The Perceptron

Rosenblatt, Frank. "The perceptron: A model for information storage and organization in the brain." Psychological review 65.6 (1958): 386.

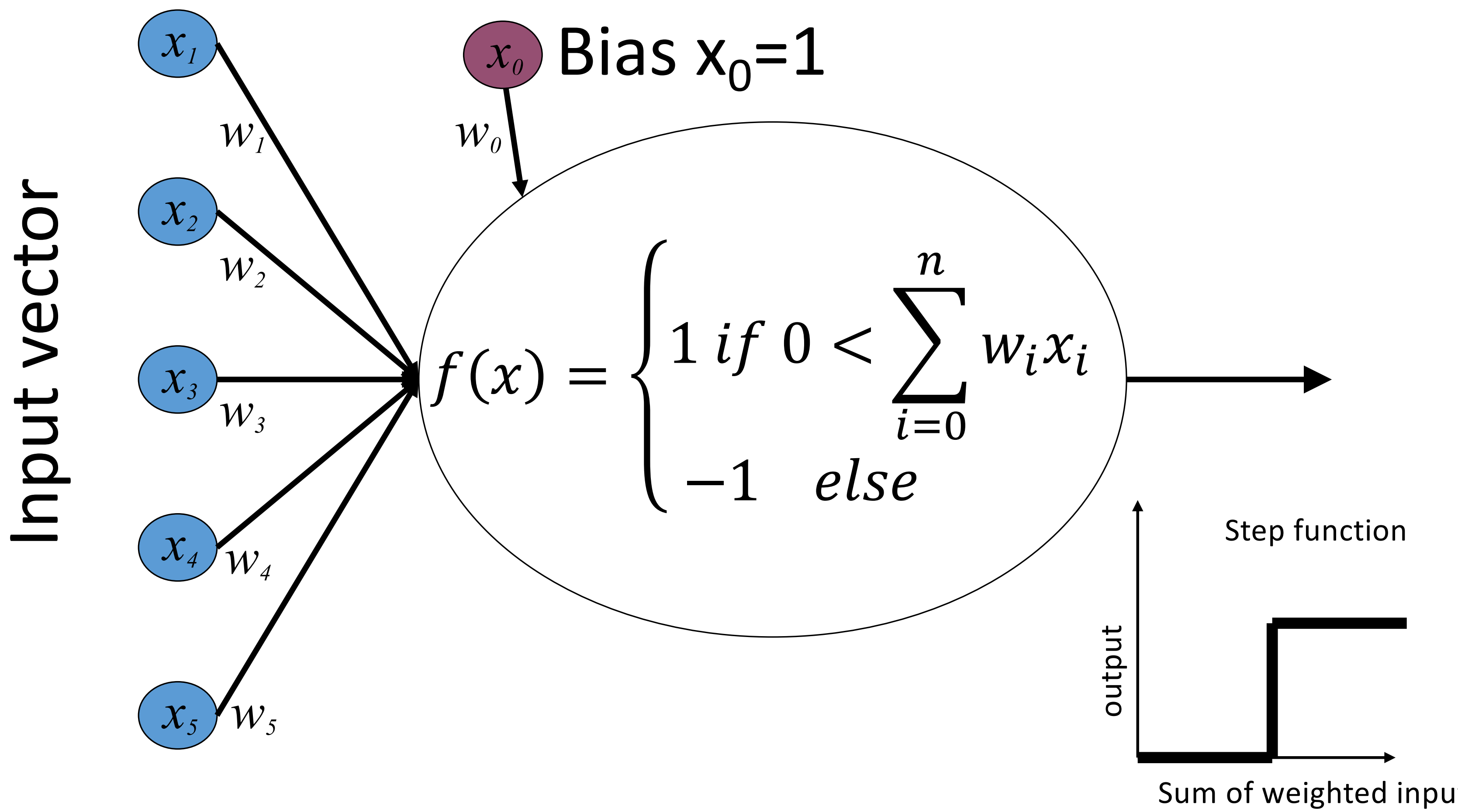
The Perceptron

- Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 386-408

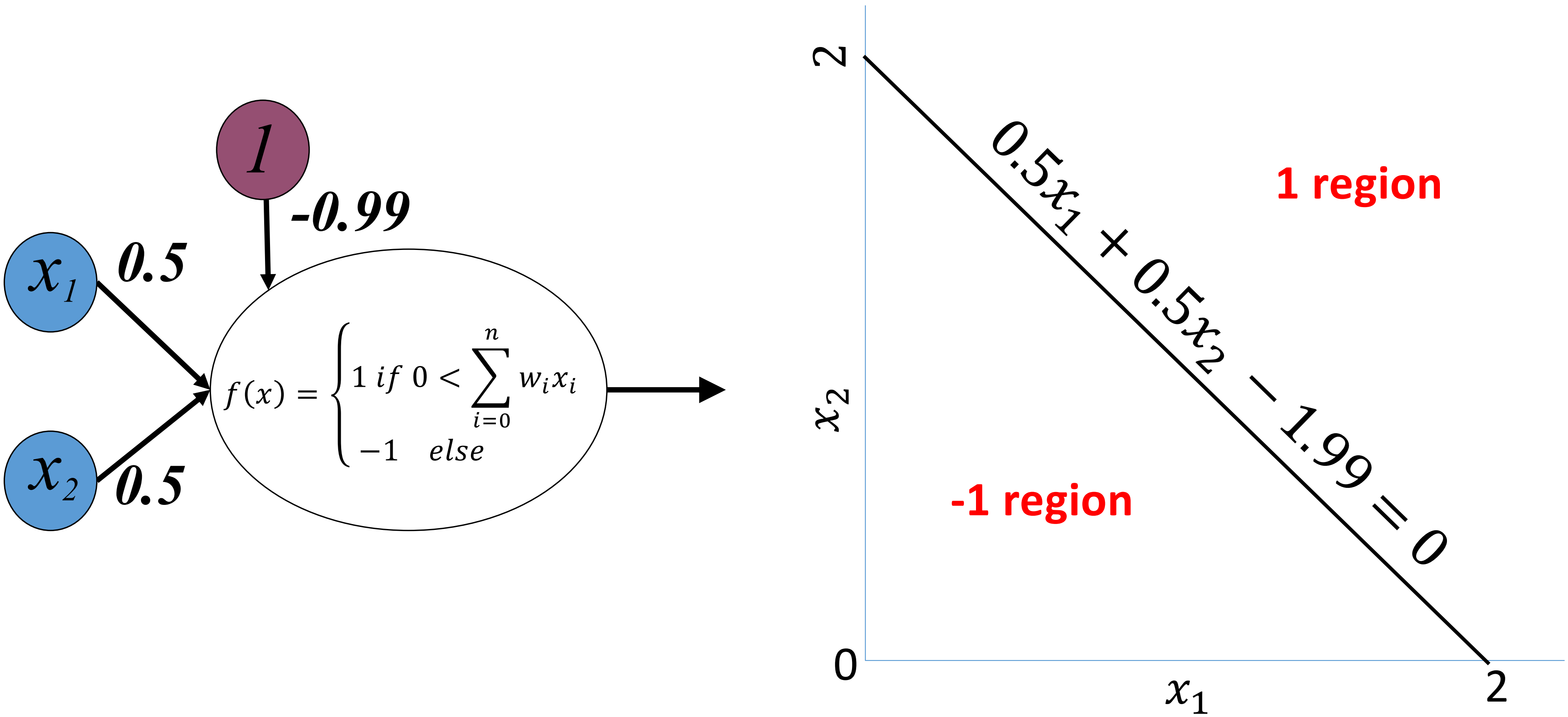
- The “first wave” in neural networks

- A linear classifier

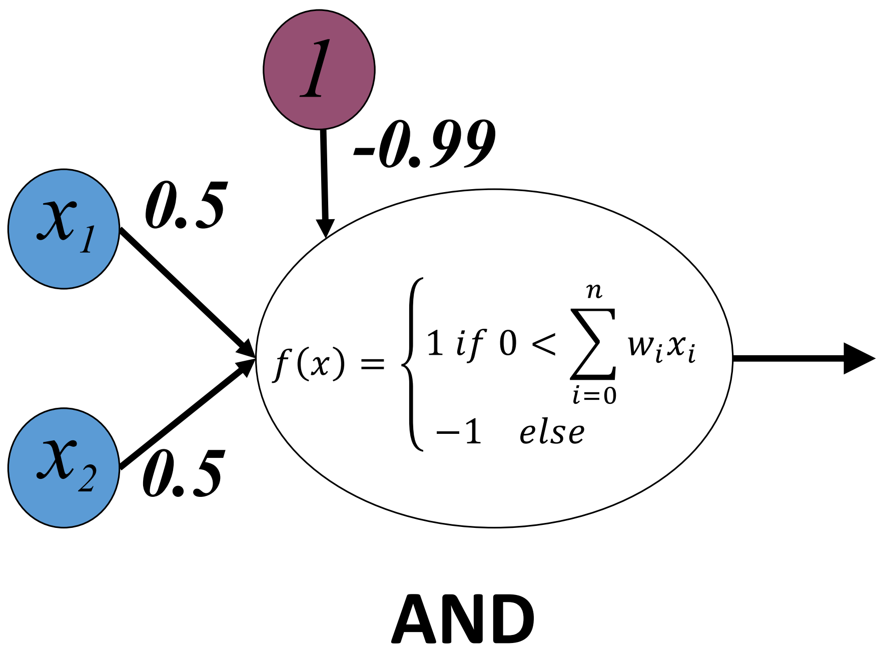
