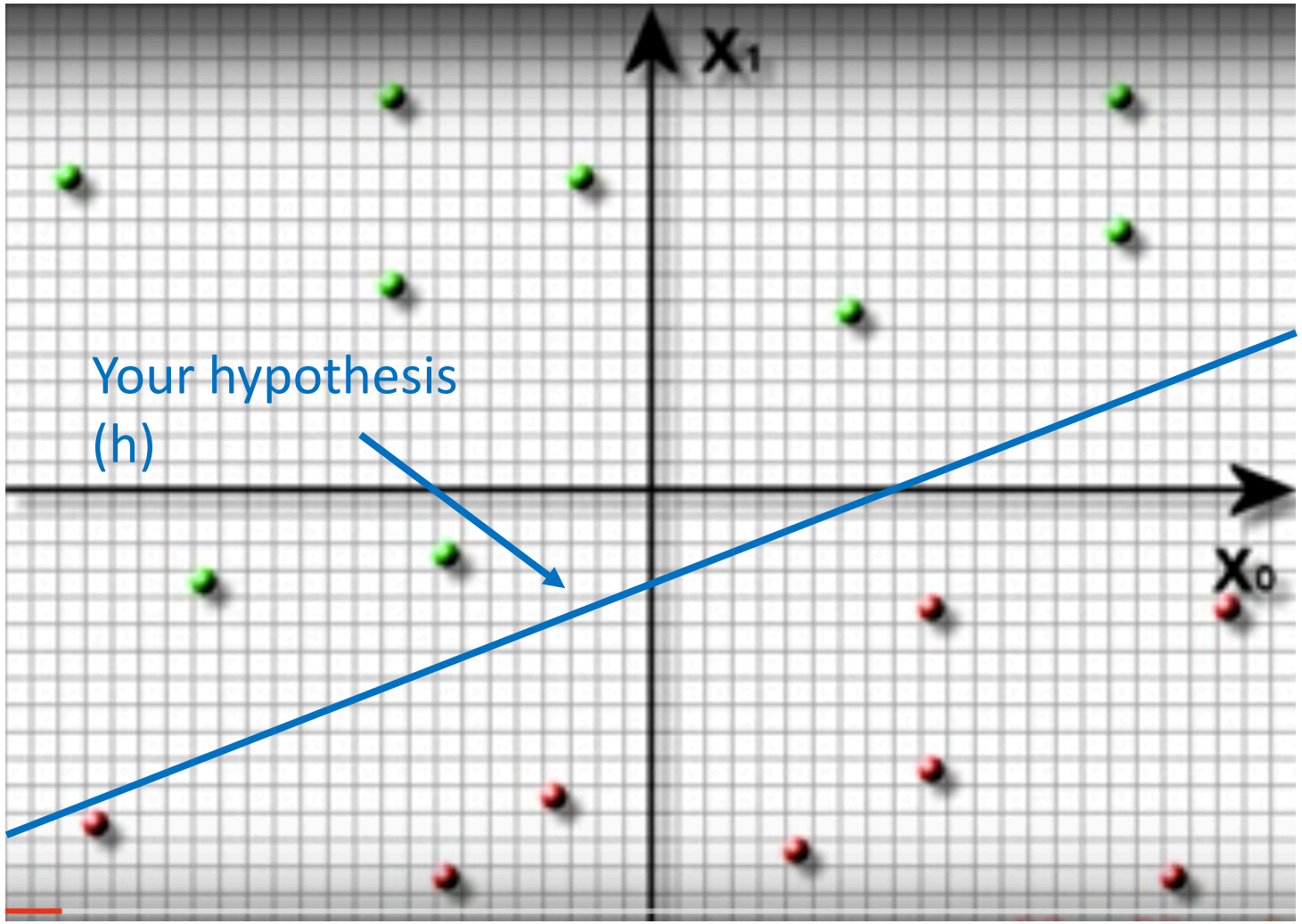
Linear Classification - Perceptron

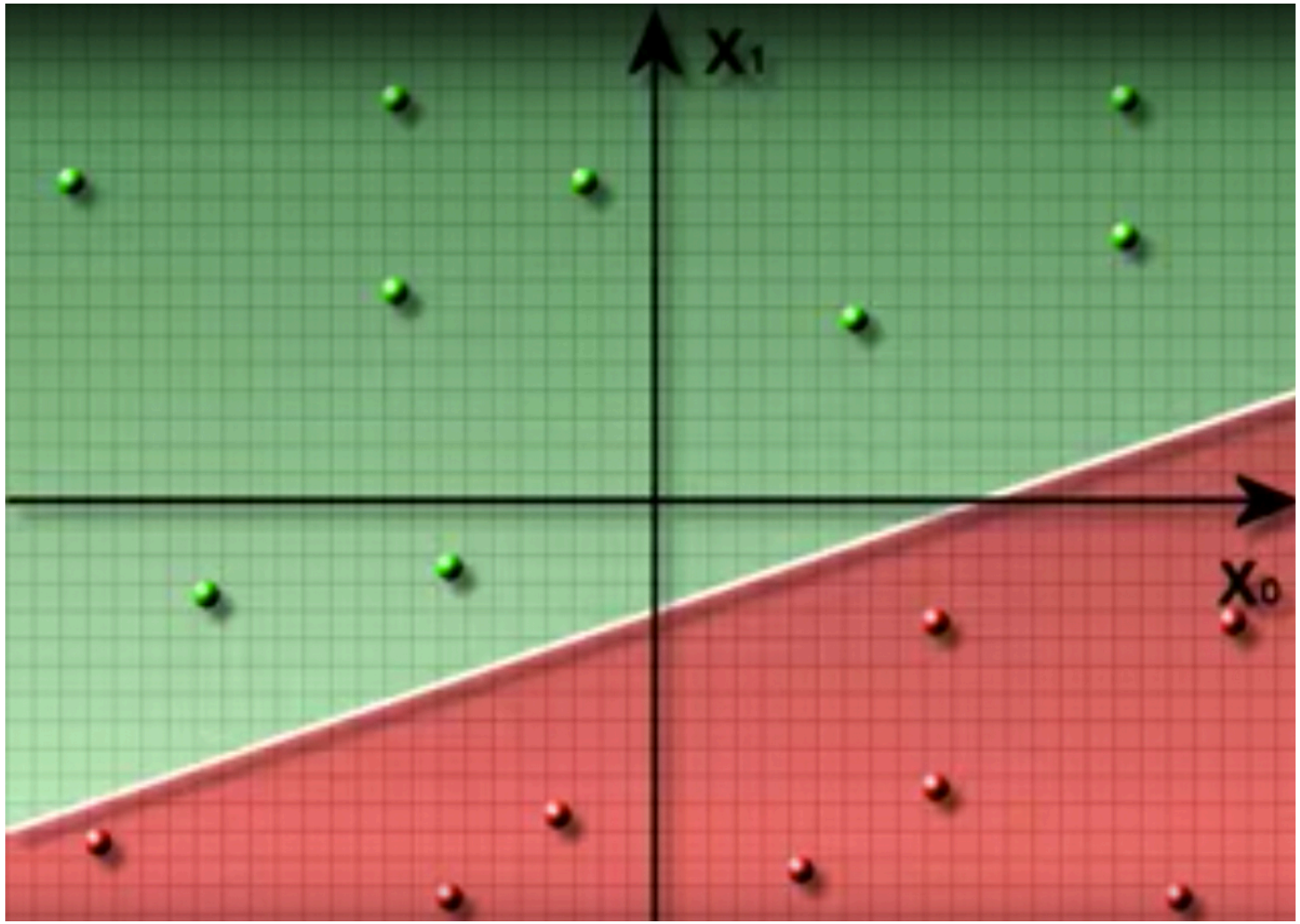
Linear Classification

- Imagine you have data as shown on the right and you want to propose a class separating hypothesis.
- Since data is linearly separable, you can propose a straight line
- Your hypothesis divides the plane into two parts
- Aim: Get those parts as pure as possible



Linear Classification

- OK, sounds simple enough
- Just draw a straight line and partition the plane into two parts.
- Limitations:
  - Works with linearly separable data only.
