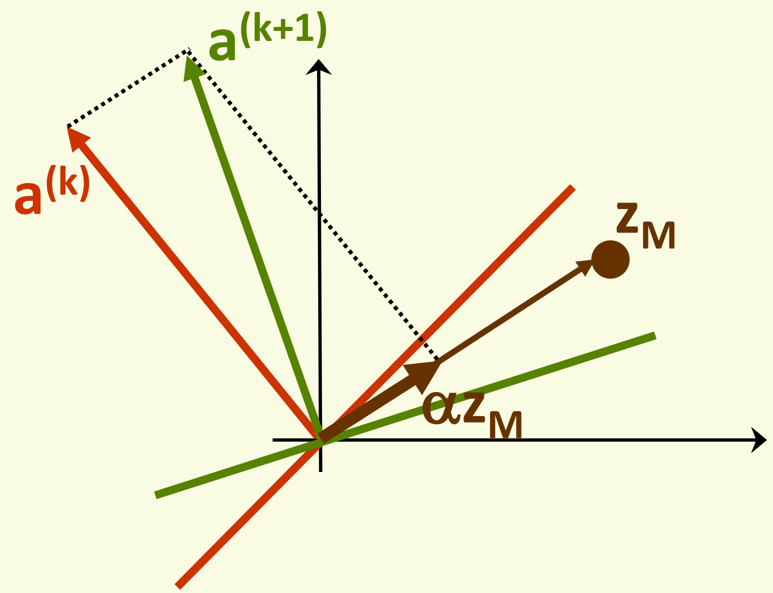
Perceptron Single Sample Rule

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



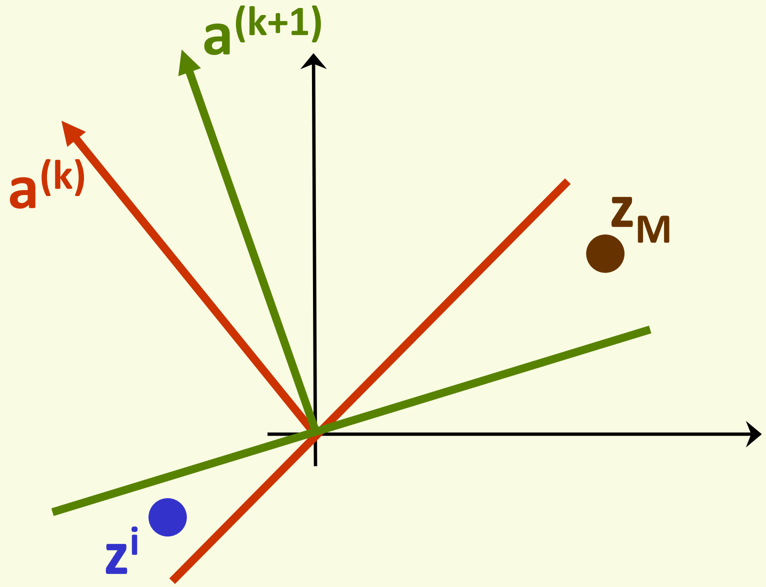
- $\mathbf{z}_M$ misclassified by $\mathbf{a}^{(k)}$

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

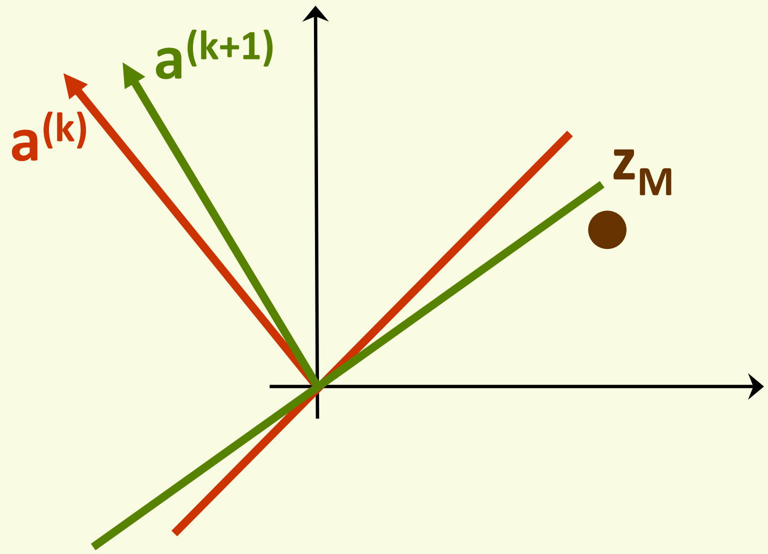
- $\mathbf{z}_M$ is on the wrong side of decision boundary
- adding $\alpha \cdot \mathbf{z}_M$ to a moves decision boundary in the right direction

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



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 $\mathbf{k} \to \infty$
     - example, inverse linear: $\alpha = \mathbf{c}/\mathbf{k}$, 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

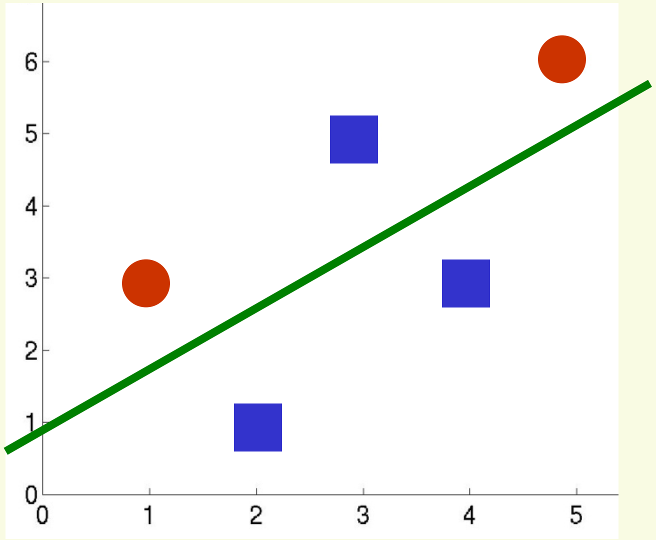
- True gradient descent, full gradient computed
- Smoother gradient because all samples are used
- Takes longer to converge

