CS4442/9542b
Artificial Intelligence II
prof. Olga Veksler

Lecture 4

Machine Learning

Linear Classifier
2 classes

Outline

- Optimization with gradient descent
- Linear Classifier
  - Two class case
    - Loss functions
    - Perceptron
      - Batch
      - Single sample
    - Logistic Regression

Optimization

- How to minimize a function of a single variable

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

- 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

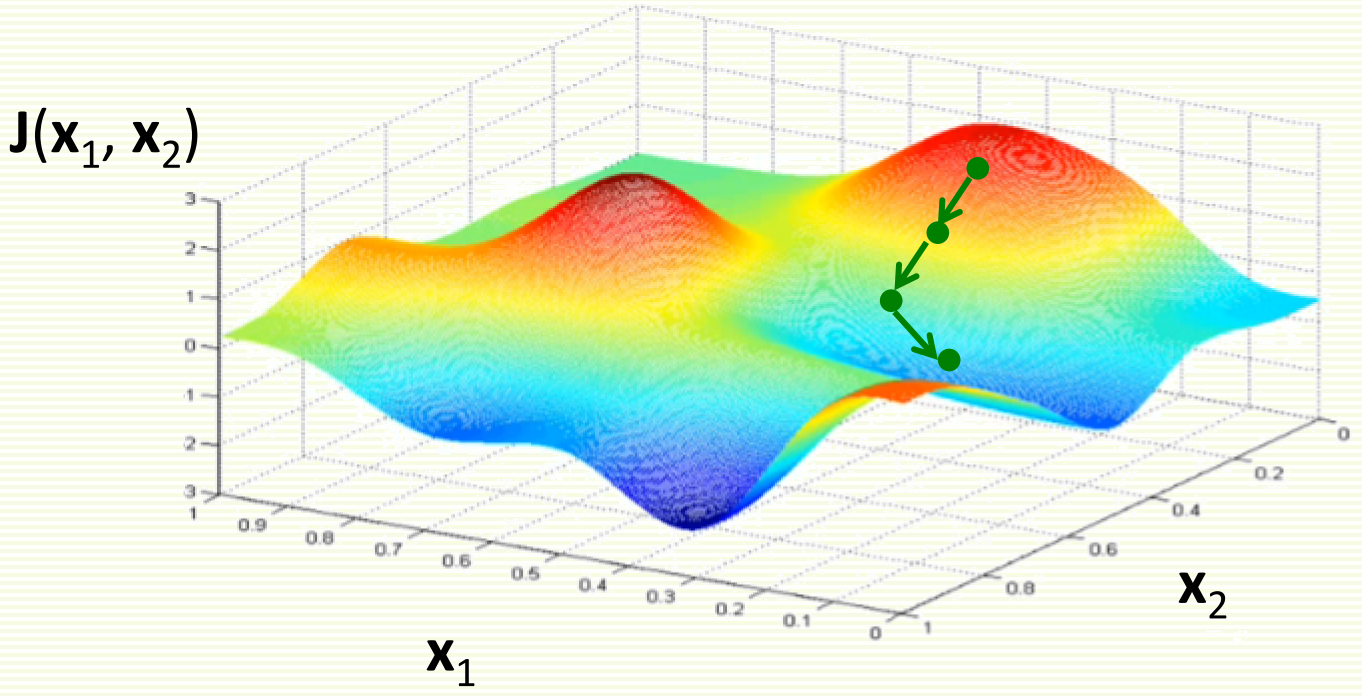
$$\mathbf{J}(\mathbf{x}) = \mathbf{J}(\mathbf{x}_1, \ldots, \mathbf{x}_d)$$

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

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

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

Optimization: Gradient Direction

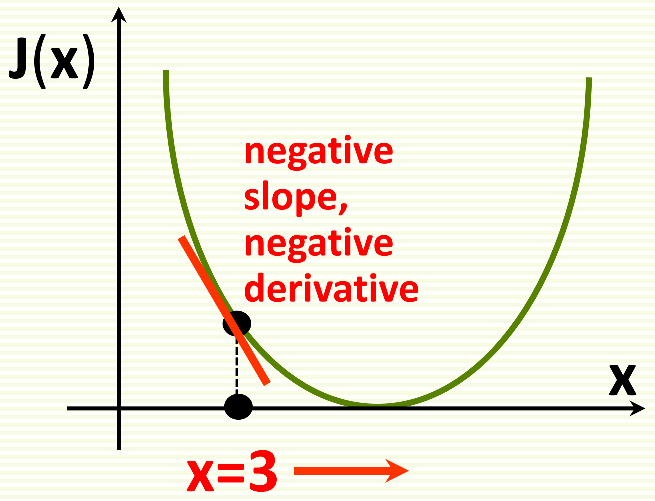


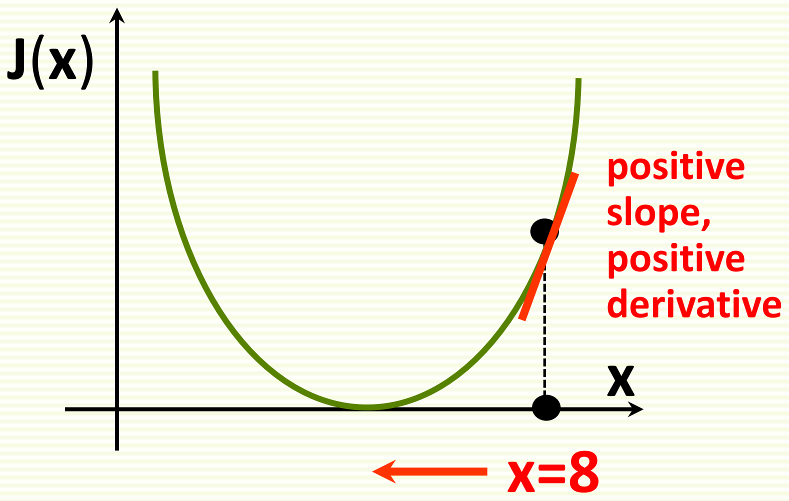
- Gradient $\nabla \mathbf{J}(\mathbf{x})$ points in the direction of steepest increase of function $\mathbf{J}(\mathbf{x})$
- $-\nabla \mathbf{J}(\mathbf{x})$ points in the direction of steepest decrease

Picture from Andrew Ng

Gradient Direction in 1D

- Gradient is just derivative in 1D
- Example: **J**(**x**) =(**x**-5)^2 and derivative is $\frac{d}{dx}J(x) = 2(x-5)$



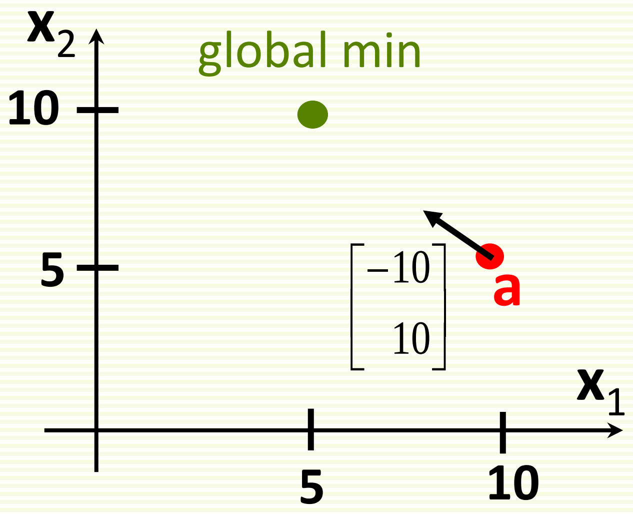


- Let **x** = 3
- $-\frac{d}{dx}J(3) = 4$
- derivative says increase **x**

- Let **x** = 8
- $-\frac{d}{dx}J(3) = -6$
- derivative says decrease **x**

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 $a = \begin{bmatrix} 10 \\ 5 \end{bmatrix}$
- $\frac{\partial}{\partial x_1} J(a) = 10$
- $\frac{\partial}{\partial x_2} J(a) = -10$
- $\nabla J(a) = \begin{bmatrix} 10 \\ -10 \end{bmatrix}$
- $-\nabla J(a) = \begin{bmatrix} -10 \\ 10 \end{bmatrix}$

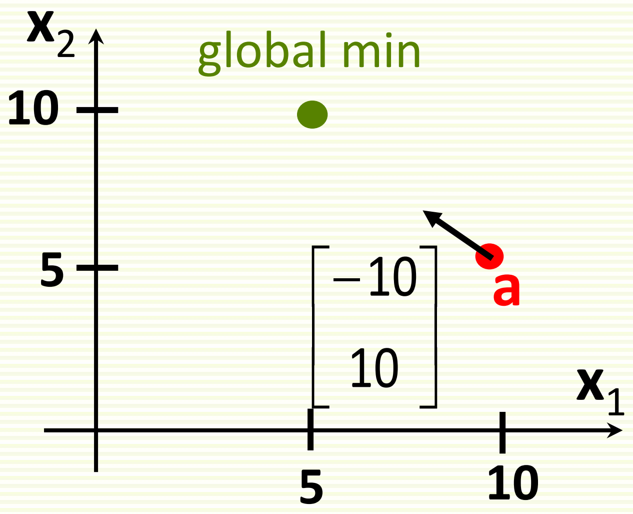


Gradient Descent: Step Size

- $\mathbf{J}(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 - 5)^2 + (\mathbf{x}_2 - 10)^2$
- Which step size to take?
- Controlled by parameter α
  - called **learning rate**
- From previous slide
  - $\mathbf{a} = \begin{bmatrix} 10 \\ 5 \end{bmatrix}, \quad -\nabla \mathbf{J}(\mathbf{a}) = \begin{bmatrix} -10 \\ 10 \end{bmatrix}$
- Let $\alpha = 0.2$

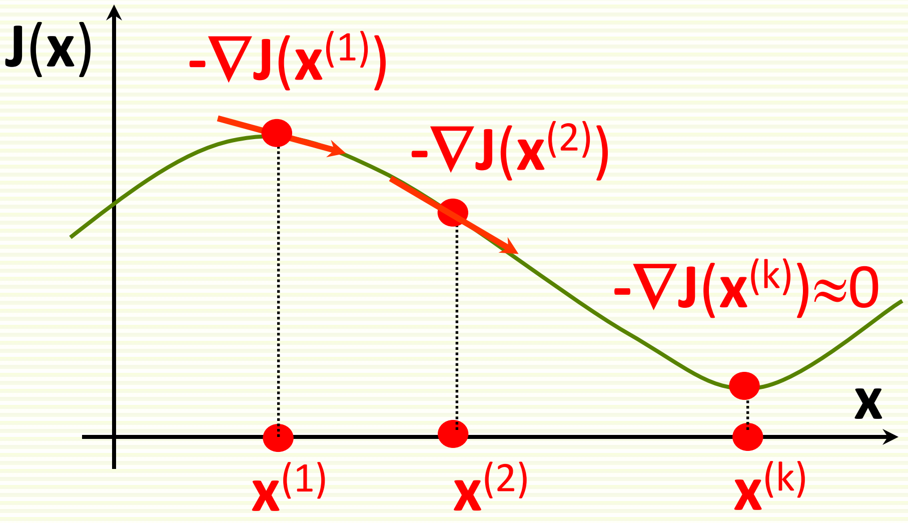
$$\mathbf{a} - \alpha \nabla \mathbf{J}(\mathbf{a}) = \begin{bmatrix} 10 \\ 5 \end{bmatrix} + 0.2 \begin{bmatrix} -10 \\ 10 \end{bmatrix} = \begin{bmatrix} 8 \\ 7 \end{bmatrix}$$

- $\mathbf{J}(10, 5) = 50$; $\mathbf{J}(8,7) = 18$



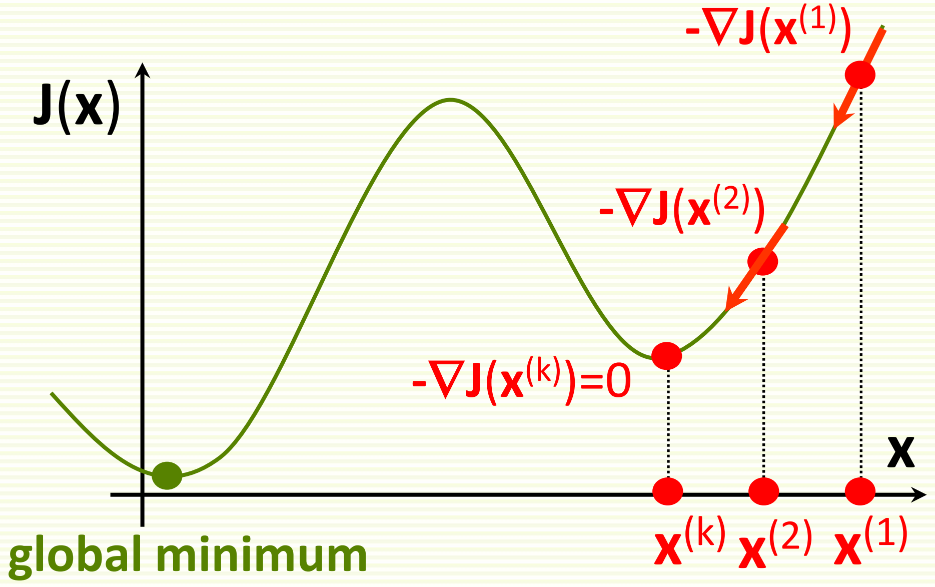
Gradient Descent Algorithm

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



Gradient Descent: Local Minimum

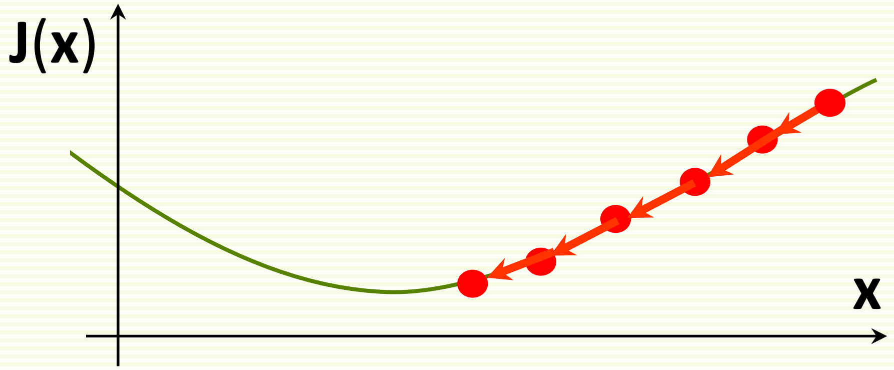
- Not guaranteed to find global minimum
  - gets stuck in local minimum



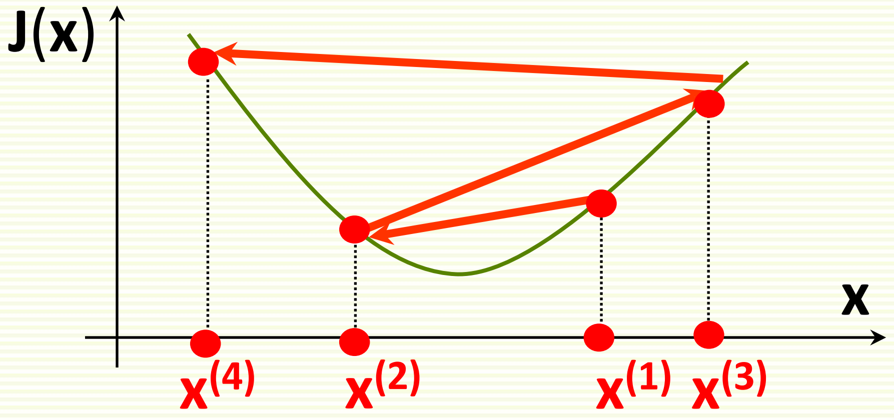
- Still gradient descent is very popular because it is simple and applicable to any differentiable function