- I can draw many lines that separate the data perfectly, which one do I choose?
- Good Question ☺



Let's do some Math

- Equation of a straight line in a plane with two dimensions x_1 and x_2:

$$w_1 x_1 + w_2 x_2 = \theta$$

- There are three parameters in the equation above w_1, w_2 and Θ.
- By changing these, you can get a different line.
- Let's call the hypothesis as:

$$g(x) = w_1 x_1 + w_2 x_2 - \theta$$

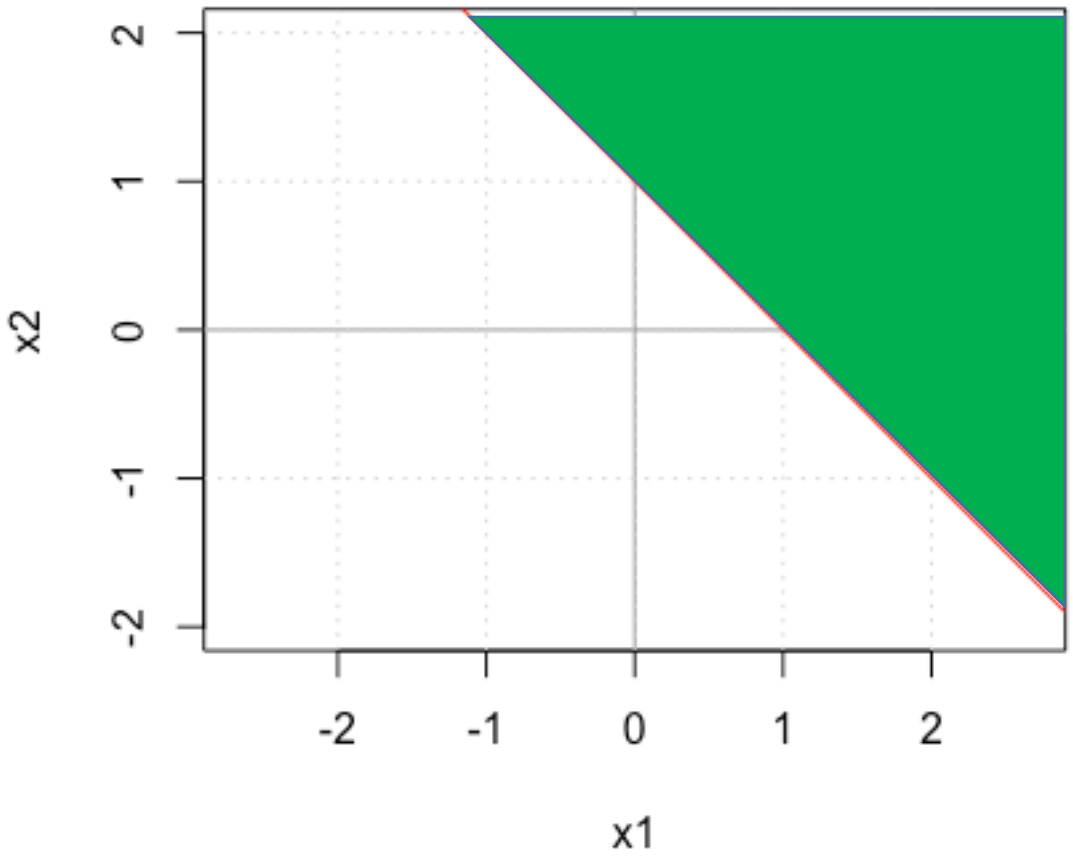
- What's the value of g for points that lie on the separating line? 0

Math

- Let's draw a line with w1 = 1, w2 = 1 and Θ = 1.
- We are trying to draw:

$$x_1 + x_2 = 1$$

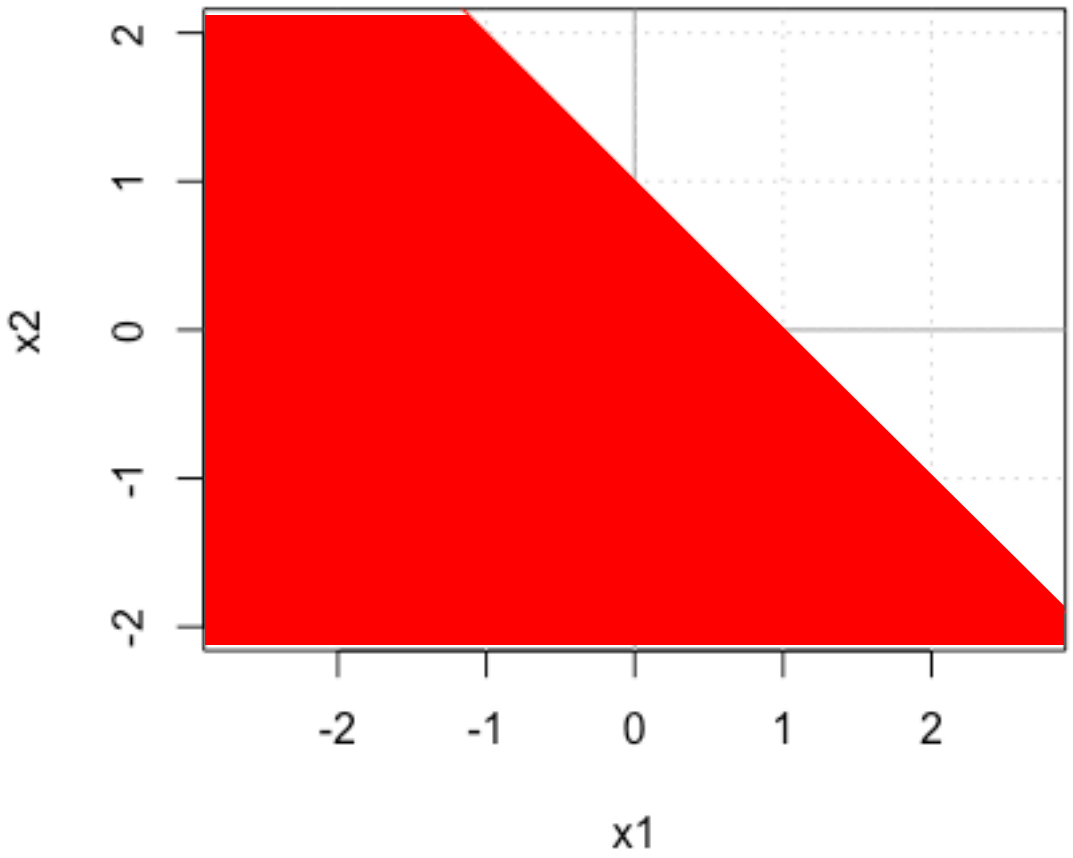
- Thus, g(x) = x_1 + x_2 - 1
- What are some points on this line:
  (1, 0), (0, 1), (½, ½), (-1, 2), …
- All the above points satisfy g(x) > 0
- Let's see what happens to points in green area.
- What is g(2, 1) ?
  g(2, 1) = 2 + 1 – 1 = 2



Math

- Let's see what happens to points in red area.
- What is g(-1, -1) ?
  g(-1, -1) = -1 - 1 – 1 = -3
- So, what can we use as a classification rule
- Check sign of hypothesis for test data point:

  if sign($g(x_{test})$) > 0 call it class +1
  else if sign($g(x_{test})$) < 0 call it class -1



Classification Rule

- Let h(x) denote the sign of g(x)
  h(x) = sign(g(x))
  sign(x) function is defined as +1 for x >= 0 and -1 for x < 0
- Predicted class label:
  Class = h(x)

Linear Classification

- If there are n input attributes x_1, x_2, …, x_n, our task is to find appropriate set of weights w_1, w_2, …, w_n, such that correct classification can be performed.
- We would like

if $\sum_{i=1}^{n} w_i x_i$ - Θ > 0 then class = +1
otherwise class = -1

for some threshold value Θ.

Note:
i starts from 1
=> We don't have any artificial attributes

- This can be states as:
  If $\mathbf{w}^T\mathbf{x}$ > Θ then class= +1
  otherwise class = -1
  where **w** is the weights vector and **x** is the attributes vector.

Linear Classification

- Can be written as:
  $w_1x_1 + w_2x_2 + \ldots + w_nx_n > \Theta$ for class +1
  or
  $-\Theta + w_1x_1 + w_2x_2 + \ldots + w_nx_n > 0$ for class +1

  or
  $w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n > 0$ for class +1

  where w_0 is referred to as the bias term.

You will find both these forms used in textbooks

Linear Classification

- Two equivalent ways:

$w^T x > \theta$ where $w = \begin{pmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{pmatrix}$ $x = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}$ for class +1

$w^T x > 0$ where $w = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_n \end{pmatrix}$ $x = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}$ for class +1

You will find both these forms used in textbooks

Graphical Meaning of Terms

- What do the terms (w1, w2, ..) and Θ mean graphically.

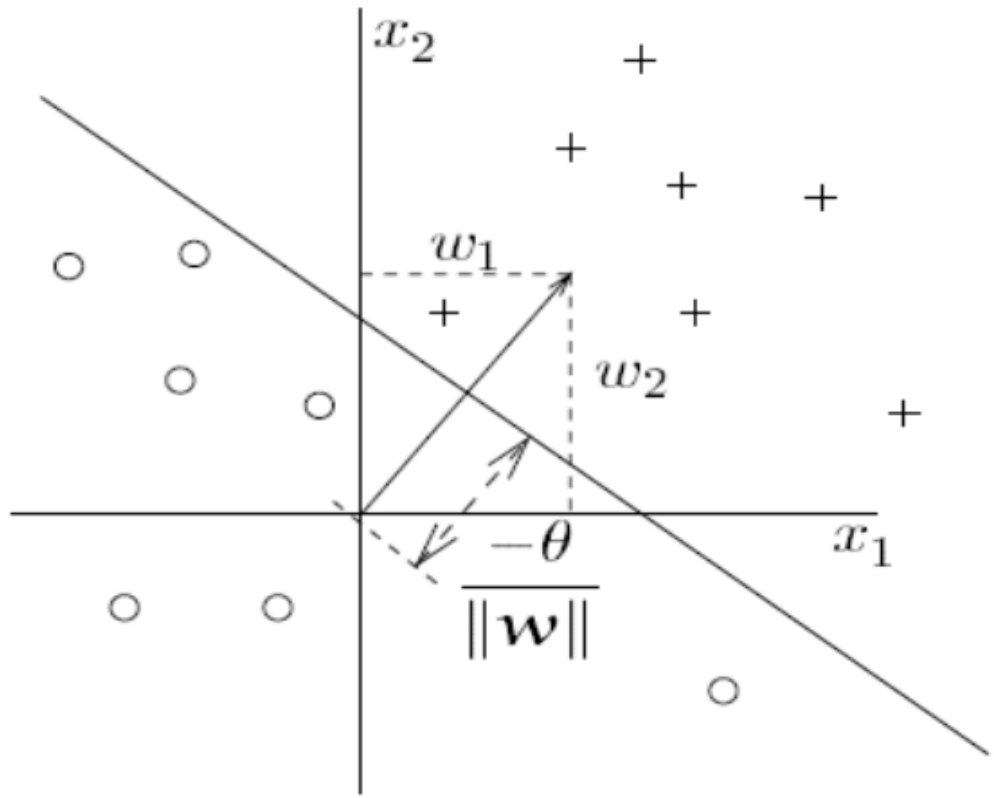
- w's correspond to the slope and Θ is a measure of intercept of the line.