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

Non-Linearly Separable Case

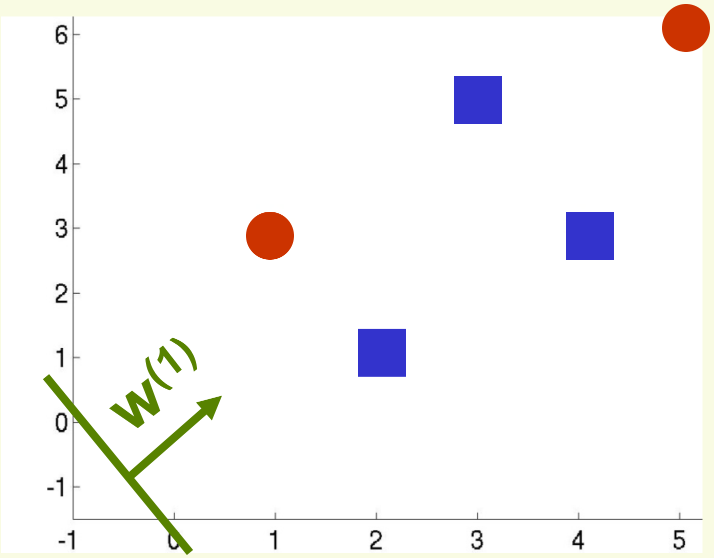
- Suppose we have examples:
  - class 1: [2,1], [4,3], [3,5]
  - class 2: [1,3] , [5,6]
  - not linearly separable
- Still wish an approximate separation
- Good line choice is shown in green
- Let us run gradient descent
  - Add extra feature and “normalize”



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

Non-Linearly Separable Case

- single sample perceptron rule
- Initial weights $\mathbf{a}^{(1)} = [1\ \ 1\ \ 1]$
- This is line $\mathbf{x}_1 + \mathbf{x}_2 + 1 = 0$
- Use fixed learning rate $\alpha = 1$
- Rule is: $\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \mathbf{z}_M$



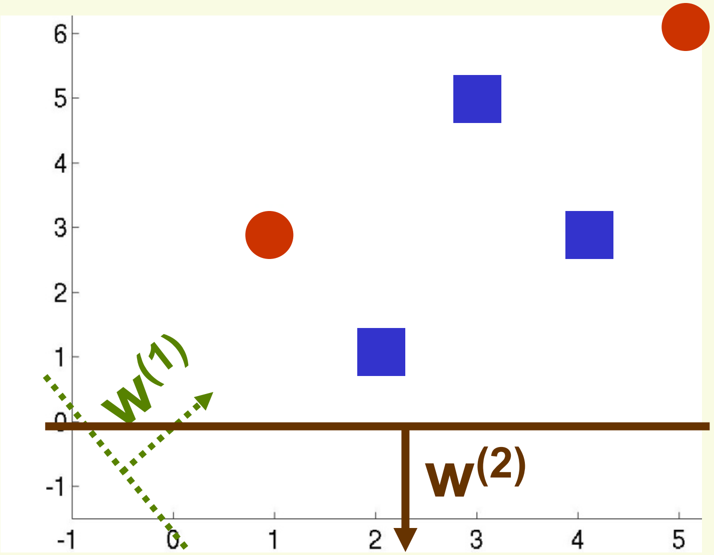
$$\mathbf{z}^1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad \mathbf{z}^2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad \mathbf{z}^3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad \mathbf{z}^4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad \mathbf{z}^5 = \begin{bmatrix} -1 \\ -5 \\ -6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

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

Non-Linearly Separable Case

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

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



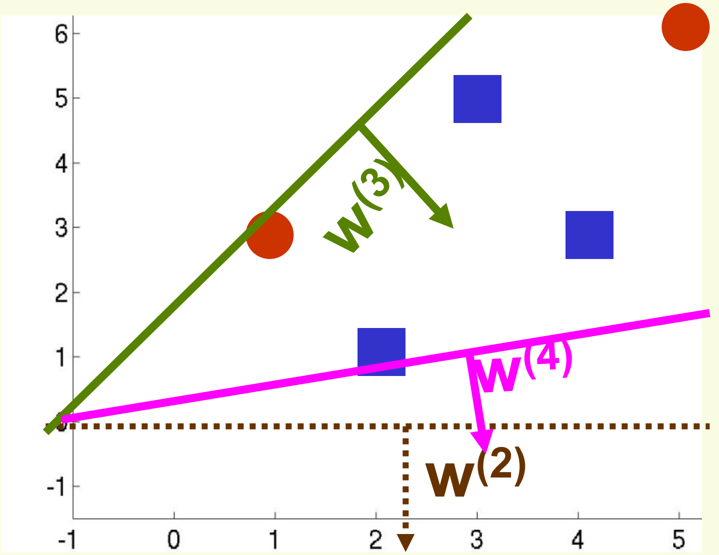
- $\mathbf{a}^t\mathbf{z}^4 = [1\;1\;1] \cdot [-1\;-1\;-3]^t = -5 < 0$
- *Update:* $\mathbf{a}^{(2)} = \mathbf{a}^{(1)} + \mathbf{z}_M = [1 \;\; 1 \;\; 1] + [-1 \;\; -1 \;\; -3] = [0 \;\; 0 \;\; -2]$

- $\mathbf{a}^t\mathbf{z}^5 = [0\;0\;-2] \cdot [-1\;-5\;-6]^t = 12 > 0$
- $\mathbf{a}^t\mathbf{z}^1 = [0\;0\;-2] \cdot [1\;2\;1]^t < 0$
- *Update:* $\mathbf{a}^{(3)} = \mathbf{a}^{(2)} + \mathbf{z}_M = [0 \;\; 0 \;\; -2] + [1 \;\; 2 \;\; 1] = [1 \;\; 2 \;\; -1]$

Non-Linearly Separable Case

- $\mathbf{a}^{(3)} = [1\ \ 2\ \ -1]$
- rule is: $\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \mathbf{z}_M$

