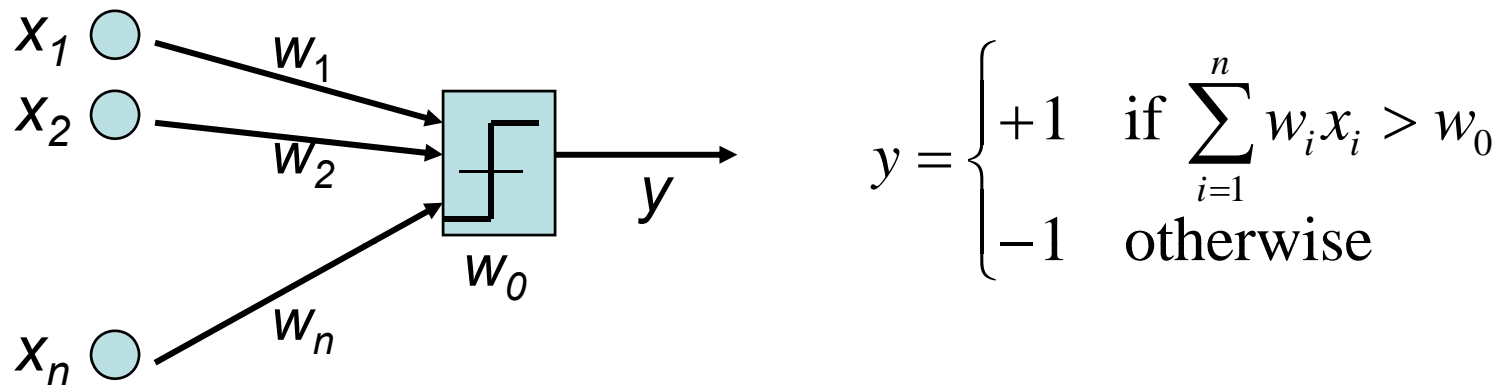


Linear Discriminant Functions: 1. Perceptron

Linear Threshold Units (McCulloch & Pitts 1943)



- Assume each feature x_j and weight w_j is a real number (we will relax this later)
- We will study three different algorithms for learning linear threshold units:
 - **Perceptron: directly learns a classifier**
 - Logistic Regression: learns the conditional distribution
 - Linear Discriminant Analysis: learns the joint distribution

What LTU Can Represent

- Conjunctions

$$x_1 \wedge x_2 \wedge x_4 \Leftrightarrow y$$

$$1 \cdot x_1 + 1 \cdot x_2 + 0 \cdot x_3 + 1 \cdot x_4 \geq 3$$

- At least *m*-of-*n*

$$\text{at - least - 2 - of } \{x_1, x_3, x_4\} \Leftrightarrow y$$

$$1 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_3 + 1 \cdot x_4 \geq 2$$

What LTU Cannot Represent

- Non-trivial disjunctions:

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \Leftrightarrow y$$

- Exclusive-OR:

$$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) \Leftrightarrow y$$

A Canonical Representation

- Given a training example: $(\langle x_1, x_2, x_3, x_4 \rangle, y)$ $y \in \{-1, 1\}$
- Transform it to canonical representation

$$(\langle 1, x_1, x_2, x_3, x_4 \rangle, y)$$

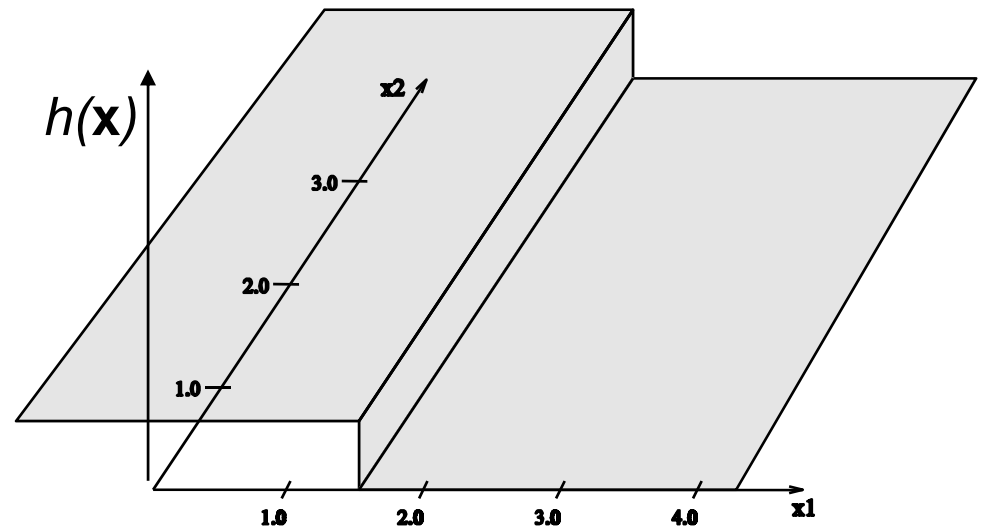
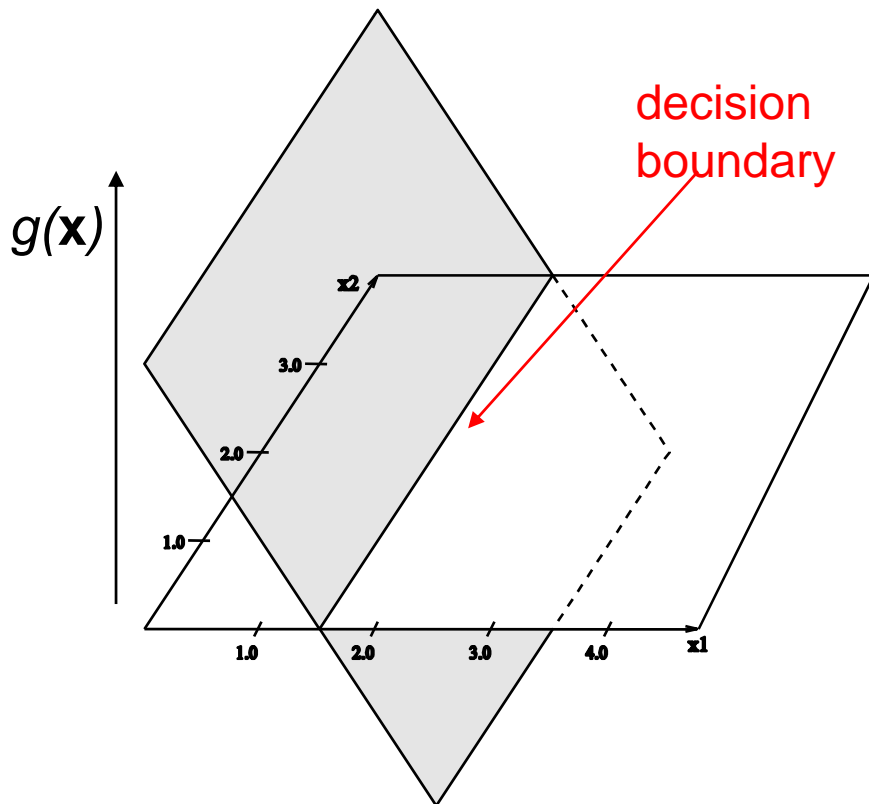
- Learn a linear function $g(\mathbf{x}, \mathbf{w}) = \mathbf{w} \cdot \mathbf{x}$, where $\mathbf{w} = \langle w_0, w_1, w_2, w_3, w_4 \rangle$
- Each \mathbf{w} corresponds to one hypothesis