The bias is proportional to the offset of the plane from the origin

The weights determine the slope of the line

The weight vector is perpendicular to the plane

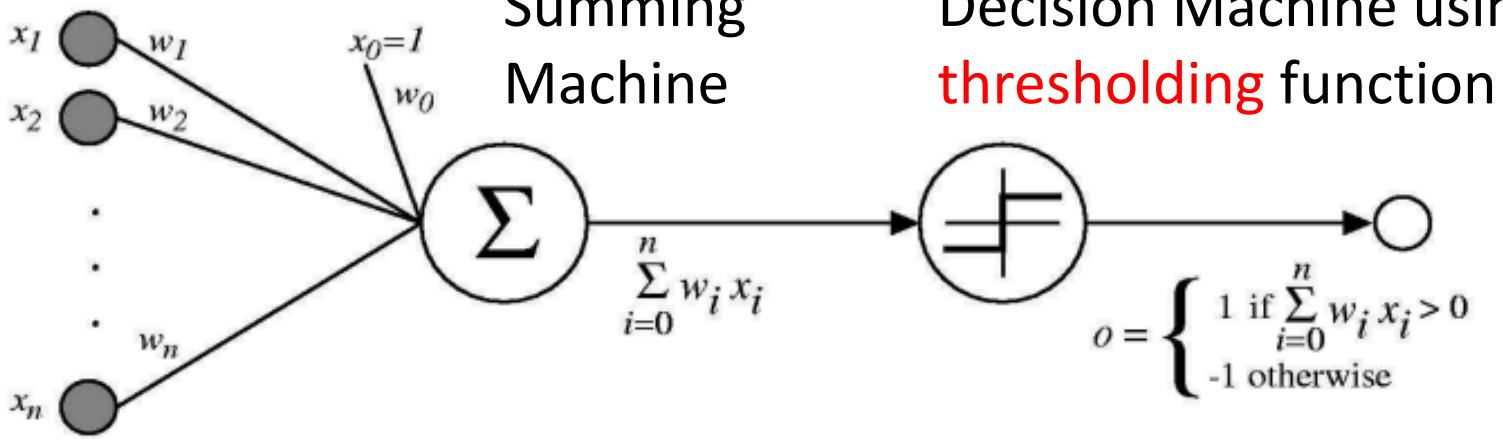
$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{\theta}{w_2}$$



Perceptron

- The idea of linear threshold classifier is used in the model of a perceptron.
- Inspired by bio-computing model of a neuron in brain.

Perceptron



Summing Machine

Decision Machine using thresholding function

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Linear Classification

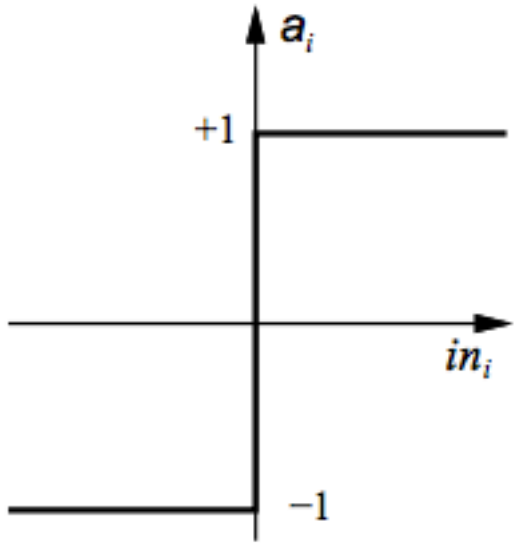
- The hypothesis can be represented as:

$h(x) = s(g(w^T x))$
where s is a function that outputs -1 or +1

What can s be?

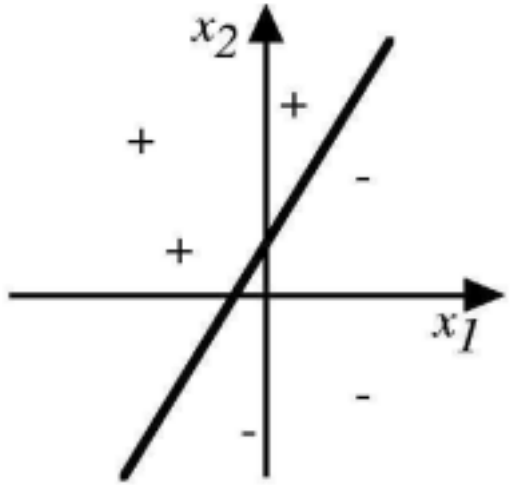
$$s(x) = \begin{cases} +1 \text{ if } x >= 0 \\ -1 \text{ if } x < 0 \end{cases}$$

It's called the sign function



Types of problems Perceptron can solve

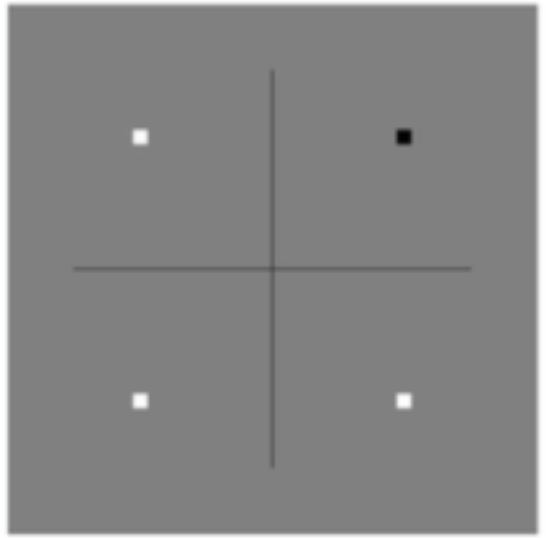
- Linearly separable dataset only.
- If you use it with non-linear data, you will not get 0% training error.
  => You will not get a consistent hypothesis on non-linear dataset



- Let's see some examples of its representational power.