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

Variable Learning Rate

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

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

$\longrightarrow$

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

Variable Learning Rate

- Usually do not keep track of all intermediate solutions

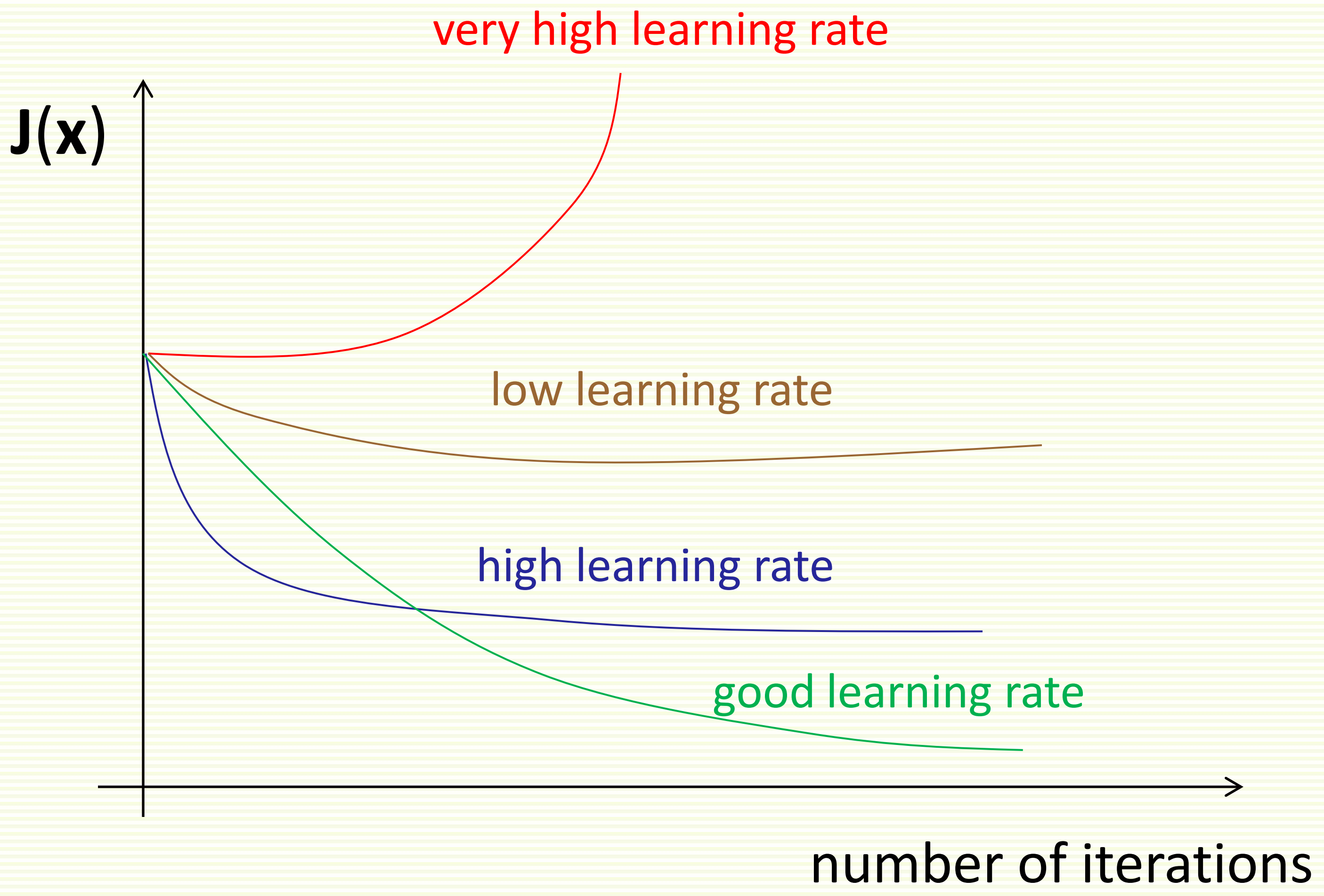
```
$\mathbf{k} = 1$
$\mathbf{x}^{(1)}$ = any initial guess
choose $\alpha, \varepsilon$
while $\alpha||\nabla J(\mathbf{x}^{(k)})|| > \varepsilon$
    $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha \nabla J(\mathbf{x}^{(k)})$
    $\mathbf{k} = \mathbf{k} + 1$
```

$\longrightarrow$

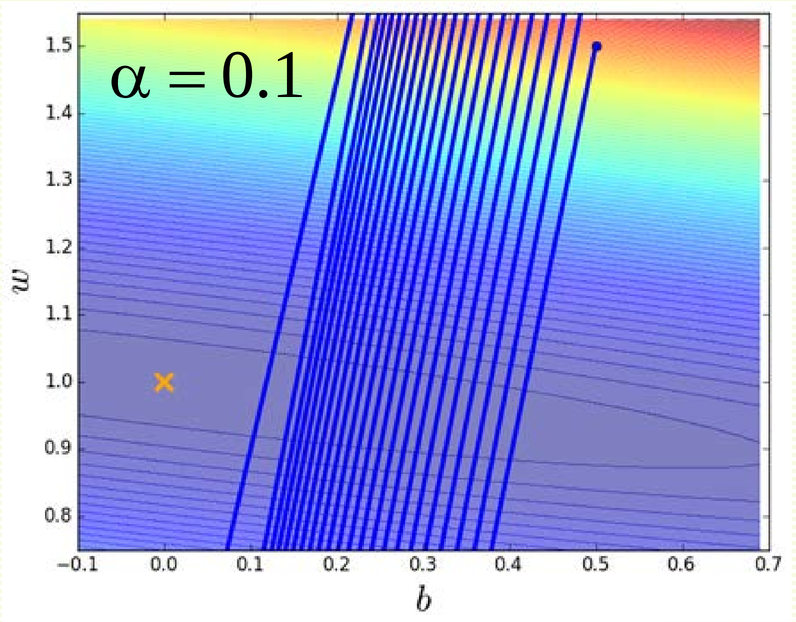
```
$\mathbf{k} = 1$
$\mathbf{x}$ = any initial guess
choose $\alpha, \varepsilon$
while $\alpha||\nabla J(\mathbf{x})|| > \varepsilon$
    $\mathbf{x} = \mathbf{x} - \alpha \nabla J(\mathbf{x})$
    $\mathbf{k} = \mathbf{k} + 1$
```

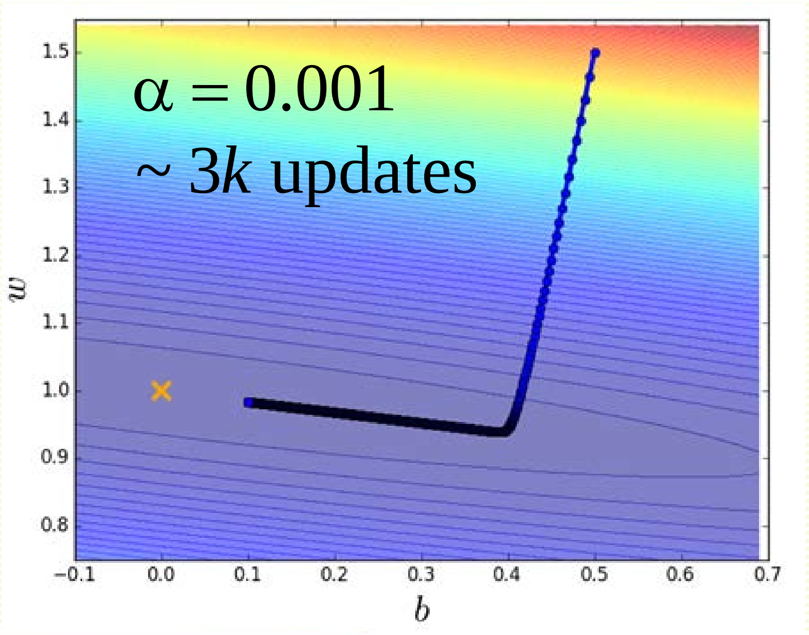
Learning Rate

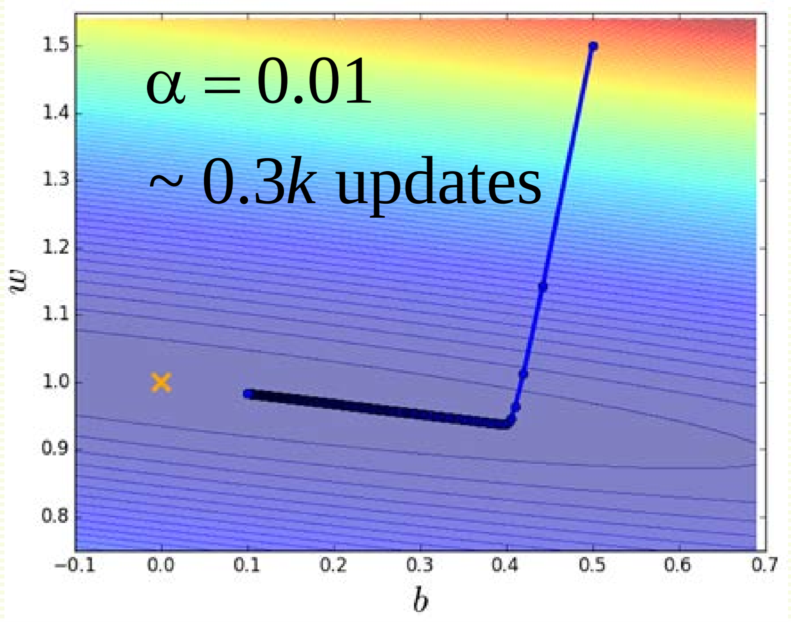
- Monitor learning rate by looking at how fast the objective function decreases



Learning Rate: Loss Surface Illustration







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 Machine Learning (Recap)

- Chose type of **f**(**x**,**w**)
  - **w** are tunable weights, **x** is the input example
  - **f**(**x**,**w**) should output the correct class of sample **x**
  - use labeled samples to tune weights **w** so that **f**(**x**,**w**) give the correct class **y** for **x**
    - with help of loss function **L**(**f**(**x**,**w**) ,**y**)
- How to choose type of **f**(**x**,**w**)?
  - many choices
  - previous lecture: kNN classifier
  - this lecture: linear classifier

Linear Classifier

- Classifier is linear if it makes a decision based on linear combination of features

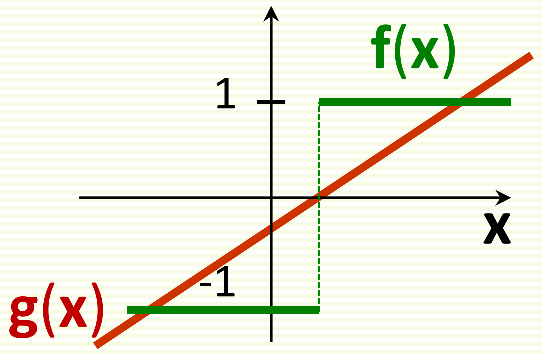
$$\mathbf{g}(\mathbf{x}) = \mathbf{w}_0+\mathbf{x}_1\mathbf{w}_1 + \dots + \mathbf{x}_d\mathbf{w}_d$$

  - **g**(**x**,**w**) sometimes called *discriminant function*

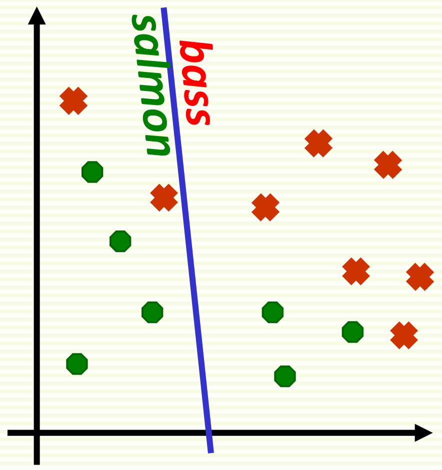
- Encode 2 classes as
  - **y** = 1 for the first class
  - **y** = -1 for the second class
- One choice for linear classifier

$$\mathbf{f}(\mathbf{x},\mathbf{w}) = \text{sign}(\mathbf{g}(\mathbf{x},\mathbf{w}))$$

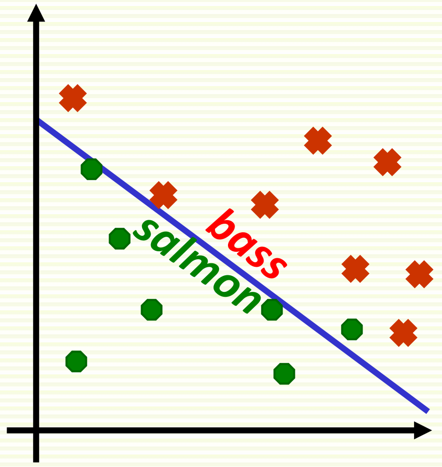
  - 1 if **g**(**x**,**w**) is positive
  - -1 if **g**(**x**,**w**) is negative



Linear Classifier: Decision Boundary



bad boundary



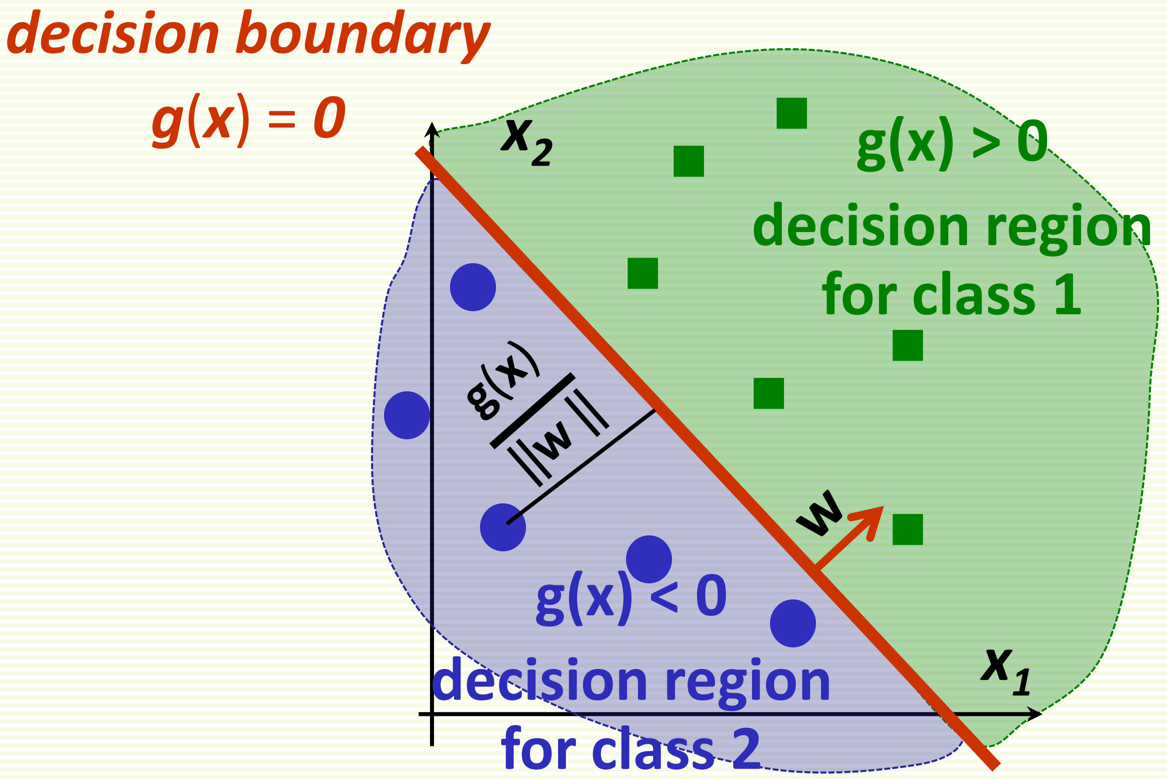
better boundary

- $\mathbf{f}(\mathbf{x},\mathbf{w}) = \text{sign}(\mathbf{g}(\mathbf{x},\mathbf{w})) = \text{sign}(\mathbf{w}_0+\mathbf{x}_1\mathbf{w}_1 + \ldots + \mathbf{x}_d\mathbf{w}_d)$
- Decision boundary is linear
- Find **w** that gives best separation of two classes with linear boundary
  - search for best $\mathbf{w} = [\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_d]$

