Linear Threshold Units

Key approaches

- Directly learn a mapping y = f(**x**)
  - No uncertainty is captured
- Learn the joint distribution i.e., learn p(y**,x)**
  - Captures uncertainty about both the attributes **x** and the target y
- Learn the conditional distribution i.e., learn p(y|**x**)
  - p(**x**,y) = p(y|x)p(x)
  - Hence this avoids modeling the distribution of **x**
  - In general, this is akin to assuming an uniform distribution over **x**
  - Can also be considered as saying "I do not care about **x** but only P(y|**x**)
- Once we learn p, how do we choose y? This is called as decision-theory

Linear Threshold Units

$$h(x) = \begin{cases} +1 \rightleftarrows & if\ w_1x_1 + \ldots + w_nx_n \geq 0 \\ -1 & otherwise \end{cases}$$

Current Assumption: Each feature x_i and each weight w_j is a real number

Three Algorithms:

1. Perceptron – Directly learns the function
2. Logisitic Regression: Conditional Distribution
3. Linear Discriminant Analysis: Joint Distribution

What can an LTU 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

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 cannot be represented

- Complex 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: (<x_1, x_2, x_3, x_4>, y) $\quad y \in \{-1, 1\}$
- Transform it to canonical representation

  (<1, x_1, x_2, x_3, x_4>, y)

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

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

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

LTU Hypotheses space

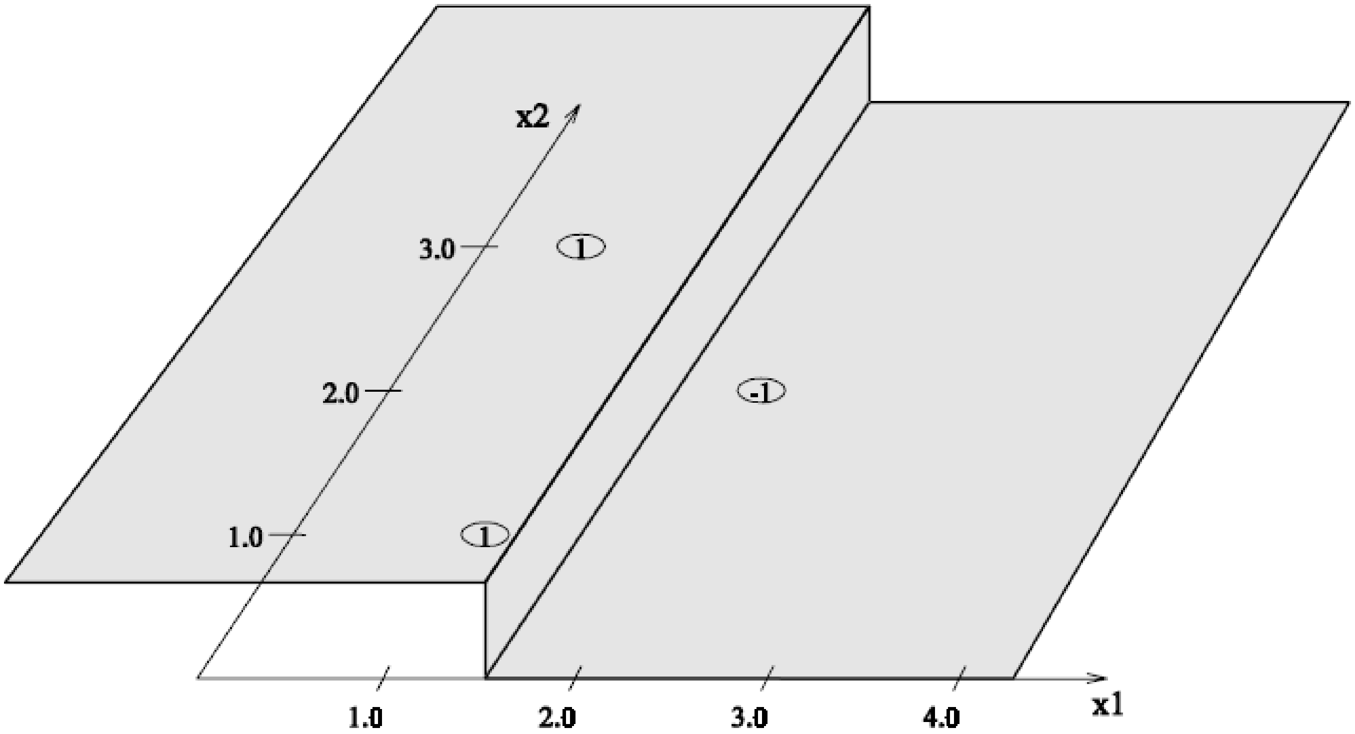
- Fixed size
- Deterministic
- Continuous parameters