Representative power

- Let's see if we can represent the AND function.
- Perceptron model:
  if $w_1x_1 + w_2x_2 > \Theta$ then Class = +1
  else class = -1
- Let's assume $w_1 = w_2 = 1$. You can vary Θ.
  if $x_1 + x_2 > \Theta$ then Class +1
  else class = -1
- Find the value of Θ
- Any number greater than 0 and less than 2
  e.g. 1.5



x1	x2	Class
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

Representative power

- Another way of representing AND.
- Perceptron model:
  if $w_0 + w_1x_1 + w_2x_2 > 0$ then Class = +1
  else class = -1
  Notice: $w_0 = - \Theta$
- Let's assume $w_1 = w_2 = 1$. You are looking for w_0 such that :
  if $w_0 + x_1 + x_2 > 0$ then Class +1
  else class = -1
- What is w_0 ?
  Any number less than 0 and greater than -2 e.g. -1.5 or -0.5

x1	x2	Class
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

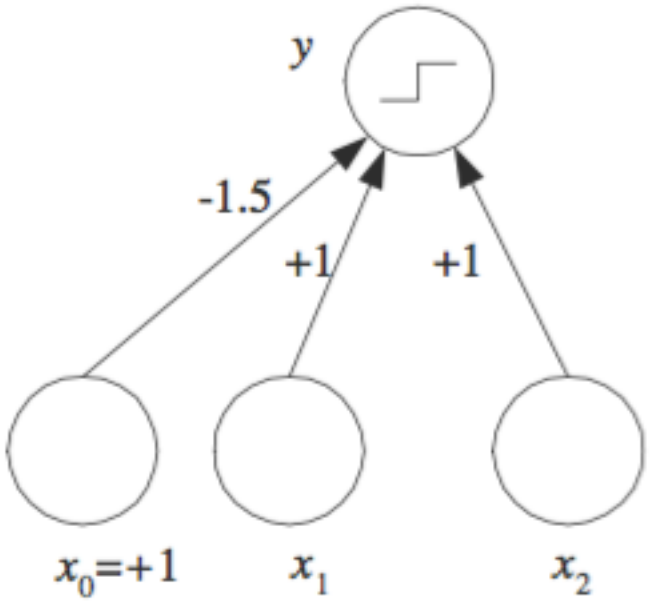


Figure 11.4 of Intro to ML textbook

Representative power

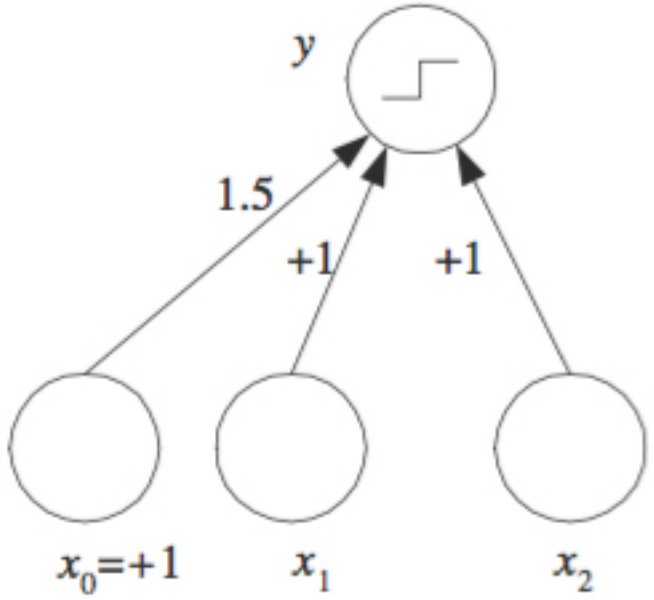
- Let's see if we can represent the OR function.
- Perceptron model:
  if $w_1x_1 + w_2x_2 > \Theta$ then Class = +1
  else class = -1
- Let's assume $w_1 = w_2 = 1$. You can vary Θ.
  if $x_1 + x_2 > \Theta$ then Class +1
  else class = -1
- What is Θ ?
- Any number less than 0, but greater than -2.
  Example: -0.5

x1	x2	Class
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

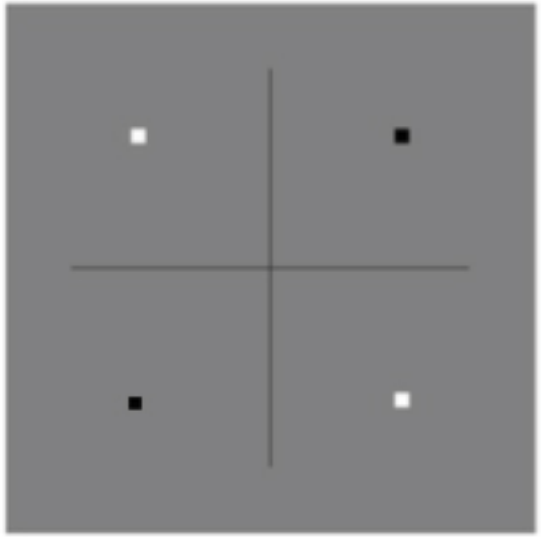
Representative power

- Another way of representing OR.
- Perceptron model:
  if $w_0 + w_1x_1 + w_2x_2 > 0$ then Class = +1
  else class = -1
  Notice: $w_0 = -\Theta$
- Let's assume $w_1 = w_2 = 1$. You are looking for w_0 such that :
  if $w_0 + x_1 + x_2 > 0$ then Class +1
  else class = -1
- What is w_0 ?
  Any number greater than 0, but less than 2.
  Example: 0.5 or 1.5

x1	x2	Class
-1	-1	-1
-1	1	1
1	-1	1
1	1	1



Representative power



- Let's see if we can represent the XOR function.
- Perceptron model:
  if $w_1x_1 + w_2x_2 > \Theta$ then Class = +1
  else class = -1
- Let's assume $w_1 = w_2 = 1$. You can vary Θ.
  if $x_1 + x_2 > \Theta$ then Class +1
  else class = -1
- What is Θ ?
  Can't find any such value

x1	x2	Class
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Canonical Representation

- In our framework, when do our hypothesis and actual value agree?
- Sign function plays a big role.
- Let's consider possible cases:

Hypothesis Prediction (h)	Actual Class (y)	Correct?
-1	-1	Yes
-1	1	No
1	1	Yes
1	-1	No

So, what do you conclude:
(h * y) should be > 0 for correct prediction
Some textbooks write it as:
$g = w^T x$
h = sign(w)
Check sign(h * y) for correct prediction

Perceptron

- So, the functional form of the hypothesis is given to you:
  $g(x) = w^T x$
- You classify a point based on sign(g)
- But the big question:
  How do you find weights?
- That's why companies are looking for you.