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

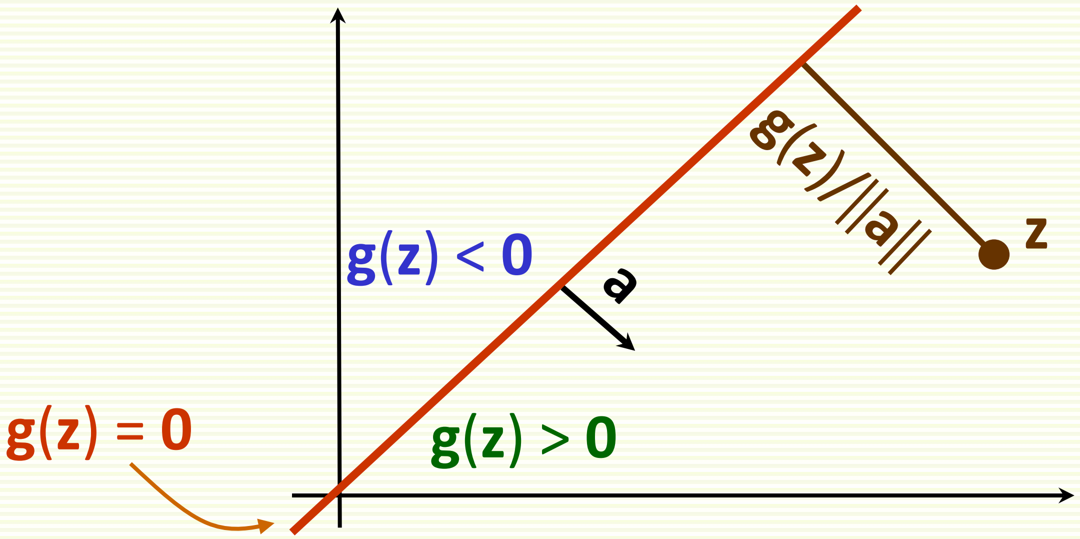
Fitting Parameters **w**

- Linear discriminant function $\mathbf{g(x,w)} = \mathbf{w}^t\mathbf{x} + \mathbf{w_0}$
- Can rewrite it $\mathbf{g(x,w)} = \underbrace{[\mathbf{w}_0 \quad \mathbf{w}^t]}_{\text{new weight vector } \boldsymbol{a}} \underbrace{\begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}}_{\text{new feature vector } \boldsymbol{z}} = \mathbf{a}^t\mathbf{z} = \mathbf{g(z,a)}$
- **z** is called augmented feature vector
- new problem equivalent to the old $\mathbf{g(z,a)} = \mathbf{a}^t\mathbf{z}$

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

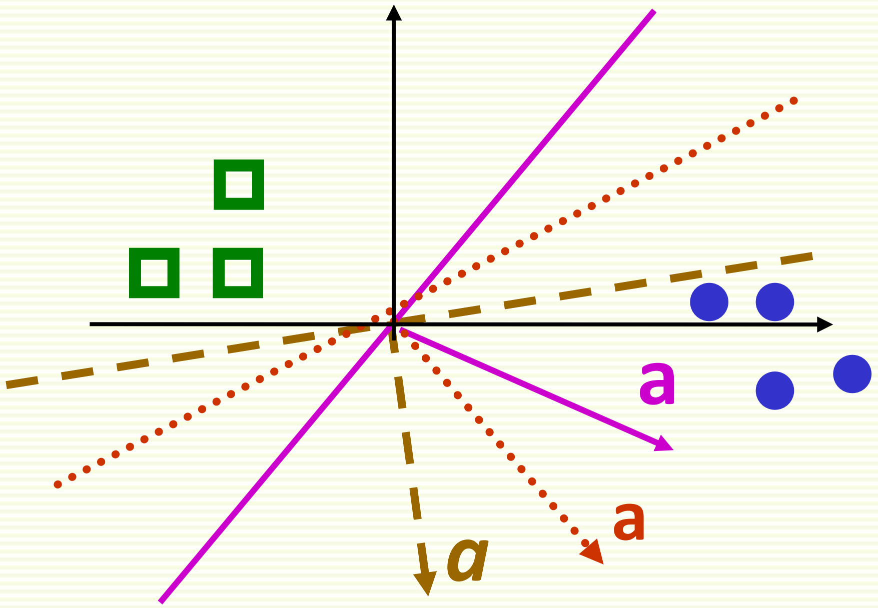
Augmented Feature Vector

- Feature augmenting simplifies notation
- Assume augmented feature vectors for the rest of lecture
    - given examples $\mathbf{x}^1, \ldots, \mathbf{x}^n$ convert them to augmented examples $\mathbf{z}^1, \ldots, \mathbf{z}^n$ by adding a new dimension of value 1
- $\mathbf{g(z,a) = a^t z}$
- $\mathbf{f(z,a) = sign(g(z,a))}$



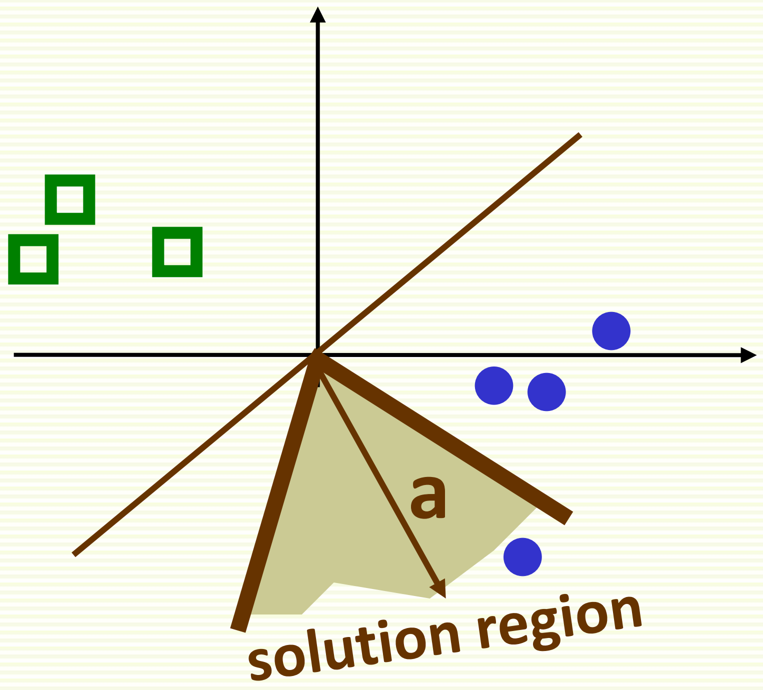
Solution Region

- If there is weight vector **a** that classifies all examples correctly, it is called a ***separating*** or ***solution*** vector
  - then there are infinitely many solution vectors **a**
  - then the original samples $\mathbf{x}^1, \ldots \mathbf{x}^n$ are also linearly separable



Solution Region

- Solution region: the set of all solution vectors **a**



Loss Function

- How to find solution vector **a**?
  - or, if no separating a exists, a good approximate solution vector **a**?
- Design a non-negative loss function **L**(**a**)
  - **L**(**a**) is small if **a** is good
  - **L**(**a**) is large if **a** is bad
- Minimize **L**(**a**) with gradient descent
- Usually design of **L**(**a**) has two steps
  1. design per-example loss $\mathbf{L}(\mathbf{f}(\mathbf{z}^i,\mathbf{a}),\mathbf{y}^i)$
     - penalizes for deviations of $\mathbf{f}(\mathbf{z}^i,\mathbf{a})$ from $\mathbf{y}^i$
  2. total loss adds up per-sample loss over all training examples

$$\mathbf{L}(\mathbf{a}) = \sum_{\mathbf{i}} \mathbf{L}\left(\mathbf{f}\left(\mathbf{z}^{\mathbf{i}},\mathbf{a}\right),\mathbf{y}^{\mathbf{i}}\right)$$

Loss Function, First Attempt

- Per-example loss function measures if error happens

$$L(f(z^i, a), y^i) = \begin{cases} 0 & \text{if } f(z^i, a) = y^i \quad ✓ \\ 1 & \text{otherwise} \quad ✗ \end{cases}$$

- Example

$$a = \begin{bmatrix} 2 \\ -3 \end{bmatrix}$$

$$z^1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad y^1 = 1$$

$$f(z^1, a) = \text{sign}(a^t z^1)$$
$$= \text{sign}(1 \cdot 2 - 3 \cdot 2)$$
$$= -1$$

$$L(f(z^1, a), y^1) = 1$$

$$z^2 = \begin{bmatrix} 1 \\ 4 \end{bmatrix} \qquad y^2 = -1$$

$$f(z^2, a) = \text{sign}(a^t z^2)$$
$$= \text{sign}(1 \cdot 2 - 3 \cdot 4)$$
$$= -1$$

$$L(f(z^2, a), y^2) = 0$$

Loss Function, First Attempt

- Per-example loss function measures if error happens

$$\mathbf{L}\left(\mathbf{f}\left(\mathbf{z}^{\mathbf{i}},\mathbf{a}\right),\mathbf{y}^{\mathbf{i}}\right)=\begin{cases}0 & \textbf{if } \mathbf{f}\left(\mathbf{z}^{\mathbf{i}},\mathbf{a}\right)=\mathbf{y}^{\mathbf{i}}\\ 1 & \textbf{otherwise}\end{cases}$$

- Total loss function

$$\mathbf{L}(\mathbf{a})=\sum_{\mathbf{i}}\mathbf{L}\left(\mathbf{f}\left(\mathbf{z}^{\mathbf{i}},\mathbf{a}\right),\mathbf{y}^{\mathbf{i}}\right)$$

- For previous example

$$\mathbf{a}=\begin{bmatrix}2\\-3\end{bmatrix}\qquad \begin{array}{l}\mathbf{L}\left(\mathbf{f}\left(\mathbf{z}^{1},\mathbf{a}\right),\mathbf{y}^{1}\right)=1\\ \mathbf{L}\left(\mathbf{f}\left(\mathbf{z}^{2},\mathbf{a}\right),\mathbf{y}^{2}\right)=0\end{array}$$

$$\mathbf{L}(\mathbf{a})=1+0=1$$

- Thus this loss function just counts the number of errors

Loss Function: First Attempt

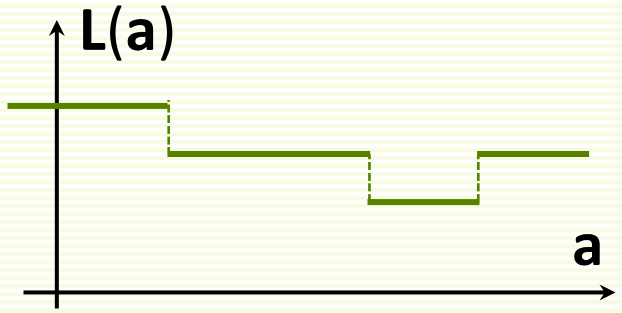
- Per-example loss

$$L\left(f\left(z^i,a\right),y^i\right)=\begin{cases}0 & \text{if } f\left(z^i,a\right)=y^i\\ 1 & \text{otherwise}\end{cases}$$

- Total loss

$$L(a)=\sum_i L\left(f\left(z^i,a\right),y^i\right)$$

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

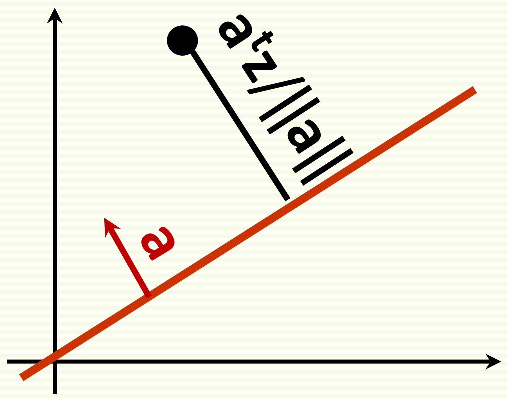


Perceptron Loss Function

- Different Loss Function: Perceptron Loss

$$L_p\left(f\left(z^i, a\right), y^i\right) = \begin{cases} 0 & \text{if } f\left(z^i, a\right) = y^i \\ -y^i\left(a^t z^i\right) & \text{otherwise} \end{cases}$$

- $L_p(a)$ is non-negative
  - positive misclassified example z^i
    - $a^t z^i < 0$
    - $y^i = 1$
    - $y^i(a^t z^i) < 0$
  - negative misclassified example z^i
    - $a^t z^i > 0$
    - $y^i = -1$
    - $y^i(a^t z^i) < 0$
  - if z^i is misclassified then $y^i(a^t z^i) < 0$
  - if z^i is misclassified then $-y^i(a^t z^i) > 0$



- $L_p(a)$ proportional to distance of misclassified example to boundary

Perceptron Loss Function

$$\mathbf{L_p}\left(\mathbf{f}\left(\mathbf{z}^i,\mathbf{a}\right),\mathbf{y}^i\right)=\begin{cases}0 & \text{if } \mathbf{f}\left(\mathbf{z}^i,\mathbf{a}\right)=\mathbf{y}^i\\ -\mathbf{y}^i\left(\mathbf{a}^t\mathbf{z}^i\right) & \text{otherwise}\end{cases}$$

- Example

$$\mathbf{a}=\begin{bmatrix}2\\-3\end{bmatrix}$$

$$\mathbf{z}^1=\begin{bmatrix}1\\2\end{bmatrix}\qquad \mathbf{y}^1=1$$

$$\mathbf{f}\left(\mathbf{z}^1,\mathbf{a}\right)=\mathbf{sign}\left(\mathbf{a}^t\mathbf{z}^1\right)$$

$$=\mathbf{sign}(1\cdot 2-3\cdot 2)$$

$$=\mathbf{sign}(-4)$$

$$=-1$$

$$\mathbf{L_p}\left(\mathbf{f}\left(\mathbf{z}^1,\mathbf{a}\right),\mathbf{y}^1\right)=4$$

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

$$\mathbf{f}\left(\mathbf{z}^2,\mathbf{a}\right)=\mathbf{sign}\left(\mathbf{a}^t\mathbf{z}^2\right)$$

$$=\mathbf{sign}(1\cdot 2-3\cdot 4)$$

$$=-1$$

$$\mathbf{L_p}\left(\mathbf{f}\left(\mathbf{z}^2,\mathbf{a}\right),\mathbf{y}^2\right)=0$$

- Total loss $\mathbf{L_p}(\mathbf{a})$ = 4 + 0 = 4

Perceptron Loss Function

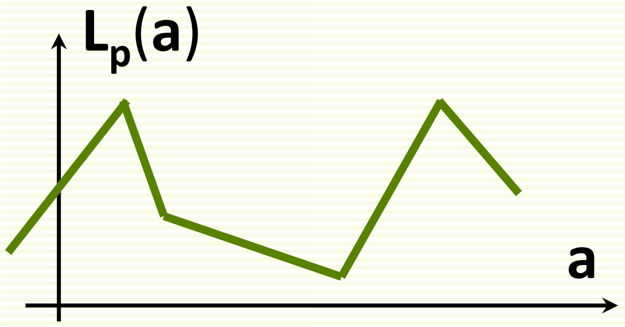
- Per-example loss

$$L_p\left(f(z^i,a),y^i\right)=\begin{cases}0 & \textbf{if } f(z^i,a)=y^i\\ -y^i\left(a^tz^i\right) & \textbf{otherwise}\end{cases}$$

- Total loss

$$L_p(a)=\sum_i L\left(f(z^i,a),y^i\right)$$

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



Optimizing with Gradient Descent

- Per-example loss

$$\mathbf{L_p}(\mathbf{f}(\mathbf{z}^i,\mathbf{a}),\mathbf{y}^i)=\begin{cases}0 & \textbf{if } \mathbf{f}(\mathbf{z}^i,\mathbf{a})=\mathbf{y}^i \\ -\mathbf{y}^i(\mathbf{a}^t\mathbf{z}^i) & \textbf{otherwise}\end{cases}$$

- Total loss

$$\mathbf{L_p}(\mathbf{a})=\sum_{\mathbf{i}}\mathbf{L}(\mathbf{f}(\mathbf{z}^i,\mathbf{a}),\mathbf{y}^i)$$

- Recall minimization with gradient descent, main step

$$\mathbf{x} = \mathbf{x} - \alpha\, \nabla\mathbf{J}(\mathbf{x})$$

- Gradient descent to minimize $\mathbf{L_p}(\mathbf{a})$, main step

$$\mathbf{a} = \mathbf{a} - \alpha\, \nabla\mathbf{L_p}(\mathbf{a})$$

- Need gradient vector $\nabla\mathbf{L_p}(\mathbf{a})$
  - has as many dimensions as dimension of **a**
  - if **a** has 3 dimensions, gradient $\nabla\mathbf{L_p}(\mathbf{a})$ has 3 dimensions

$$\mathbf{a}=\begin{bmatrix}2\\3\\1\end{bmatrix} \qquad \nabla\mathbf{L_p}(\mathbf{a})=\begin{bmatrix}\frac{\partial\mathbf{L_p}}{\partial\mathbf{a}_1}\\[2ex]\frac{\partial\mathbf{L_p}}{\partial\mathbf{a}_2}\\[2ex]\frac{\partial\mathbf{L_p}}{\partial\mathbf{a}_3}\end{bmatrix}$$

