# Machine Learning

# Topic: Linear Discriminants

Bryan Pardo and Zach Wood-Doughty, EECS 349 Machine Learning, 2021

Thanks to Mark Cartwright for his contributions to these slides
Thanks to Alpaydin, Bishop, and Duda/Hart/Stork for images and ideas

# Recall: Regression Learning Task

There is a set of possible examples $X = \{\mathbf{x_1}, \ldots \mathbf{x_n}\}$

Each example is a **vector** of k **real valued attributes**

$$\mathbf{x}_i = <x_{i1}, \ldots, x_{ik}>$$

A target function maps $X$ onto some **real value** $Y$

$$f : X \rightarrow Y$$

The DATA is a set of tuples <example, response value>

$$\{<\mathbf{x_1}, y_1>, \ldots <\mathbf{x_n}, y_n>\}$$

Find a hypothesis **h** such that…

$$\forall \mathbf{x}, h(\mathbf{x}) \approx f(\mathbf{x})$$

# Discrimination Learning Task

There is a set of possible examples $X = \{\mathbf{x_1}, \ldots \mathbf{x_n}\}$

Each example is a **vector** of k **real valued attributes**

$$\mathbf{x}_i = <x_{i1}, \ldots, x_{ik}>$$

A target function maps $X$ onto some **categorical variable** $Y$

$$f : X \rightarrow Y$$

The DATA is a set of tuples <example, response value>

$$\{<\mathbf{x_1}, y_1>, \ldots <\mathbf{x_n}, y_n>\}$$

Find a hypothesis **h** such that...

$$\forall \mathbf{x}, h(\mathbf{x}) \approx f(\mathbf{x})$$

# Reminder about notation

- $\mathbf{x}$ is a vector of attributes $<x_1, x_2,...x_k>$

- $\mathbf{w}$ is a vector of weights $<w_1, w_2,...w_k>$

- Given this...

$$g(x) = w_0 + w_1 x_1 + w_2 x_2 .... + w_k x_k$$

- We can notate it with linear algebra as

$$g(x) = w_0 + \mathbf{w^T x}$$
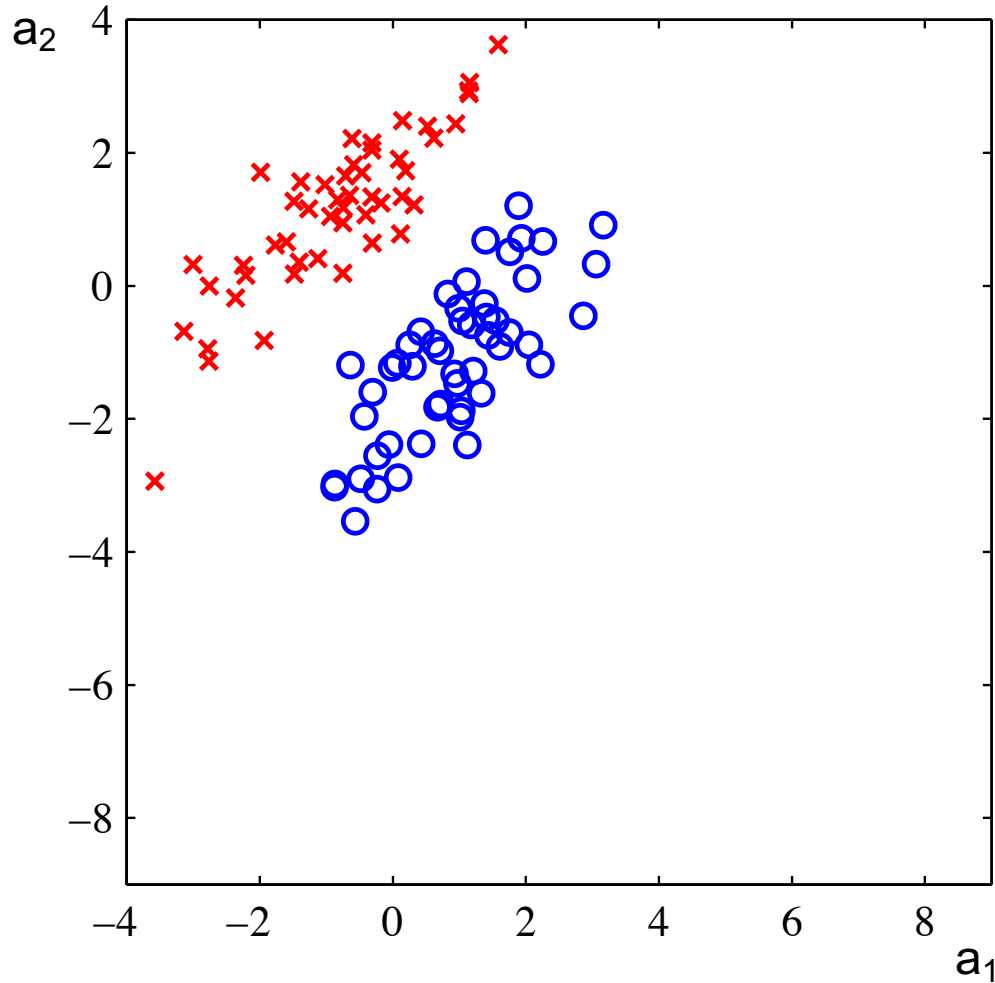
# **Recall:** $w_0$

- $g(x) = w_0 + \mathbf{w^T x}$ is ALMOST what we want, but that pesky offset $w_0$ is not in the linear algebra part yet.

- If we define $\mathbf{w}$ to include $w_0$ and $\mathbf{x}$ to include an $x_0$ that is always 1, now…

  $\mathbf{x}$ is a vector of attributes $<1, x_1, x_2, …x_k>$

  $\mathbf{w}$ is a vector of weights $<w_0, w_1, w_2, …w_k>$

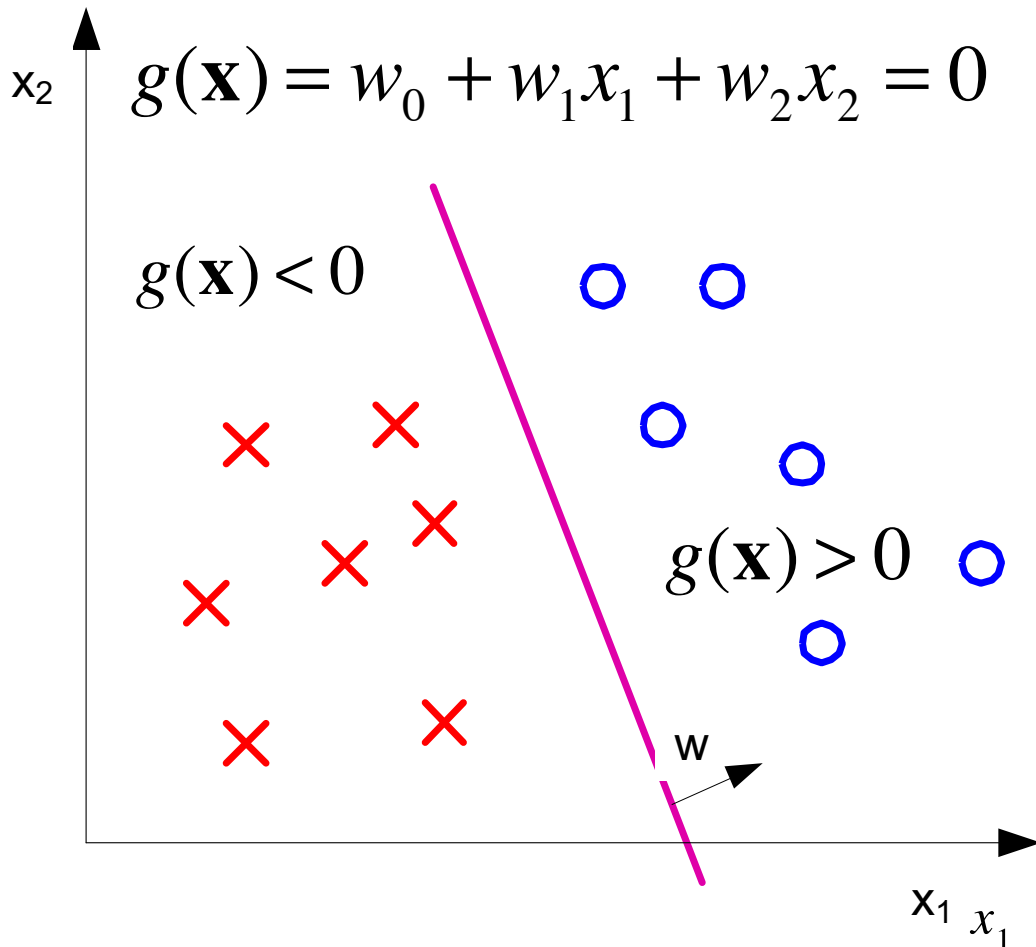- This lets us notate things as…
  $$g(x) = \mathbf{w^T x}$$

# Visually: Where to draw the line?

# Two-Class Classification

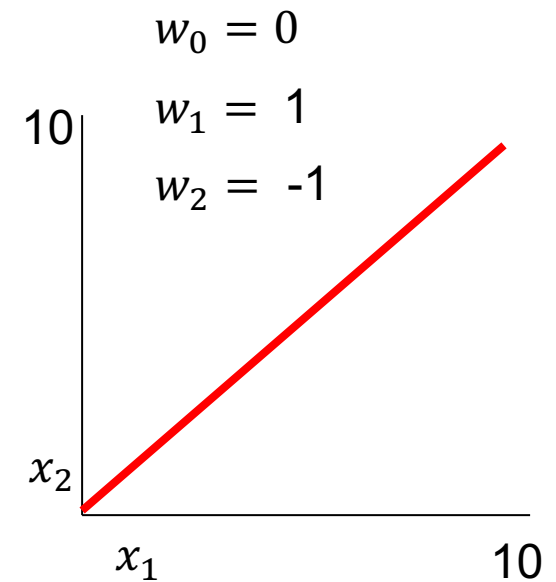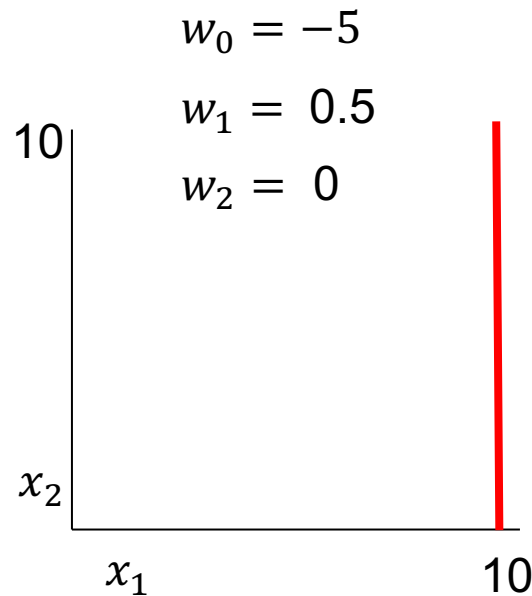$g(\mathbf{x}) = 0$ defines a decision boundary that splits the space in two

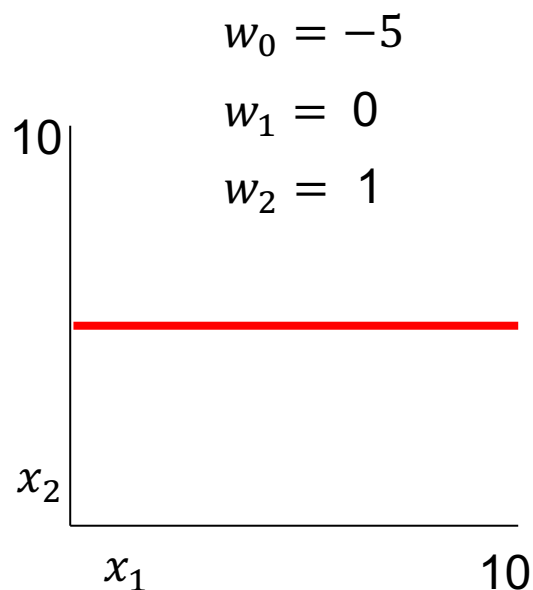$$g(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 = 0$$

If a line exists that does this without error, the classes are *linearly separable*

$g(\mathbf{x}) < 0$

$g(\mathbf{x}) > 0$

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } g(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