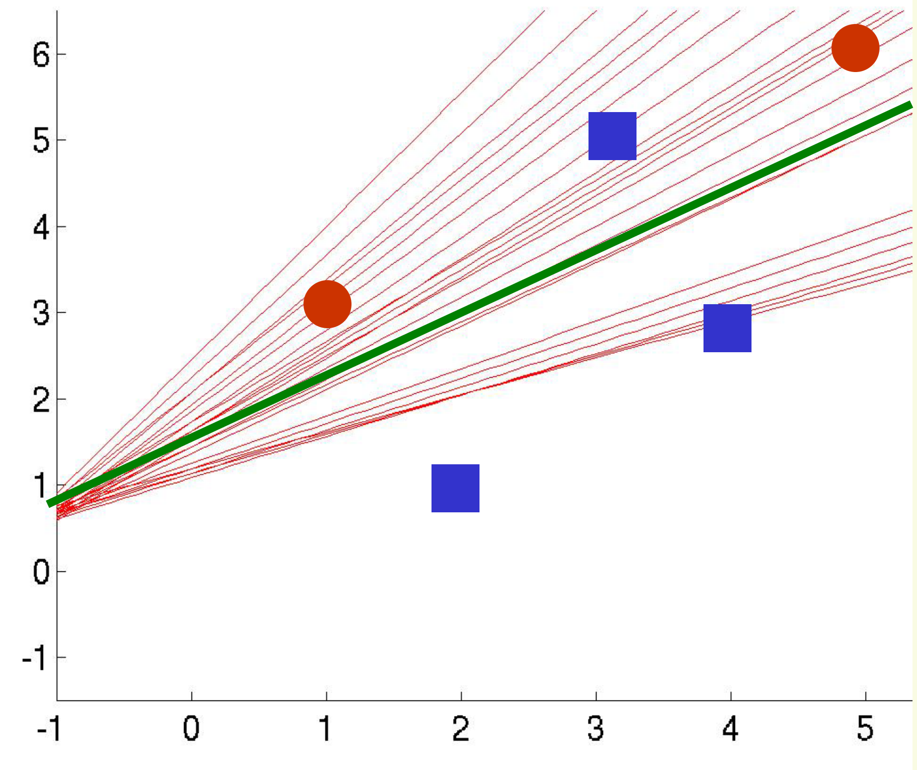
$$\mathbf{z}^1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad \mathbf{z}^2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad \mathbf{z}^3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad \mathbf{z}^4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad \mathbf{z}^5 = \begin{bmatrix} -1 \\ -5 \\ -6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



- $\mathbf{a}^t\mathbf{z}^2 = [1\ 4\ 3] \cdot [1\ \ 2\ -1]^t = 6 > 0$
- $\mathbf{a}^t\mathbf{z}^3 = [1\ 3\ 5] \cdot [1\ \ 2\ -1]^t = 2 > 0$
- $\mathbf{a}^t\mathbf{z}^4 = [-1\ -1\ -3] \cdot [1\ \ 2\ -1]^t = 0$
- *Update:* $\mathbf{a}^{(4)} = \mathbf{a}^{(3)} + \mathbf{z}_M = [1\ \ \ 2\ \ -1] + [-1\ \ -1\ \ -3] = [0\ \ 1\ \ -4]$

Non-Linearly Separable Case

- 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?

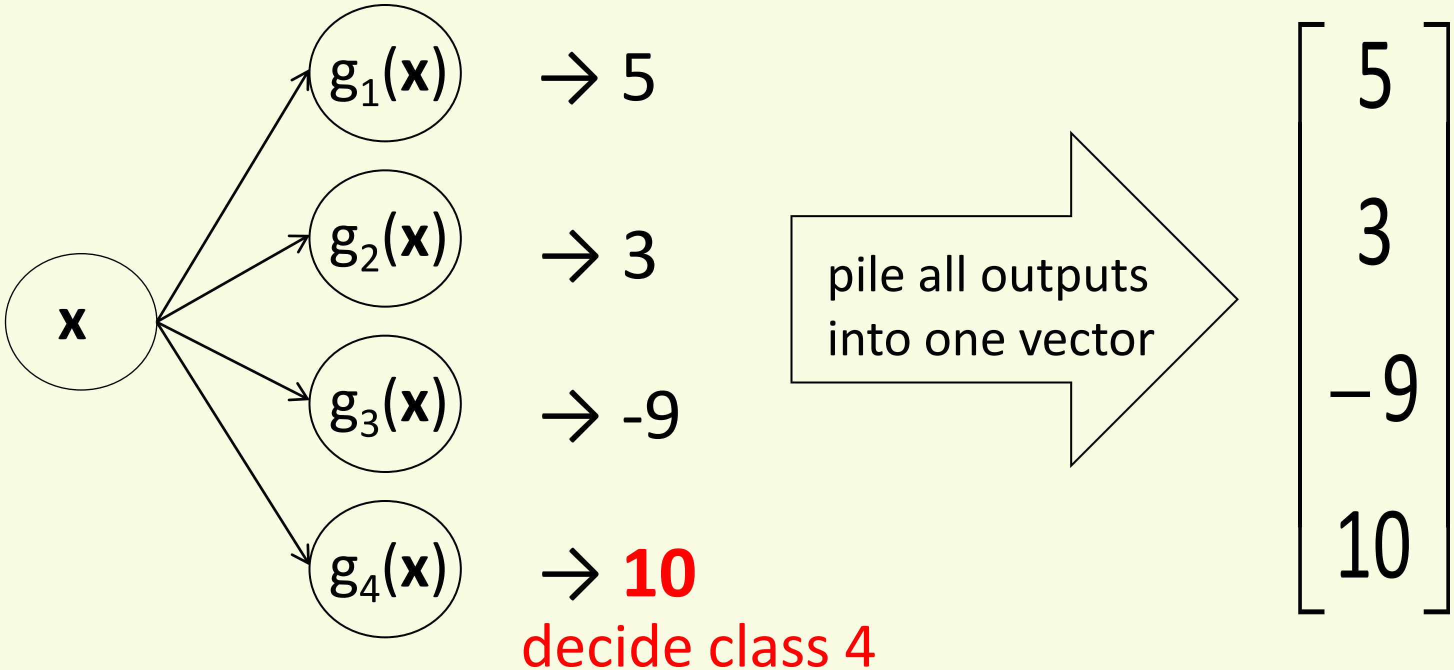


Linear Classifier: Multiple Classes

- Can extend to **m** class case
- Augment samples with 1 as the first feature
    - but no "normalization"
- Define **m** discriminant functions

$$\mathbf{g}_i(\mathbf{x}) = \mathbf{w}_i^t\mathbf{x} \quad \text{for } \mathbf{i} = 1, 2, \dots \mathbf{m}$$