Geometric View

- Let us consider 3 training examples:

$(\langle 1.0, 1.0\rangle, +1)\ (\langle 0.5, 3.0\rangle, +1)\ (\langle 2.0, 2.0\rangle, -1\ )$

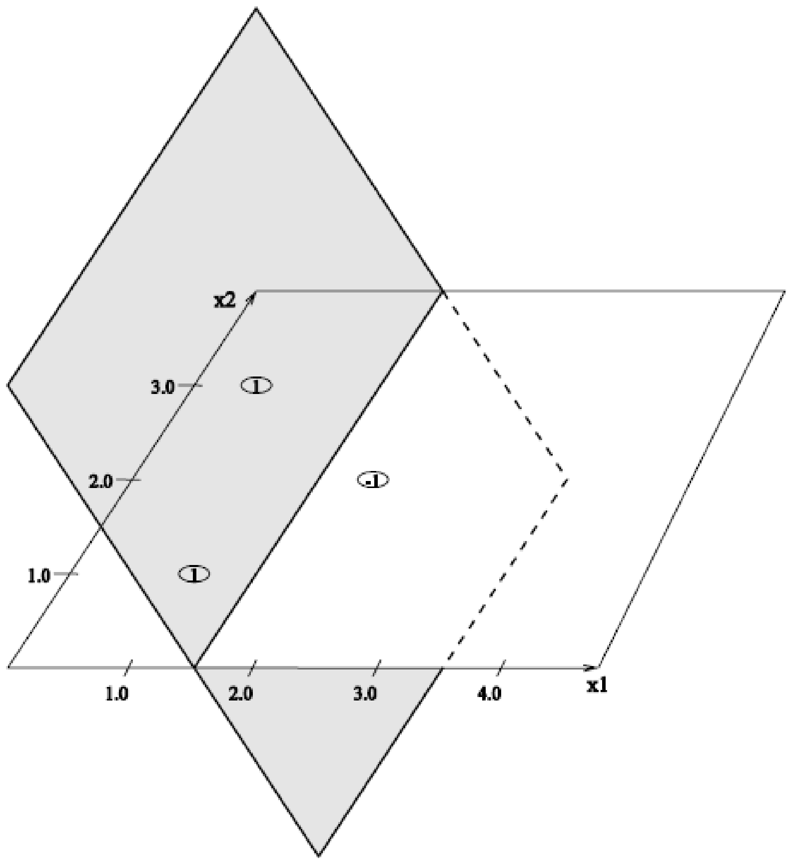
The classifier should look like the following



The discriminant function is a hyperplane

- The equation $u(\mathbf{x}) = \mathbf{w} \cdot x$ is a plane

$$\hat{y} = \begin{cases} +1 \; if \;\; u(x) \geq 0 \\ -1 \;\; otherwise \end{cases}$$



We can view this problem as an optimization problem

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

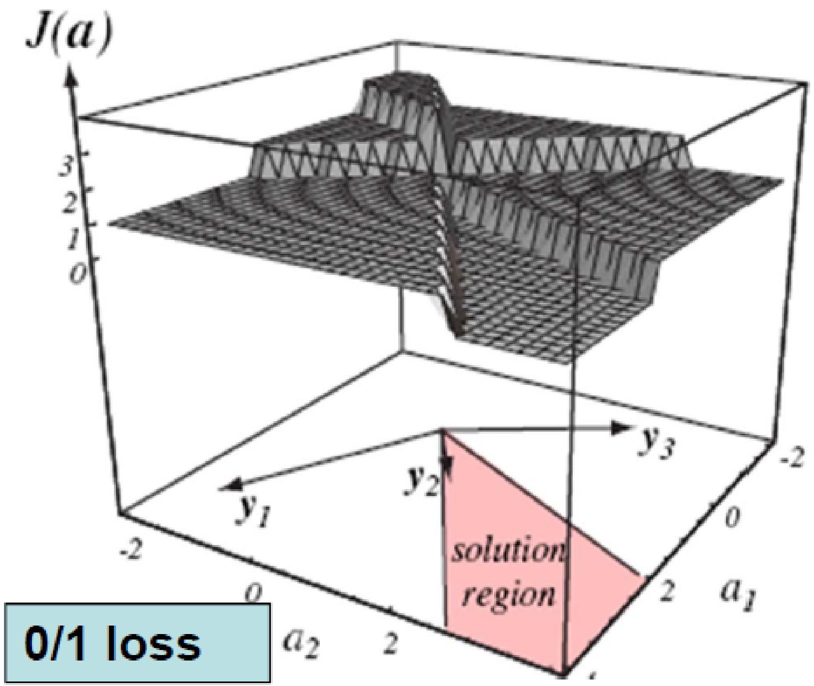
$$J(w) = \frac{1}{N}\sum_{i=1}^{N} L(w \cdot x_i, y_i)$$