$x_2$

$x_1$ $x_1$

w

# Example 2-D decision boundaries

$$0 = g(x) = w_0 + w_1 x_1 + w_2 x_2 = \mathbf{w}^T\mathbf{x}$$

$w_0 = -5$
$w_1 = 0$
$w_2 = 1$

$w_0 = -5$
$w_1 = 0.5$
$w_2 = 0$

$w_0 = 0$
$w_1 = 1$
$w_2 = -1$
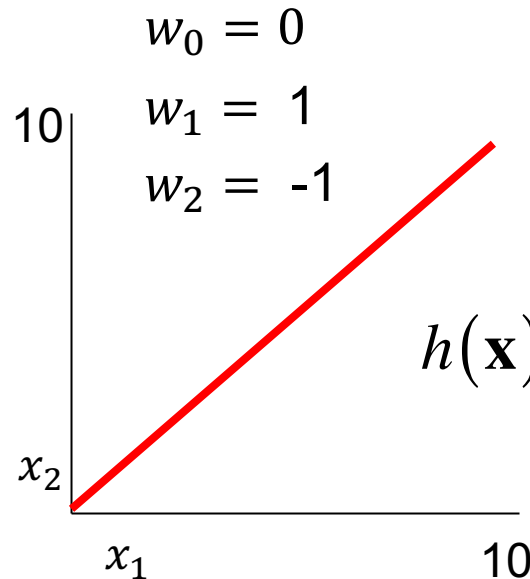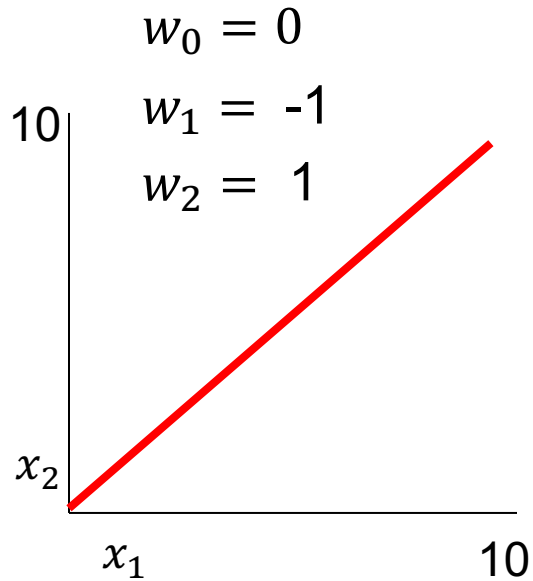
# What's the difference?

$$0 = g(x) = w_0 + w_1 x_1 + w_2 x_2 = \mathbf{w}^T \mathbf{x}$$

What's the difference between these two?

$w_0 = 0$
$w_1 = -1$
$w_2 = 1$

10

$x_2$

$x_1$      10

$w_0 = 0$
$w_1 = 1$
$w_2 = -1$

10

$x_2$

$x_1$      10

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } g(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

# Loss/Objective function

- To train a model (e.g. learn the weights of a useful line) we define a measure of the "goodness" of that model. (e.g. the number of misclassified points).

- We make that measure a function of the parameters of the model (and the data).

- This is called a loss function, or an objective function.

- We want to minimize the loss (or maximize the objective) by picking good model parameters.

# Classification via regression

- Linear regression's loss function is the the squared distance from a data point to the line, summed over all data points.

- The line that minimizes this function can be calculated by applying a simple formula.

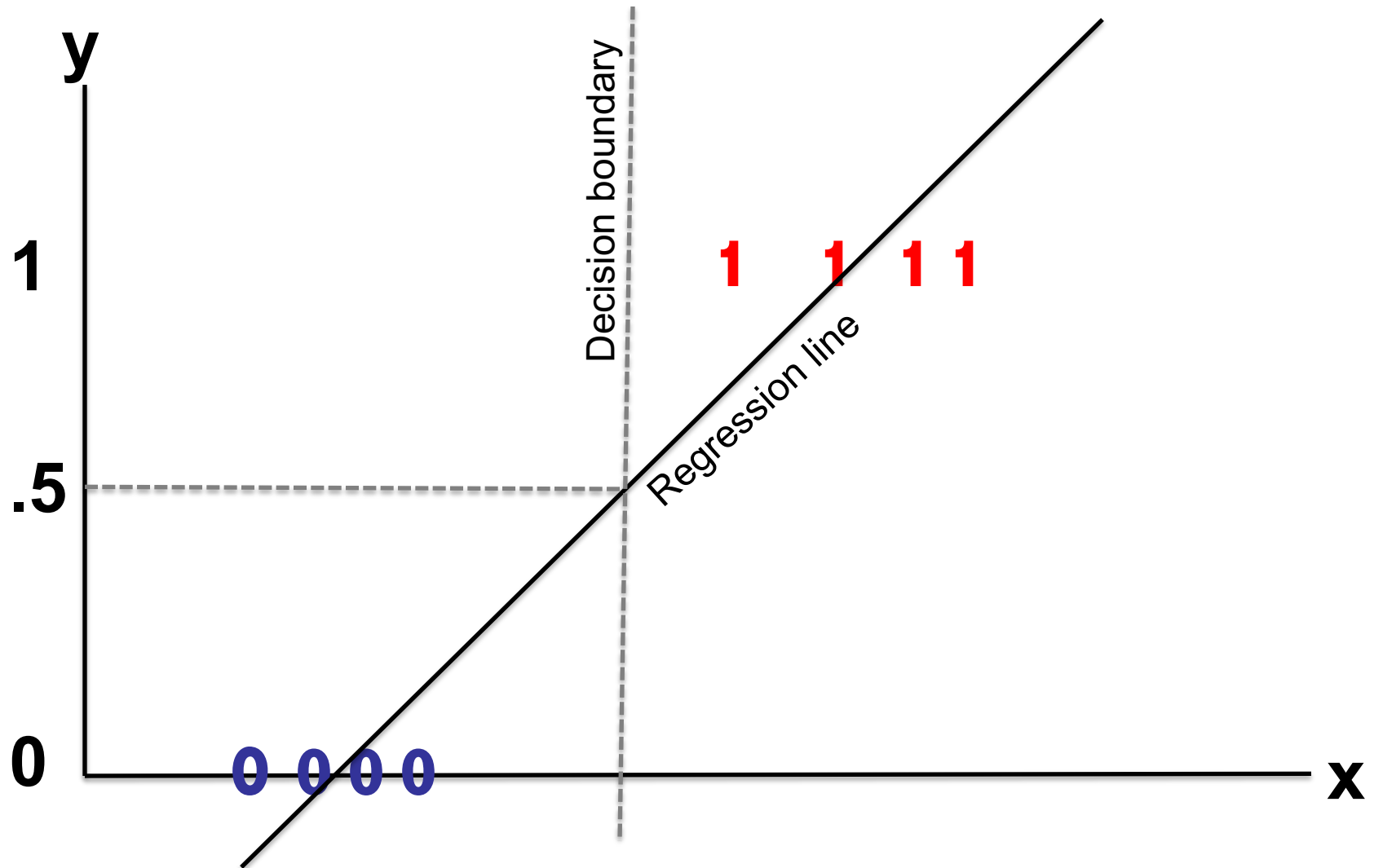$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

- Can we find a decision boundary in one step, by just repurposing the math we used for finding a regression line?
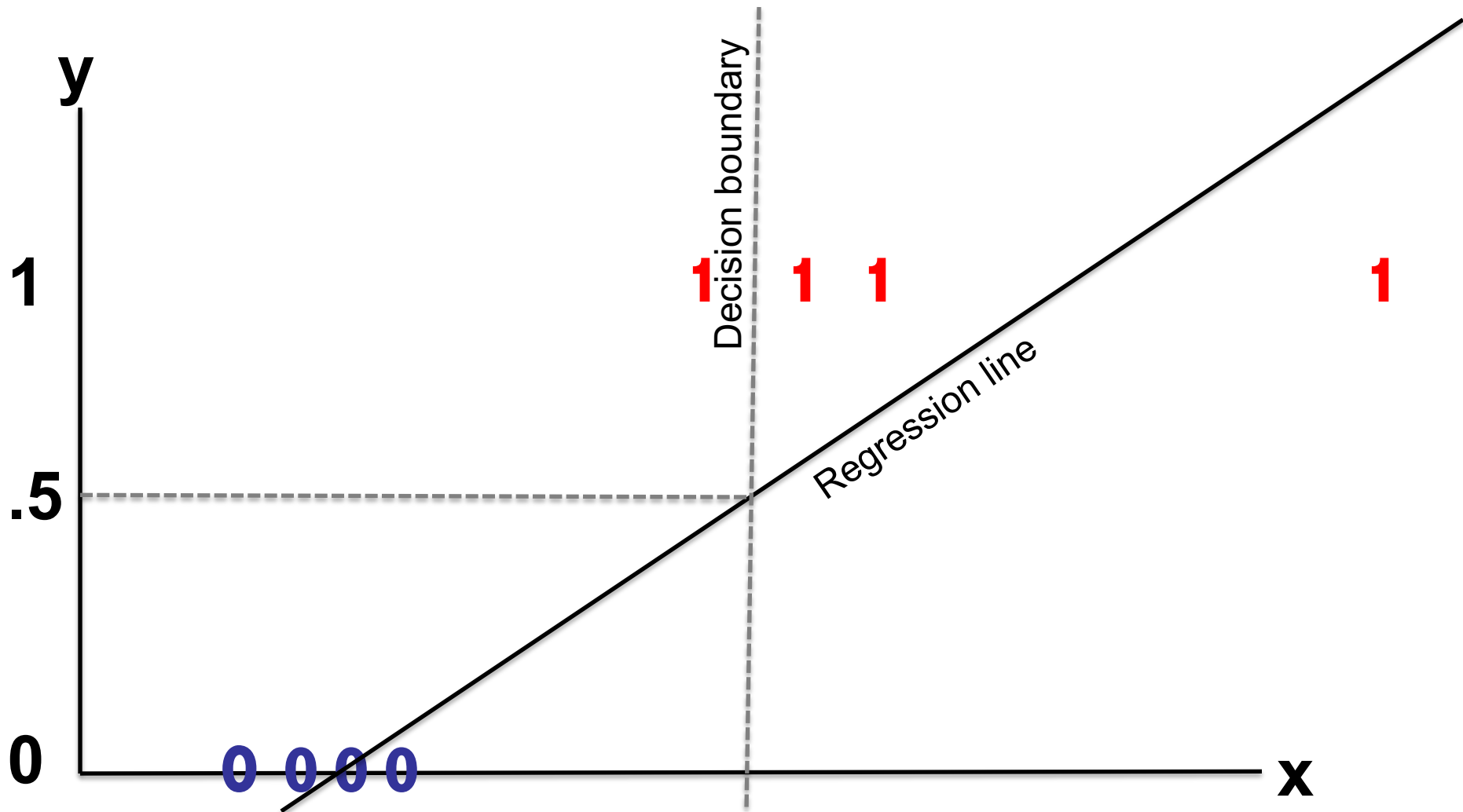
# Classification via regression

- Label each class by a number

- Call that number the response variable

- Derive closed-form regression solution

- Round the regression prediction to the nearest label number

# An example

# What happens now?

# **Classification via regression take-away**

- Closed form solution: simple formula for getting the regression line

- Residual sum of squares is a bad fit for classification: very sensitive to outliers

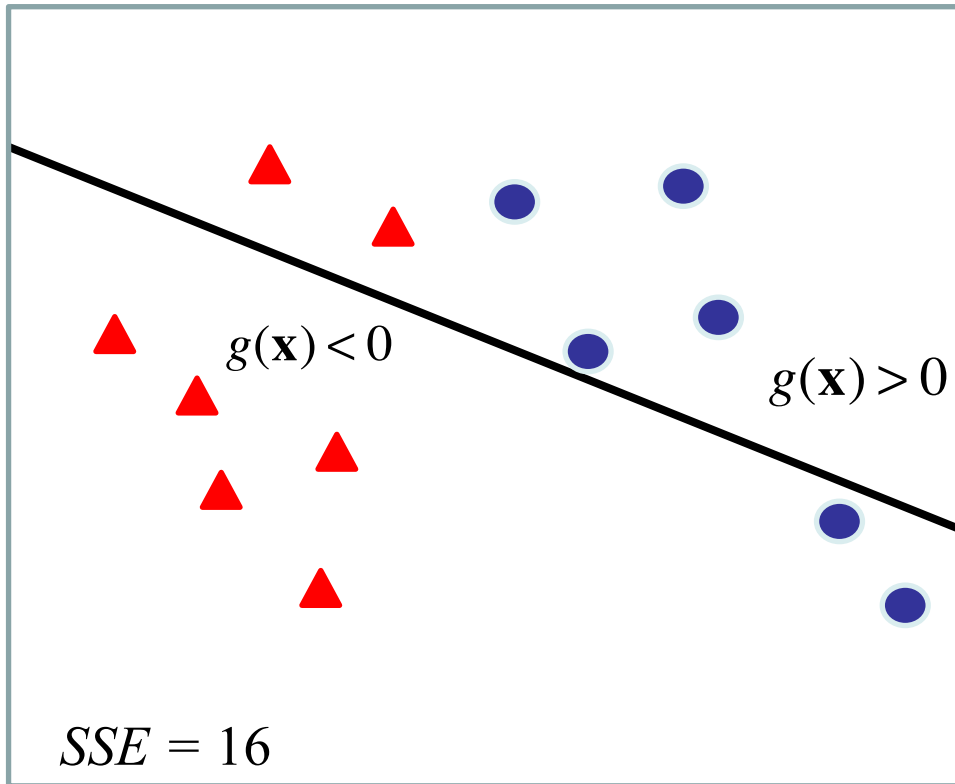- What's the natural mapping from categories to the real numbers?