Learning Weights of a Perceptron

- Two techniques available: (Tom Mitchell 4.4.2 and 4.4.3)

1. Perceptron Training Rule – Change training weights according to:

$$w^{new} = w^{old} + \Delta w$$

where $$\Delta w = \eta(t - o)x_i$$

Start with random weights and apply above rule to each training example and modify each weight w_i according to above rule.

η is the learning rate

t is the target value that you want (given for training examples)

o is the current output of the perceptron.

Note: t and o can only have values in the set {-1, +1}

Perceptron Training Rule only works for linearly separable data.

Learning Weights of a Perceptron

- Two techniques available: (Tom Mitchell 4.4.2 and 4.4.3)

2. Gradient Descent Rule – First consider unthresholded output of the perceptron:

$$o = \mathbf{w} \cdot \mathbf{x}$$

Specify an error function:

$$E(w) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$

where *D* is the set of all training examples.

This works even if the data is not linearly separable

Just like before, run the gradient descent algorithm on this function by starting with random weights.

1. Perceptron Training Rule

Example:

- Given:
  - A set of instances $X_1, X_2, \ldots, X_D$ each containing n attributes
  i.e. $X_i = <x_1, x_2, \ldots, x_n>$
  * For LC, we assume the binary attributes can take values of +1 or -1 *
  - Labels for each instance $y_1, y_2, \ldots, y_D$ (can be +1 or -1)
- Example: UTD Internship Problem

GPA > 3.5 (x1)	Exp > 2 yrs (x2)	Internship (y)
1	-1	-1
1	1	1
-1	1	1

Can each attribute have different weight?
I want to give twice the weight to work experience as GPA.

Perceptron Training Rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

This is the practical rule that you will use to train a perceptron

Where:

- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., 0.1) called *learning rate*

Perceptron Training Algorithm

Perceptron_Training_Algorithm(training_examples, $\boldsymbol{\eta}$) {

Initialize weights vector w to some random value
Until convergence do {

For each training example {

Compute output o using activation function

For each linear unit (attribute) *i*, update using

$$w_i = w_i + \eta\,(t - o)\,x_i$$

}

}

}

Convergence

Will it converge?

Yes, provided:

- Data is linearly separable.
- The learning rate is small enough.

Example

- You initialize the weights vector as:

$$w = \begin{pmatrix} -2 \\ 1 \\ 2 \end{pmatrix}$$

Assume a learning rate $\boldsymbol{\eta}$ = 0.5. You are given following data points and target values:

x1	x2	t
0.5	1.5	+1
-0.5	0.5	-1
0.5	0.5	+1

Example

- For first point,

g(x) = sign(w.x) = sign(-2 + 0.5 +3) = +1

=> no change needed since t(x) = g(x)

- For second point,

g(x) = sign(w.x) = sign(-2-0.5+1.0) = -1

=> no change needed since t(x) = g(x)

- For third point,

g(x) = sign(w.x) = sign(-2+0.5+1.0) = -1

=> Need to update weights

Example

- Updating weights. Start with $\Delta w = 0$; t - o = 1 - (-1) = 2

Δw_0 = 0.5 * 2 * 1 = 1

Δw_1 = 0.5 * 2 * 0.5 = 0.5

Δw_2 = 0.5 * 2 * 0.5 = 0.5

So new values will be:

w_0 = -2 + 1 = -1.0

w_1 = 1 + 0.5 = 1.5

w_2 = 2 + 0.5 = 2.5

Example

So new values will be:

$w_0 = -2 + 1 = -1.0$

$w_1 = 1 + 0.5 = 1.5$

$w_2 = 2 + 0.5 = 2.5$

You can verify that this value of the weights vector will classify points correctly.

Example 2

No.	input values	desired output value
1.	$x^1 = (1, 0, 1)^T$	-1
2.	$x^2 = (0, -1, -1)^T$	1
3.	$x^3 = (-1, -0.5, -1)^T$	1

The learning constant is assumed to be 0.1. The initial weight vector is $w^0 = (1, -1, 0)^T$.

Example 2

- **[Step 1]** Input x^1, desired output is -1:

$$< w^0, x^1 >= (1, -1, 0) \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = 1$$

Correction in this step is needed since

$$y_1 = -1 \neq \text{sign}(1).$$

We thus obtain the updated vector

$$w^1 = w^0 + 0.1(-1 - 1)x^1$$

Plugging in numerical values we obtain

$$w^1 = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix} - 0.2 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.8 \\ -1 \\ -0.2 \end{pmatrix}$$

Example 2

[Step 2] Input is x^2, desired output is 1. For the present w^1 we compute the activation value

$$< w^1, x^2 >= (0.8, -1, -0.2) \begin{pmatrix} 0 \\ -1 \\ -1 \end{pmatrix} = 1.2$$

Correction is not performed in this step since $1 = \text{sign}(1.2)$, so we let $w^2 := w^1$.

Example 2

- **[Step 3]** Input is x_3, desired output is 1.

$$< w^2, x^3 >= (0.8, -1, -0.2) \begin{pmatrix} -1 \\ -0.5 \\ -1 \end{pmatrix} = -0.1$$

Correction in this step is needed since $y_3 = 1 \neq \text{sign}(-0.1)$.
We thus obtain the updated vector

$$w^3 = w^2 + 0.1(1+1)x^3$$

Plugging in numerical values we obtain

$$w^3 = \begin{pmatrix} 0.8 \\ -1 \\ -0.2 \end{pmatrix} + 0.2 \begin{pmatrix} -1 \\ -0.5 \\ -1 \end{pmatrix} = \begin{pmatrix} 0.6 \\ -1.1 \\ -0.4 \end{pmatrix}$$

Example 2

- **[Step 4]** Input x^1, desired output is -1:

$$< w^3, x^1 >= (0.6, -1.1, -0.4) \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = 0.2$$

Correction in this step is needed since

$$y_1 = -1 \neq \text{sign}(0.2).$$

We thus obtain the updated vector

$$w^4 = w^3 + 0.1(-1 - 1)x^1$$

Plugging in numerical values we obtain

$$w^4 = \begin{pmatrix} 0.6 \\ -1.1 \\ -0.4 \end{pmatrix} - 0.2 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.4 \\ -1.1 \\ -0.6 \end{pmatrix}$$

Example 2

- **[Step 5]** Input is x^2, desired output is 1. For the present w^4 we compute the activation value

$$< w^4, x^2 >= (0.4, -1.1, -0.6) \begin{pmatrix} 0 \\ -1 \\ -1 \end{pmatrix} = 1.7$$

Correction is not performed in this step since $1 = \text{sign}(1.7)$, so we let $w^5 := w^4$.