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

- A prediction is correct if $y(\mathbf{w} \cdot \mathbf{x}) > 0$
- Goal of learning is to find a good \mathbf{w}
 - e.g., a \mathbf{w} such that $h(\mathbf{x})$ makes few mis-predictions

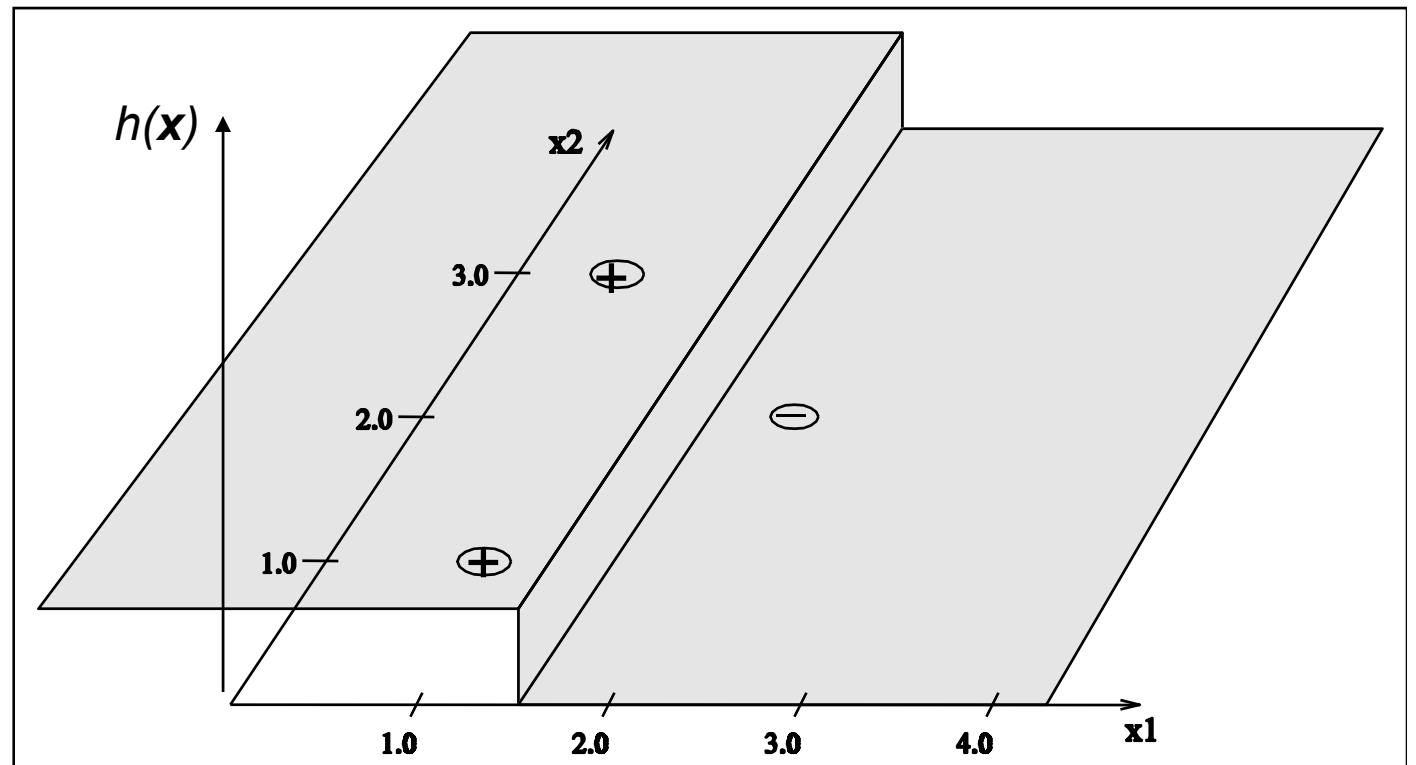
Geometrical View: Hyperplane

- For a 2-d feature space $g(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$ is a 2-d plane
 - $g(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} = 0$ defines a line in the 2-d feature space called the decision boundary
 - $g(\mathbf{x})$ changes sign when crossing boundary
 - $h(\mathbf{x})$ predicts + on one side and – on the other side