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

0/1 Loss function

- 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



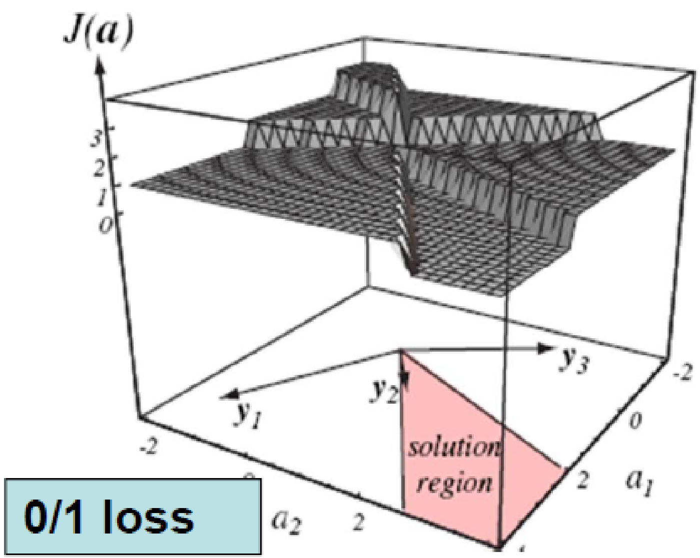
0/1 loss

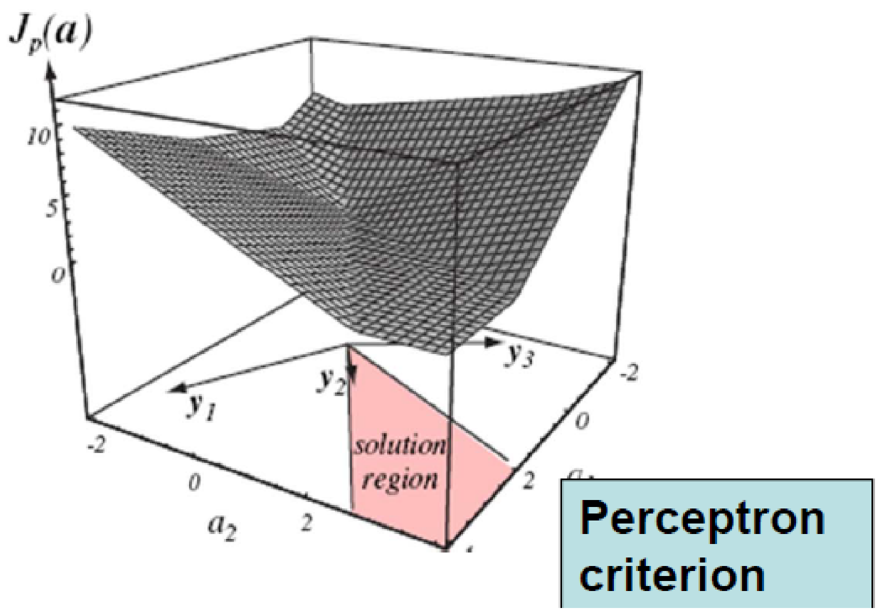
Other alternative – modified hinge loss

- Instead we will consider the "**perceptron criterion**" (a slightly modified version of hinge loss):