A single perceptron

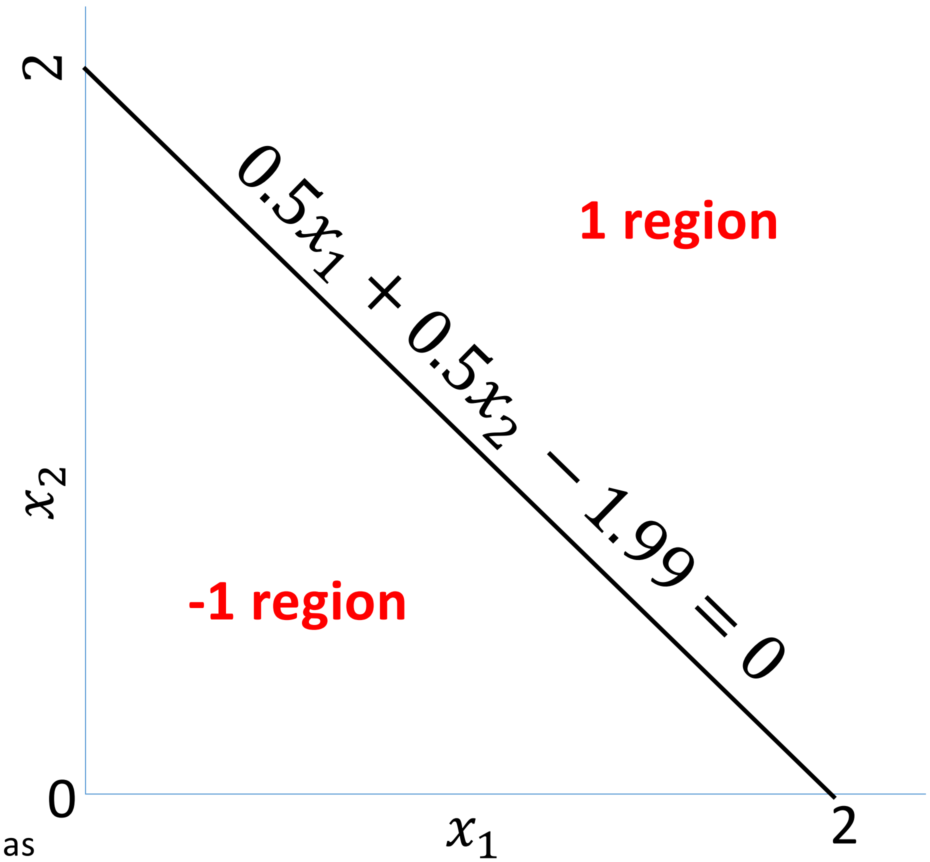


Weights define a hyperplane in the input space



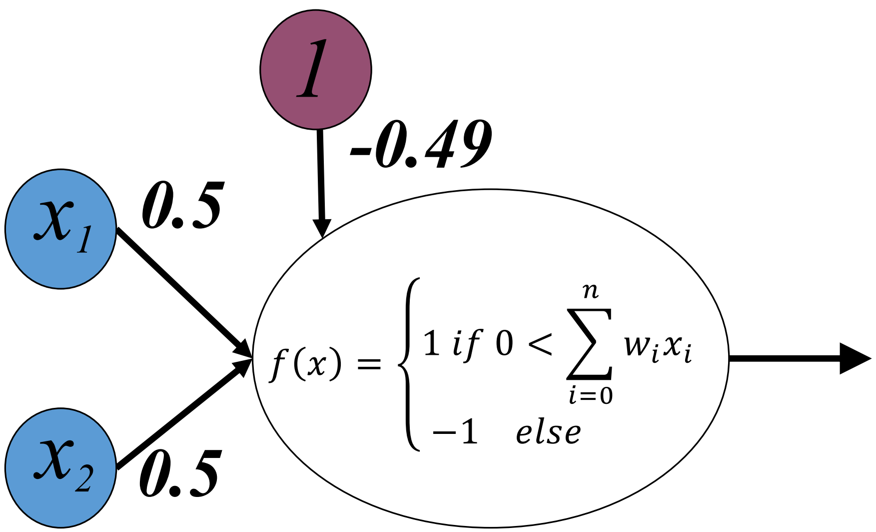
Different logical functions are possible