# What can we do instead?

- Let's define an objective (aka "loss") function that directly measures **classification** quality

- Then find the line that minimizes that loss

- How about basing our loss function on the number of misclassifications?
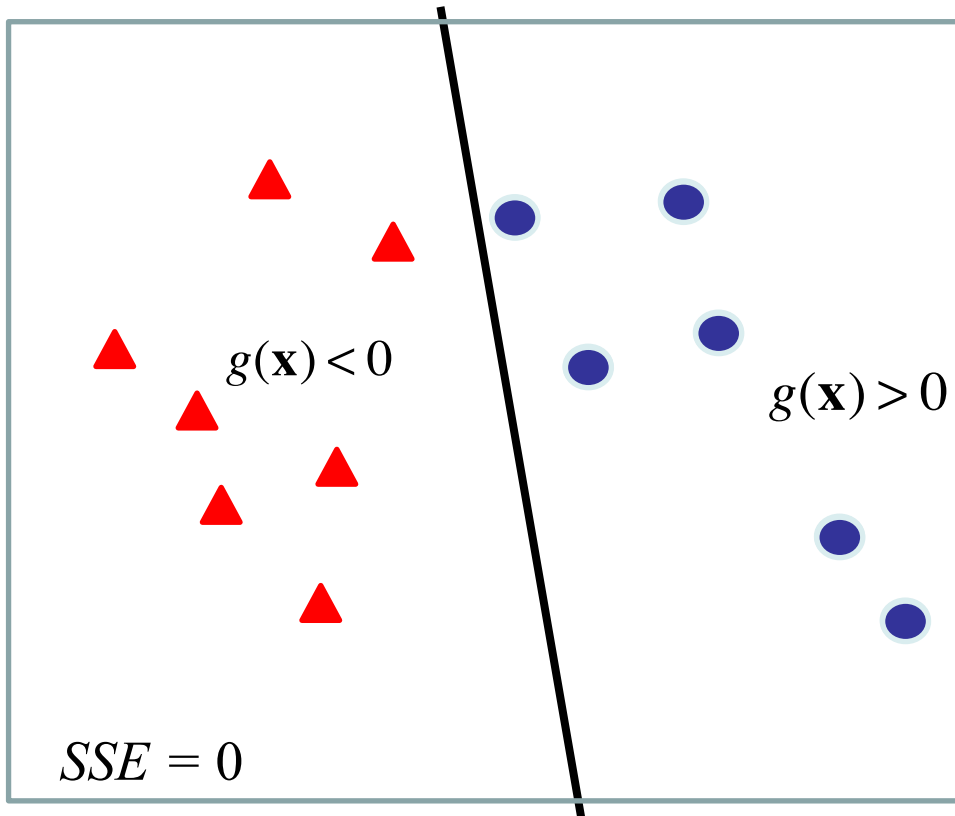
# sum of squared errors (SSE)



$$g(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 = 0$$
$$= \mathbf{w}^T \mathbf{x}$$

$g(\mathbf{x}) < 0$

$g(\mathbf{x}) > 0$

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } g(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$SSE = \sum_{i}^{n} (y_i - h(\mathbf{x}_i))^2$$

$SSE = 16$

# sum of squared errors (SSE)

$$g(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 = 0$$
$$= \mathbf{w}^T \mathbf{x}$$

$g(\mathbf{x}) < 0$

$g(\mathbf{x}) > 0$

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } g(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

$SSE = 0$

$$SSE = \sum_{i}^{n} (y_i - h(\mathbf{x}_i))^2$$

# No closed form solution!

- For many objective functions we can't find a formula to to get the best model parameters, like we could with regression.

- The objective function from the previous slide is one of those "no closed form solution" functions.

- This means we have to try various guesses for what the weights should be and try them out.

- Let's look at the perceptron approach.

# Let's learn a decision boundary

- We'll do 2-class classification
- We'll learn a linear decision boundary
    $$0 = g(x) = \mathbf{w^T x}$$
-  Things on each side of 0 get their class labels according to the sign of what g(x) outputs.

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } g(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

- We will use the Perceptron algorithm.

# The Perceptron

- The "first wave" in neural networks

- A linear classifier

# A single perceptron

$x_1$

$x_0$ Bias $x_0 = 1$

Input vector

$w_1$

$w_0$

$x_2$

$w_2$

$x_3$

$w_3$

$x_4$ $w_4$

$x_5$ $w_5$

$$f(x) = \begin{cases} 1 \ if \ 0 < \sum_{i=0}^{n} w_i x_i \\ \\ 0 \ else \end{cases}$$

Step function

output

Sum of weighted inp
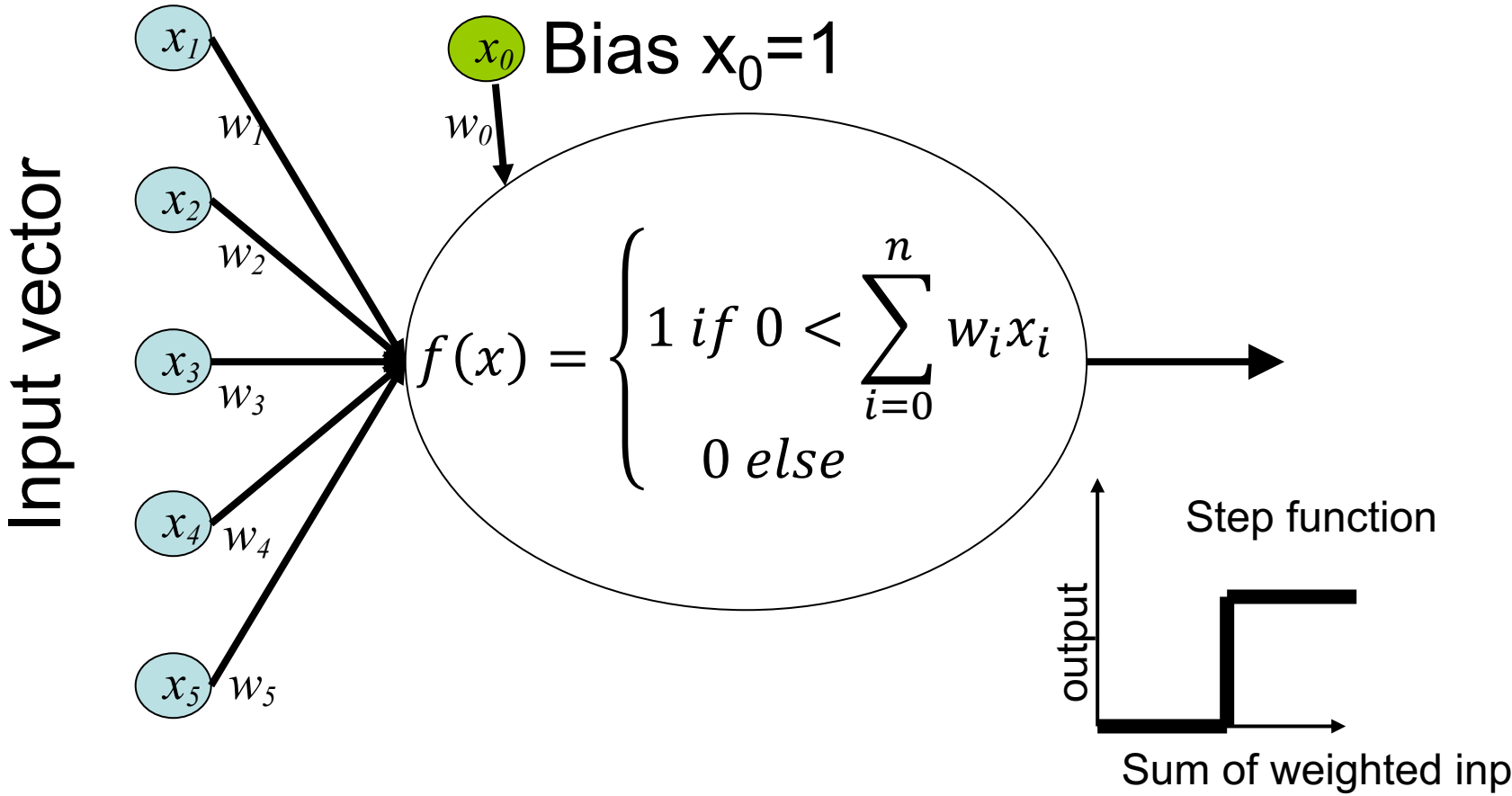
# Weights define a hyperplane in the input space



$$f(x) = \begin{cases} 1 \ if \ 0 < \sum_{i=0}^{n} w_i x_i \\ 0 \ else \end{cases}$$

**AND**

$0.5x_1 + 0.5x_2 - 0.6 = 0$

**1 region**

**0 region**
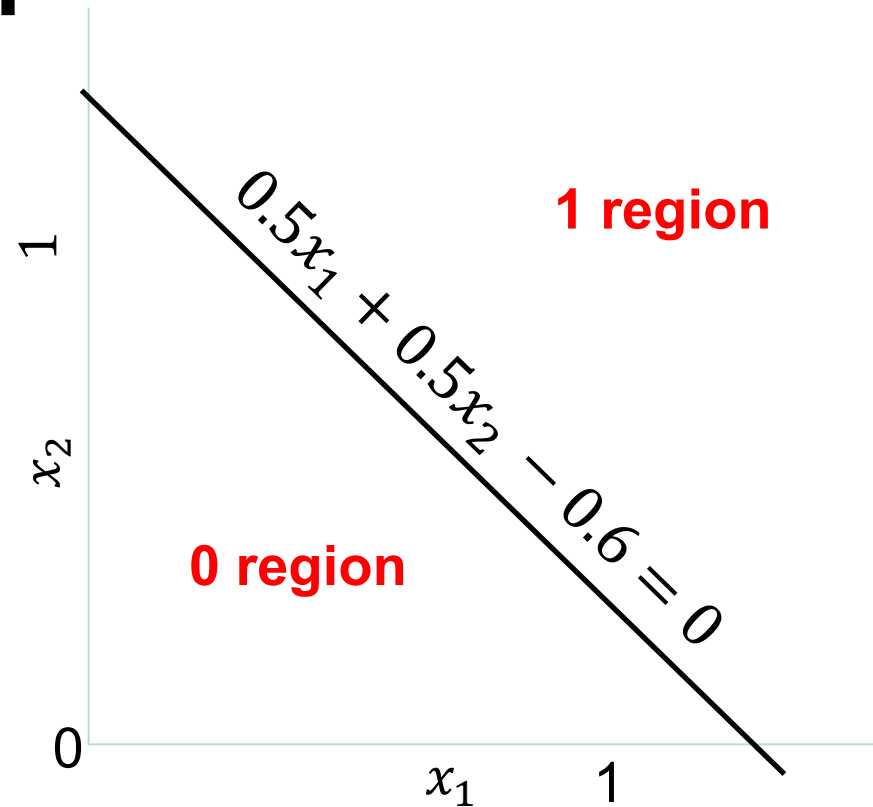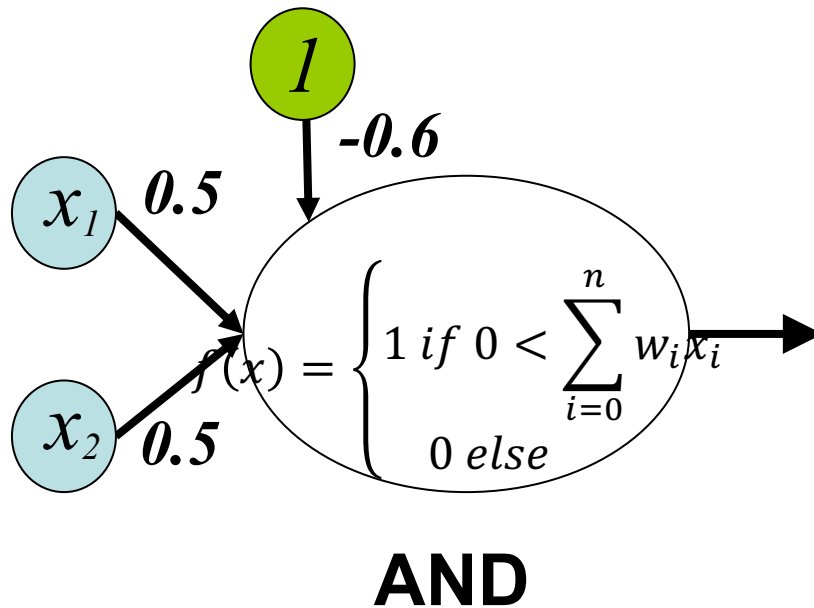
# Classifies any (linearly separable) data



$$f(x) = \begin{cases} 1 \ if \ 0 < \sum_{i=0}^{n} w_i x_i \\ 0 \ else \end{cases}$$