- Assign **x** to **i** that gives maximum $\mathbf{g}_i(\mathbf{x})$



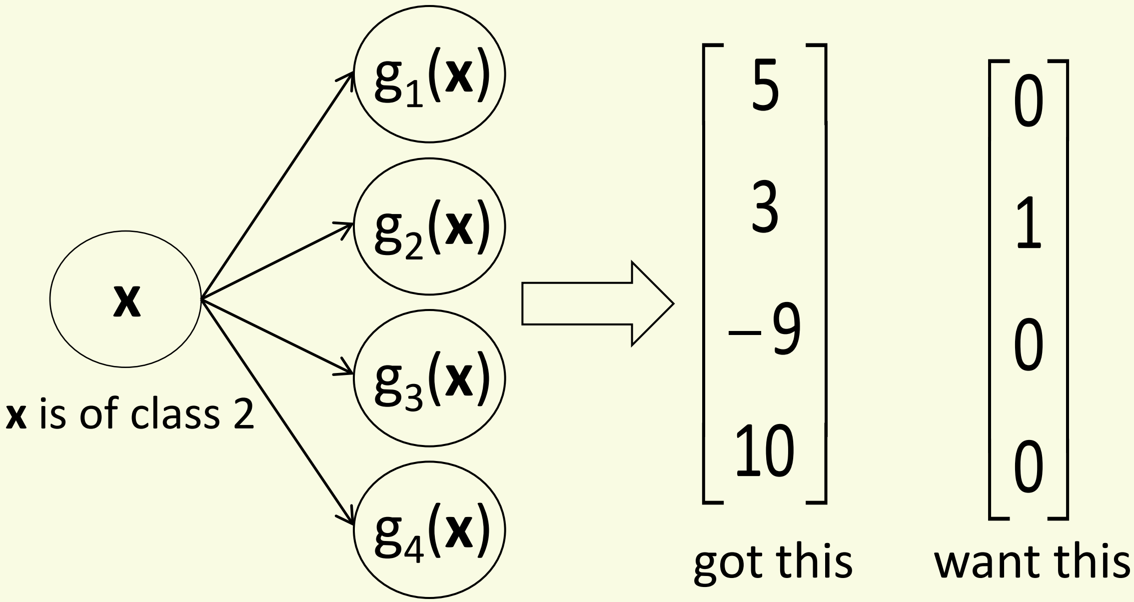
Linear Classifier: Multiple Classes

- Could use one dimensional output $y_i \in \{1,2,3,\ldots m\}$
- Convenient to use multi-dimensional outputs

$$y^j = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad y^j = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad y^j = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \qquad y^j = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

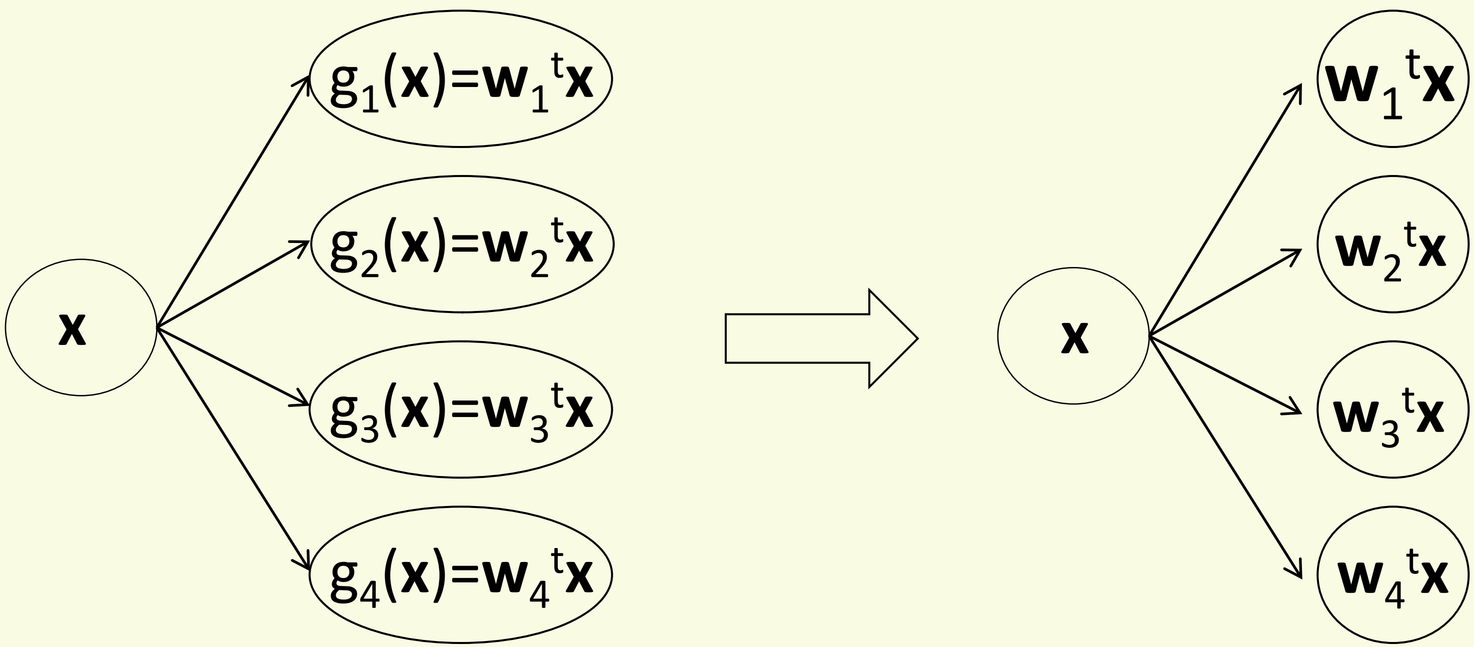
class 1 ***class 2*** ***class 3*** ***class 4***

- For training, if sample is of class **i**, want output vector to be 0 everywhere except position **i**, where it should be 1



Linear Classifier: Multiple Classes

- Assign **x** to **i** that gives maximum $\mathbf{g_i}(\mathbf{x}) = \mathbf{w}_i^t\mathbf{x}$



- In matrix notation

$$\underbrace{\begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix}}_{\mathbf{W}} \begin{matrix} \leftarrow \mathbf{w}_1 \\ \leftarrow \mathbf{w}_2 \\ \leftarrow \mathbf{w}_3 \\ \leftarrow \mathbf{w}_4 \end{matrix} \underbrace{\begin{bmatrix} 1 \\ 7 \\ 4 \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} 2 \\ -4 \\ \boxed{47} \\ -43 \end{bmatrix}}_{\mathbf{Wx}}$$

- Assign **x** to class that corresponds to largest row of **Wx**

Linear Multiclass Classifier: Loss Function

- Assign sample $\mathbf{x}^i$ to class that corresponds to largest row of $\mathbf{W}\mathbf{x}^i$
- Loss function?