Optimizing with Gradient Descent

- Per-example loss

$$\mathbf{L_p}\left(\mathbf{f}\left(\mathbf{z^i},\mathbf{a}\right),\mathbf{y^i}\right)=\begin{cases}0 & \textbf{if } \mathbf{f}\left(\mathbf{z^i},\mathbf{a}\right)=\mathbf{y^i}\\ -\mathbf{y^i}\left(\mathbf{a^t z^i}\right) & \textbf{otherwise}\end{cases}$$

- Total loss

$$\mathbf{L_p}\left(\mathbf{a}\right)=\sum_{\mathbf{i}}\mathbf{L}\left(\mathbf{f}\left(\mathbf{z^i},\mathbf{a}\right),\mathbf{y^i}\right)$$

- Gradient descent to minimize $\mathbf{L_p(a)}$, main step

$$\mathbf{a}= \mathbf{a} - \alpha\, \nabla \mathbf{L_p(a)}$$

- Need gradient vector $\nabla\mathbf{L_p(a)}$

$$\nabla\mathbf{L_p}\left(\mathbf{a}\right)=\nabla\sum_{\mathbf{i}}\mathbf{L_p}\left(\mathbf{f}\left(\mathbf{z^i},\mathbf{a}\right),\mathbf{y^i}\right)=\sum_{\mathbf{i}}\nabla\mathbf{L_p}\left(\mathbf{f}\left(\mathbf{z^i},\mathbf{a}\right),\mathbf{y^i}\right)$$

per example gradient

$$\begin{bmatrix}\dfrac{\partial\mathbf{L_p}\left(\mathbf{f}\left(\mathbf{z^i},\mathbf{a}\right),\mathbf{y^i}\right)}{\partial\mathbf{a}_1}\\[2ex] \dfrac{\partial\mathbf{L_p}\left(\mathbf{f}\left(\mathbf{z^i},\mathbf{a}\right),\mathbf{y^i}\right)}{\partial\mathbf{a}_2}\\[2ex] \dfrac{\partial\mathbf{L_p}\left(\mathbf{f}\left(\mathbf{z^i},\mathbf{a}\right),\mathbf{y^i}\right)}{\partial\mathbf{a}_3}\end{bmatrix}$$

- Compute and add up per example gradient vectors

Per Example Loss Gradient

- Per-example loss has two cases

$$L_p\left(f\left(z^i,a\right),y^i\right)=\begin{cases}0 & \text{if } f\left(z^i,a\right)=y^i\\ -y^i\left(a^t z^i\right) & \text{otherwise}\end{cases}$$

- First case, $f(z^i,a) = y^i$

$$\nabla L_p\left(f\left(z^i,a\right),y^i\right)=\begin{cases}\begin{bmatrix}0\\0\\0\end{bmatrix} & \text{if } f\left(z^i,a\right)=y^i\\ ? & \text{otherwise}\end{cases}$$

- To save space, rewrite

$$\nabla L_p\left(f\left(z^i,a\right),y^i\right)=\begin{cases}0 & \text{if } f\left(z^i,a\right)=y^i\\ ? & \text{otherwise}\end{cases}$$

Per Example Loss Gradient

- Per-example loss has two cases

$$L_p(f(z^i,a),y^i) = \begin{cases} 0 & \text{if } f(z^i,a) = y^i \\ -y^i(a^t z^i) & \text{otherwise} \end{cases}$$

- Second case, $f(z^i,a) \neq y^i$

$$\nabla L_p(f(z^i,a),y^i) = \begin{bmatrix} \frac{\partial L}{\partial a_1}(-y^i(a^t z^i)) \\ \frac{\partial L}{\partial a_2}(-y^i(a^t z^i)) \\ \frac{\partial L}{\partial a_3}(-y^i(a^t z^i)) \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial a_1}(-y^i(a_1 z_1^i + a_2 z_2^i + a_3 z_3^i)) \\ \frac{\partial L}{\partial a_2}(-y^i(a_1 z_1^i + a_2 z_2^i + a_3 z_3^i)) \\ \frac{\partial L}{\partial a_3}(-y^i(a_1 z_1^i + a_2 z_2^i + a_3 z_3^i)) \end{bmatrix} = \begin{bmatrix} -y^i z_1^i \\ -y^i z_2^i \\ -y^i z_3^i \end{bmatrix}$$

$$= -y^i z^i$$

- Combining both cases, gradient for per-example loss

$$\nabla L_p(f(z^i,a),y^i) = \begin{cases} 0 & \text{if } f(z^i,a) = y^i \\ -y^i z^i & \text{otherwise} \end{cases}$$

Optimizing with Gradient Descent

- Gradient for per-example loss

$$\nabla L_p\left(f(z^i, a), y^i\right) = \begin{cases} 0 & \textbf{if } f(z^i, a) = y^i \\ -y^i z^i & \textbf{otherwise} \end{cases}$$

- Total gradient $\nabla L_p(a) = \sum_i \nabla L_p\left(f(z^i, a), y^i\right)$

- Simpler formula $\nabla L_p(a) = \sum_{\substack{\text{misclassified} \\ \text{examples } i}} -y^i z^i$

- Gradient decent update rule for $L_p(a)$

$$a = a + \alpha \sum_{\substack{\text{misclassified} \\ \text{examples } i}} y^i z^i$$

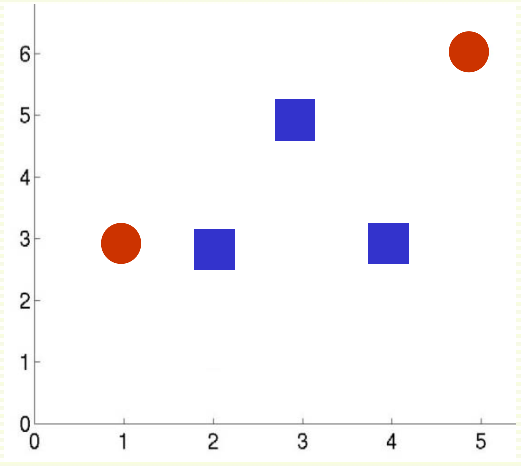
  - called **batch** because it is based on all examples
  - can be slow if number of examples is very large

Perceptron Loss Batch Example

- Examples

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

class 1 class 2



- Labels

$$\mathbf{y}^1 = 1 \quad \mathbf{y}^2 = 1 \quad \mathbf{y}^3 = 1 \quad \mathbf{y}^4 = -1 \quad \mathbf{y}^5 = -1$$

- Add extra feature

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

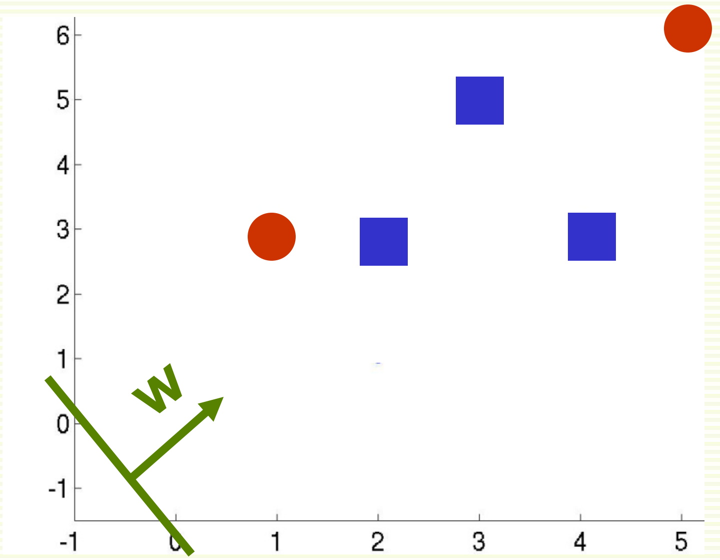
$$\mathbf{Z} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$

- Pile all examples as rows in matrix **Z**
- Pile all labels into column vector **Y**

Perceptron Loss Batch Example

- Examples in **Z**, labels in **Y**

$$\mathbf{Z} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$

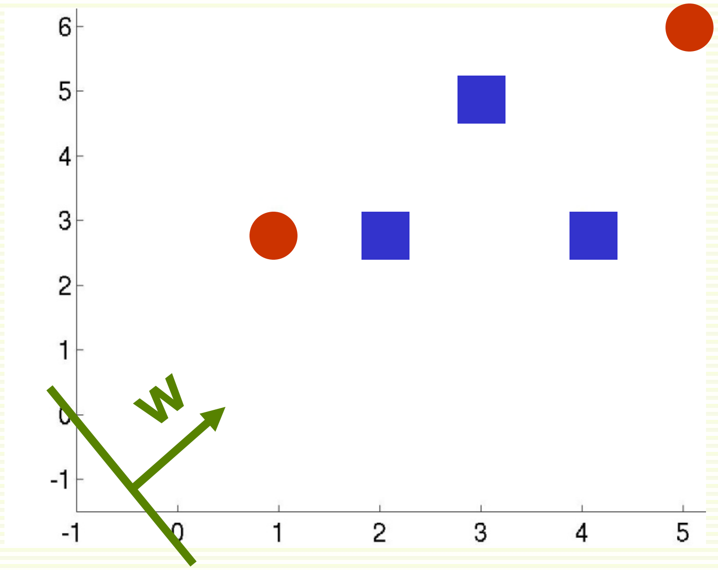


- Initial weights $\mathbf{a} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

- This is line $\mathbf{x}_1 + \mathbf{x}_2 + 1 = 0$

Perceptron Loss Batch Example

$$\mathbf{Z} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \qquad \mathbf{a} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$



- Perceptron Batch

$$\mathbf{a} = \mathbf{a} + \alpha \sum_{\substack{\text{misclassified} \\ \text{examples } i}} \mathbf{y}^i \mathbf{z}^i$$

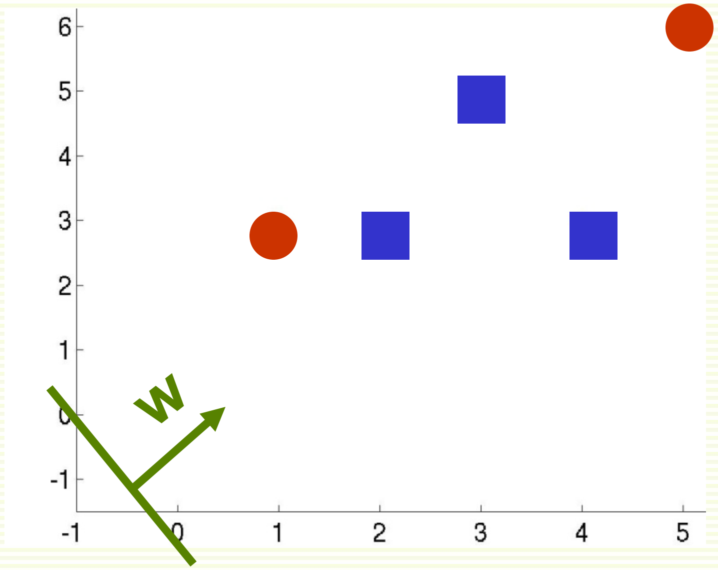
- Let us use learning rate α = 0.2

$$\mathbf{a} = \mathbf{a} + 0.2 \sum_{\substack{\text{misclassified} \\ \text{examples } i}} \mathbf{y}^i \mathbf{z}^i$$

- Sample misclassified if $\mathbf{y}(\mathbf{a}^t\mathbf{z}) < 0$

Perceptron Loss Batch Example

$$\mathbf{Z} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \qquad \mathbf{a} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$



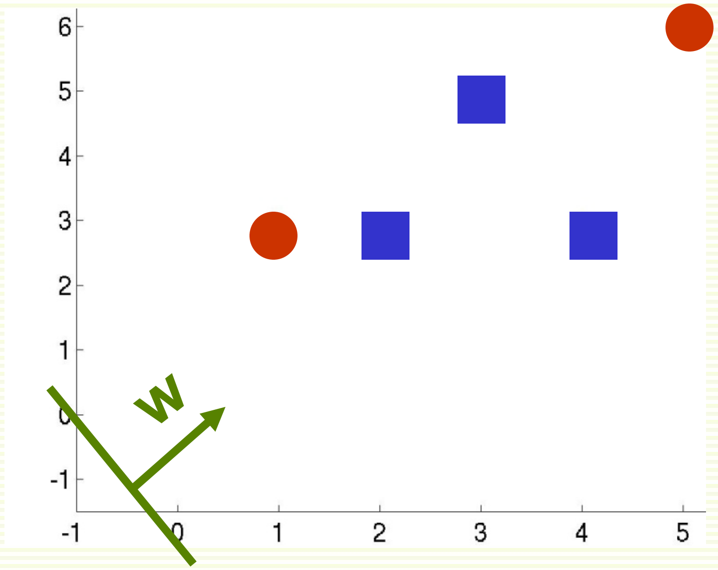
- Sample misclassified if $\mathbf{y}(\mathbf{a}^t\mathbf{z}) < 0$
- Find all misclassified samples with one line in matlab
- Could have for loop to compute $\mathbf{a}^t\mathbf{z}$
- For $\mathbf{i} = 1$

$$\mathbf{y}^1\mathbf{a}^t\mathbf{z}^1 = 1 \cdot \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = 6 > 0 \quad ✓$$

- Repeat for $\mathbf{i}$ = 2, 3, 4, 5

Perceptron Loss Batch Example

$$\mathbf{Z} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \qquad \mathbf{a} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$

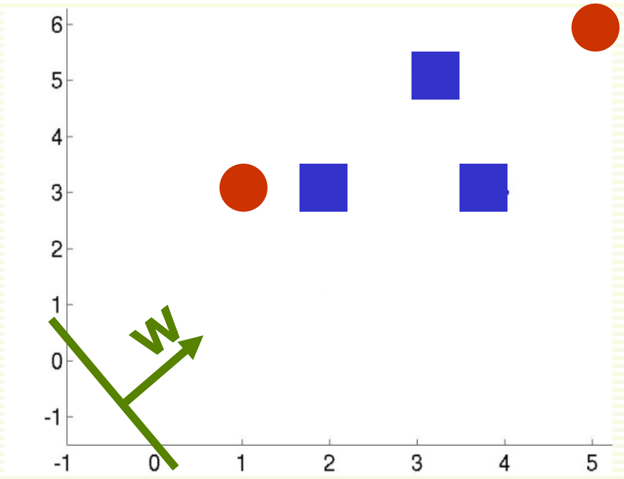


- Sample misclassified if $\mathbf{y}(\mathbf{a}^t\mathbf{z}) < 0$
- Find all misclassified samples with one line in matlab
- Can compute $\mathbf{a}^t\mathbf{z}$ for all samples

$$\begin{bmatrix} \mathbf{a}^t\mathbf{z}^1 \\ \mathbf{a}^t\mathbf{z}^2 \\ \mathbf{a}^t\mathbf{z}^3 \\ \mathbf{a}^t\mathbf{z}^4 \\ \mathbf{a}^t\mathbf{z}^5 \end{bmatrix} = \mathbf{Z} * \mathbf{a} = \begin{bmatrix} 6 \\ 8 \\ 9 \\ 5 \\ 12 \end{bmatrix}$$

Perceptron Loss Batch Example

$$\mathbf{Z} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$



- Sample misclassified if $\mathbf{y}(\mathbf{a}^t\mathbf{z}) < 0$
- Can compute $\mathbf{y}(\mathbf{a}^t\mathbf{z})$ for all samples in one line