**1**

**-0.6**

$x_1$ **0.5**

$x_2$ **0.5**

**AND**

**1 region**

**0 region**

$0.5x_1 + 0.5x_2 - 0.6 = 0$

$x_2$

$x_1$

1
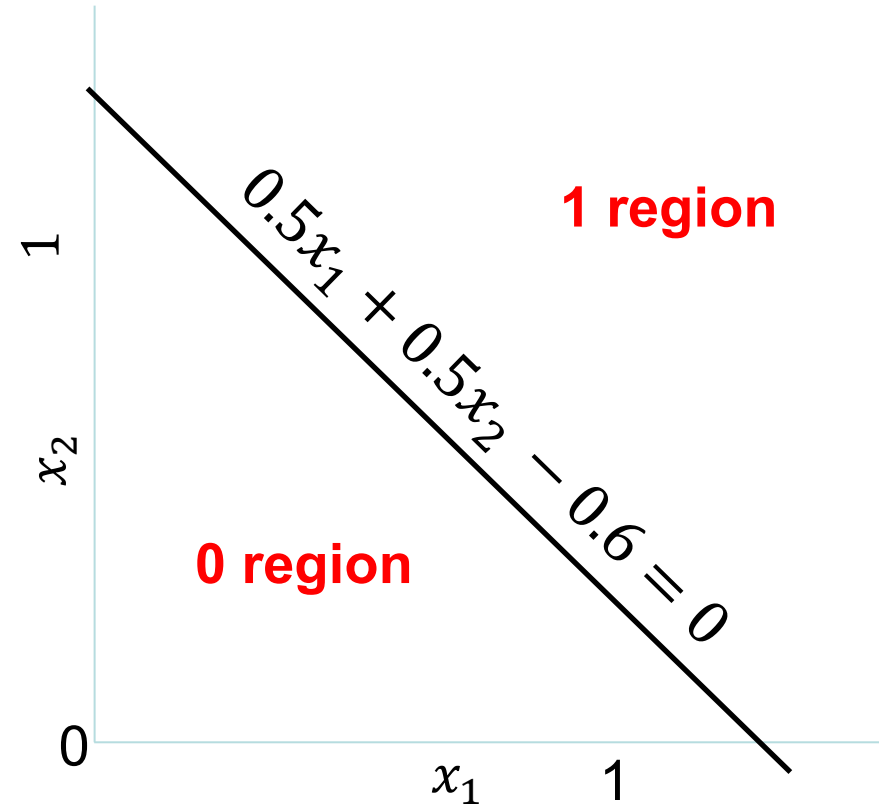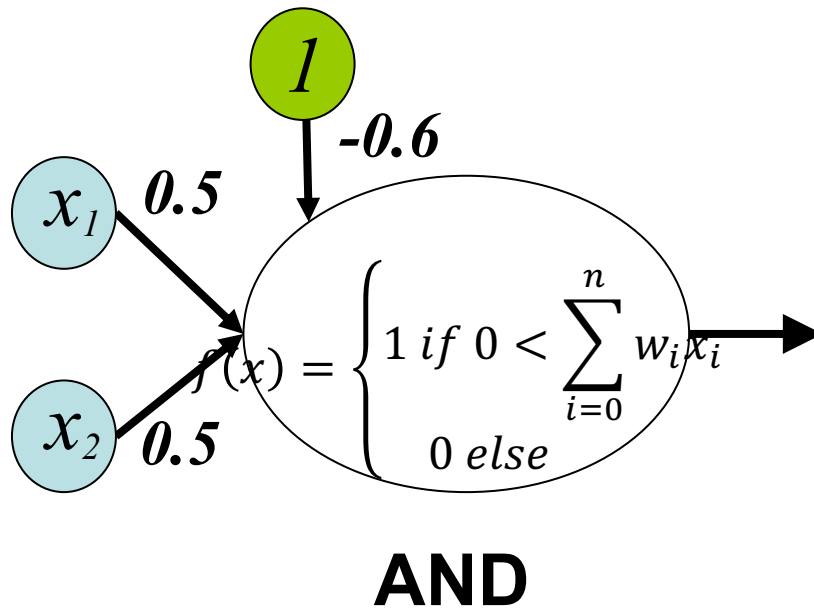
# Different logical functions are possible



$$f(x) = \begin{cases} 1 \; if \; 0 < \sum_{i=0}^{n} w_i x_i \\ 0 \; else \end{cases}$$

**OR**

$0.5 x_1 + 0.5 x_2 - 0.3 = 0$

**1 region**

**0 region**
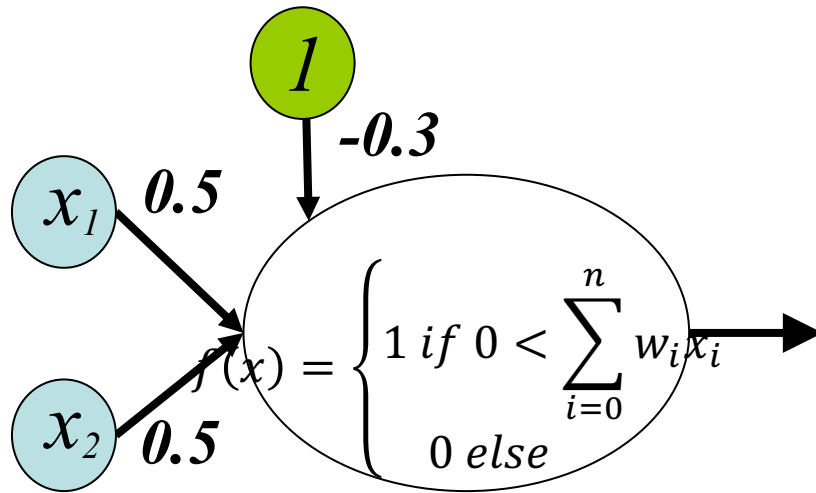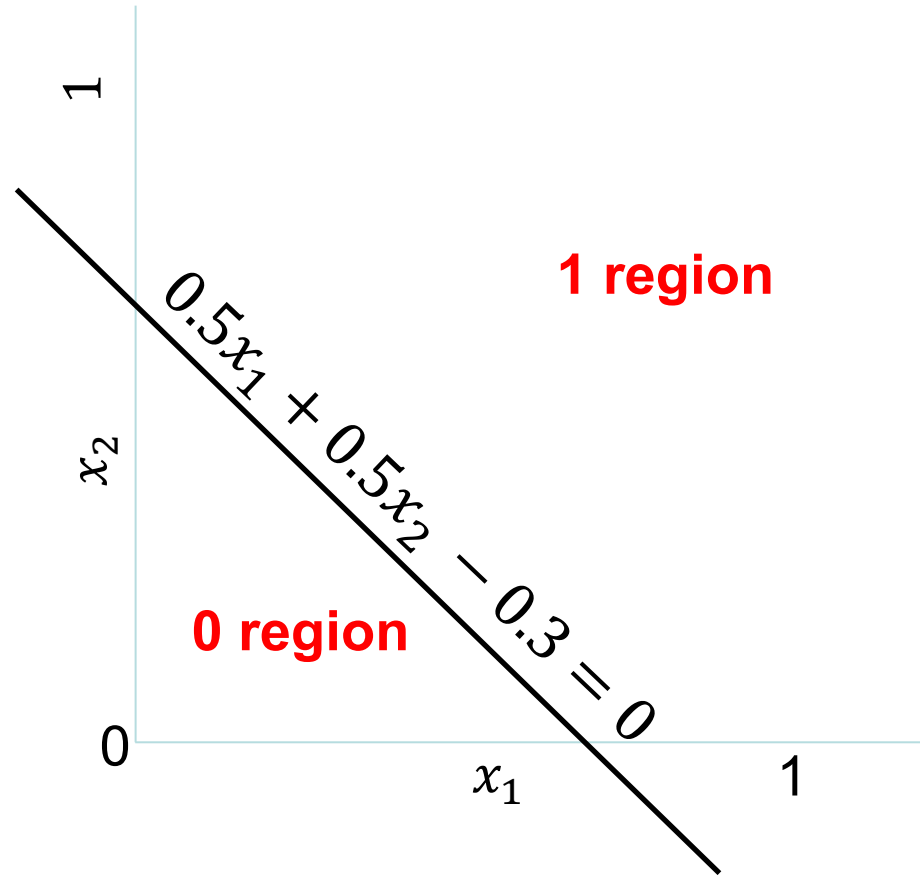
# And, Or, Not are easy to define

1

0 region

$-x_2 = 0$

$x_2$

1 region

-1

1

0

$x_1$ 0

$x_2$ -1

$f(x) = \begin{cases} 1 \ if \ 0 < \sum_{i=0}^{n} w_i x_i \\ 0 \ else \end{cases}$