- **[Step 6]** Input is x_3, desired output is 1.

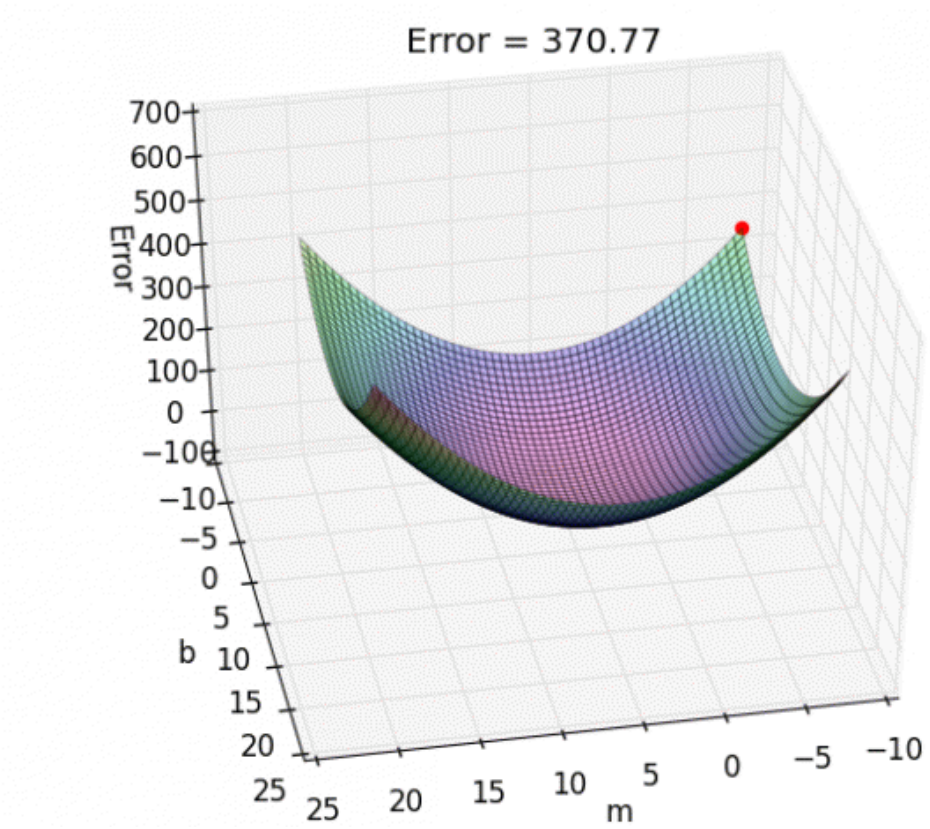
$$< w^5, x^3 >= (0.4, -1.1, -0.6) \begin{pmatrix} -1 \\ -0.5 \\ -1 \end{pmatrix} = 0.75$$

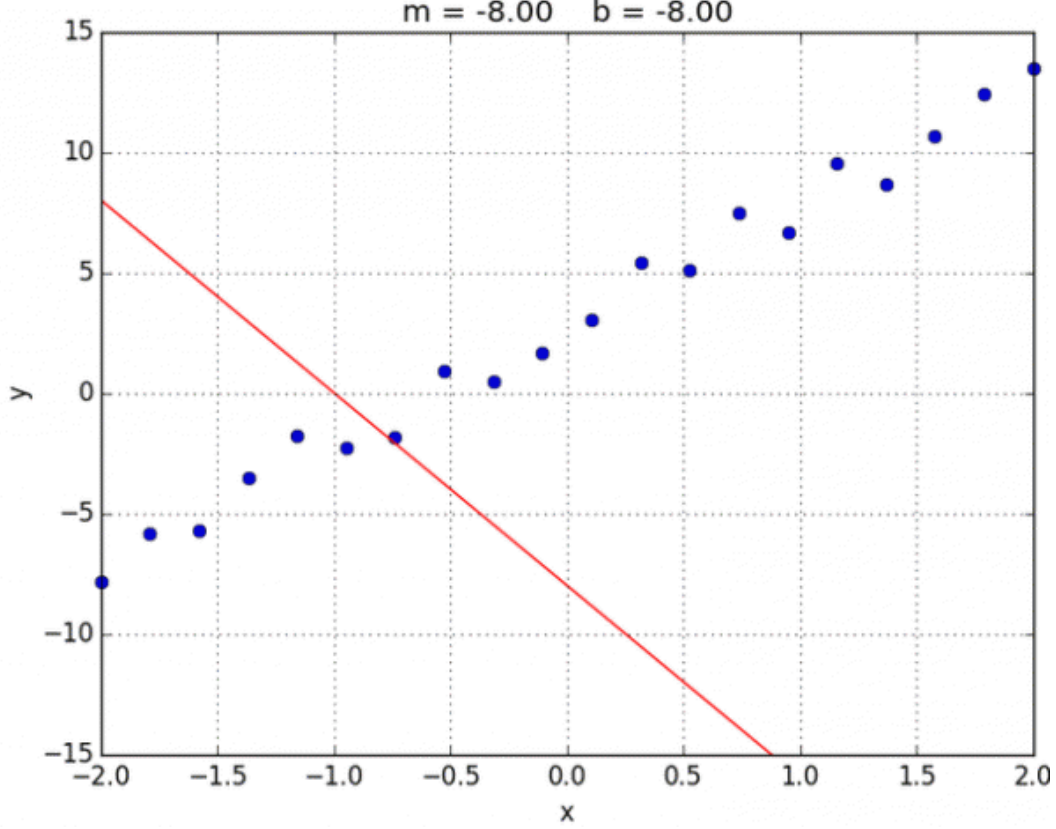
Correction is not performed in this step since $1 = \text{sign}(0.75)$, so we let $w^6 := w^5$.