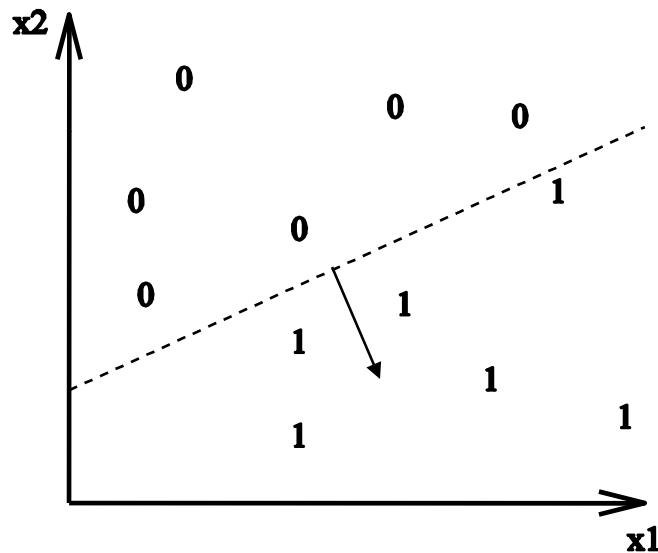
Geometrical View

- Consider three training examples: $(\langle 1.0, 1.0 \rangle, +1)$
 $(\langle 0.5, 3.0 \rangle, +1)$
 $(\langle 2.0, 2.0 \rangle, -1)$
- We want to find a \mathbf{w} such that the decision boundary separates the positive and negative instances if possible



Geometrical View: Linear Separability

Equation $w_0 + w_1x_1 + \dots + w_nx_n = 0$ defines a decision boundary that separates input space into decision regions.

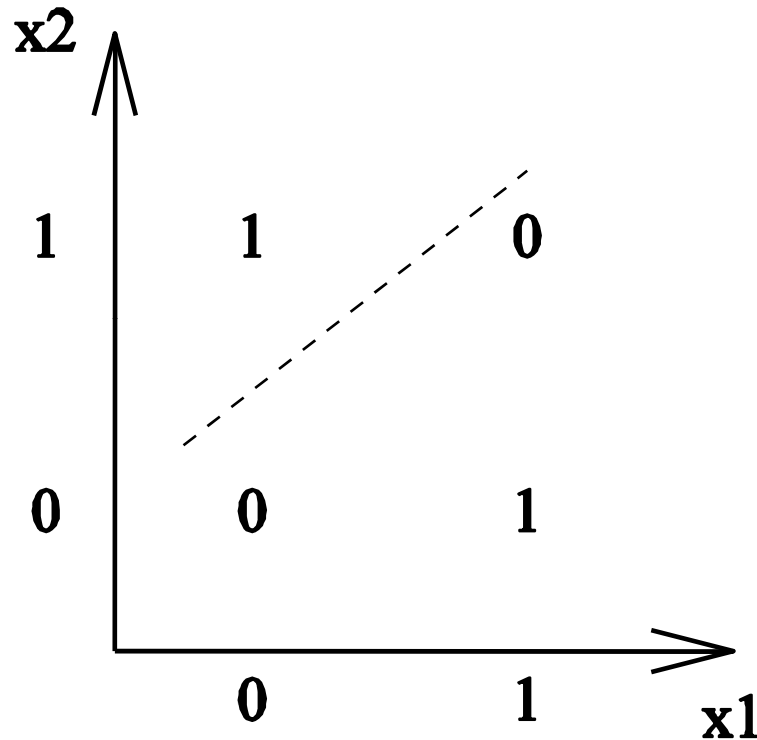


The orientation of the surface is determined by (w_1, w_2, \dots, w_n) .

The location of the surface is determined by the bias w_0

- A set of points that can be separated by a linear decision surface is called **linearly separable**

XOR: Not Linearly Separable



In such cases we might try to find a decision boundary that makes the fewest mistakes on the training data.

Learning w :

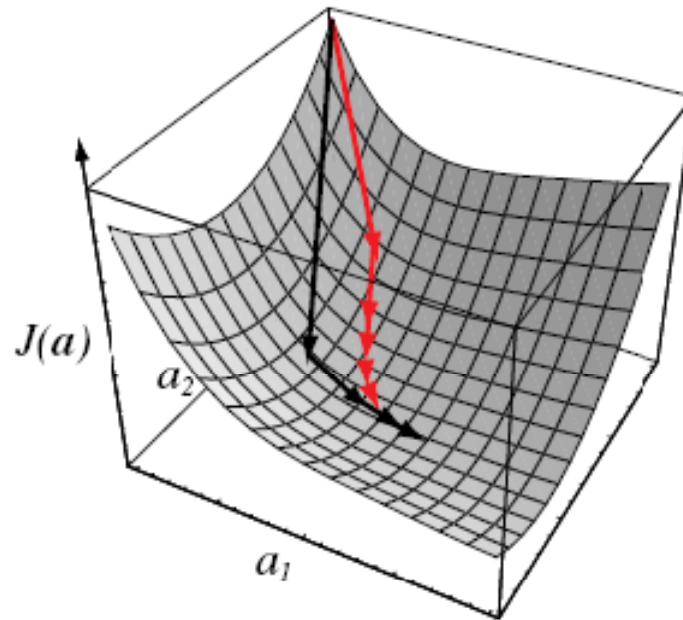
An Optimization Problem

- Formulate learning problem as an optimization problems
 - Given:
 - A set of N training examples
 $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$
 - A loss function L
 - Find the weight vector \mathbf{w} that minimizes the objective function - the expected/average loss on training data

$$J(w) = \frac{1}{N} \sum_{i=1}^N L(w \cdot x_i, y_i)$$

- Many machine learning algorithms apply some optimization algorithm to find a good hypothesis.