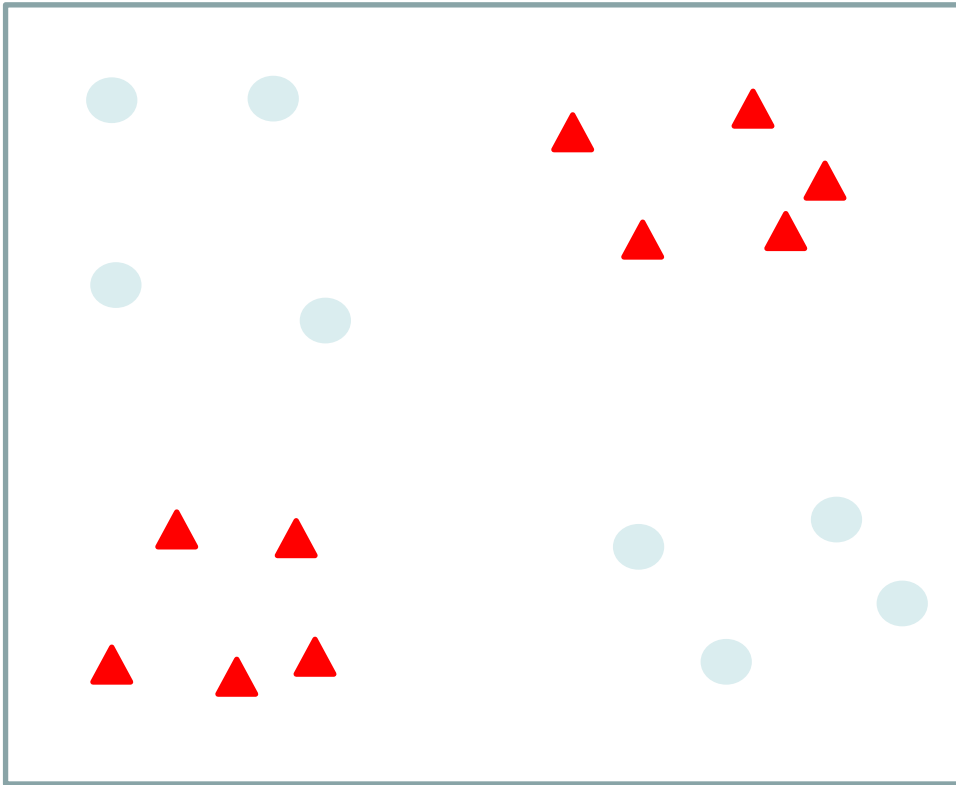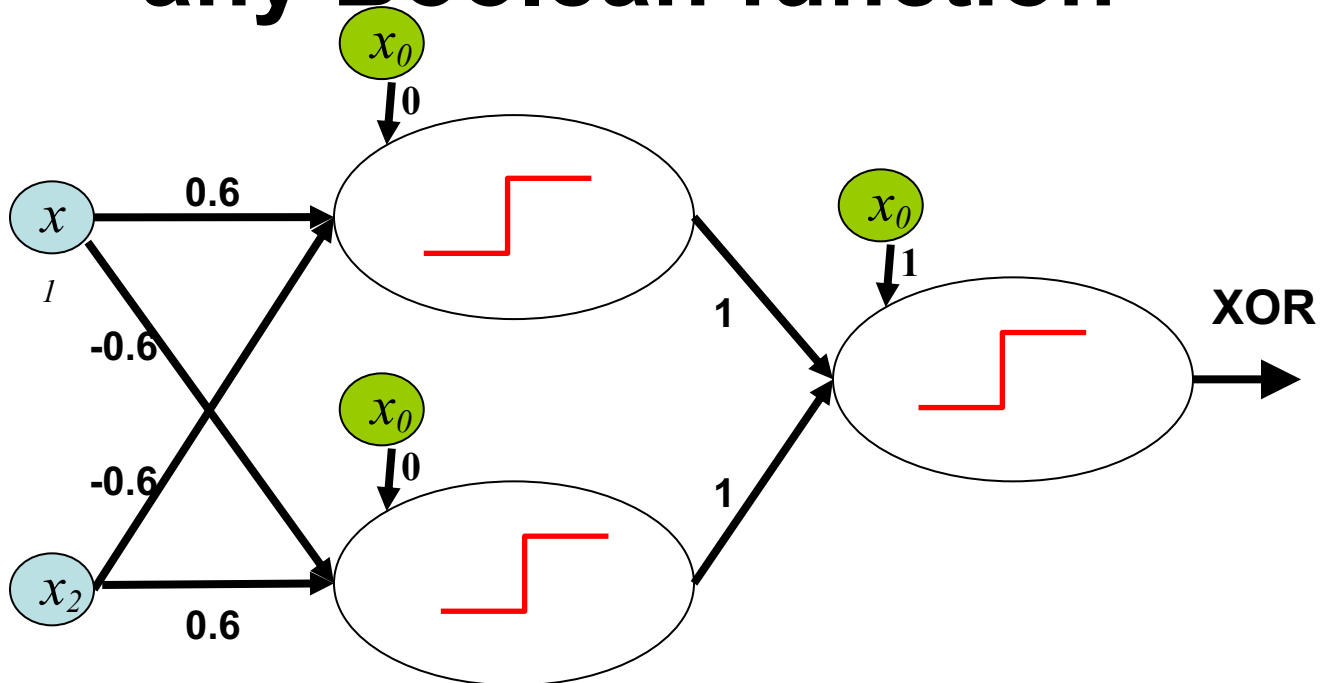**NOT**
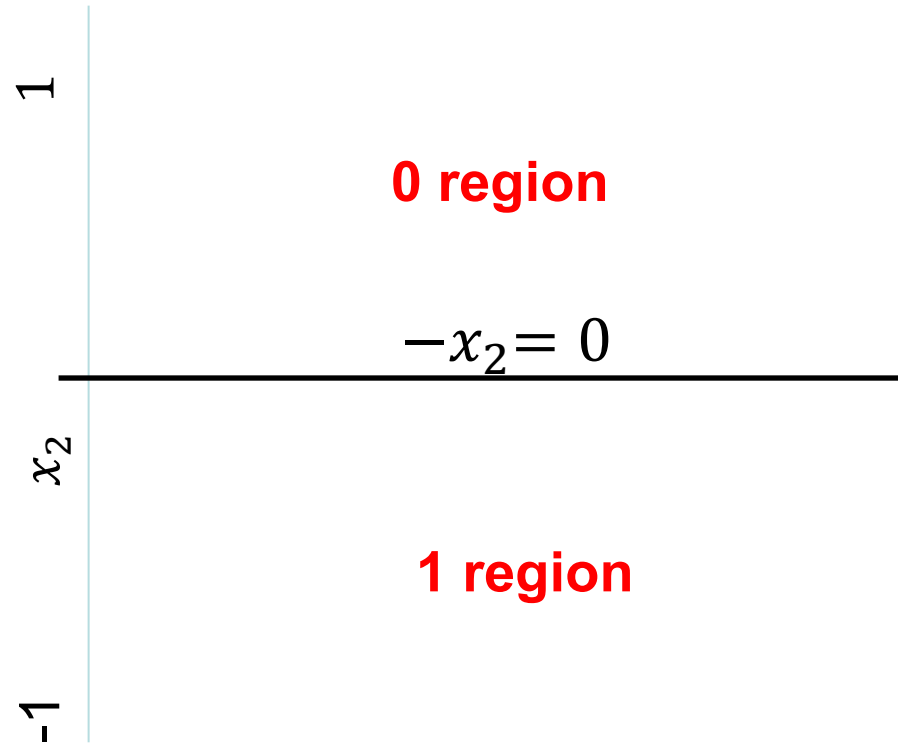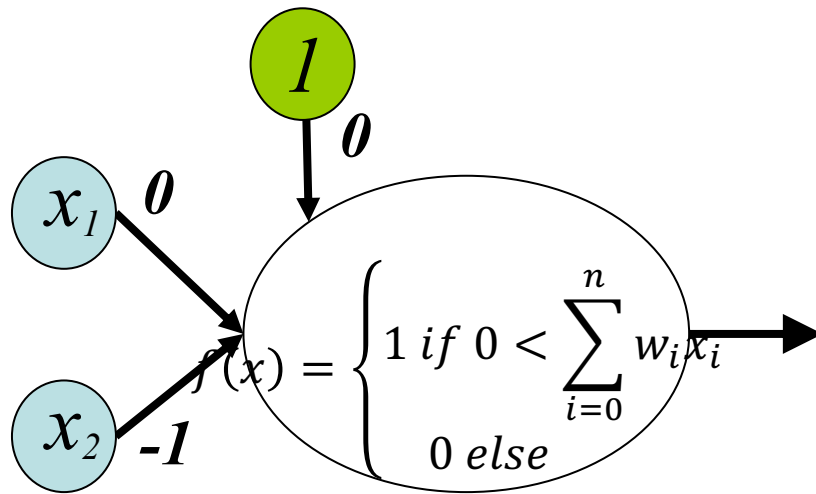
# One perceptron: Only linear decisions

This is XOR.

It can't learn XOR.

# Combining perceptrons can make any Boolean function



…if you can set the weights & connections right

# Defining our goal

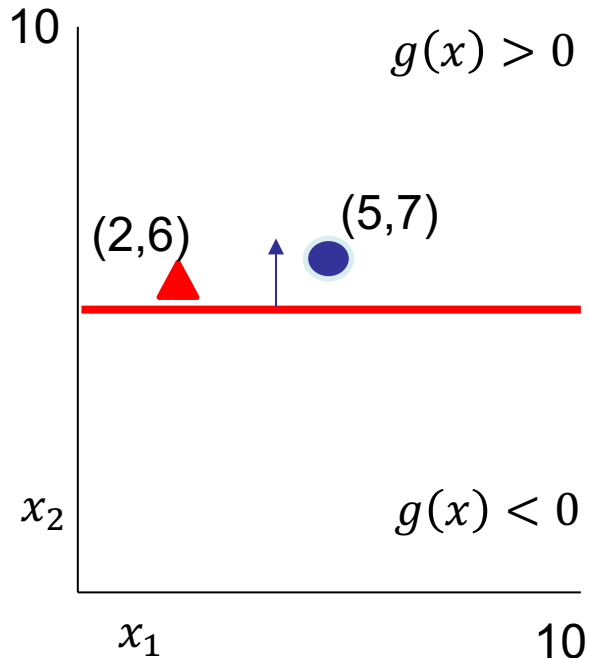$D$ is our data, consisting of training examples $< \mathbf{x}, y >$. Remember y is the true label (drawn from {1,-1} and **x** is the thing being labeled.

Our goal : make $(\mathbf{w^T x})y > 0$ for all $< \mathbf{x}, y > \in D$

Why is this the goal?

# An example



Goal: classify ● as +1 and ▲ as -1 by putting a line between them.

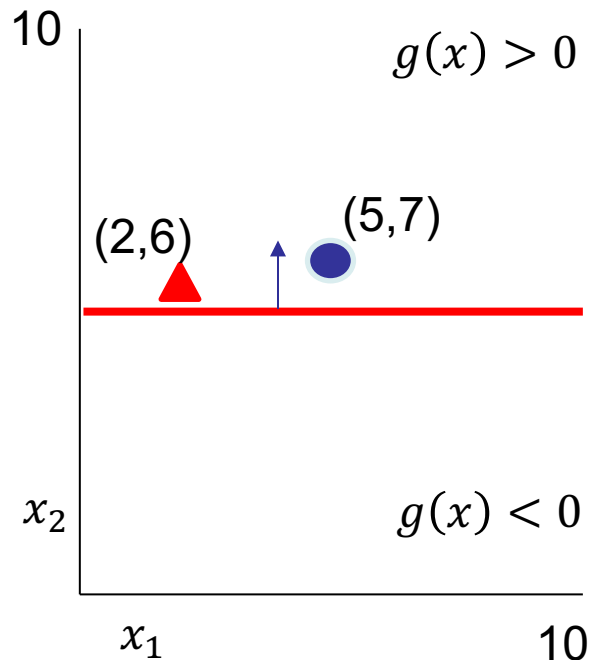Our objective function is…

$$(\mathbf{w}^T\mathbf{x})y > 0$$

Start with a randomly placed line.

$$\mathbf{w} = [w_0, w_1, w_2] = [-5, 0, 1]$$

Measure the objective for each point.

Move the line if the objective isn't met.

# An example

10

$g(x) > 0$

(2,6)   (5,7)

$x_2$   $g(x) < 0$

$x_1$   10

Objective met. Don't move the line.

Goal: classify ● as +1 and ▲ as -1 by putting a line between them.

Our objective function is…

$$(\mathbf{w}^T\mathbf{x})y > 0$$

Start with a randomly placed line.

$$\mathbf{w} = [w_0, w_1, w_2] = [-5, 0, 1]$$

● $(\mathbf{w}^T\mathbf{x})y = [-5,0,1]^T[1,5,7](1)$
$= 2$
$> 0$

# An example

10

$g(x) > 0$

(2,6)    (5,7)

$x_2$    $g(x) < 0$

$x_1$                    10

Goal:  classify ● as +1 and ▲ as -1 by putting a line between them.
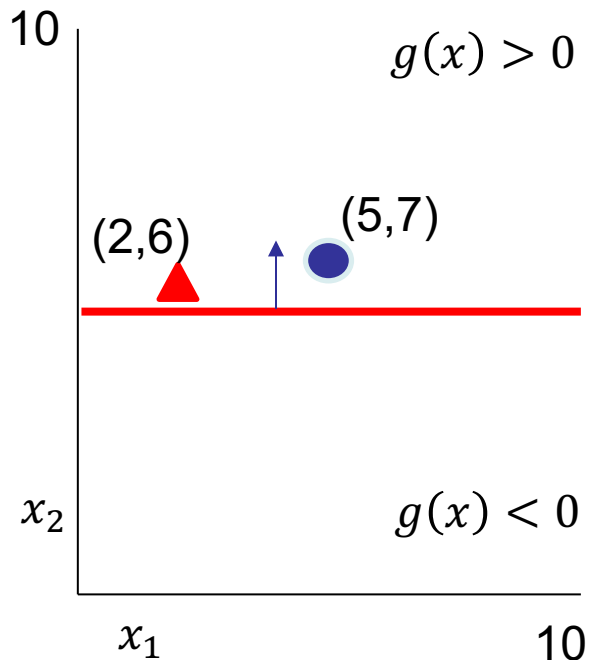
Our objective function is…

$$(\mathbf{w^T x})y > 0$$

Start with a randomly placed line.

$$\mathbf{w} = [w_0, w_1, w_2] = [-5, 0, 1]$$

$$
\begin{aligned}
▲\ (\mathbf{w^T x})y &= [-5,0,1]^T[1,2,6](-1) \\
&= (-5+6)(-1) \\
&= -1
\end{aligned}
$$

Objective not met. Move the line.        $< 0$

# An example

10

$g(x) > 0$

(2,6)    (5,7)

$x_2$    $g(x) < 0$

$x_1$                    10

Goal: classify ● as +1 and ▲ as -1 by putting a line between them.

Our objective function is…

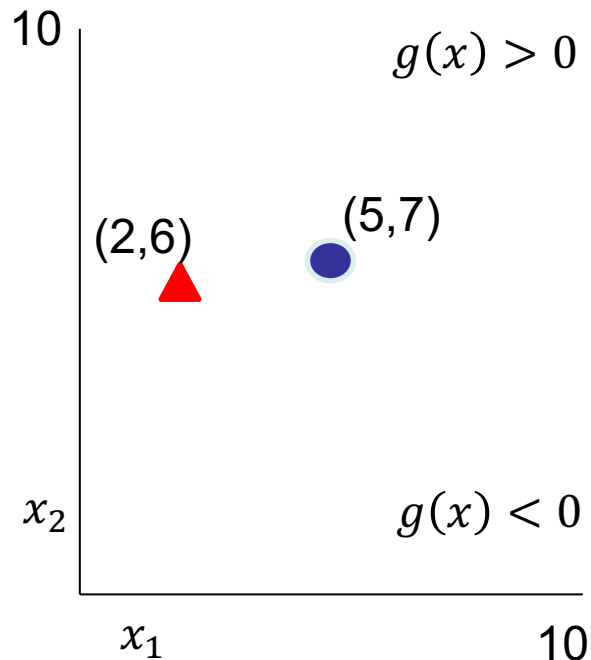$$(\mathbf{w}^T\mathbf{x})y > 0$$

Start with a randomly placed line.

$$\mathbf{w} = [w_0, w_1, w_2] = [-5, 0, 1]$$

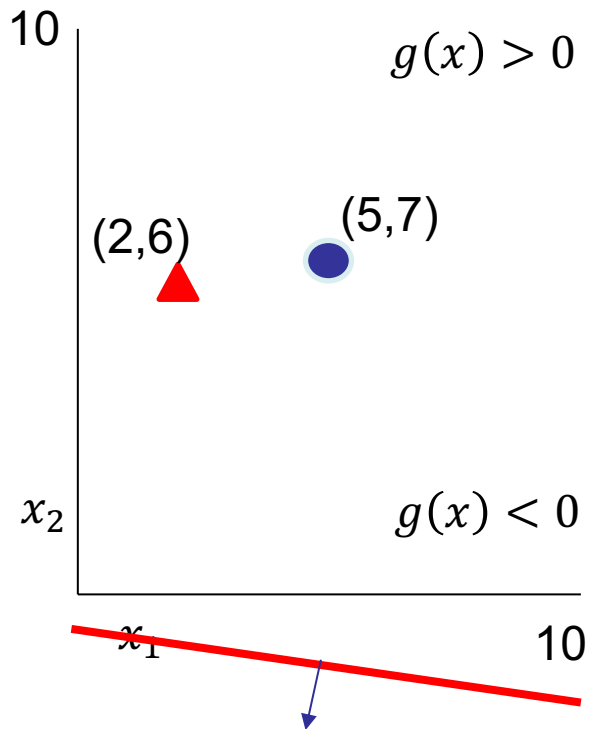Let's update the line by doing $\mathbf{w} = \mathbf{w} + \mathbf{x}(y)$.

$$\mathbf{w} = \mathbf{w} + \mathbf{x}(y) = [-5,0,1] + [1,2,6](-1)$$
$$= [-6, -2, -5]$$

# Now what ?

- What does the decision boundary look like when $\mathbf{w} = [-6, -2, -5]$ ? Does it misclassify the blue dot now?