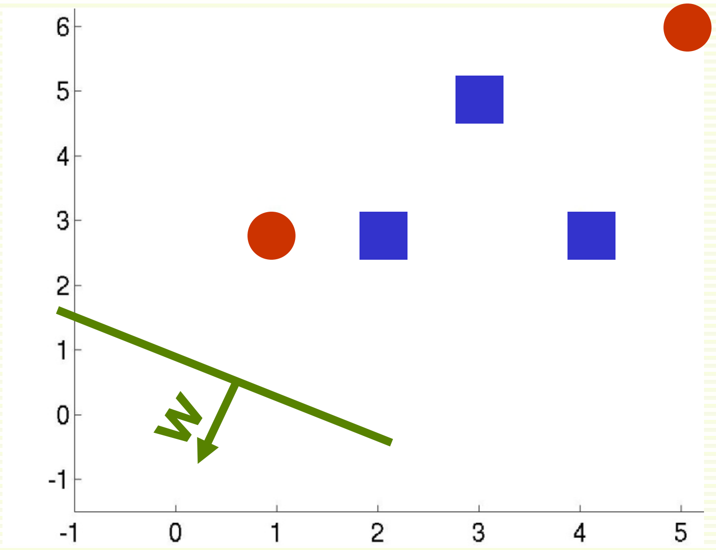
$$\begin{bmatrix} \mathbf{y}^1(\mathbf{a}^t\mathbf{z}^1) \\ \mathbf{y}^2(\mathbf{a}^t\mathbf{z}^2) \\ \mathbf{y}^3(\mathbf{a}^t\mathbf{z}^3) \\ \mathbf{y}^4(\mathbf{a}^t\mathbf{z}^4) \\ \mathbf{y}^5(\mathbf{a}^t\mathbf{z}^5) \end{bmatrix} = \mathbf{Y} * .(\mathbf{Z} * \mathbf{a}) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} * . \begin{bmatrix} 6 \\ 8 \\ 9 \\ 5 \\ 12 \end{bmatrix} = \begin{bmatrix} 6 \\ 8 \\ 9 \\ -5 \\ -12 \end{bmatrix} \begin{matrix} \\ \\ \\ \text{✗} \\ \text{✗} \end{matrix}$$

Total loss is
$L(\mathbf{a})$ = 5+12 = 17

- Per example loss is $$\mathbf{L_p}\left(\mathbf{f}(\mathbf{z}^i, \mathbf{a}), \mathbf{y}^i\right) = \begin{cases} 0 & \textbf{if } \mathbf{f}(\mathbf{z}^i, \mathbf{a}) = \mathbf{y}^i \\ -\mathbf{y}^i(\mathbf{a}^t\mathbf{z}^i) & \textbf{otherwise} \end{cases}$$

Perceptron Loss Batch Example

$$\mathbf{Z} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \qquad \mathbf{a} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$



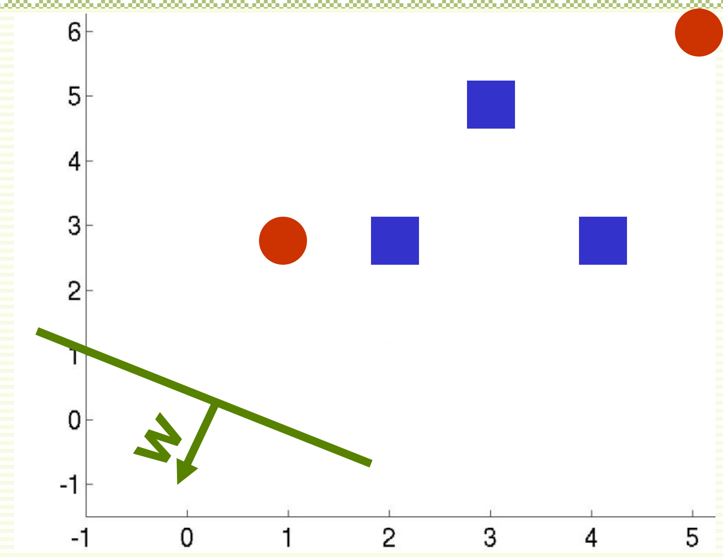
- Samples 4 and 5 misclassified
- Perceptron Batch rule update $\mathbf{a} = \mathbf{a} + 0.2 \sum_{\substack{\text{misclassified} \\ \text{examples } i}} \mathbf{y}^i \mathbf{z}^i$

$$\mathbf{a} = \mathbf{a} + 0.2\left(-1 \cdot \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix} - 1 \cdot \begin{bmatrix} 1 \\ 5 \\ 6 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0.2 \\ 0.2 \\ 0.6 \end{bmatrix} - \begin{bmatrix} 0.2 \\ 1 \\ 1.2 \end{bmatrix} = \begin{bmatrix} 0.6 \\ -0.2 \\ -0.8 \end{bmatrix}$$

- This is line $-0.2\mathbf{x}_1 - 0.8\,\mathbf{x}_2 + 0.6 = 0$

Perceptron Loss Batch Example

$$\mathbf{Z} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} 0.6 \\ -0.2 \\ -0.8 \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$



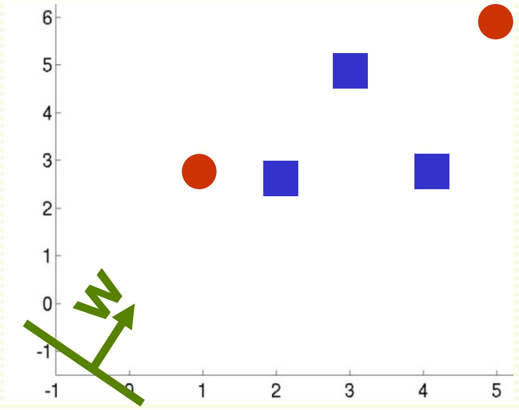
- Sample misclassified if $\mathbf{y}(\mathbf{a}^t\mathbf{z}) < 0$
- Find all misclassified samples

$$(\mathbf{Z} * \mathbf{a}) .* \mathbf{Y} = \begin{array}{l} \left[\begin{array}{r} -2.2 \\ -2.6 \\ -4.0 \\ 2 \\ 5.2 \end{array}\right] \begin{array}{l} ✗ \\ ✗ \\ ✗ \\ \\ \\ \end{array} \end{array}$$

- Total loss is **L**(**a**) = 2.2 + 2.6 +4= 8.8
  - previous loss was 17 with 2 misclassified examples

Perceptron Loss Batch Example

$$\mathbf{Z}=\begin{bmatrix}1&2&3\\1&4&3\\1&3&5\\1&1&3\\1&5&6\end{bmatrix}\quad \mathbf{a}=\begin{bmatrix}0.6\\-0.2\\-0.8\end{bmatrix}\quad \mathbf{Y}=\begin{bmatrix}1\\1\\1\\-1\\-1\end{bmatrix}\qquad (\mathbf{Z}*\mathbf{a}).*\mathbf{Y}=\begin{bmatrix}-2.2\\-2.6\\-4.0\\2\\5.2\end{bmatrix}\begin{array}{c}\times\\\times\\\times\\ \\ \end{array}$$



- Perceptron Batch rule update

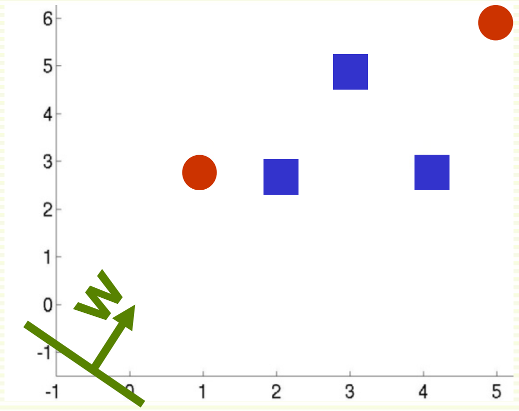
$$\mathbf{a}=\mathbf{a}+0.2\sum_{\substack{\textbf{misclassified}\\\textbf{examples } i}}\mathbf{y}^i\mathbf{z}^i$$

$$\mathbf{a}=\mathbf{a}+0.2\left(1\cdot\begin{bmatrix}1\\2\\3\end{bmatrix}+1\cdot\begin{bmatrix}1\\4\\3\end{bmatrix}+1\cdot\begin{bmatrix}1\\3\\5\end{bmatrix}\right)=\begin{bmatrix}0.6\\-0.2\\-0.8\end{bmatrix}+\begin{bmatrix}0.2\\0.4\\0.6\end{bmatrix}+\begin{bmatrix}0.2\\0.8\\0.6\end{bmatrix}+\begin{bmatrix}0.2\\0.6\\1\end{bmatrix}=\begin{bmatrix}1.2\\1.6\\1.4\end{bmatrix}$$

- This is line $1.6\mathbf{x}_1+1.4\,\mathbf{x}_2+1.2=0$

Perceptron Loss Batch Example

$$\mathbf{Z} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \qquad \mathbf{a} = \begin{bmatrix} 1.2 \\ 1.6 \\ 1.4 \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$



- Sample misclassified if $\mathbf{y}(\mathbf{a}^t\mathbf{z}) < 0$
- Find all misclassified samples

$$(\mathbf{Z} * \mathbf{a}).* \mathbf{Y} = \begin{bmatrix} 8.6 \\ 11.8 \\ 13.0 \\ -7 \\ -17.6 \end{bmatrix} \begin{matrix} \\ \\ \\ \times \\ \times \end{matrix}$$

- Total loss is **L**(**a**) = 7 + 17.6 = 24.6
  - previous loss was 8.8 with 3 misclassified examples
  - loss went up, means learning rate of 0.2 is too high

Perceptron Single Sample Gradient Descent

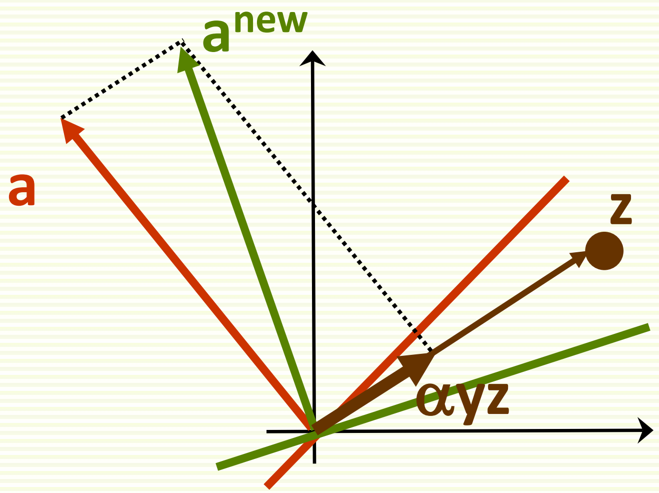
- Batch Perceptron can be slow to converge if lots of examples
- **Single sample** optimization
  - update weights **a** as soon as possible, after seeing 1 example
- One iteration (epoch)
  - go over all examples, as soon as find misclassified example, update

$$\mathbf{a} = \mathbf{a} + \alpha \cdot \mathbf{y}\,\mathbf{z}$$

  - **z** is misclassified example, **y** is its label
- Geometric intuition
  - **z** misclassified by **a** means

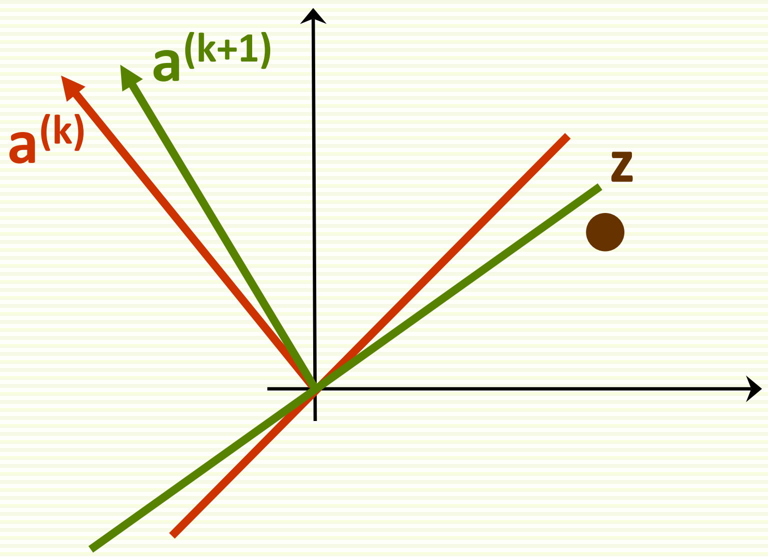
$$\mathbf{a}^t \mathbf{y}\mathbf{z} \le 0$$

  - **z** is on the wrong side of decision boundary
  - adding $\alpha \cdot \mathbf{y}\,\mathbf{z}$ moves decision boundary in the right direction
  - Illustration for positive example **z**
- Best to go over examples in random order

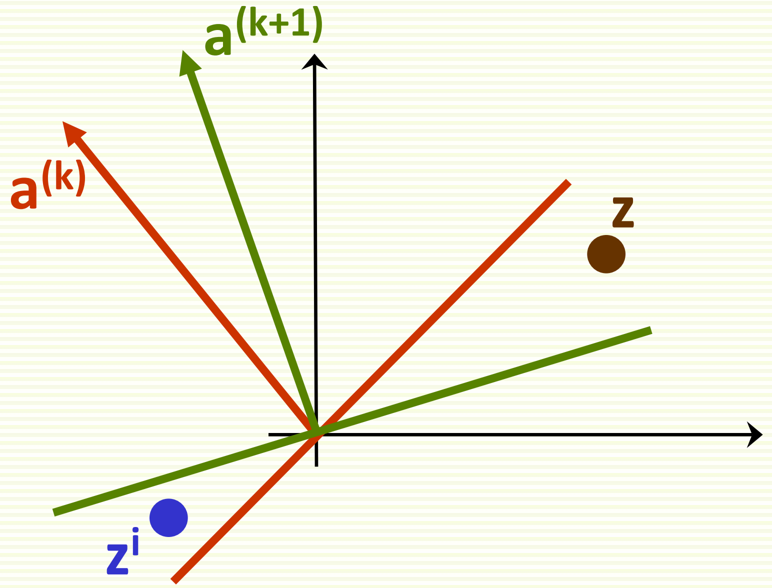


Perceptron Single Sample Rule

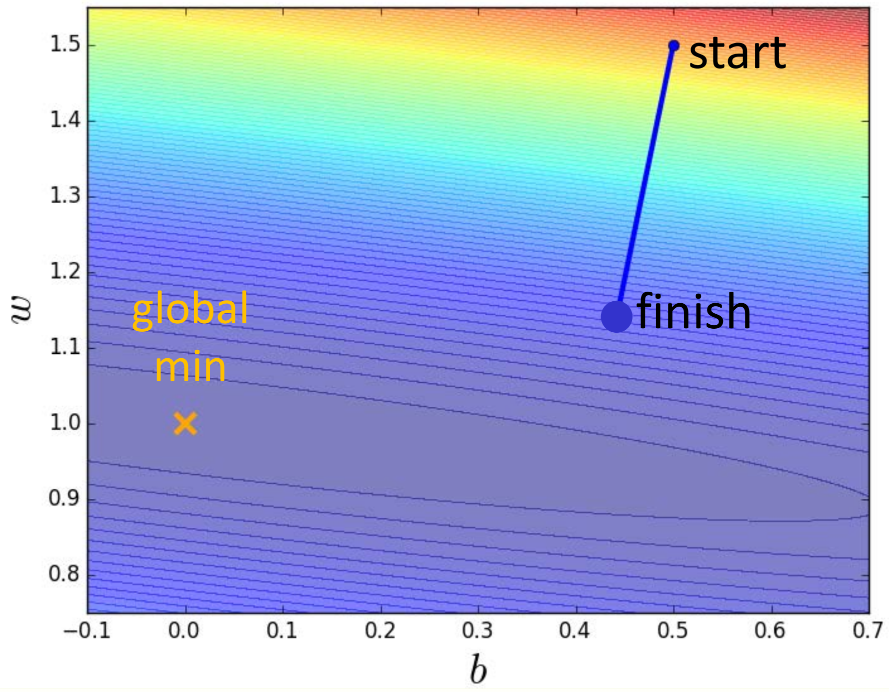
if α is too small, **z** is still misclassified