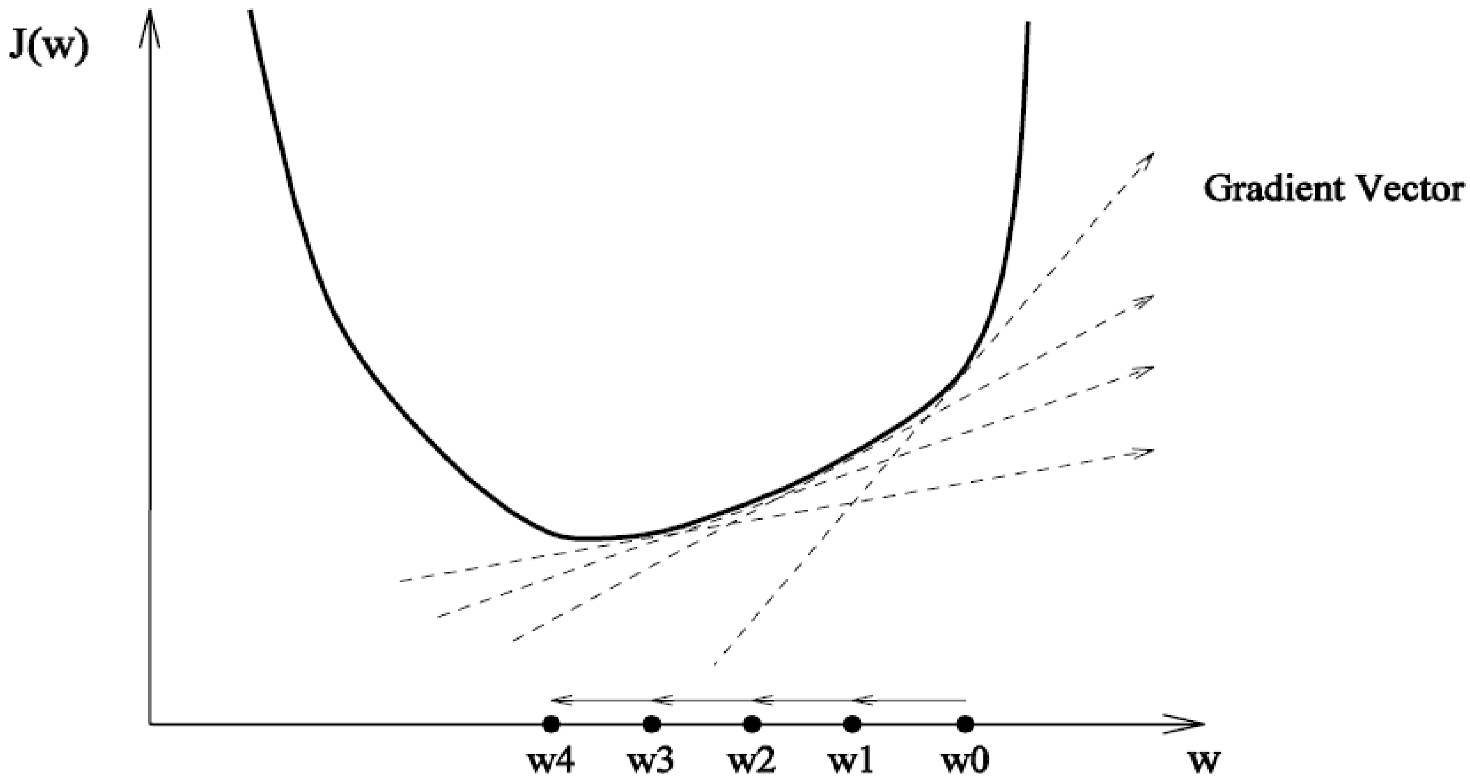
$$J_p(w) = \frac{1}{N}\sum_{i=1}^{N} \max(0, -y_i w \cdot x_i)$$

- 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 Descent minimizes the loss function



-Start with weight vector $\mathbf{w} = (w_0,\ldots, w_n)$

-Compute gradient $\nabla J(\mathbf{w}_0) = \left(\frac{\partial J(\mathbf{w})}{\partial w_0}, \frac{\partial J(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial J(\mathbf{w})}{\partial w_n}\right)_{\mathbf{w}_0}$

- Compute $w_1 = w_0 - \eta \nabla \bar{J}(w_0)$ where η is "step size"
- Repeat until convergence

Gradient Descent

- The objective function consists of a sum over data points---we can update the parameter after observing each example
- This is referred to as Stochastic gradient descent approach

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

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

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

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

After observing $(\mathbf{x}_i, y_i)$, if it is a mistake $\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x}_i$