$$\underbrace{\begin{bmatrix} 2 \\ -4 \\ 47 \\ -43 \end{bmatrix}}_{\mathbf{W}\mathbf{x}^i} \qquad \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}_{\mathbf{y}^i}$$

- Can use quadratic loss per sample $\mathbf{x}^i$ as $\frac{1}{2}\|\mathbf{W}\mathbf{x}^i - \mathbf{y}^i\|^2$
  - for example above, loss $(2^2 + 4^2 + 47^2 + 44^2)/2$
  - total loss on all training samples $L(\mathbf{W}) = \frac{1}{2}\sum_i \|\mathbf{W}\mathbf{x}^i - \mathbf{y}^i\|^2$
  - gradient of the loss

$$\nabla \mathbf{L}(\mathbf{W}) = \sum_{\mathbf{i}} \left(\mathbf{W}\mathbf{x}^{\mathbf{i}} - \mathbf{y}^{\mathbf{i}}\right)\left(\mathbf{x}^{\mathbf{i}}\right)^{\mathbf{t}}$$

  - batch gradient descent updates

$$\mathbf{W} = \mathbf{W} - \alpha \sum_{\mathbf{i}} \left(\mathbf{W}\mathbf{x}^{\mathbf{i}} - \mathbf{y}^{\mathbf{i}}\right)\left(\mathbf{x}^{\mathbf{i}}\right)^{\mathbf{t}}$$

Linear Multiclass: Quadratic Loss

- Consider gradient descent update, single sample **x** with $\alpha = 1$

$$\mathbf{W} = \mathbf{W} - (\mathbf{Wx} - \mathbf{y})\mathbf{x}^t$$

- Suppose $\mathbf{x} = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$ and is in class 2 and $\mathbf{W} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix}$

ok
need to decrease
need to increase

$$\mathbf{Wx} - \mathbf{y} = \begin{bmatrix} 0 \\ 4 \\ 23 \\ -17 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 23 \\ -17 \end{bmatrix}$$

- update rule

$$\mathbf{W} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 3 \\ 23 \\ -17 \end{bmatrix} \begin{bmatrix} 1 & 3 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix} - \begin{bmatrix} 0\cdot(1 \quad 2 \quad 3) \\ 3\cdot(1 \quad 2 \quad 3) \\ 23\cdot(1 \quad 2 \quad 3) \\ -17\cdot(1 \quad 2 \quad 3) \end{bmatrix}$$

Linear Multiclass: Quadratic Loss

ok
need to decrease
need to increase

$$\mathbf{Wx} - \mathbf{y} = \begin{bmatrix} 0 \\ 4 \\ 23 \\ -17 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 23 \\ -17 \end{bmatrix}$$

- update rule

$$\mathbf{W} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 3 \\ 23 \\ -17 \end{bmatrix} \begin{bmatrix} 1 & 3 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix} - \begin{bmatrix} 0\cdot(& 1 & 2 & 3\ ) \\ 3\cdot(& 1 & 2 & 3\ ) \\ 23\cdot(& 1 & 2 & 3\ ) \\ -17\cdot(& 1 & 2 & 3\ ) \end{bmatrix} = \begin{bmatrix} 2 & 4 & -7 \\ 6 & -12 & -4 \\ -19 & -64 & -44 \\ 19 & 44 & 35 \end{bmatrix}$$

- With new **W**

$$\mathbf{Wx} = \begin{bmatrix} 0 \\ -38 \\ -299 \\ 221 \end{bmatrix}$$

Linear Multiclass: Perceptron Loss Function

- Assign sample $\mathbf{x}^i$ to class that corresponds to largest row of $\mathbf{Wx}^i$
- Another loss function?

$$\underset{\mathbf{Wx}^i}{\begin{bmatrix} 2 \\ -4 \\ 47 \\ -43 \end{bmatrix}} \qquad \underset{\mathbf{y}^i}{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}$$

- Perceptron loss on sample $\mathbf{x}^i$: $\mathbf{L}_i(\mathbf{W}) = \max_{\mathbf{k}}[(\mathbf{Wx}^i)_k - (\mathbf{Wx}^i)_{\mathbf{c}}]$, where
  - $(\mathbf{Wx}^i)_{\mathbf{k}}$ is the entry in row **k** of vector $\mathbf{Wx}^i$
  - **c** is the correct class of sample $\mathbf{x}^i$
  - in words, find the largest entry in $\mathbf{Wx}^i$, subtract from it the entry in the row corresponding to the true class of sample $\mathbf{x}^i$
    - loss is zero if correct classification, positive otherwise
  - for the example above, loss is 47-(-43)= 90 since sample is of class 4

Linear Multiclass: Perceptron Loss Function

- $\mathbf{L}_i(\mathbf{W}) = \max_{\mathbf{k}}[(\mathbf{Wx}^i)_{\mathbf{k}} - (\mathbf{Wx}^i)_{\mathbf{c}}]$
- Gradient, single sample rule
  - let **c** be the correct row, and **r** be row where $\mathbf{Wx}^i$ gives the largest output
  - if **r** = **c**, $\nabla \mathbf{L}_i(\mathbf{W}) = 0$
  - otherwise, $\nabla \mathbf{L}_i(\mathbf{W}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ & \mathbf{x}^i & & \\ 0 & 0 & 0 & 0 \\ & -\mathbf{x}^i & & \end{bmatrix}$ $\begin{matrix} \\ \text{row } \mathbf{r} \\ \\ \text{row } \mathbf{c} \end{matrix}$
  - for the example, $\nabla \mathbf{L}_i(\mathbf{W}) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 3 & 2 \\ -1 & -3 & -2 \end{bmatrix}$

$$\mathbf{x}^i = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} \quad \underset{\mathbf{Wx}^i}{\begin{bmatrix} 2 \\ -4 \\ 47 \\ -43 \end{bmatrix}} \quad \underset{\mathbf{y}^i}{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}$$

Linear Multiclass: Perceptron Loss Function

- For the example,

$$\nabla \mathbf{L}_i(\mathbf{W}) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 3 & 2 \\ -1 & -3 & -2 \end{bmatrix} \qquad \mathbf{x}^i = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \mathbf{y}^i$$

- With α = 1, new $\mathbf{W} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 3 & 2 \\ -1 & -3 & -2 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 3 & 2 & 0 \\ 3 & -4 & 3 \end{bmatrix}$