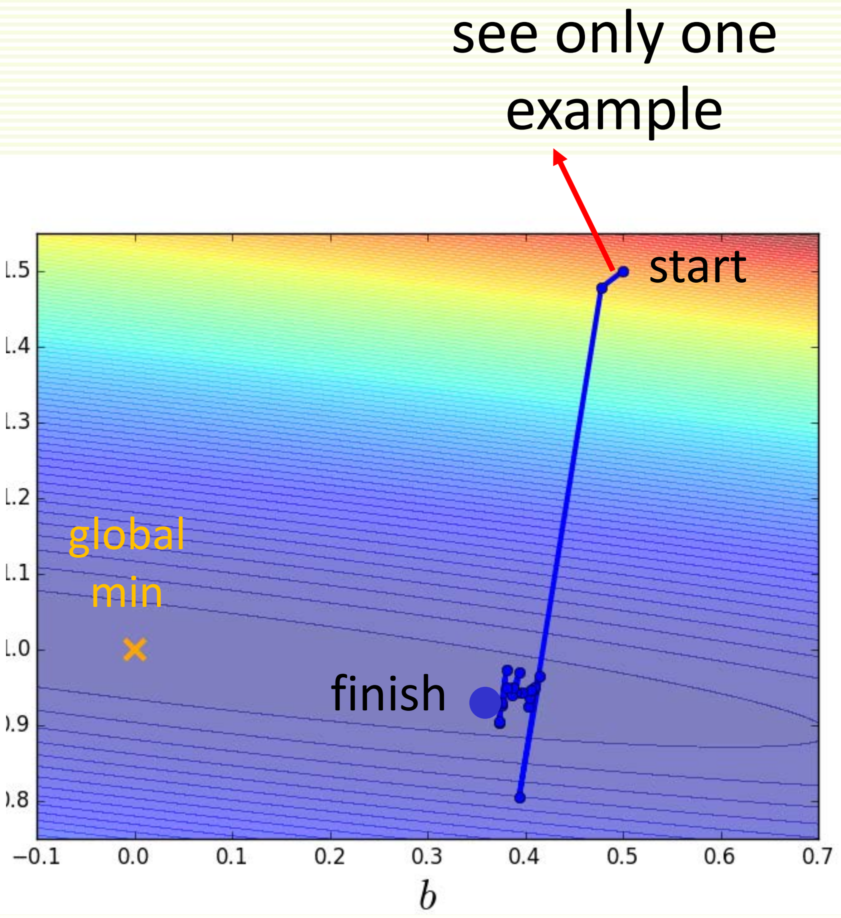
if α is too large, previously correctly classified sample $\mathbf{z}^{\mathbf{i}}$ is now misclassified



Batch Size: Loss Surface Illustration



Batch Gradient Descent, one iteration



Single sample gradient descent, one iteration

Perceptron Single Sample Rule Example

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

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

Perceptron Single Sample Rule Example

- Convert attributes to numerical values

	features				y
name	***good attendance?***	***tall?***	***sleeps in class?***	***chews gum?***	
Jane	1	1	-1	-1	1
Steve	1	1	1	1	-1
Mary	-1	-1	-1	1	-1
Peter	1	-1	1	1	1

Augment Feature Vector

	features					**y**
name	***extra***	***good attendance?***	***tall?***	***sleeps in class?***	***chews gum?***	
Jane	*1*	1	1	-1	-1	1
Steve	*1*	1	1	1	1	-1
Mary	*1*	-1	-1	-1	1	-1
Peter	*1*	1	-1	1	1	1

- convert samples $\mathbf{x}^1,\ldots,\mathbf{x}^n$ to augmented samples $\mathbf{z}^1,\ldots,\mathbf{z}^n$ by adding a new dimension of value 1

Apply Single Sample Rule

	features					y
name	***extra***	***good attendance?***	***tall?***	***sleeps in class?***	***chews gum?***	
Jane	*1*	1	1	-1	-1	1
Steve	*1*	1	1	1	1	-1
Mary	*1*	-1	-1	-1	1	-1
Peter	*1*	1	-1	1	1	1

- Set fixed learning rate to $\alpha = 1$
- Gradient descent with single sample rule
  - visit examples in random order
  - example misclassified if $\mathbf{y}(\mathbf{a}^t\mathbf{z}) < 0$
  - when misclassified example **z** found, update $\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \mathbf{yz}$

Apply Single Sample Rule

- initial weights $\mathbf{a}^{(1)} = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}$

- for simplicity, we will visit all samples sequentially
- example misclassified if $\mathbf{y}(\mathbf{a}^t\mathbf{z}) < 0$

name	**y**	**y(a*ᵗ*z)**	***misclassified?***
Jane	1	0.25*1+0.25*1+0.25*1+0.25*(-1)+0.25*(-1) > 0	no
Steve	-1	-1 * (0.25*1+0.25*1+0.25*1+0.25*1+0.25*1) < 0	yes

- new weights $\mathbf{a}^{(2)} = \mathbf{a}^{(1)} + \mathbf{yz} = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.75 \\ 0.75 \\ 0.75 \\ 0.75 \\ 0.75 \end{bmatrix}$

Apply Single Sample Rule

$$\mathbf{a}^{(2)} = \begin{bmatrix} 0.75 \\ 0.75 \\ 0.75 \\ 0.75 \\ 0.75 \end{bmatrix}$$

name	**y**	**y(a*ᵗ*z)**	***misclassified?***
Mary	-1	-1*(-0.75*1-0.75*(-1) -0.75 *(-1) -0.75 *(-1) -0.75*1) < 0	*yes*

- new weights $\mathbf{a}^{(3)} = \mathbf{a}^{(2)} + \mathbf{yz} = \begin{bmatrix} 0.75 \\ 0.75 \\ 0.75 \\ 0.75 \\ 0.75 \end{bmatrix} - \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1.75 \\ 0.25 \\ 0.25 \\ 0.25 \\ -1.75 \end{bmatrix}$

Apply Single Sample Rule

$$\mathbf{a}^{(3)} = \begin{bmatrix} -1.75 \\ 0.25 \\ 0.25 \\ 0.25 \\ -1.75 \end{bmatrix}$$

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

- new weights $\mathbf{a}^{(4)} = \mathbf{a}^{(3)} + \mathbf{yz} = \begin{bmatrix} -1.75 \\ 0.25 \\ 0.25 \\ 0.25 \\ -1.75 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.75 \\ 1.25 \\ -0.75 \\ -0.75 \\ -0.75 \end{bmatrix}$

Single Sample Rule: Convergence

$$\mathbf{a}^{(4)} = \begin{bmatrix} -0.75 \\ 1.25 \\ -0.75 \\ -0.75 \\ -0.75 \end{bmatrix}$$

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

Single Sample Rule: Convergence

$$\mathbf{a}^{(4)} = \begin{bmatrix} -0.75 \\ 1.25 \\ -0.75 \\ -0.75 \\ -0.75 \end{bmatrix}$$

- Discriminant function is

$$\mathbf{g}(\mathbf{z}) = -0.75\,\mathbf{z}_0 + 1.25\mathbf{z}_1 - 0.75\mathbf{z}_2 - 0.75\mathbf{z}_3 - 0.75\mathbf{z}_4$$

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

$$\mathbf{g}(\mathbf{x}) = 1.25x_1 - 0.75\mathbf{x}_2 - 0.75\mathbf{x}_3 - 0.75\mathbf{x}_4 - 0.75$$

Final Classifier

- Trained LDF: $\mathbf{g}(\mathbf{x}) = 1.25x_1 - 0.75\mathbf{x}_2 - 0.75\mathbf{x}_3 - 0.75\mathbf{x}_4 - 0.75$
- Leads to classifier:

$$\underset{\substack{\swarrow \\ \text{good attendance}}}{1.25x_1} - \underset{\substack{\swarrow \\ \text{tall}}}{0.75\mathbf{x}_2} - \underset{\substack{\searrow \\ \text{sleeps in class}}}{0.75\mathbf{x}_3} - \underset{\substack{\searrow \\ \text{chews gum}}}{0.75\mathbf{x}_4} > 0.75 \Rightarrow \text{grade } \mathbf{A}$$

- This is just *one* possible solution vector

- With $\mathbf{a}^{(1)}=[0,0.5, 0.5, 0, 0]$, solution is $[-1,1.5, -0.5, -1, -1]$

$$1.5x_1 - 0.5\mathbf{x}_2 - \mathbf{x}_3 - \mathbf{x}_4 > 1 \Rightarrow \text{grade } \mathbf{A}$$

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

Convergence under Perceptron Loss

1. Classes are linearly separable
   - with fixed learning rate, both single sample and batch versions 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

- 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, may concentrates more than necessary on any isolated training examples (those could be noise)
- Converges faster

Mini-Batch

- Update weights after seeing *batchSize* examples
- Faster convergence than the Batch rule
- Less susceptible to noisy examples than Single Sample Rule

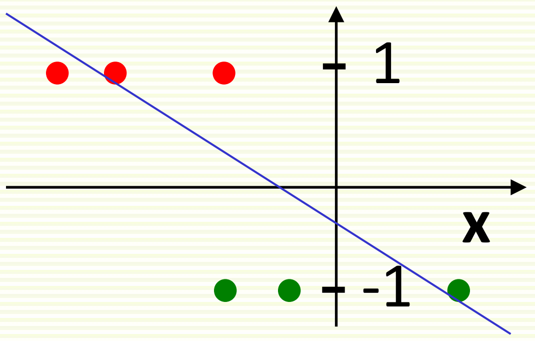
Linear Classifier: Quadratic Loss

- Other loss functions are possible for our classifier

$$\mathbf{f}(\mathbf{x}^i,\mathbf{w}) = \mathbf{sign}(\mathbf{x}^t\mathbf{w}^i)$$

- Quadratic per-example loss

$$\mathbf{L}_p(\mathbf{f}(\mathbf{x}^i,\mathbf{w}),\mathbf{x}^i) = \frac{1}{2}(\mathbf{y}^i - \mathbf{w}^t\mathbf{x}^i)^2$$



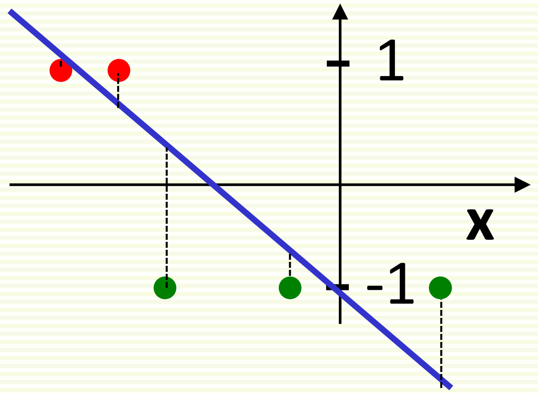
- Total loss $\mathbf{L}_p(\mathbf{w}) = \sum_i \mathbf{L}_p(\mathbf{f}(\mathbf{x}^i,\mathbf{w}),\mathbf{y}^i)$
- This is just standard line fitting (linear regression)
- Can find optimal weight **a** analytically with least squares
  - expensive for large problems
- Gradient descent more efficient for a larger problem

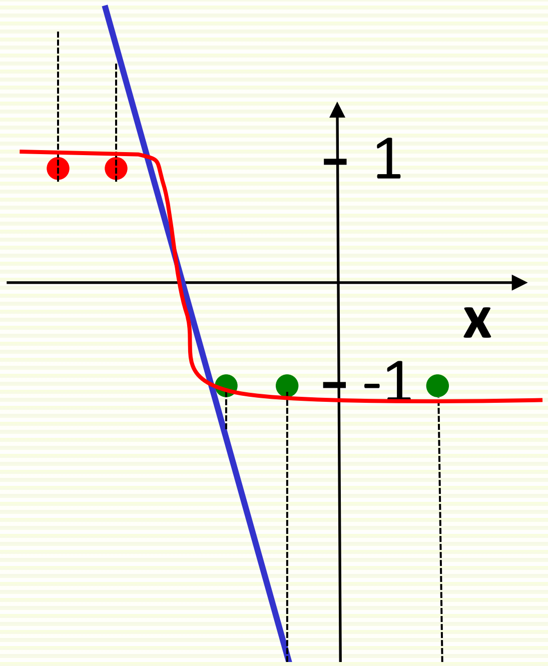
$$\nabla\mathbf{L}_p(\mathbf{w}) = -\sum_i(\mathbf{y}^i - \mathbf{w}^t\mathbf{x}^i)\mathbf{x}^i$$

- Batch update rule $\mathbf{w} = \mathbf{w} + \alpha\sum_i(\mathbf{y}^i - \mathbf{w}^t\mathbf{x}^i)\mathbf{x}^i$

Linear Classifier: Quadratic Loss

- Quadratic loss is an inferior choice for classification





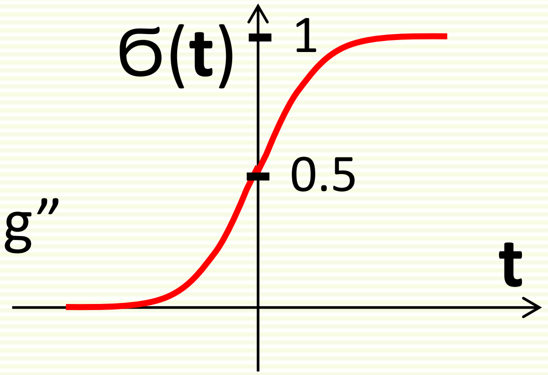
- Optimal classifier under quadratic loss
  - smallest squared errors
  - one sample misclassified

- Classifier found with Perceptron loss
  - huge squared errors
  - all samples classified correctly

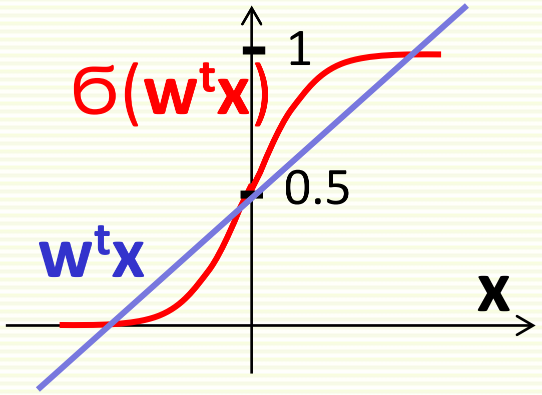
- Idea: instead of trying to get $\mathbf{w}^t\mathbf{x}$ close to $\mathbf{y}$, use some differentiable function $\boldsymbol{\sigma}(\mathbf{w}^t\mathbf{x})$ with "squished range", and try to get $\boldsymbol{\sigma}(\mathbf{w}^t\mathbf{x})$ close to $\mathbf{y}$