Online Perceptron Algo

Let w = (0,0,0,…,0) be the initial weight vector

Repeat forever

Accept training example i: $\langle \mathbf{x}_i, y_i \rangle$

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

IF $(y_i \cdot u_i < 0)$

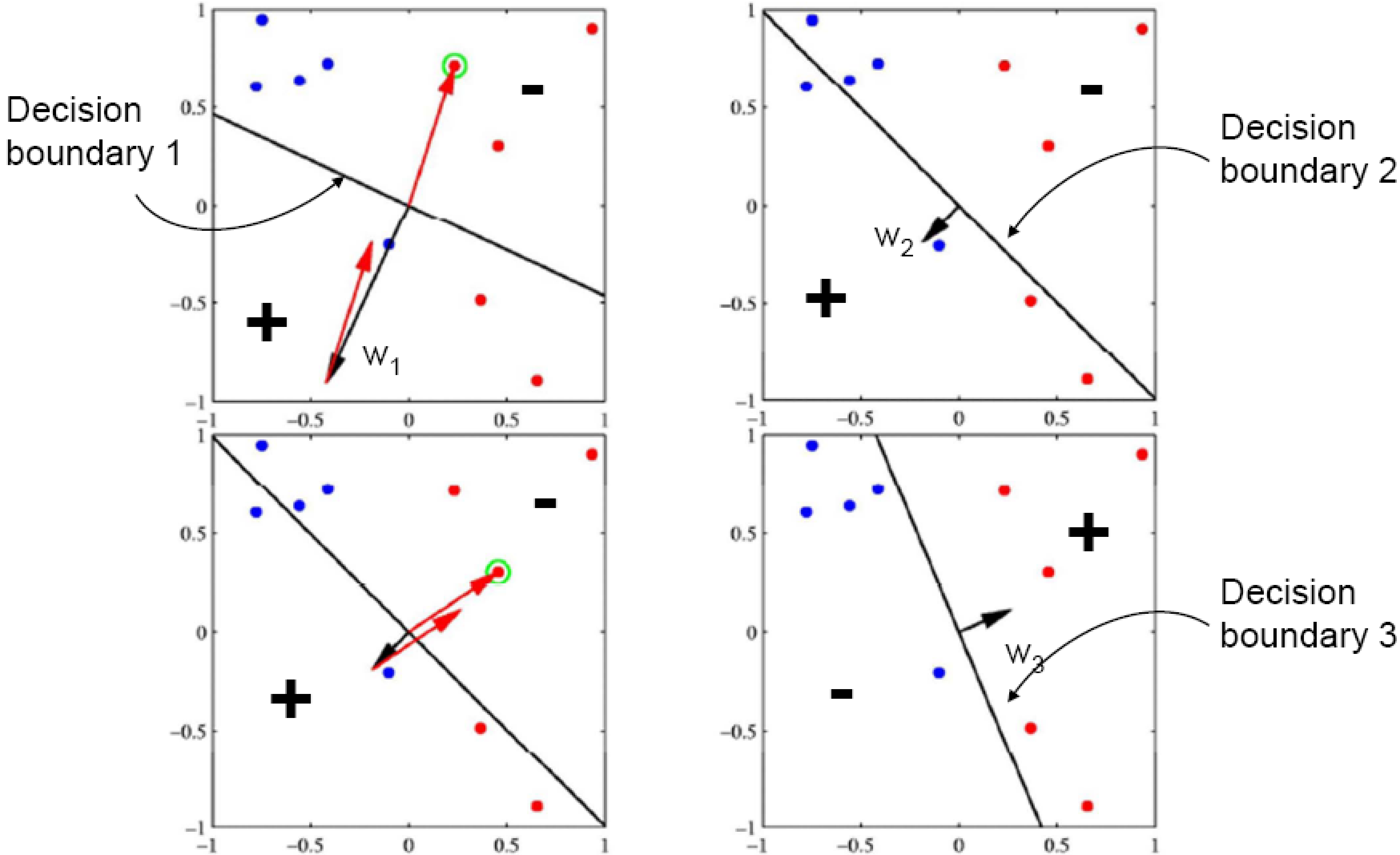
For j = 1 to *n* do // For every feature, compute gradient