Gradient Descent Search



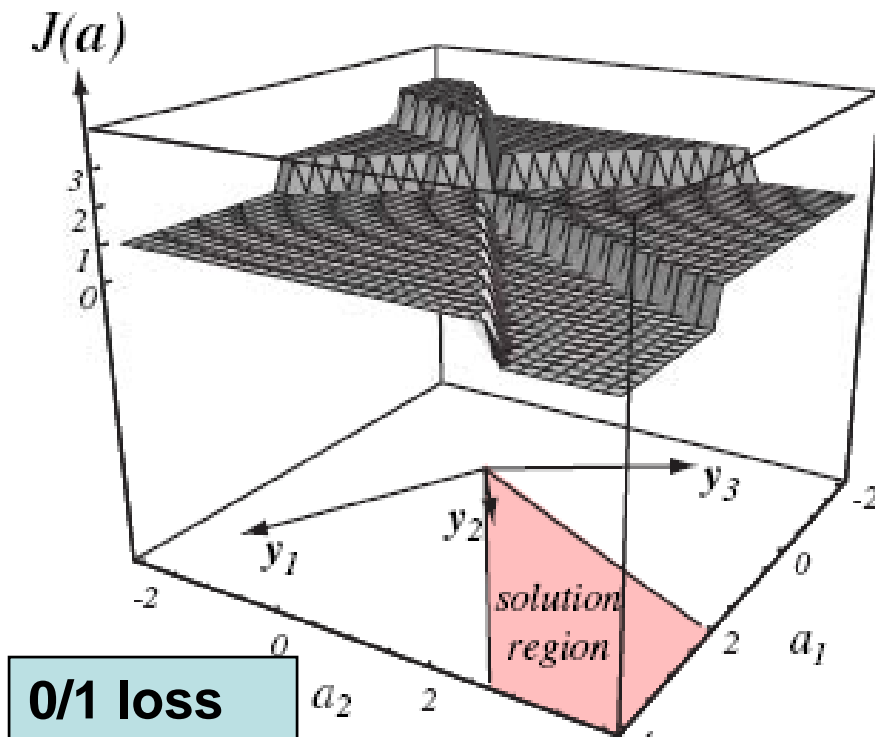
- Start with initial $\mathbf{w}(0) = (w_0, \dots, w_n)$
- Compute gradient $\nabla J(\mathbf{w}_0) = \left(\frac{\partial J(\mathbf{w})}{\partial w_0}, \frac{\partial J(\mathbf{w})}{\partial w_1}, \dots, \frac{\partial J(\mathbf{w})}{\partial w_n} \right)_{\mathbf{w}_0}$
- $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla J(\mathbf{w}_t)$, Where η is the “step size” parameter
- Repeat until convergence

Loss Functions

- 0/1 Loss function: $J_{0/1}(w) = \frac{1}{N} \sum_{i=1}^N L(\text{sgn}(w \cdot x_i), y_i)$

$L(y', y) = 0$ when $y' = y$, otherwise $L(y', y) = 1$

- Does not produce useful gradient since the surface of J is flat

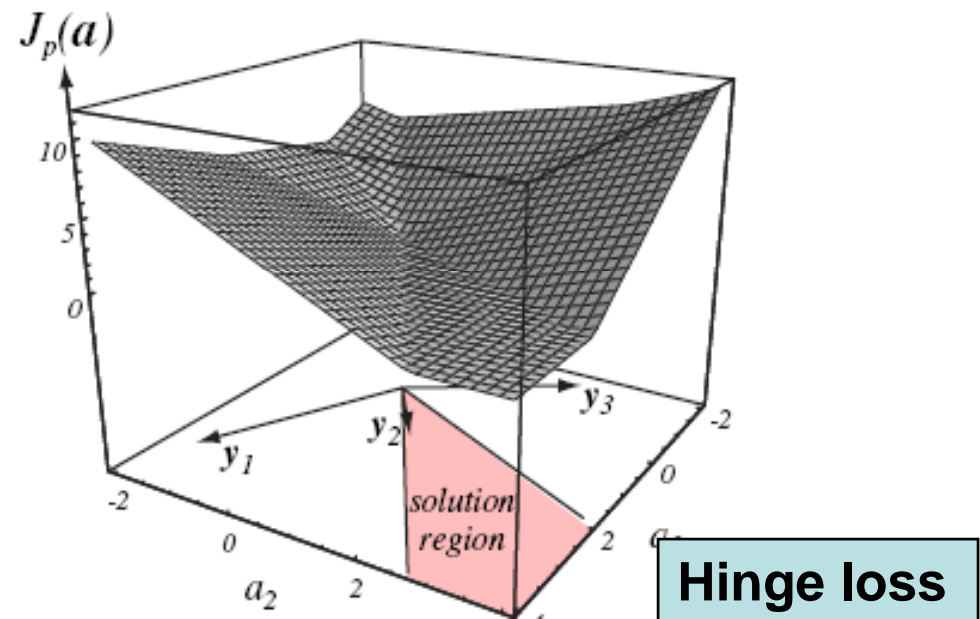
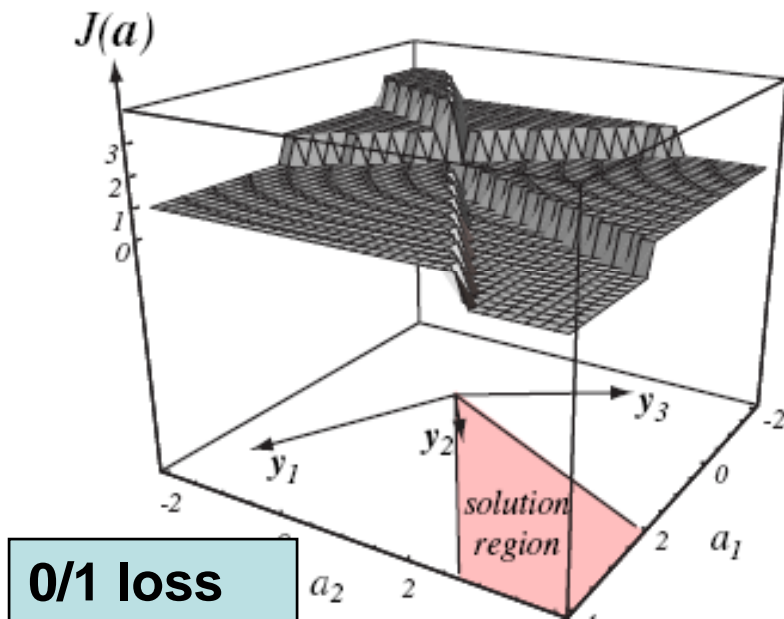