- With new weights:

$$\mathbf{W}\mathbf{x}^i = \begin{bmatrix} 0 \\ 4 \\ 9 \\ -3 \end{bmatrix}$$

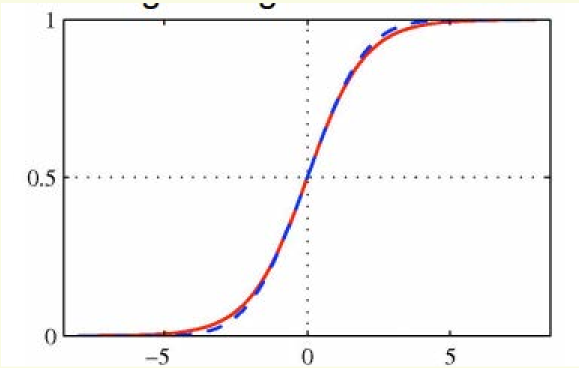
- Compare to the old weights:

$$\mathbf{W}_{old}\mathbf{x}^i = \begin{bmatrix} 0 \\ 4 \\ 23 \\ -17 \end{bmatrix}$$

Three Approaches to Classification

1. Directly design discriminant function **f**(**x**,**w**) for classification
   - design differentiable loss function that makes intuitive sense
   - find **w** that minimize loss function
   - Choose class that maximizes discriminant function
2. Model conditional class probabilities P(class=k|**x**,**w**)
   - Choose loss function with probabilistic interpretation and minimize it
     - Loss function is usually (–log probability)
     - Parameters **w** are tuned so as to maximize probability of the training data
   - Choose class that maximizes discriminant function
3. Model probability of training data **x** under class-specific generative models p(**x**,**w**)
   - Use training data to fit parameters **w** for each class independently
     - i.e. fit Gaussians to samples from each class
   - Choose the class that makes **x** most probable

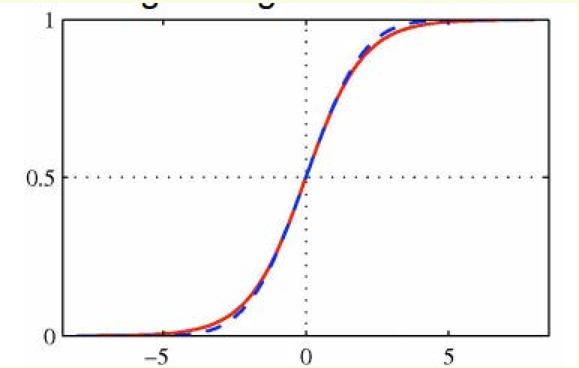
Linear Machine: Logistic Regression



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

- Has probabilistic interpretation
- Model **P**(class 1|**x**,**w**) and **P**(class 2|**x**,**w**)
- Uses logistic sigmoid function
  - denote classes with 1 and 0 now
    - $\mathbf{y}^i$ = 1 for class 1, $\mathbf{y}^i$ = 0 for class 2
- $\mathbf{g}(\mathbf{x},\mathbf{w}) = \mathbf{w}^T\mathbf{x}$
- let $\mathbf{f}(\mathbf{x},\mathbf{w}) = \sigma(\mathbf{g}(\mathbf{x},\mathbf{w})) = \sigma(\mathbf{w}^T\mathbf{x})$
  - assume **x** is augmented with 1
  - bigger 0.5 if $\mathbf{w}^T\mathbf{x}$ is positive, decide class 1
  - less 0.5 if $\mathbf{w}^T\mathbf{x}$ is negative, decide class 2
- Probabilistic interpretation
  - **P**(class 1|**x**,**w**) = $\sigma(\mathbf{w}^T\mathbf{x})$
  - **P**(class 2|**x**,**w**) = 1 - P(class 1|**x**,**w**)
- Despite the name, logistic regression is used for classification, not regression
  - Side note: sigmoid is a continuous function, good for gradient descent

Linear Machine: Logistic Regression



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

- **f**(**x**,**w**) = σ(**w**^{T}**x**)
- Probabilistic interpretation
  - **P**(class 1|**x**,**w**) = σ(**w**^{T}**x**)
  - **P**(class 2|**x**,**w**) = 1 - **P**(class 1|**x**,**w**)
- Per sample loss function: -log(**P**(**y**^{i}|**x**^{i}))
  - if sample **x**^{i} of class 1, loss is –log(σ(**w**^{T}**x**^{i}))
  - if sample **x**^{i} of class 2, loss is –log(1-σ(**w**^{T}**x**^{i}))
- Convex, can be optimized exactly with gradient descent
- Gradient descent update rule

$$w = w + \alpha \sum_j \left(y^j - \sigma\left(w^t x^j\right)\right) x^j$$

Linear Machine: Softmax Regression

- In case of **m** classes, define **m** functions

$$\mathbf{g}_i(\mathbf{x}) = \mathbf{w}_i^t\mathbf{x} \quad \text{for } \mathbf{i} = 1, 2, \ldots \mathbf{m}$$

- Instead of raw scores
- Softmax scores for probabilistic interpretation

$$\mathbf{f}(\mathbf{x},\mathbf{w}) = \begin{bmatrix} w_1^T x \\ w_2^T x \\ \ldots \\ w_m^T x \end{bmatrix} = \begin{bmatrix} 100 \\ 5 \\ \ldots \\ -6 \end{bmatrix}$$

$$f(x,w) = \begin{bmatrix} \dfrac{\exp(w_1^T x)}{\sum_{j=1}^{m} \exp(w_j^T x)} \\ \dfrac{\exp(w_2^T x)}{\sum_{j=1}^{m} \exp(w_j^T x)} \\ \\ \dfrac{\exp(w_m^T x)}{\sum_{j=1}^{m} \exp(w_j^T x)} \end{bmatrix} = \begin{bmatrix} \Pr(\textbf{class } 1 \,|\, \mathbf{x}, \mathbf{w}) \\ \Pr(\textbf{class } 2 \,|\, \mathbf{x}, \mathbf{w}) \\ \ldots \\ \Pr(\textbf{class m} \,|\, \mathbf{x}, \mathbf{w}) \end{bmatrix} = \begin{bmatrix} 0.6 \\ 0.1 \\ \ldots \\ 0.06 \end{bmatrix}$$