$g_j := y_i \cdot x_{ij}$

$\mathbf{w} := \mathbf{w} + \eta g$

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

When an error is made, moves the weight in a direction that corrects the error



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

Batch Perceptron Algo

Let w = (0,0,0,…,0) be the initial weight vector

Let g = (0,0,0,…,0) be the initial gradient vector

Repeat until convergence

For *i = 1* to *N* do

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

IF $\left(y_i \cdot u_i < 0 \right)$

For j = 1 to *n* do

$$g_j := g_j - y_i \cdot x_{ij}$$

$$g := g / N$$

$$w := w - \eta g$$

When $\eta = 1$ it is a fixed increment perceptron

Step size

- Referred to as learning rate -- an important factor in many learning algorithms
- Learning rate must decrease to zero in order for the algorithm to converge

$$\lim_{t \to \infty} \eta_t = 0$$

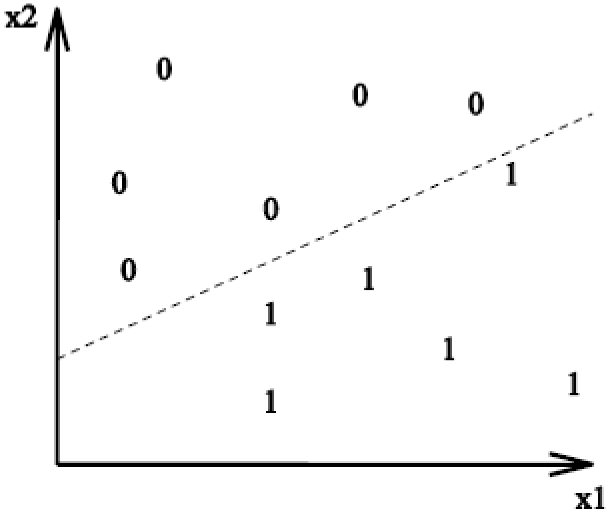
$$\sum_{t=0}^{\infty} \eta_t = \infty$$

$$\sum_{t=0}^{\infty} \eta_t^2 < \infty$$

- Some optimization algorithms set the step size automatically and converge faster
- For LTUs, there is only one basin. i.e., local minimum is global minimum. Choosing good step size will result in faster convergence

Decision Boundaries

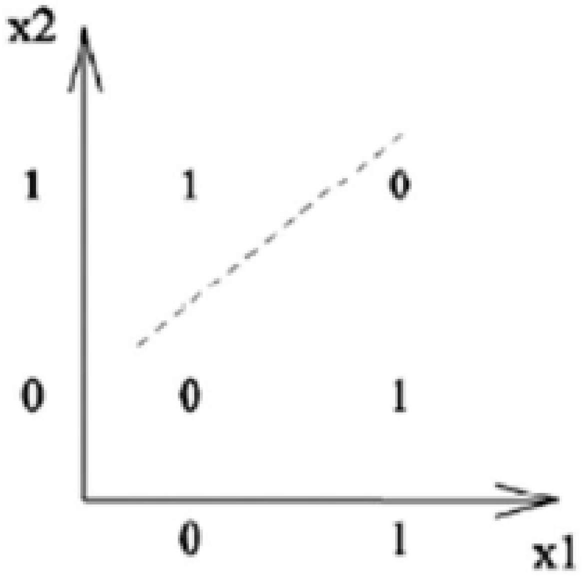
- A classifier can be viewed as partitioning the input space or feature X into decision regions



- A set of points that can be separated by a linear decision boundary is said to be linearly separable.

Not Linearly Separable

- X-Or



Perceptron – Key Result

If training set is linearly separable, perceptron WILL find it in finite steps

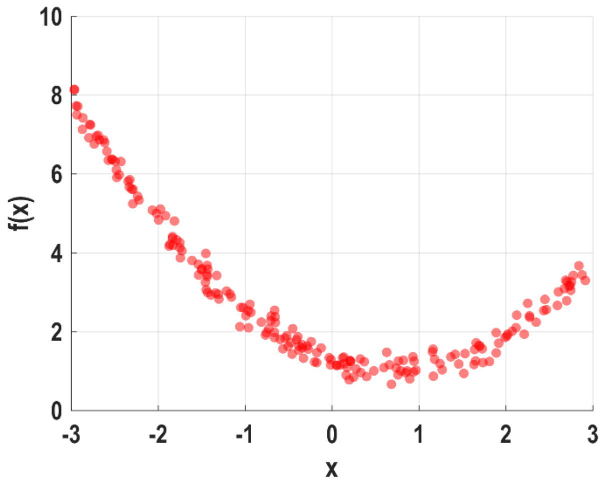
If training data is not linearly separable, perceptron will never converge