Linear Classifier: Logistic Regression

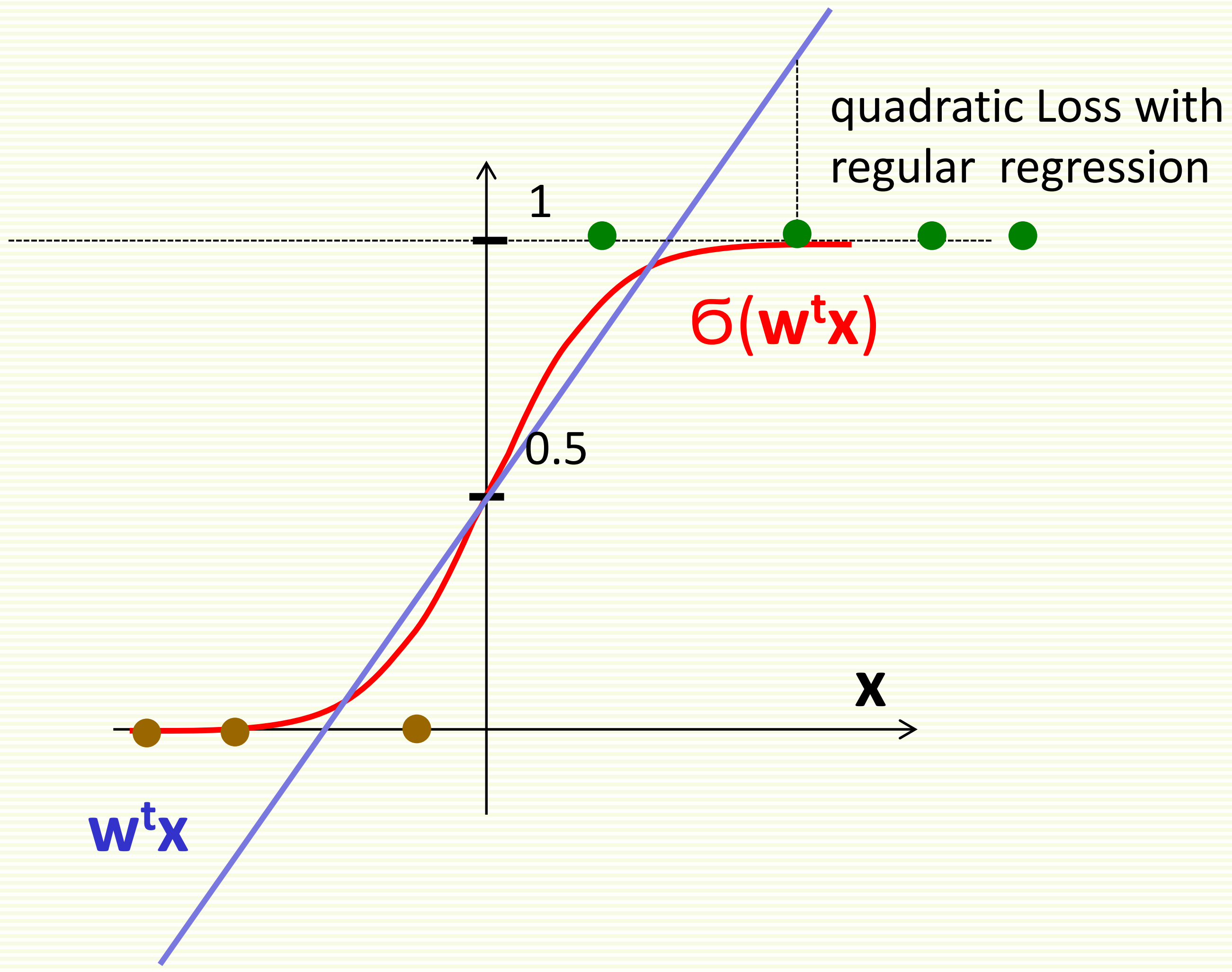
- denote classes with 1 and 0 now
  - $\mathbf{y}^i = 1$ for positive class, $\mathbf{y}^i = 0$ for negative
- Use logistic sigmoid function Ϭ(t) for "squishing"
- Discriminant function $\mathbf{g}(\mathbf{x},\mathbf{w}) = \mathbf{w}^T\mathbf{x}$
- $\sigma(\mathbf{g}(\mathbf{x},\mathbf{w})) = \sigma(\mathbf{w}^T\mathbf{x})$
- As before
  - decide class 1 if $\mathbf{w}^T\mathbf{x} > 0$
  - decide class 0 if $\mathbf{w}^T\mathbf{x} < 0$
- Equivalently, in terms of sigmoid function
  - Decide class 1 if $\sigma(\mathbf{w}^T\mathbf{x}) > 0.5$
  - Decide class 0 if $\sigma(\mathbf{w}^T\mathbf{x}) < 0.5$



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



Logistic Regression vs. Regresson



- Despite the name, logistic regression is used for classification, not regression

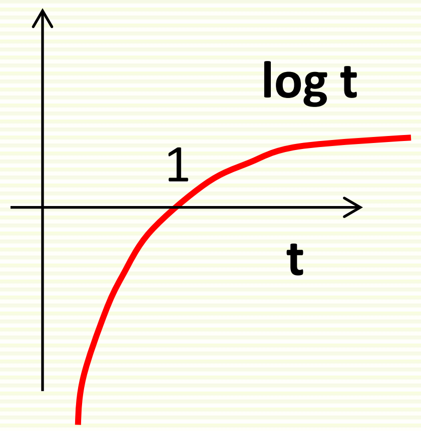
Logistic Regression: Loss Function

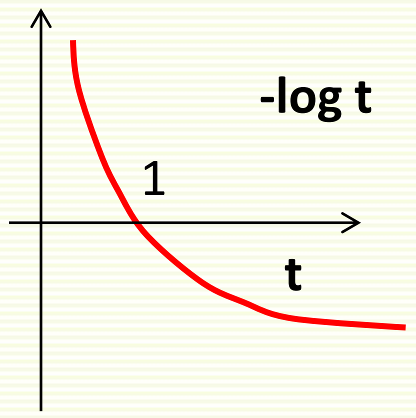
- Could use $(\mathbf{y}^i - \sigma(\mathbf{w}^t\mathbf{x}))^2$ as per-example loss function
- Instead use a different loss
  - if example $\mathbf{x}$ has label 1, want $\sigma(\mathbf{w}^T\mathbf{x})$ close to 1, define loss as

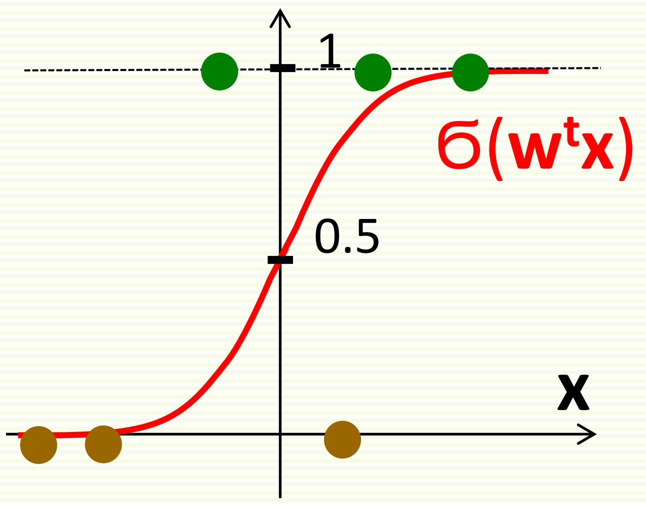
$$-\log[\sigma(\mathbf{w}^T\mathbf{x})]$$

  - if example $\mathbf{x}$ has label 0, want $\sigma(\mathbf{w}^T\mathbf{x})$ close to 0, define loss as

$$-\log[1-\sigma(\mathbf{w}^T\mathbf{x})]$$





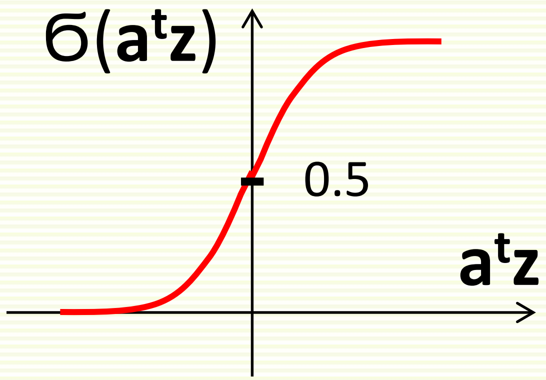


Logistic Regression: Loss Function

- Per-example loss function
  - if example **x** has label 1, loss is

    $-\log [\sigma(\mathbf{a}^T\mathbf{z})]$
  - if example **x** has label 0, loss is

    $-\log [1-\sigma(\mathbf{a}^T\mathbf{z})]$



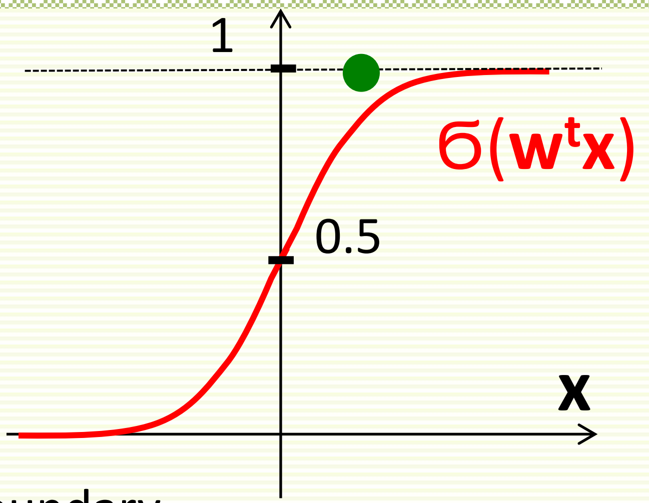
- Total loss is sum over per-example losses
- Convex, can be optimized exactly with gradient descent
- Gradient descent batch update rule

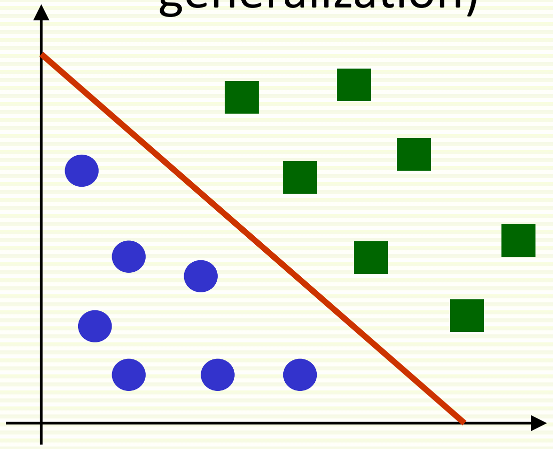
$$\mathbf{a} = \mathbf{a} + \alpha \sum_{i} \left(\mathbf{y}^i - \sigma\left(\mathbf{a}^t \mathbf{z}^i\right)\right) \mathbf{z}^i$$

- Logistic Regression has interesting probabilistic interpretation
  - **P**(class 1) = $\sigma(\mathbf{w}^T\mathbf{x})$
  - **P**(class 0) = 1 - **P**(class 1)
  - Therefore loss function is -log **P**(**y**) (negative log-likelihood)
    - standard objective in statistics

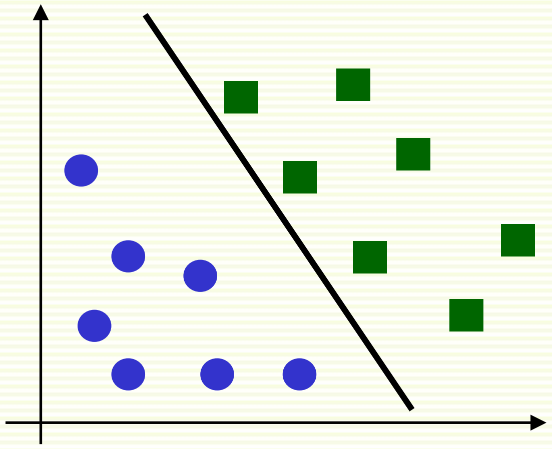
Logistic Regression vs. Perceptron

- Green example classified correctly, but close to decision boundary
  - Suppose $\mathbf{w}^t\mathbf{x}$ = 0.8 for green example
  - classified correctly, no loss under Perceptron
  - loss of $-\log(\sigma(0.8)) = 0.37$ under logistic regression
  - Logistic Regression (LR) encourages decision boundary move away from any training sample
  - may work better for new samples (better generalization)

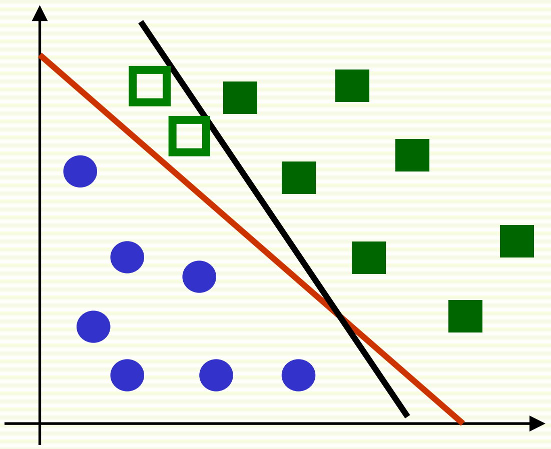




- zero Perceptron loss
- smaller LR loss



- zero Perceptron loss
- larger LR loss

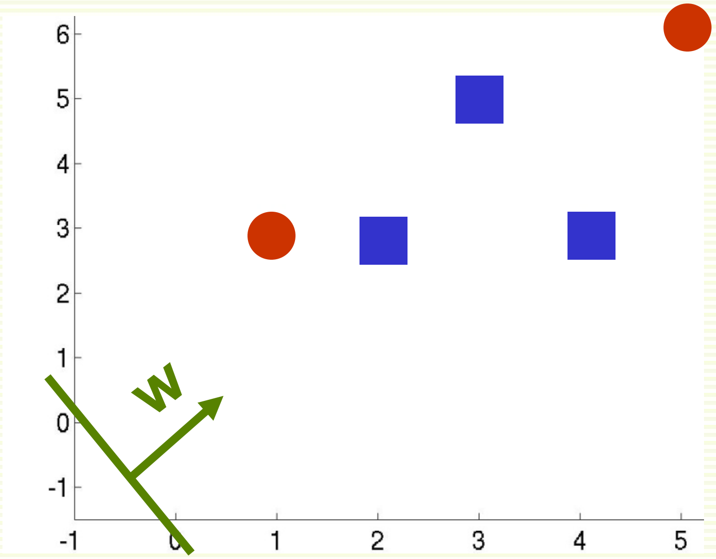


- red classifier works better for new data

Linear Classifier: Logistic Regression

- Examples in **Z**, labels in **Y**

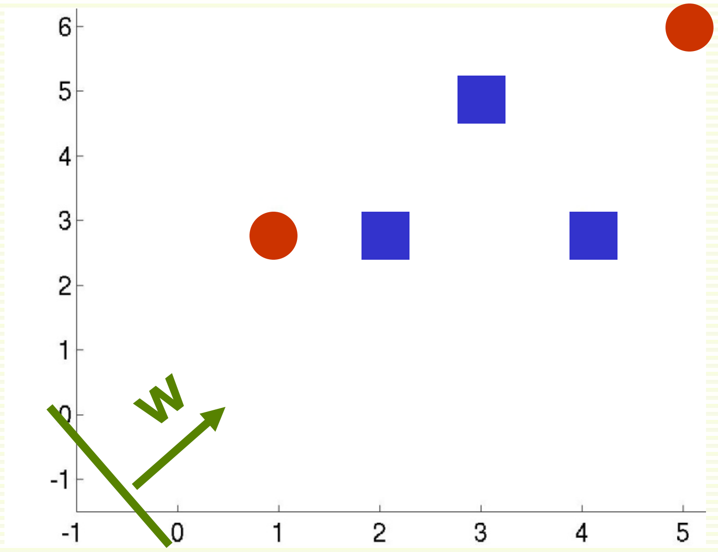
$$\mathbf{Z} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$



- Batch Logistic Regression with learning rate α=1
- Initial weights $\mathbf{a} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$
- This is line $\mathbf{x}_1 + \mathbf{x}_2 + 1 = 0$

Linear Classifier: Logistic Regression

$$\mathbf{Z} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \qquad \mathbf{a} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$



- Logistic Regression Batch rule update with α = 1

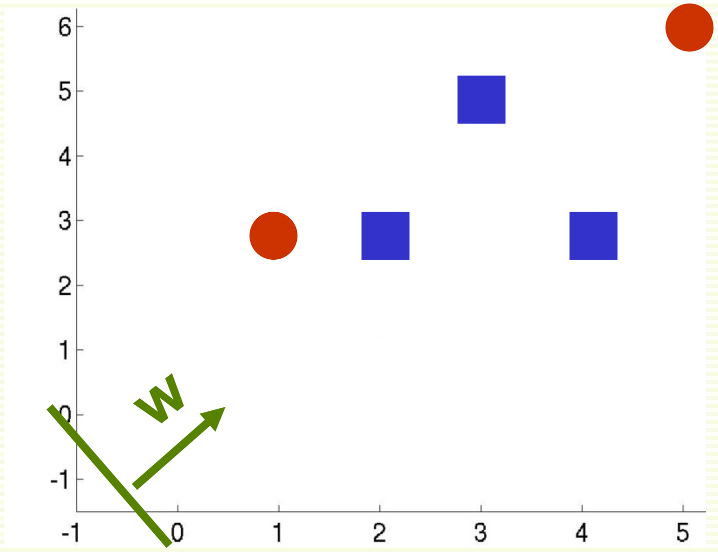
$$\mathbf{a} = \mathbf{a} + \sum_{\mathbf{i}} \left(\mathbf{y}^{\mathbf{i}} - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^{\mathbf{i}}\right)\right)\mathbf{z}^{\mathbf{i}}$$

- Can compute each $(\mathbf{y}^{\mathbf{i}} - \sigma(\mathbf{a}^{\mathbf{t}}\mathbf{z}^{\mathbf{i}}))\,\mathbf{z}^{\mathbf{i}}$ with **for** loop, and add them up
- For **i** = 1,

$$\left(\mathbf{y}^{1} - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^{1}\right)\right)\mathbf{z}^{1} = \left(1 - \sigma\left(\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}\right)\right)\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \left(1 - \sigma(6)\right)\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = 0.0025\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.0025 \\ 0.005 \\ 0.0075 \end{bmatrix}$$