Loss Functions

- Instead we will consider the “**hinge loss**”:

$$J_p(w) = \frac{1}{N} \sum_{i=1}^N \max(0, -y_i w \cdot x_i)$$

- The term $\max(0, -y_i w \cdot x_i)$ is 0 when y_i is predicted correctly otherwise it is equal to the “confidence” in the mis-prediction
- Has a nice gradient leading to the solution region



Gradient For Hinge Loss Function

$$J(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N \max(0, -y_i \mathbf{w} \cdot \mathbf{x}_i)$$

$$J_i(\mathbf{w}) = \max(0, -y_i \mathbf{w} \cdot \mathbf{x}_i)$$

$$\frac{\partial J_i}{\partial w_j} = \begin{cases} 0 & \text{if } y_i \mathbf{w} \cdot \mathbf{x}_i > 0 \\ -y_i x_{ij} & \text{otherwise} \end{cases}$$

$$\nabla J_i = \begin{cases} \mathbf{0} & \text{if } y_i \mathbf{w} \cdot \mathbf{x}_i > 0 \\ -y_i \mathbf{x}_i & \text{otherwise} \end{cases}$$

$$\nabla J = \frac{1}{N} \sum_{y_i \mathbf{w} \cdot \mathbf{x}_i < 0} -y_i \mathbf{x}_i$$

Batch Perceptron Algorithm

Given : training examples (\mathbf{x}_i, y_i) , $i = 1, \dots, N$

Let $\mathbf{w} \leftarrow (0, 0, 0, \dots, 0)$

do

$\mathit{delta} \leftarrow (0, 0, 0, \dots, 0)$

for $i = 1$ to N do

$u_i \leftarrow \mathbf{w} \cdot \mathbf{x}_i$

if $y_i \cdot u_i \leq 0$

$\mathit{delta} \leftarrow \mathit{delta} - y_i \cdot x_i$

$\mathit{delta} \leftarrow \mathit{delta} / N$

$\mathbf{w} \leftarrow \mathbf{w} - \eta \mathit{delta}$

until $|\mathit{delta}| < \varepsilon$

Simplest case: $\eta = 1$ and don't normalize – 'Fixed increment perceptron'

η – the step size

- Also referred as the learning rate
- In practice, recommend to decrease η as learning continues
- Some optimization approaches set step-size automatically, e.g., by line search, and converge faster
- If linearly separable, there is only one basin for the hinge loss, thus local minimum is the global minimum

Online Perceptron Algorithm

Let $\mathbf{w} \leftarrow (0,0,0,\dots,0)$

Repeat

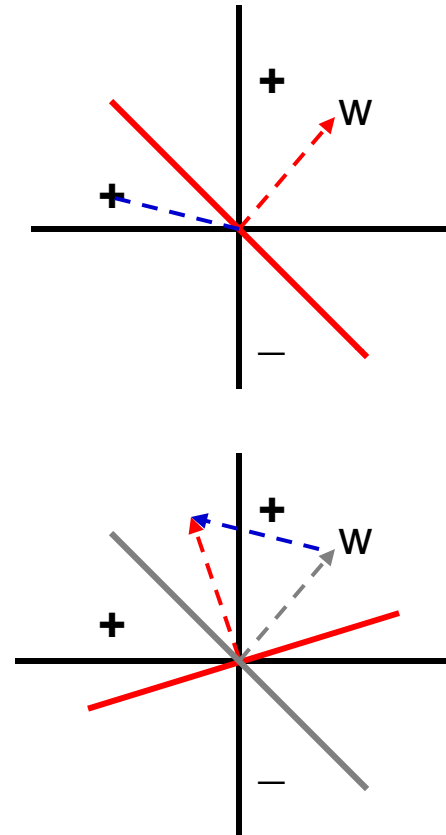
Accept training example $i : (\mathbf{x}_i, y_i)$