Example 2

This terminates the learning process, because

$$< w^6, x_1 >= -0.2 < 0,$$
$$< w^6, x_2 >= 1.70 > 0,$$
$$< w^6, x_3 >= 0.75 > 0.$$





2. Gradient Descent

Gradient Descent

To understand, consider simpler *linear unit*, where

$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

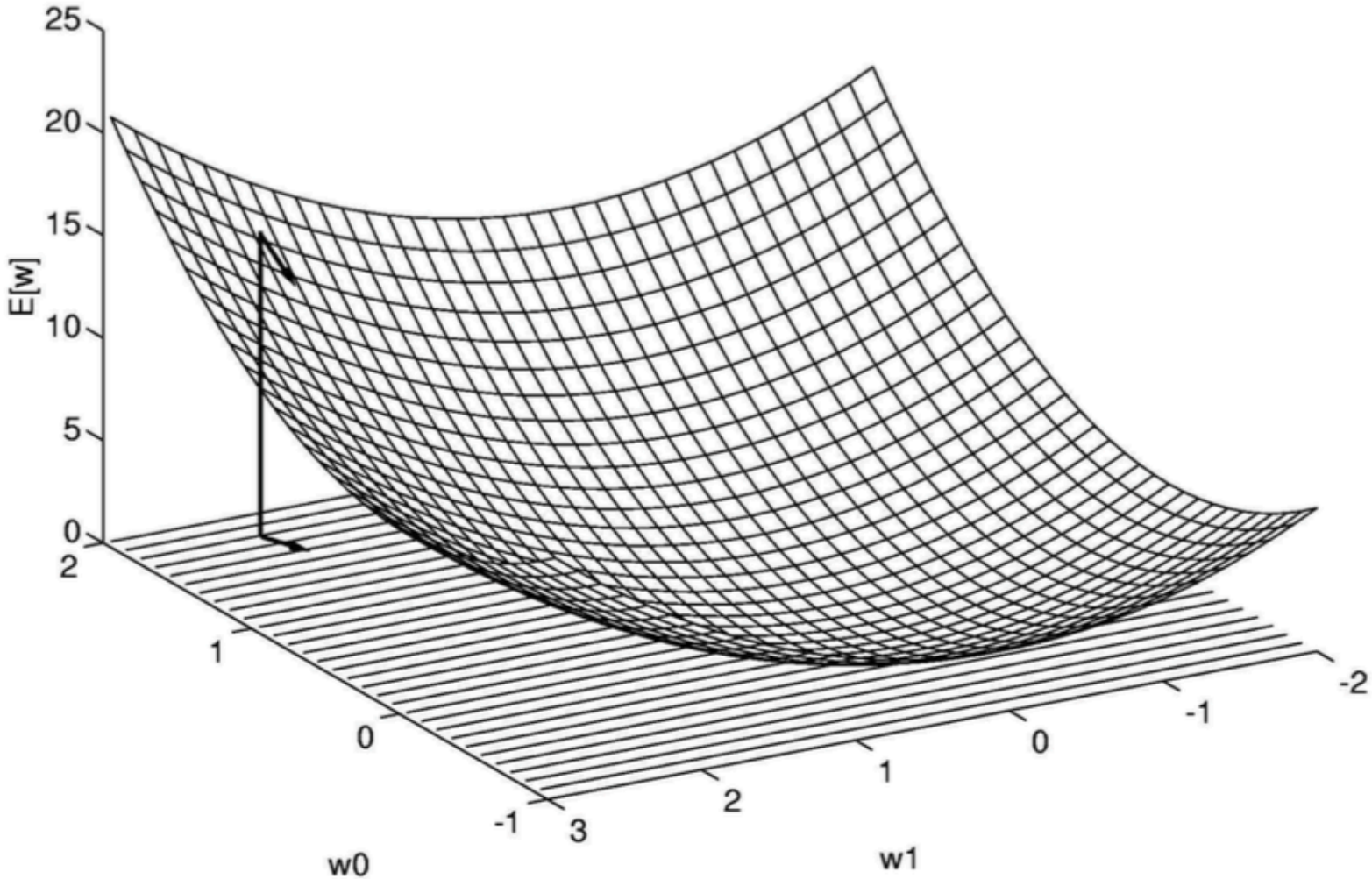
Note: Now the output o is not restricted to values 0 and 1 only, even though the input x values may be Boolean.

Let's learn w_i's that minimize the squared error

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Where D is set of training examples

Gradient Descent



Gradient:

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

I.e.:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Gradient Descent

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\
&= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x_d}) \\
\frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d)(-x_{i,d})
\end{aligned}
$$

Gradient Descent

- The value of Δw_i can be obtained as:

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id}$$

where

η is the learning rate

D is the set of all training example

t_d is the target output for the d^{th} training example

o_d is the actual output for the d^{th} training example

x_{id} is the value of the i^{th} attribute for the d^{th} training example

Two ways to update weights

Traditional Gradient Descent

Start with a random value of w
and $\Delta w_i = 0$ for all attributes i
for all training examples $x^{(j)}$ {

Compute output o
Compare with t and compute
each $\Delta w_i^{(j)}$

Update $\Delta w_i = \Delta w_i + \Delta w_i^{(j)}$
}

Update each w_i as
$w_i = w_i + \Delta w_i$

Stochastic Gradient Descent

Start with a random value of w
and $\Delta w_i = 0$ for all attributes i
for all training examples $x^{(j)}$ {

Compute output o
Compare with t and compute
each $\Delta w_i^{(j)}$

Update each w_i as
$w_i = w_i + \Delta w_i^{(j)}$

This is the difference

}

Difference

Traditional gradient descent can lead to:

1. Convergence to local minimum can be quite slow (many thousand of gradient descent steps)
2. If there are multiple local minima in the error surface, then there is no guarantee that the procedure will find global minimum.

Stochastic gradient descent can alleviate these problems.
(Section 4.4.3.3 of Tom Mitchell)

Gradient Descent

GRADIENT-DESCENT($training\_examples, \eta$)

Initialize each w_i to some small random value

Until the termination condition is met, Do

- Initialize each Δw_i to zero.
- For each $\langle \vec{x}, t \rangle$ in $training\_examples$, Do
  - Input instance $\vec{x}$ to unit and compute output o
  - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

- For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \Delta w_i$$

Thought Questions:

- Pros and Cons
  - Only works with linearly separable data
  - Is error driven
  - Does it converge? If you input data that is not separable, what would happen?
  - Efficient?
  - Easy to visualize
  - Can weight attributes
  - Activation function concept is nice

What is next?

- How do we extend it to non-linear datasets?