Linear Classifier: Logistic Regression

$$\mathbf{Z} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \qquad \mathbf{a} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$



- Logistic Regression Batch rule update with α = 1

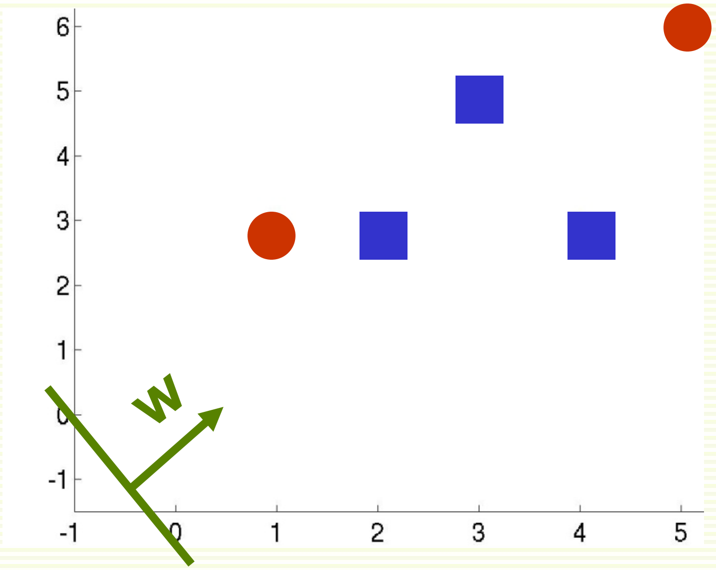
$$\mathbf{a} = \mathbf{a} + \sum_{\mathbf{i}} \left(\mathbf{y}^{\mathbf{i}} - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^{\mathbf{i}}\right)\right)\mathbf{z}^{\mathbf{i}}$$

- But also can compute update with a few lines in Matlab, no need for a loop
- First compute $\mathbf{a}^{\mathbf{t}}\mathbf{z}^{\mathbf{i}}$ for all examples

$$\begin{bmatrix} \mathbf{a}^{\mathbf{t}}\mathbf{z}^1 \\ \mathbf{a}^{\mathbf{t}}\mathbf{z}^2 \\ \mathbf{a}^{\mathbf{t}}\mathbf{z}^3 \\ \mathbf{a}^{\mathbf{t}}\mathbf{z}^4 \\ \mathbf{a}^{\mathbf{t}}\mathbf{z}^5 \end{bmatrix} = \mathbf{Z} * \mathbf{a} = \begin{bmatrix} 6 \\ 8 \\ 9 \\ 5 \\ 12 \end{bmatrix}$$

Linear Classifier: Logistic Regression

$$\mathbf{Z} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \qquad \mathbf{a} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$



- Batch update rule

$$\mathbf{a} = \mathbf{a} + \sum_{\mathbf{i}} \left(\mathbf{y}^{\mathbf{i}} - \sigma\left(\mathbf{a}^{\mathbf{t}} \mathbf{z}^{\mathbf{i}}\right) \right) \mathbf{z}^{\mathbf{i}}$$

- Apply sigmoid to each row

$$\begin{bmatrix} \mathbf{a}^{\mathbf{t}}\mathbf{z}^1 \\ \mathbf{a}^{\mathbf{t}}\mathbf{z}^2 \\ \mathbf{a}^{\mathbf{t}}\mathbf{z}^3 \\ \mathbf{a}^{\mathbf{t}}\mathbf{z}^4 \\ \mathbf{a}^{\mathbf{t}}\mathbf{z}^5 \end{bmatrix} = \mathbf{Z} * \mathbf{a} = \begin{bmatrix} 6 \\ 8 \\ 9 \\ 5 \\ 12 \end{bmatrix}$$

$$\begin{bmatrix} \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^1\right) \\ \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^2\right) \\ \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^3\right) \\ \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^4\right) \\ \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^5\right) \end{bmatrix} = \begin{bmatrix} \sigma(6) \\ \sigma(8) \\ \sigma(9) \\ \sigma(5) \\ \sigma(12) \end{bmatrix} = \begin{bmatrix} 0.9975 \\ 0.9997 \\ 0.9999 \\ 0.9933 \\ 1.000 \end{bmatrix}$$

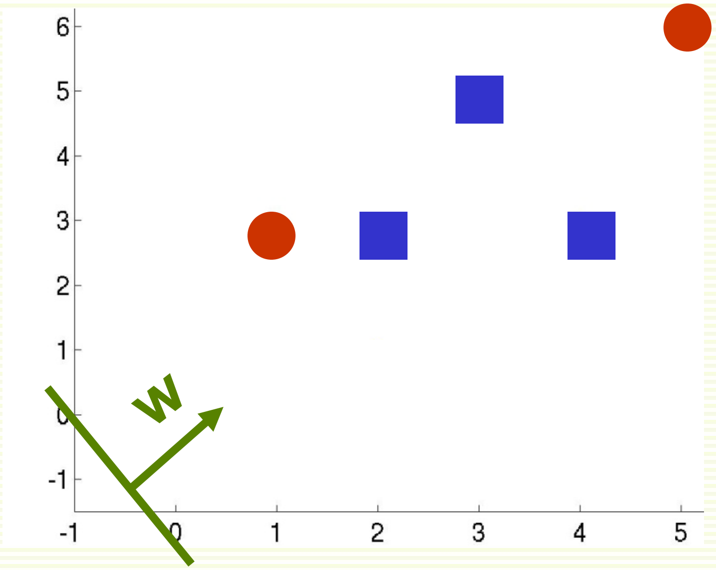
Linear Classifier: Logistic Regression

- Assume you have sigmoid function **σ**(**t**) implemented
    - takes scalar t as an input, outputs **σ**(**t**)
- To apply sigmoid to each element of column vector with one line, use *arrayfun*(*functionPtr, A*) in matlab

$$\begin{bmatrix} \sigma(6) \\ \sigma(8) \\ \sigma(9) \\ \sigma(5) \\ \sigma(12) \end{bmatrix} = \begin{bmatrix} 0.9975 \\ 0.9997 \\ 0.9999 \\ 0.9933 \\ 1.000 \end{bmatrix}$$

Linear Classifier: Logistic Regression

$$\mathbf{Z} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \qquad \mathbf{a} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$



- Batch rule update

$$\mathbf{a} = \mathbf{a} + \sum_{\mathbf{i}} \left(\mathbf{y}^{\mathbf{i}} - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^{\mathbf{i}}\right) \right) \mathbf{z}^{\mathbf{i}}$$

$$\begin{bmatrix} \sigma(\mathbf{a}^{\mathbf{t}}\mathbf{z}^1) \\ \sigma(\mathbf{a}^{\mathbf{t}}\mathbf{z}^2) \\ \sigma(\mathbf{a}^{\mathbf{t}}\mathbf{z}^3) \\ \sigma(\mathbf{a}^{\mathbf{t}}\mathbf{z}^4) \\ \sigma(\mathbf{a}^{\mathbf{t}}\mathbf{z}^5) \end{bmatrix} = \begin{bmatrix} 0.9975 \\ 0.9997 \\ 0.9999 \\ 0.9933 \\ 1.000 \end{bmatrix}$$

- Subtract from labels **Y**

$$\begin{bmatrix} \mathbf{y}^1 - \sigma(\mathbf{a}^{\mathbf{t}}\mathbf{z}^1) \\ \mathbf{y}^2 - \sigma(\mathbf{a}^{\mathbf{t}}\mathbf{z}^2) \\ \mathbf{y}^3 - \sigma(\mathbf{a}^{\mathbf{t}}\mathbf{z}^3) \\ \mathbf{y}^4 - \sigma(\mathbf{a}^{\mathbf{t}}\mathbf{z}^4) \\ \mathbf{y}^5 - \sigma(\mathbf{a}^{\mathbf{t}}\mathbf{z}^5) \end{bmatrix} = \mathbf{Y} - \begin{bmatrix} 0.9975 \\ 0.9997 \\ 0.9999 \\ 0.9933 \\ 1.000 \end{bmatrix} = \begin{bmatrix} 0.0025 \\ 0.0003 \\ 0.0001 \\ -0.9933 \\ -1.000 \end{bmatrix}$$

Linear Classifier: Logistic Regression

$$\mathbf{Z} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \qquad \mathbf{a} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

- Batch rule update

$$\mathbf{a} = \mathbf{a} + \sum_{\mathbf{i}} \left(\mathbf{y}^{\mathbf{i}} - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^{\mathbf{i}}\right)\right)\mathbf{z}^{\mathbf{i}}$$

$$\mathbf{v} = \begin{bmatrix} \mathbf{y}^1 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^1\right) \\ \mathbf{y}^2 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^2\right) \\ \mathbf{y}^3 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^3\right) \\ \mathbf{y}^4 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^4\right) \\ \mathbf{y}^5 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^5\right) \end{bmatrix} = \begin{bmatrix} 0.0025 \\ 0.0003 \\ 0.0001 \\ -0.9933 \\ -1.000 \end{bmatrix}$$

- Multiply by corresponding example

$$\begin{bmatrix} \left[\mathbf{y}^1 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^1\right)\right]\mathbf{z}^1 \\ \left[\mathbf{y}^2 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^2\right)\right]\mathbf{z}^2 \\ \left[\mathbf{y}^3 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^3\right)\right]\mathbf{z}^3 \\ \left[\mathbf{y}^4 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^4\right)\right]\mathbf{z}^4 \\ \left[\mathbf{y}^5 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^5\right)\right]\mathbf{z}^5 \end{bmatrix} = \mathbf{repmat}(\mathbf{v},1,3).*\mathbf{Z} = \begin{bmatrix} 0.0025 & 0.0025 & 0.0025 \\ 0.0003 & 0.0003 & 0.0003 \\ 0.0001 & 0.0001 & 0.0001 \\ -0.99 & -0.99 & -0.99 \\ -1.00 & -1.00 & -1.00 \end{bmatrix}.*\mathbf{Z}$$

Linear Classifier: Logistic Regression

$$\mathbf{Z} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

- Batch rule update

$$\mathbf{a} = \mathbf{a} + \sum_{\mathbf{i}} \left(\mathbf{y}^{\mathbf{i}} - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^{\mathbf{i}}\right)\right)\mathbf{z}^{\mathbf{i}}$$

$$\mathbf{v} = \begin{bmatrix} \mathbf{y}^1 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^1\right) \\ \mathbf{y}^2 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^2\right) \\ \mathbf{y}^3 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^3\right) \\ \mathbf{y}^4 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^4\right) \\ \mathbf{y}^5 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^5\right) \end{bmatrix} = \begin{bmatrix} 0.0025 \\ 0.0003 \\ 0.0001 \\ -0.9933 \\ -1.000 \end{bmatrix}$$

- Multiply by corresponding example continued

$$\begin{bmatrix} 0.0025 & 0.0025 & 0.0025 \\ 0.0003 & 0.0003 & 0.0003 \\ 0.0001 & 0.0001 & 0.0001 \\ -0.99 & -0.99 & -0.99 \\ -1.00 & -1.00 & -1.00 \end{bmatrix} .*\mathbf{Z} = \begin{bmatrix} 0.0025 & 0.0049 & 0.0074 \\ 0.0003 & 0.0013 & 0.001 \\ 0.0001 & 0.0004 & 0.0006 \\ -0.99 & -0.99 & -2.98 \\ -1.00 & -5.0 & -6.0 \end{bmatrix}$$

Linear Classifier: Logistic Regression

- Batch rule update $\mathbf{a} = \mathbf{a} + \sum_{\mathbf{i}} \left(\mathbf{y}^{\mathbf{i}} - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^{\mathbf{i}}\right)\right)\mathbf{z}^{\mathbf{i}}$

$$\begin{bmatrix} \left[\mathbf{y}^1 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^1\right)\right]\mathbf{z}^1 \\ \left[\mathbf{y}^2 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^2\right)\right]\mathbf{z}^2 \\ \left[\mathbf{y}^3 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^3\right)\right]\mathbf{z}^3 \\ \left[\mathbf{y}^4 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^4\right)\right]\mathbf{z}^4 \\ \left[\mathbf{y}^5 - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^5\right)\right]\mathbf{z}^5 \end{bmatrix} = \begin{bmatrix} 0.0025 & 0.0049 & 0.0074 \\ 0.0003 & 0.0013 & 0.001 \\ 0.0001 & 0.0004 & 0.0006 \\ -0.99 & -0.99 & -2.98 \\ -1.00 & -5.0 & -6.0 \end{bmatrix} = \mathbf{A}$$

- Add up all rows

$$\mathbf{sum}(\mathbf{A},1) = \begin{bmatrix} -1.99 & -5.99 & -8.97 \end{bmatrix}$$

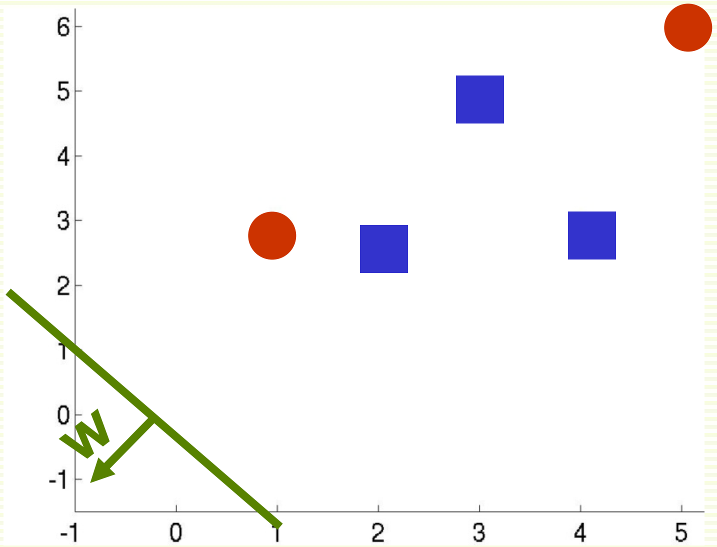
- Transpose to get the needed update

$$\begin{bmatrix} -1.99 & -5.99 & -8.97 \end{bmatrix}^{\mathbf{t}} = \begin{bmatrix} -1.99 \\ -5.99 \\ -8.97 \end{bmatrix} = \sum_{\mathbf{i}} \left(\mathbf{y}^{\mathbf{i}} - \sigma\left(\mathbf{a}^{\mathbf{t}}\mathbf{z}^{\mathbf{i}}\right)\right)\mathbf{z}^{\mathbf{i}}$$

Linear Classifier: Logistic Regression

- Batch rule update

$$\underbrace{\mathbf{a}}_{\begin{bmatrix}1\\1\\1\end{bmatrix}} = \mathbf{a} + \underbrace{\sum_i \left(\mathbf{y}^i - \sigma\left(\mathbf{a}^t\mathbf{z}^i\right)\right)\mathbf{z}^i}_{\begin{bmatrix}-1.99\\-5.99\\-8.97\end{bmatrix}}$$



- Finally update $\mathbf{a} = \begin{bmatrix}1\\1\\1\end{bmatrix} + \begin{bmatrix}-1.99\\-5.99\\-8.97\end{bmatrix} = \begin{bmatrix}-0.99\\-4.99\\-7.97\end{bmatrix}$

- This is line $-4.99\mathbf{x}_1 - 7.97\,\mathbf{x}_2 - 0.99 = 0$

More General Discriminant Functions

- Linear discriminant functions
  - simple decision boundary
  - should try simpler models first to avoid overfitting
  - optimal for certain type of data
  - Gaussian distributions with equal covariance
  - May not be optimal for other data distributions
- Discriminant functions can be more general than linear
  - For example, polynomial discriminant functions
  - Decision boundaries more complex than linear
  - Later will look more at non-linear discriminant functions

Summary

- Linear classifier works well when examples are linearly separable, or almost separable
- Two Linear Classifiers
  - Perceptron
    - find a separating hyperplane in the linearly separable case
    - uses gradient descent for optimization
    - does not converge in the non-separable case
    - can force convergence by using a decreasing learning rate
  - Logistic Regression
    - has probabilistic interpretation
    - can be optimized exactly with gradient descent