$$u_i \leftarrow \mathbf{w} \cdot \mathbf{x}_i$$

$$\text{if } y_i \cdot u_i \leq 0$$

$$\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$$

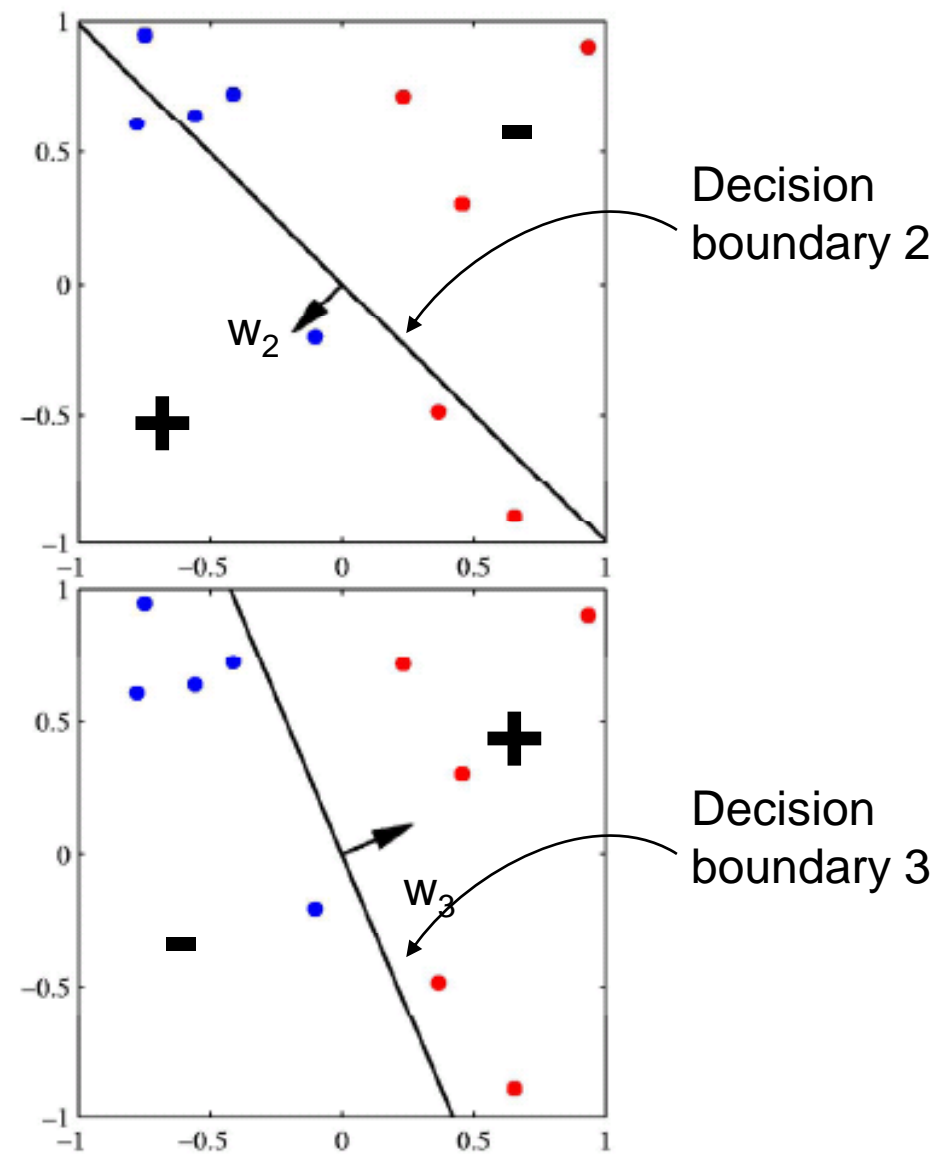
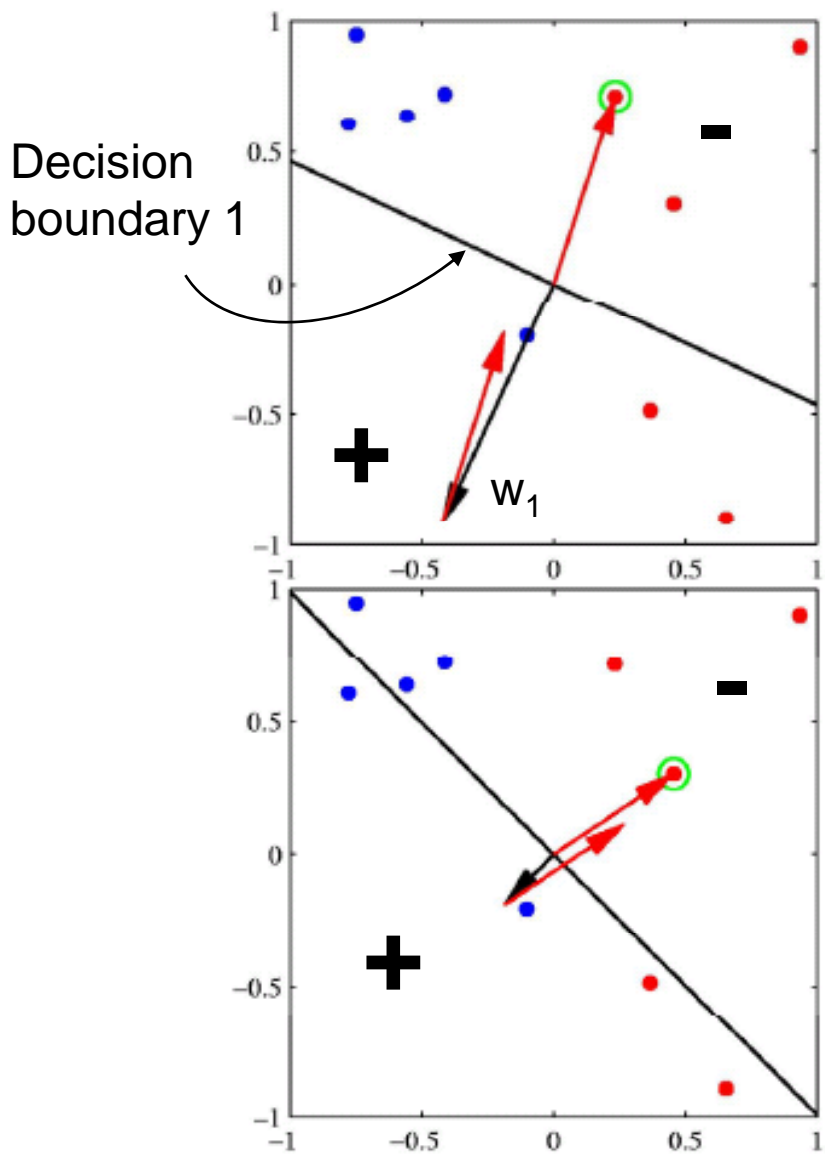
When an error is made, moves the weight in a direction toward correcting the error