Linear Machine: Softmax Regression

- Also optimize under -log(Pr($\mathbf{y}^i | \mathbf{x}^i$)) loss function

$$f(x,w) = \begin{bmatrix} \dfrac{\exp(w_1^T x)}{\sum_{j=1}^{m} \exp(w_j^T x)} \\ \dfrac{\exp(w_2^T x)}{\sum_{j=1}^{m} \exp(w_j^T x)} \\ \\ \dfrac{\exp(w_m^T x)}{\sum_{j=1}^{m} \exp(w_j^T x)} \end{bmatrix} = \begin{bmatrix} \Pr(\textbf{class } 1 | \mathbf{x}, \mathbf{w}) \\ \Pr(\textbf{class } 2 | \mathbf{x}, \mathbf{w}) \\ \dots \\ \Pr(\textbf{class m} | \mathbf{x}, \mathbf{w}) \end{bmatrix} = \begin{bmatrix} 0.6 \\ 0.1 \\ \dots \\ 0.06 \end{bmatrix}$$

if sample of class 2, take –log of the number in row 2 for the loss

Linear Machine: Softmax Regression

- Define softmax(**a**) function for vector **a** as

$$\begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \vdots \\ \mathbf{a}_m \end{bmatrix} \xrightarrow{\text{softmax}} \begin{bmatrix} \dfrac{\exp(\mathbf{a}_1)}{\sum_{j=1}^{m}\exp(\mathbf{a}_j)} \\ \dfrac{\exp(\mathbf{a}_2)}{\sum_{j=1}^{m}\exp(\mathbf{a}_j)} \\ \vdots \\ \dfrac{\exp(\mathbf{a}_m)}{\sum_{j=1}^{m}\exp(\mathbf{a}_j)} \end{bmatrix}$$

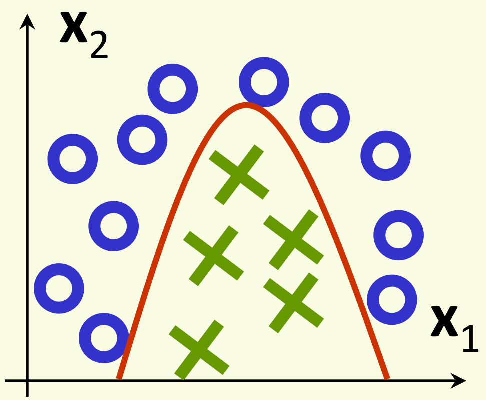
- Update rule for weight matrix **W**

$$\mathbf{W} = \mathbf{W} + \alpha \sum_{j} \left(\mathbf{y}^{j} - \sigma\left(\mathbf{w}^{T}\mathbf{x}^{j}\right)\right)\left(\mathbf{x}^{j}\right)^{t}$$

Generalized Linear Classifier

- Can use other discriminant functions, like quadratics

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



- Methodology is almost the same as in the linear case
  - $\mathbf{f(x)} = \text{sign}(\mathbf{w}_0+\mathbf{w}_1\mathbf{x}_1+\mathbf{w}_2\mathbf{x}_2+\mathbf{w}_{12}\mathbf{x}_1\mathbf{x}_2+\mathbf{w}_{11}\mathbf{x}_1^2+\mathbf{w}_{22}\mathbf{x}_2^2)$
  - $\mathbf{z} = [\,1 \quad \mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_1\mathbf{x}_2 \quad \mathbf{x}_1^2 \quad \mathbf{x}_2^2\,]$
  - $\mathbf{a} = [\,\mathbf{w}_0 \quad \mathbf{w}_1 \quad \mathbf{w}_2 \quad \mathbf{w}_{12} \quad \mathbf{w}_{11} \quad \mathbf{w}_{22}\,]$
  - “normalization”: multiply negative class samples by -1
  - all the other procedures remain the same, i.e. gradient descent to minimize Perceptron loss function, any other loss function

Generalized Linear Classifier

- In general, to the liner function:

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

- can add quadratic terms:

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

- This is still a linear function in its parameters **w**
- $\mathbf{g}(\mathbf{y},\mathbf{v}) = \mathbf{v}_0+\mathbf{v}^t\mathbf{y}$

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

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

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

- Can use all the same training methods as before

Generalized Linear Classifier

- Generalized linear classifier

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

- $\mathbf{h}(\mathbf{x})$ 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 **w**

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

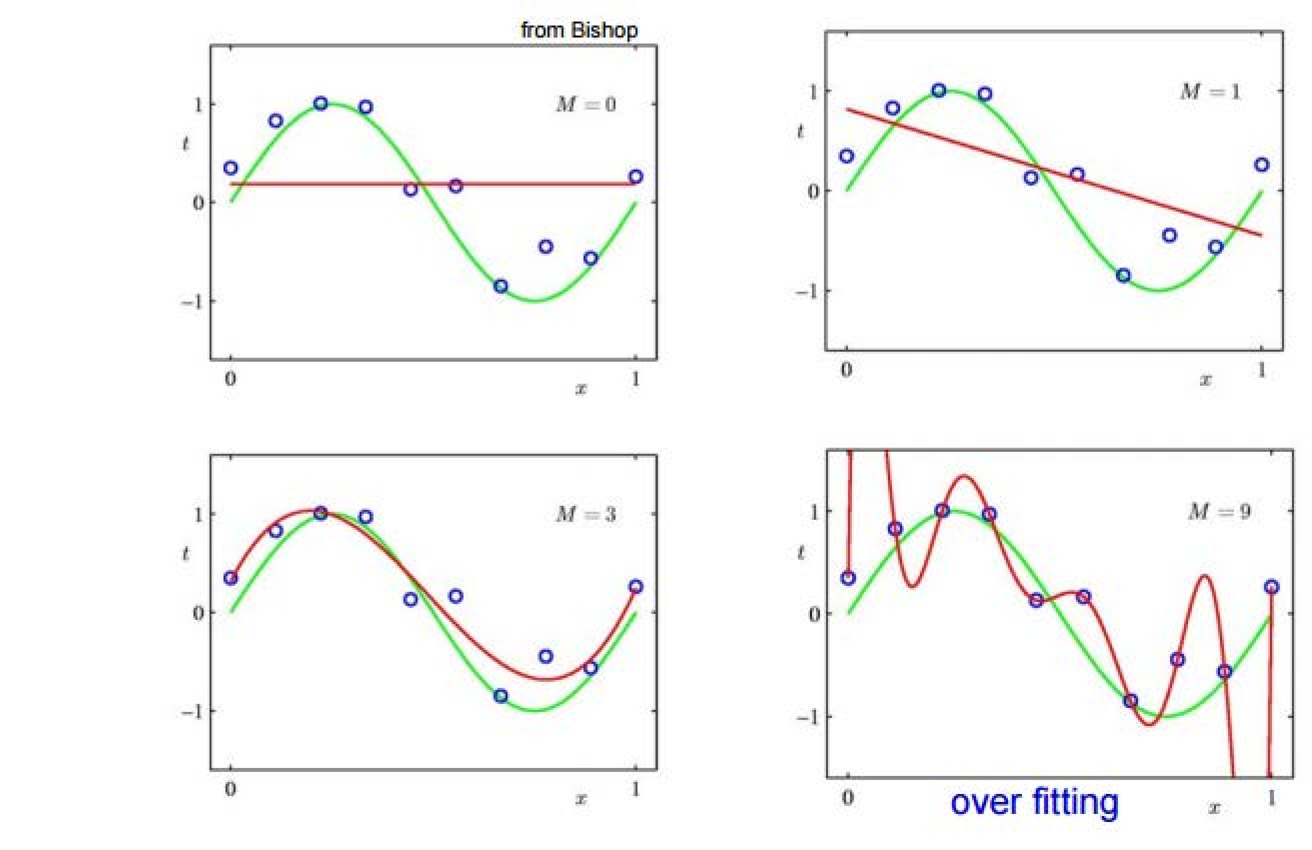
$$\mathbf{h} = [\mathbf{h}_1(\mathbf{x})\ \ \mathbf{h}_2(\mathbf{x})\ \ \ldots\ \ \mathbf{h}_m(\mathbf{x})]$$