- What if we update it the same way, each time we find a misclassified point?

- Could this approach be used to find a good separation line for a lot of data?

# An example

$g(x) > 0$

$(5,7)$

$(2,6)$

$x_2$

$g(x) < 0$

$x_1$

$10$

Goal: classify ● as +1 and ▲ as -1 by putting a line between them.

Our objective function is…

$$(\mathbf{w}^T\mathbf{x})y > 0$$

New line:

$$\mathbf{w} = [w_0, w_1, w_2] = [-6, -2, -5]$$

$$g(x) = [w_0, w_1, w_2]^T[1, x_1, x_2] = 0$$

$$0 = -6 - 2x_1 - 5x_2$$

$$-1.2 - .4x_1 = x_2$$

# Perceptron Algorithm

**The decision boundary**

$$0 = g(x) = \mathbf{w^T x}$$

$$m = |D| = \textit{size of data set}$$

**The classification function**

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } g(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

**The weight update algorithm**

$$\mathbf{w} = \textit{some random setting}$$

Do

$\quad k = (k + 1)\text{mod}(m)$

$\quad \text{if } h(\mathbf{x}_k) != y_k$

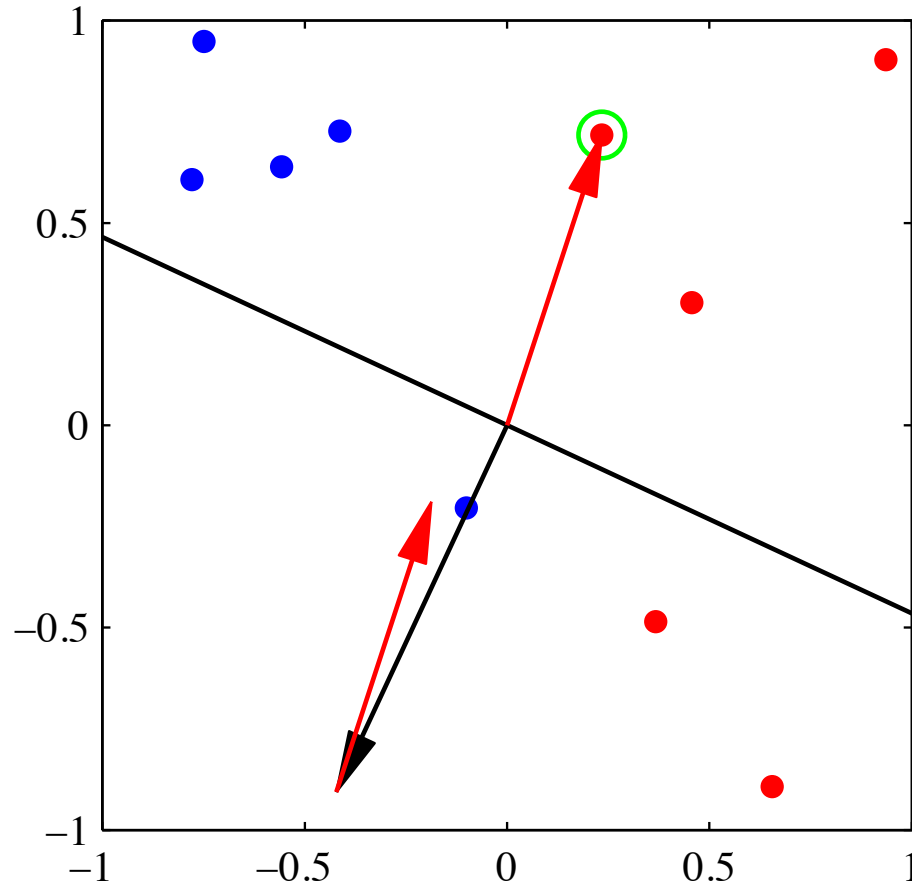$\quad\quad \mathbf{w} = \mathbf{w} + \mathbf{x}_k y$

Until $\forall k, \; h(\mathbf{x}_k) = y_k$

**Warning: Only guaranteed to terminate if classes are linearly separable!**

**This means you have to add another exit condition for when you've gone through the data too many times and suspect you'll never terminate.**
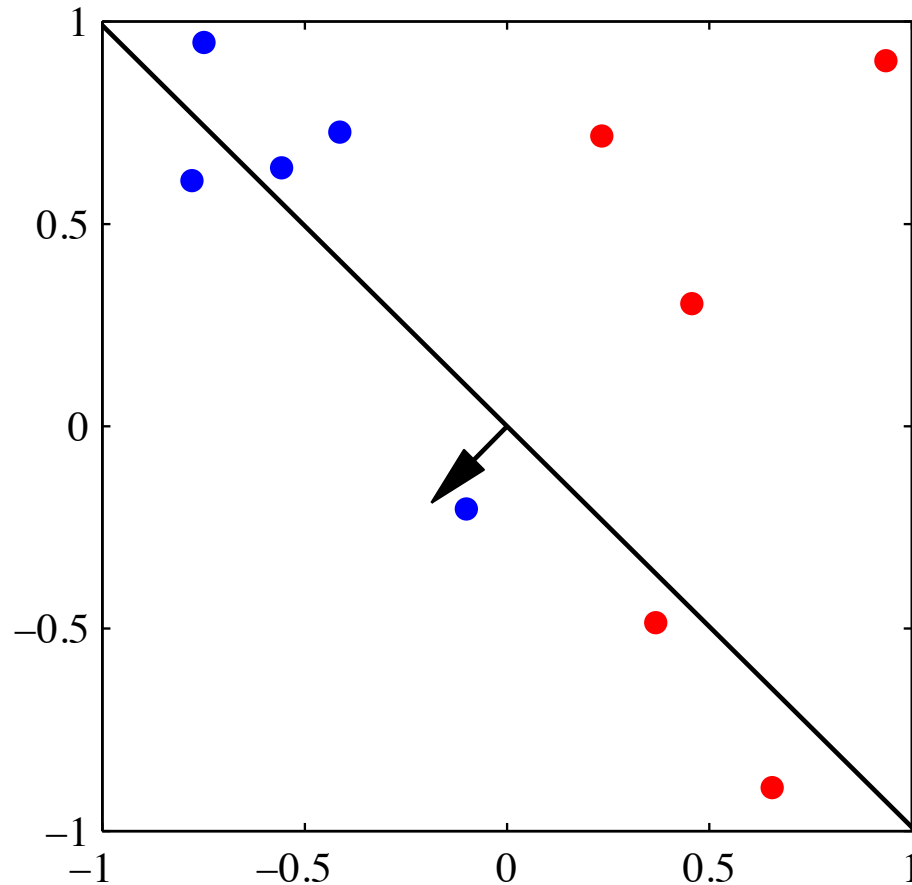
# Perceptron Algorithm

- Example:



Red is the positive class

Blue is the negative class

# Perceptron Algorithm

- Example (cont'd):

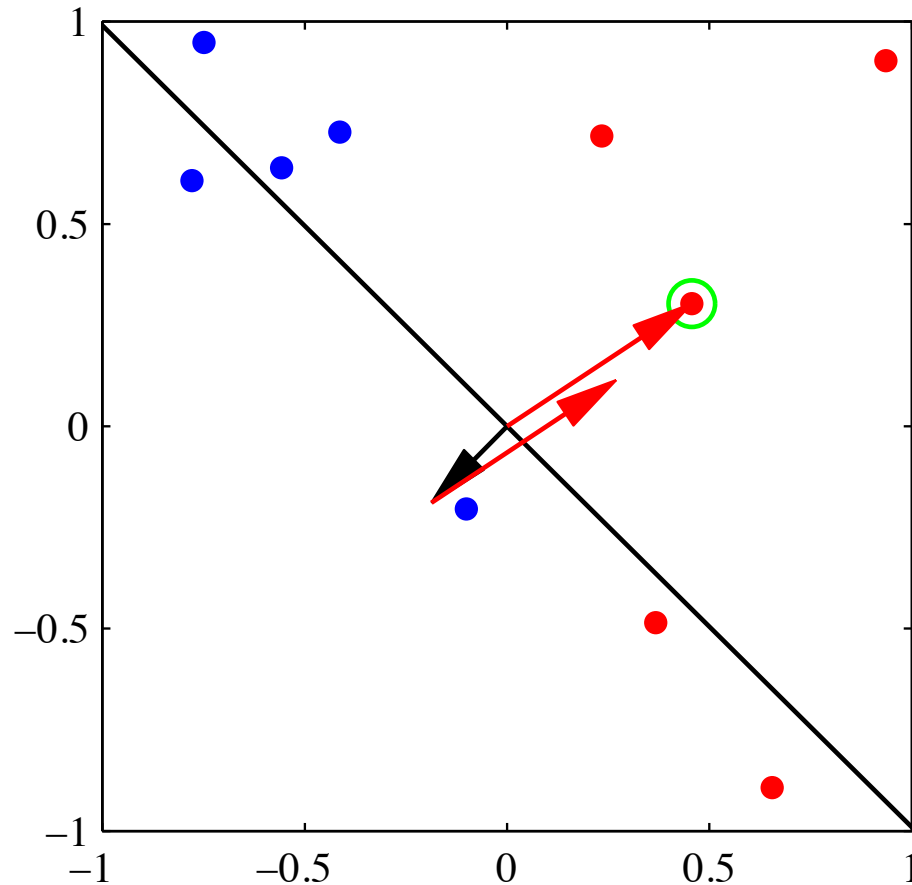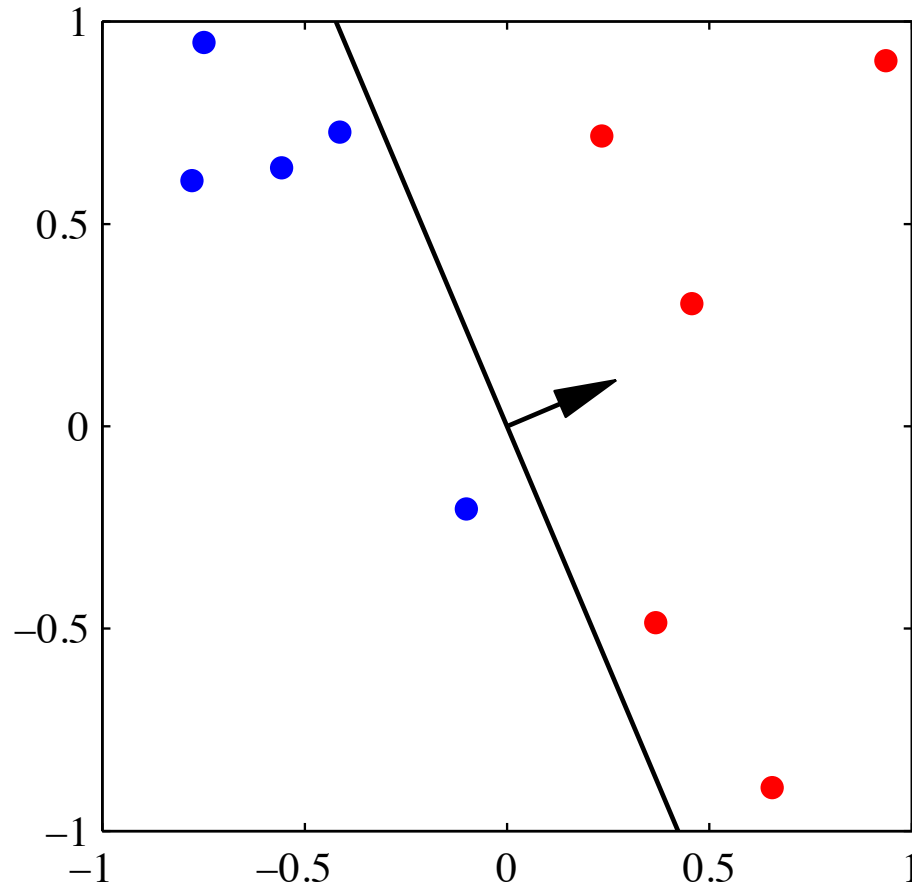

Red is the positive class

Blue is the negative class

# Perceptron Algorithm

- Example (cont'd):