This is called stochastic gradient descent because the overall gradient is approximated by the gradient from each individual example

Online VS. Batch Perceptron

- Batch learning learns with a batch of examples collectively
- Online learning learns with one example at a time
- Both learning mechanisms are useful in practice
- Online Perceptron is sensitive to the order that training examples are received
- In batch training, the correction incurred by each mistake is accumulated and applied at once at the end of the iteration
- In online training, each correction is applied immediately once a mistake is encountered, which will change the decision boundary, thus different mistakes maybe encountered for online and batch training
- Online training performs stochastic gradient descent, an approximation to real gradient descent, which is used by the batch training



Red points belong to the positive class,
blue points belong to the negative class

Convergence Theorem

(Block, 1962, Novikoff, 1962)

Given training example sequence $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$.

If $\forall i, \|\mathbf{x}_i\| \leq D$, and $\exists \mathbf{u}, \|\mathbf{u}\| = 1$ and $y_i \mathbf{u} \cdot \mathbf{x}_i \geq \gamma > 0$ for all i ,

then the number of mistakes that the perceptron algorithm

makes is at most $(D / \gamma)^2$.

Note that $\|\cdot\|$ is the Euclidean length of a vector.

Proof

To show convergence, we just need to show that each update moves the weight vector closer to a solution vector by a lower bounded amount
Note that $\alpha \mathbf{u}$ is also a solution vector, given that \mathbf{u} is a solution vector, where α is an arbitrary scaling factor

Let \mathbf{x}_k be the k th mistake, we have $\mathbf{w}(k+1) = \mathbf{w}(k) + y_k \mathbf{x}_k$

$$\begin{aligned} & \|\mathbf{w}(k+1) - \alpha \mathbf{u}\|^2 \\ &= \|\mathbf{w}(k) + y_k \mathbf{x}_k - \alpha \mathbf{u}\|^2 = \|(\mathbf{w}(k) - \alpha \mathbf{u}) + y_k \mathbf{x}_k\|^2 \\ &= \|\mathbf{w}(k) - \alpha \mathbf{u}\|^2 + 2y_k [\mathbf{x}_k \cdot (\mathbf{w}(k) - \alpha \mathbf{u})] + (y_k)^2 \|\mathbf{x}_k\|^2 \\ &= \|\mathbf{w}(k) - \alpha \mathbf{u}\|^2 + 2y_k \mathbf{x}_k \cdot \mathbf{w}(k) - 2y_k \alpha \mathbf{u} \cdot \mathbf{x}_k + \|\mathbf{x}_k\|^2 \\ &\leq \|\mathbf{w}(k) - \alpha \mathbf{u}\|^2 + 2y_k \mathbf{x}_k \cdot \mathbf{w}(k) - 2y_k \alpha \mathbf{u} \cdot \mathbf{x}_k + D^2 \quad , \text{ because } \|\mathbf{x}_k\| \leq D \\ &\leq \|\mathbf{w}(k) - \alpha \mathbf{u}\|^2 - 2\alpha \mathbf{u} \cdot \mathbf{x}_k + D^2 \quad , \text{ because } y_k \mathbf{x}_k \cdot \mathbf{w}(k) \leq 0 \\ &\leq \|\mathbf{w}(k) - \alpha \mathbf{u}\|^2 - 2\alpha \gamma + D^2 \quad , \text{ because } y_k \mathbf{u} \cdot \mathbf{x}_k \geq \gamma \end{aligned}$$

Because α is an arbitrary scaling factor, we can set $\alpha = \frac{D^2}{\gamma}$

$$\|\mathbf{w}(k+1) - \alpha \mathbf{u}\|^2 \leq \|\mathbf{w}(k) - \alpha \mathbf{u}\|^2 - D^2$$

Proof (cont.)