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

- Can use all the same training methods as before

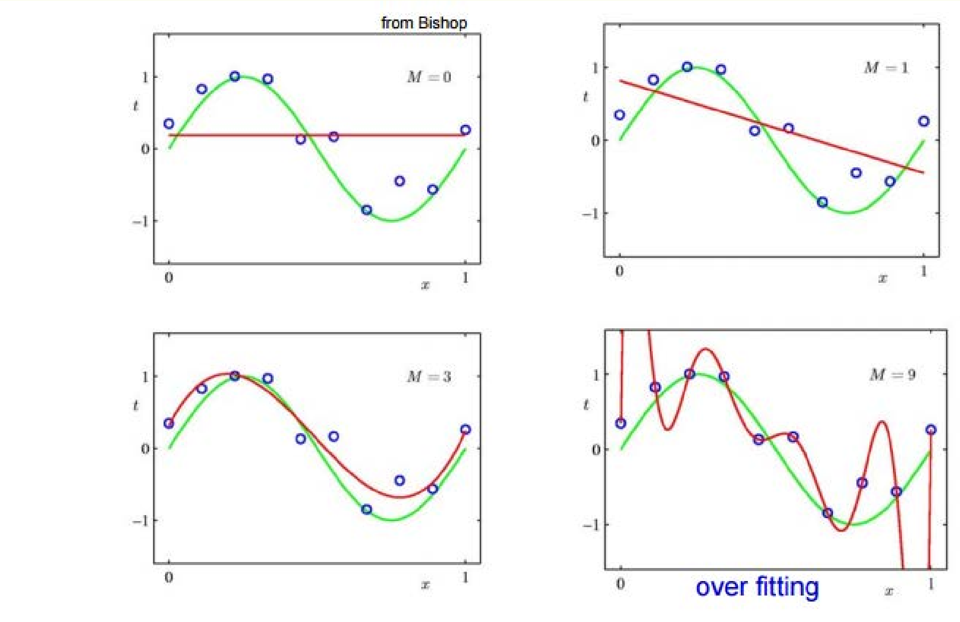
Generalized Linear Classifier

- Usually face severe overfitting
  - too many degrees of freedom
  - boundary can “curve” to fit to the noise in the data
- Regression example



Generalized Linear Classifier

- Helps to regularize by keeping **w** small
  - small **w** means the boundary is not as curvy
- Regression example



Polynomial Coefficients

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Generalized Linear Classifier

- Helps to *regularize* by keeping **w** small
  - small **w** means the boundary is not as curvy
- For example, add $\boldsymbol{\lambda}||\mathbf{w}||^2$ to the loss function
- Recall quadratic loss function

$$\mathbf{L} = \tfrac{1}{2}\Sigma_i || \mathbf{f}(\mathbf{x}^i, \mathbf{w}) - \mathbf{y}^i ||^2$$

- Regularized version

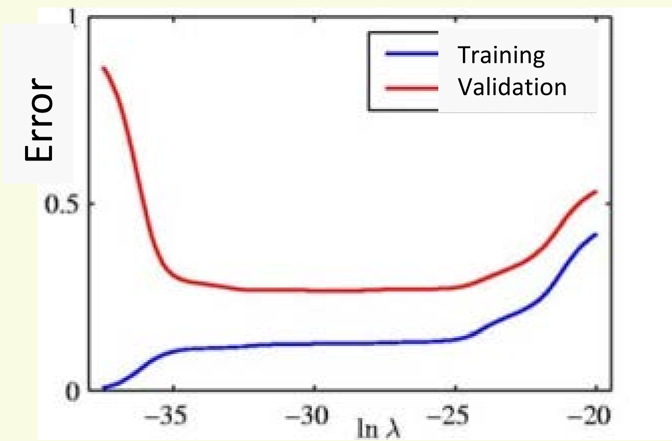
$$\mathbf{L} = \tfrac{1}{2}\Sigma_i || \mathbf{f}(\mathbf{x}^i, \mathbf{w}) - \mathbf{y}^i ||^2 + \boldsymbol{\lambda}||\mathbf{w}||^2$$

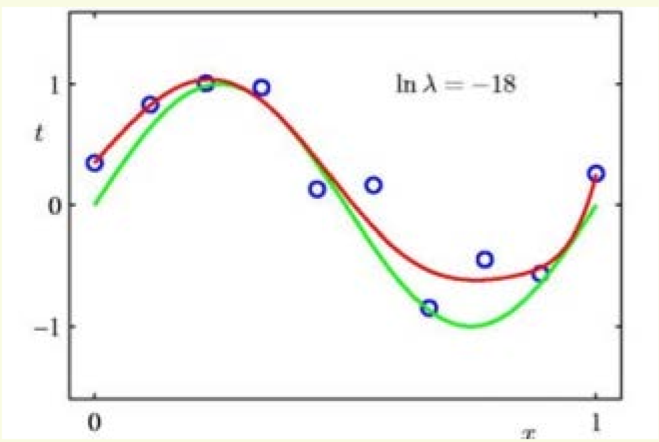
- Regression example, polynomial coefficients for degree M = 9

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Generalized Linear Classifier

- How to set **λ**?
- With validation or cross-validation
- Consider polynomial of degree M=9 regression





Learning by Gradient Descent

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