Red is the positive class
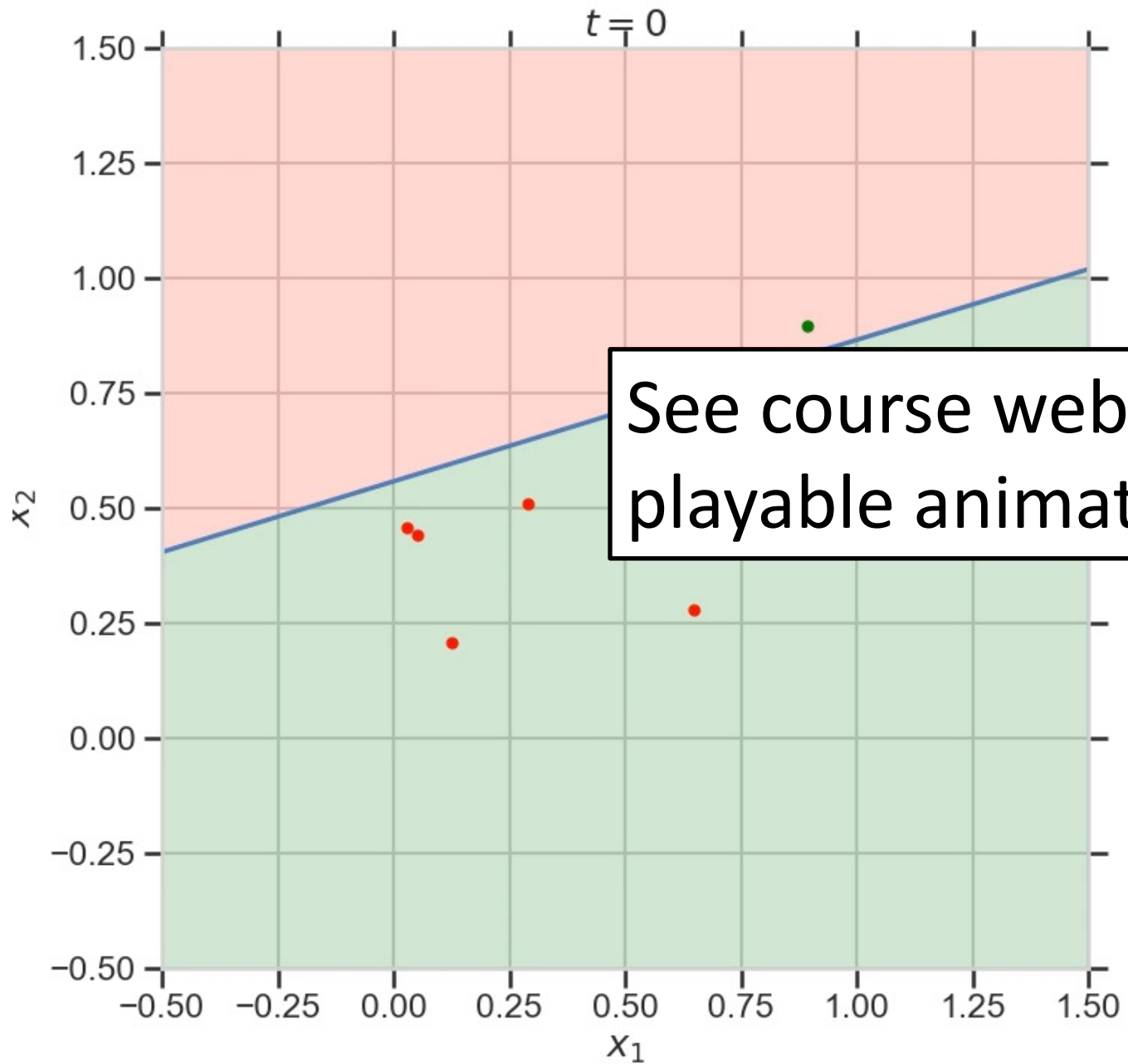
Blue is the negative class

# Perceptron Algorithm

- Example (cont'd):



Red is the positive class

Blue is the negative class

See course website for playable animations

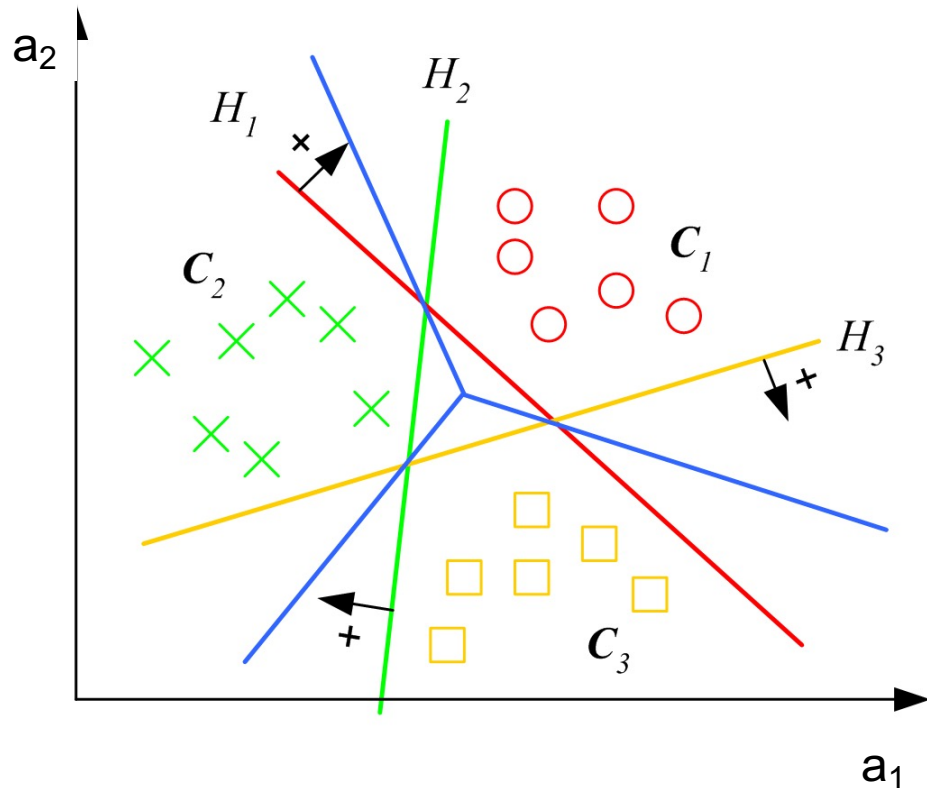See course website for playable animations
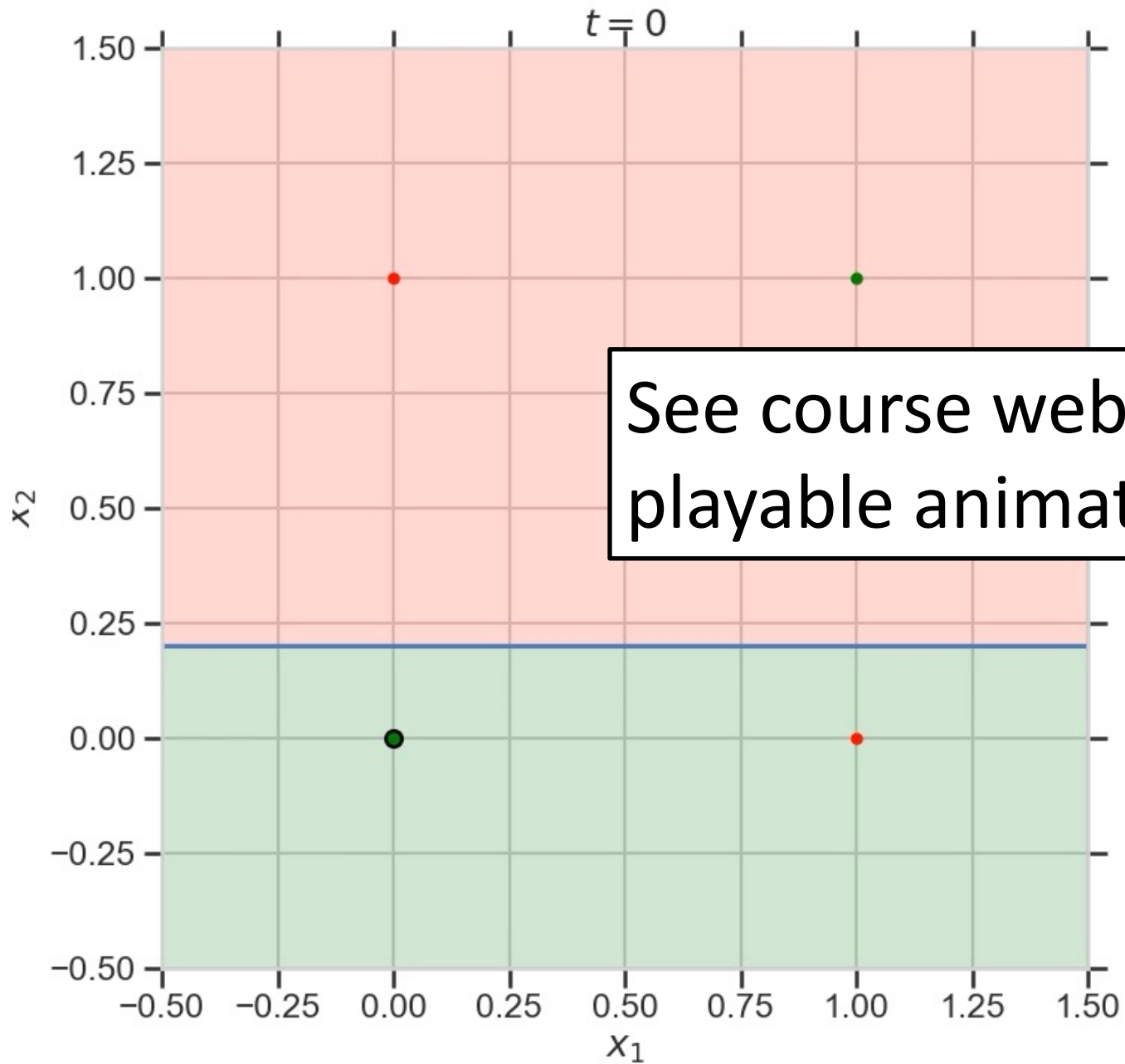
# Multi-class Classification



When there are N classes you can classify using N discriminant functions.

Choose the class c from the set of all classes C whose function $g_c(\mathbf{x})$ has the maximum output

Geometrically divides feature space into N **convex** decision regions

$$h(\mathbf{x}) = \underset{c \in C}{\mathrm{argmax}}\; g_c(\mathbf{x})$$

# Multi-class Classification

A class label

$$c = h(\mathbf{x}) = \underset{c \in C}{\operatorname{argmax}}\, g_c(\mathbf{x})$$

Set of all classes

Remember $g_c(\mathbf{x})$ is the inner product of the feature vector for the example $(\mathbf{x})$ with the weights of the decision boundary hyperplane for class c. If $g_c(\mathbf{x})$ is getting more positive, that means $(\mathbf{x})$ is deeper inside its "yes" region.

Therefore, if you train a bunch of 2-way classifiers (one for each class) and pick the output of the classifier that says the example is deepest in its region, you have a multi-class classifier.