By induction on k , we can show that

$$\|\mathbf{w}(k+1) - \alpha \mathbf{u}\|^2 \leq \|\mathbf{w}(1) - \alpha \mathbf{u}\|^2 - kD^2 = \alpha^2 \|\mathbf{u}\|^2 - kD^2 = \alpha^2 - kD^2$$

$$\Leftrightarrow \alpha^2 - kD^2 \geq 0$$

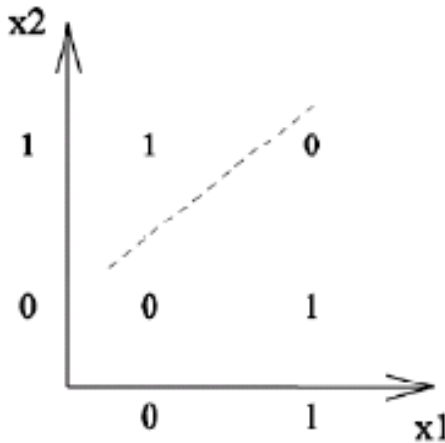
$$\Leftrightarrow k \leq \frac{\alpha^2}{D^2} \quad \left(\alpha = \frac{D^2}{\gamma}\right)$$

$$\Leftrightarrow k \leq (D/\gamma)^2$$

Margin

- γ is referred to as the margin
 - The minimum distance from data points to the decision boundary
 - The bigger the margin, the easier the classification problem is
 - The bigger the margin, the more confident we are about our prediction
- We will see later in the course this concept leads to one of the recent most exciting developments in the ML field – support vector machines

Not linearly separable case



- In such cases the algorithm will never stop! How to fix?
- Look for decision boundary that make as few mistakes as possible – NP-hard!

Fixing the Perceptron

- Idea one: only go through the data once, or a fixed number of times

```
Let  $\mathbf{w} \leftarrow (0,0,0,\dots,0)$ 
for  $i = 1,\dots,N$ 
    Take training example  $i : (\mathbf{x}_i, y_i)$ 
     $u_i \leftarrow \mathbf{w} \cdot \mathbf{x}_i$ 
    if  $y_i \cdot u_i < 0$ 
         $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$ 
```

- At least this stops
- Problem: the final \mathbf{w} might not be good e.g. right before it stops the algorithm might perform an update on a total outlier

Voted Perceptron

- Keep intermediate hypotheses and have them vote [Freund and Schapire 1998]

```
Let  $w_0 = (0,0,0, \dots, 0)$   
 $c_0 = 0$   
repeat  
  Take example  $i : (x_i, y_i)$   
   $u_i \leftarrow w_n \cdot x_i$   
  if  $y_i \cdot u_i \leq 0$   
     $w_{n+1} \leftarrow w_n + y_i x_i$   
     $c_{n+1} = 0$   
     $n = n + 1$   
  else  
     $c_n = c_n + 1$ 
```

Store a collection of linear separators w_0, w_1, \dots , along with their survival time c_0, c_1, \dots

The c 's can be good measures of the reliability of the w 's

For classification, take a weighted vote among all separators:

$$\text{sgn}\left\{\sum_{n=0}^N c_n \text{sgn}(w_n \cdot x)\right\}$$

Summary of Perceptron

- Learns a Classifier $\hat{y} = f(\mathbf{x})$ directly
- Applies gradient descent search to optimize the hinge loss function
 - Online version performs stochastic gradient descent
- Guaranteed to converge in finite steps if linearly separable
 - There exists an upper bound on the number of corrections needed
 - Inversely proportional to the margin of the optimal decision boundary
- If not linearly separable, use voted perceptrons

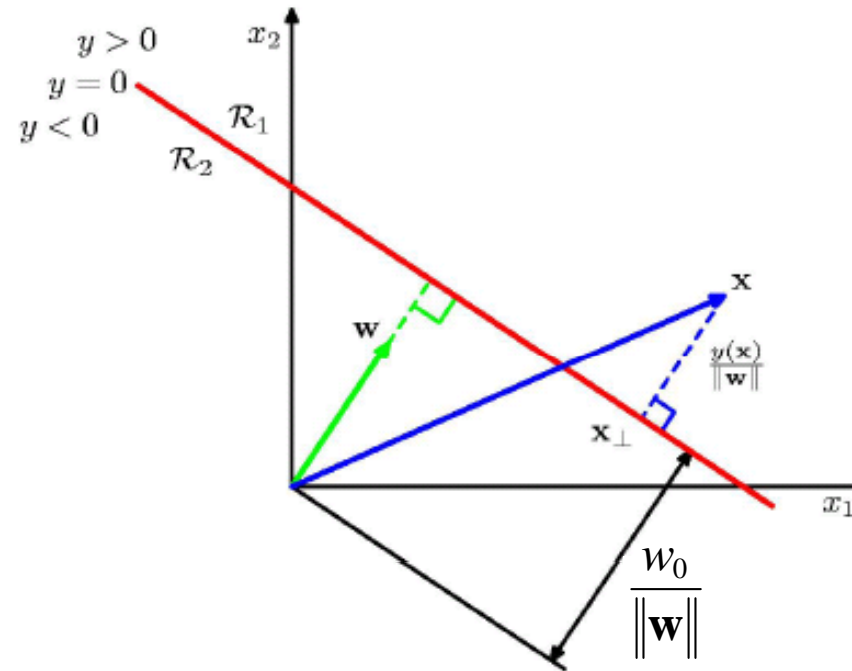
Geometric Interpretation of Linear Discriminant Functions

- Two classes

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$$

if $y(\mathbf{x}) \geq 0$, assign to C_1
otherwise, assign to C_2

- Decision boundary: $y(\mathbf{x})=0$
- Decision boundary is perpendicular to \mathbf{w}



The signed distance (positive if \mathbf{x} is on the positive side, negative otherwise) from any point \mathbf{x} to the decision boundary is: $\frac{y(\mathbf{x})}{\|\mathbf{w}\|}$

Note that in Perceptron, due to the adoption of the canonical representation, all training points will lie on the hyperplane $x_0=0$, and the decision boundary will always go through the origin.