Summary of Perceptron

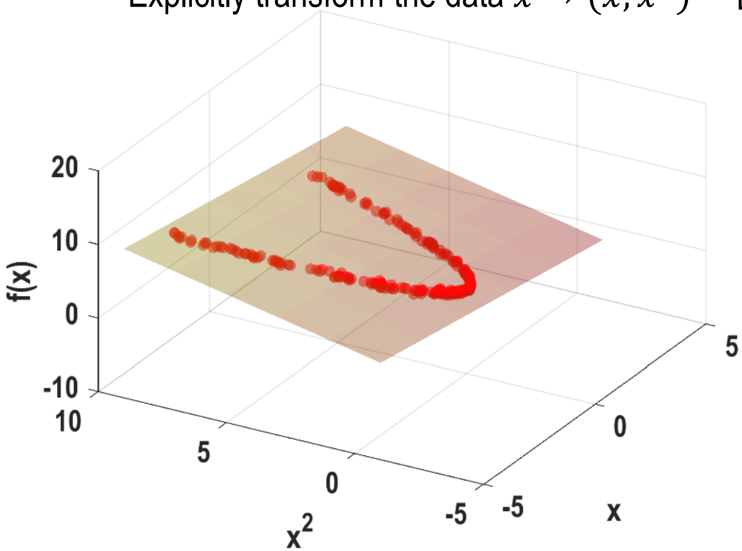
- Directly learns a classifier
- Local Search
  - Begins with an initial weight vector. Modifies it iteratively to minimize a loss function. The error function is loosely related to the goal of minimizing classification errors
- Eager
  - The classifier is constructed from training examples
  - The examples can then be discarded
- Online or Batch versions

Limitations of Linear Hypotheses

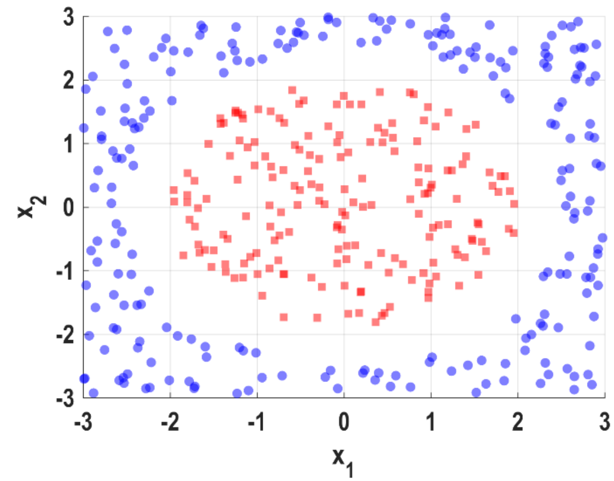
Regression: Non-linear regression functions



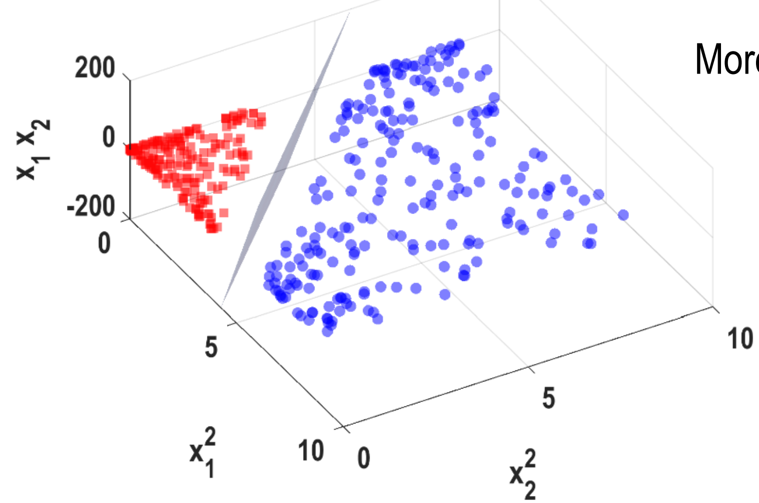
Explicitly transform the data $x \rightarrow (x, x^2)$



Classification: Linearly inseparable classes



Explicitly transform the data $(x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1x_2)$



A potential solution:
Transforming the data into a **higher-dimensional** space makes the problem linear in that space at expense of adding more features.

Coming up with such transformations is not easy.

More on this later in the course!

Next

- Logistic Regression
- Linear Discriminant Analysis
- **First Assignment:** Due the following week