# Pairwise Multi-class Classification

If they are not linearly separable (singly connected convex regions), may still be pair-wise separable, using N(N-1)/2 linear discriminants.
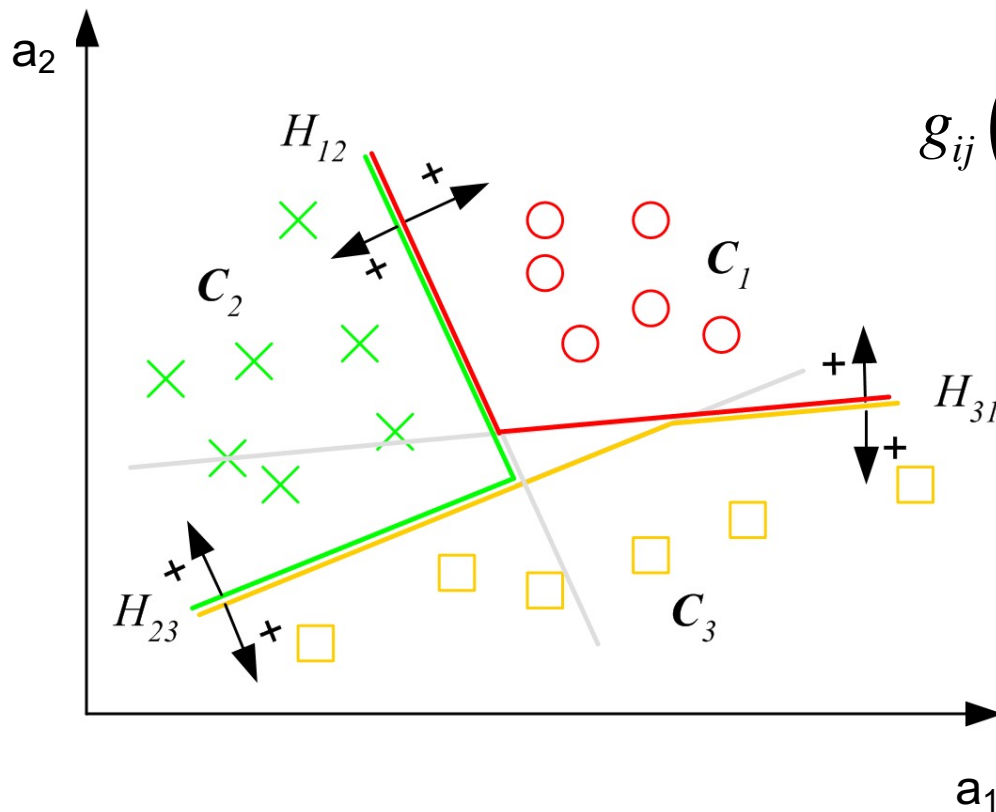


$$g_{ij}\left(\vec{x}\mid\vec{w}_{ij},w_{ij0}\right)=w_{ij0}+\sum_{l=1}^{K}w_{ijl}x_l$$

choose $C_i$ if

$$\forall j\neq i, g_{ij}(\mathbf{x})>0$$

# Appendix

(stuff I didn't have time to discuss in class…and for which I haven't updated the notation. )

# Linear Discriminants

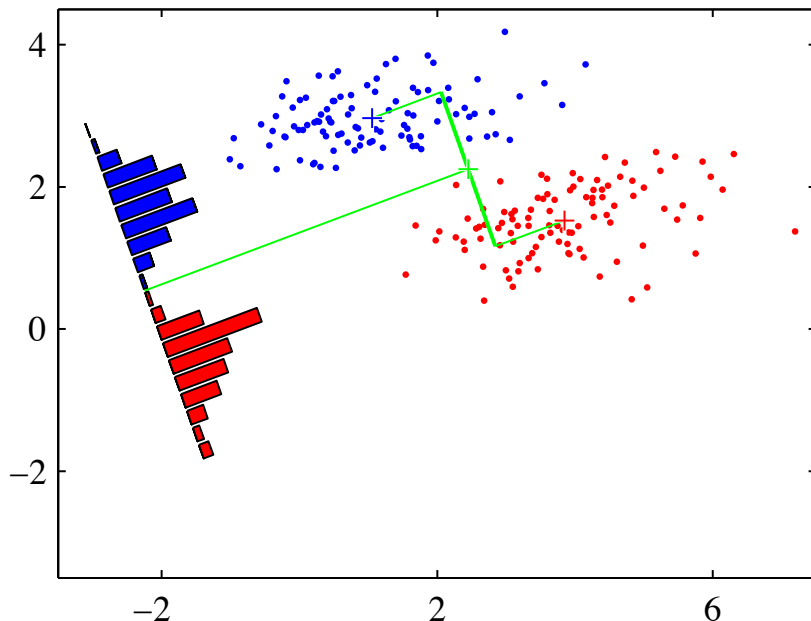- A linear combination of the attributes.

$$g\left(\vec{x} \mid \vec{w}, w_0\right) = w_0 + \vec{w}^T \vec{x} = w_0 + \sum_{i=1}^{k} w_i a_i$$

- Easily interpretable

- Are optimal when classes are Gaussian and share a covariance matrix

# Fisher Linear Discriminant Criteria

- Can think of $\vec{w}^T \vec{x}$ as dimensionality reduction from K-dimensions to 1
- Objective:
  - Maximize the difference between class means
  - Minimize the variance within the classes



$$J(\vec{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$$

where $s_i$ and $m_i$ are the sample variance and mean for class i in the projected dimension. We want to maximize $J$.

# Fisher Linear Discriminant Criteria
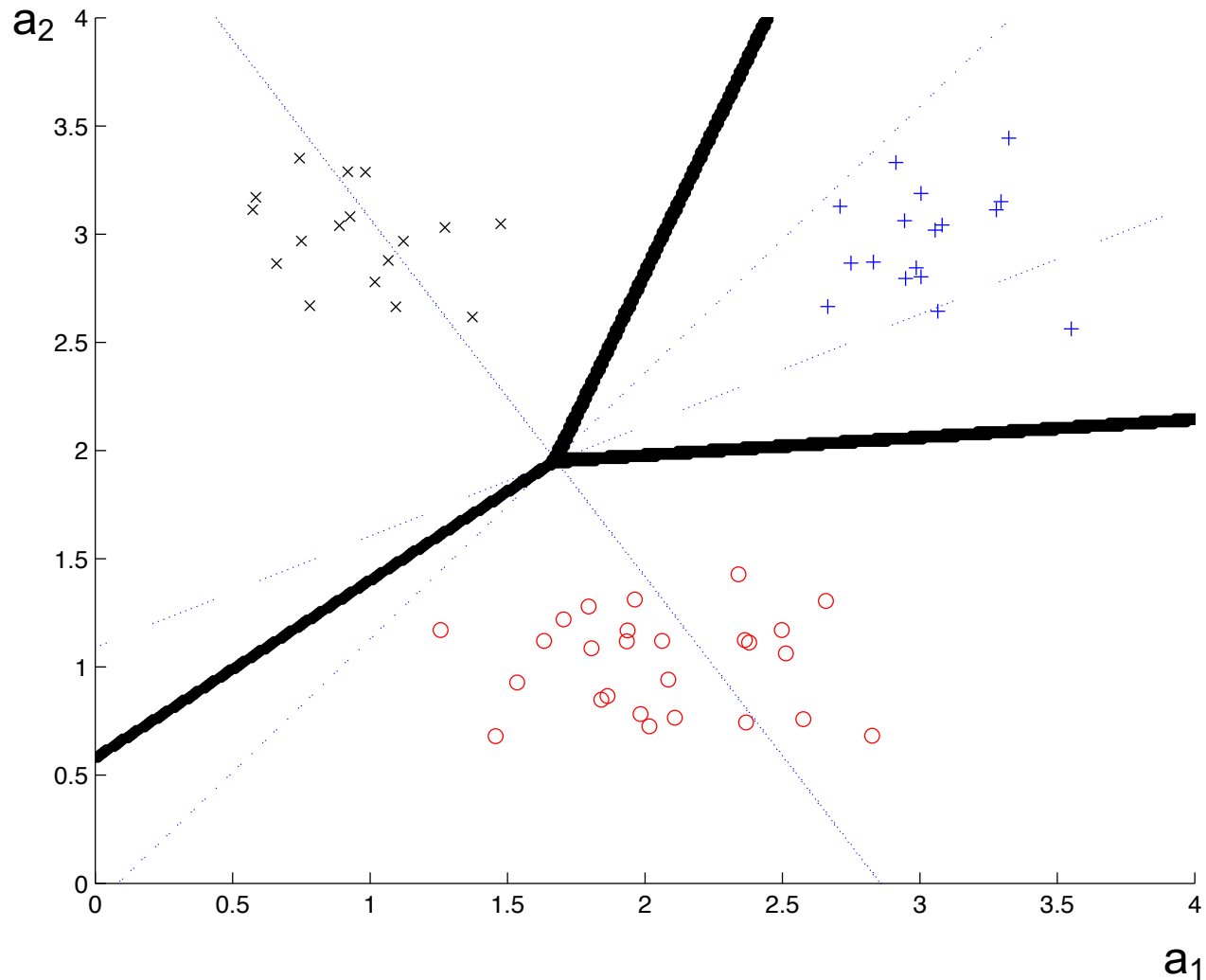
- Solution:

$$\vec{w} = \mathbf{S}_W^{-1}(\vec{m}_2 - \vec{m}_1)$$

where

$$\mathbf{S}_W = \sum_{n \in C_1} (\vec{x}_n - \vec{m}_1)(\vec{x}_n - \vec{m}_1)^T + \sum_{n \in C_2} (\vec{x}_n - \vec{m}_2)(\vec{x}_n - \vec{m}_2)^T$$

- However, while this finds finds the direction ($\vec{w}$) of decision boundary. Must still solve for $w_0$ to find the threshold.

- Can be expanded to multiple classes

# Logistic Regression (Discrimination)

# Logistic Regression (Discrimination)

- Discriminant model but well-grounded in probability

- Flexible assumptions (exponential family class-conditional densities)

- Differentiable error function ("cross entropy")

- Works very well when classes are linearly separable

# Logistic Regression (Discrimination)

- Probabilistic discriminative model
- Models posterior probability $p(C_1|\vec{x})$
- To see this, let's start with the 2-class formulation:

$$p(C_1|x) = \frac{p(\vec{x}|C_1)p(C_1)}{p(\vec{x}|C_1)p(C_1) + p(\vec{x}|C_2)p(C_2)}$$

$$= \frac{1}{1 + \exp\left(-\log\frac{p(\vec{x}|C_1)p(C_1)}{p(\vec{x}|C_2)p(C_2)}\right)}$$

$$= \frac{1}{1 + \exp(-\alpha)} \qquad \text{logistic sigmoid function}$$
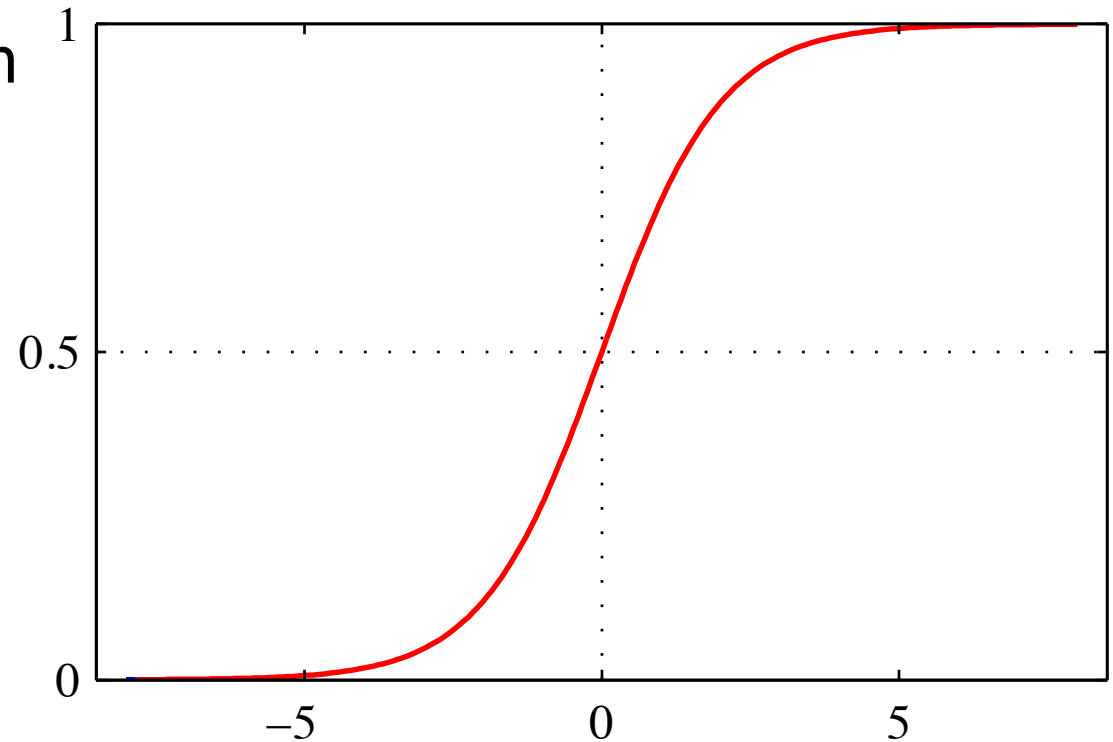
$$= \sigma(\alpha)$$

where

$$\alpha = \log\frac{p(\vec{x}|C_1)p(C_1)}{p(\vec{x}|C_2)p(C_2)}$$

# **Logistic Regression (Discrimination)**

logistic sigmoid function

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



"Squashing function" that maps $(-\infty, +\infty) \rightarrow (0, 1)$

# Logistic Regression (Discrimination)

For exponential family of densities,

$$\alpha \quad = \quad \log \frac{p(\overrightarrow{x}|C_1)p(C_1)}{p(\overrightarrow{x}|C_2)p(C_2)}$$

is a linear function of x.

Therefore we can model the posterior probability as a logistic sigmoid acting on a linear function of the attribute vector, and simply solve for the weight vector **w** (e.g. treat it as a discriminant model):

$$y = p(C_1|\overrightarrow{x}) = \sigma(w_0 + \sum_{i=1}^{k} w_i a_i) \qquad p(C_2|\overrightarrow{x}) = 1 - p(C_1|\overrightarrow{x})$$

To classify: $h(\overrightarrow{x}_i) = \begin{cases} C_1 & y_i > 0.5 \\ C_2 & o.w. \end{cases}$