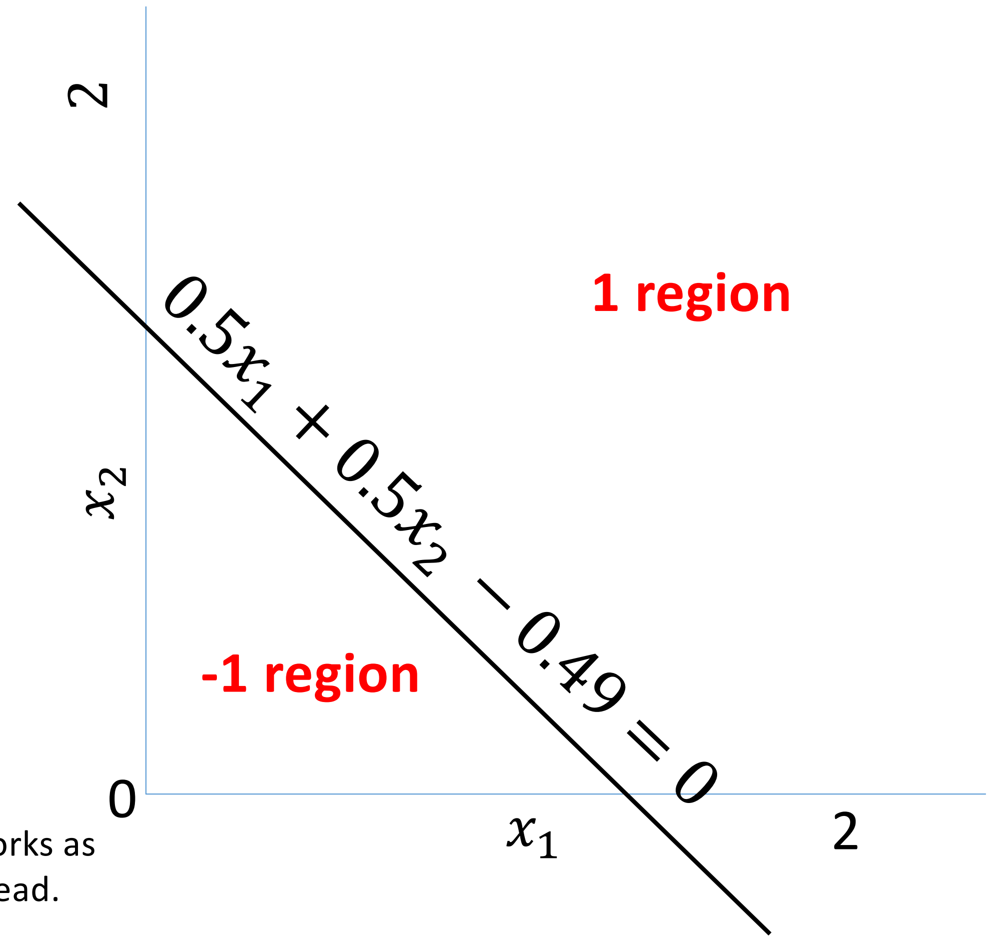


If we say that the input data is all binary (0 or 1) then this line works as an AND….if, when the function outputs -1, we map that to 0 instead.

Different logical functions are possible

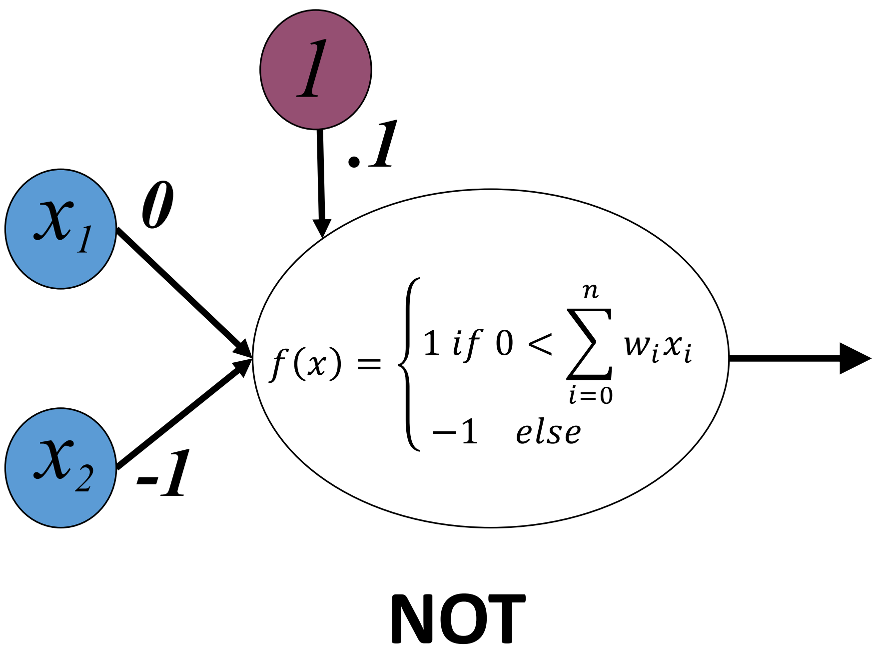


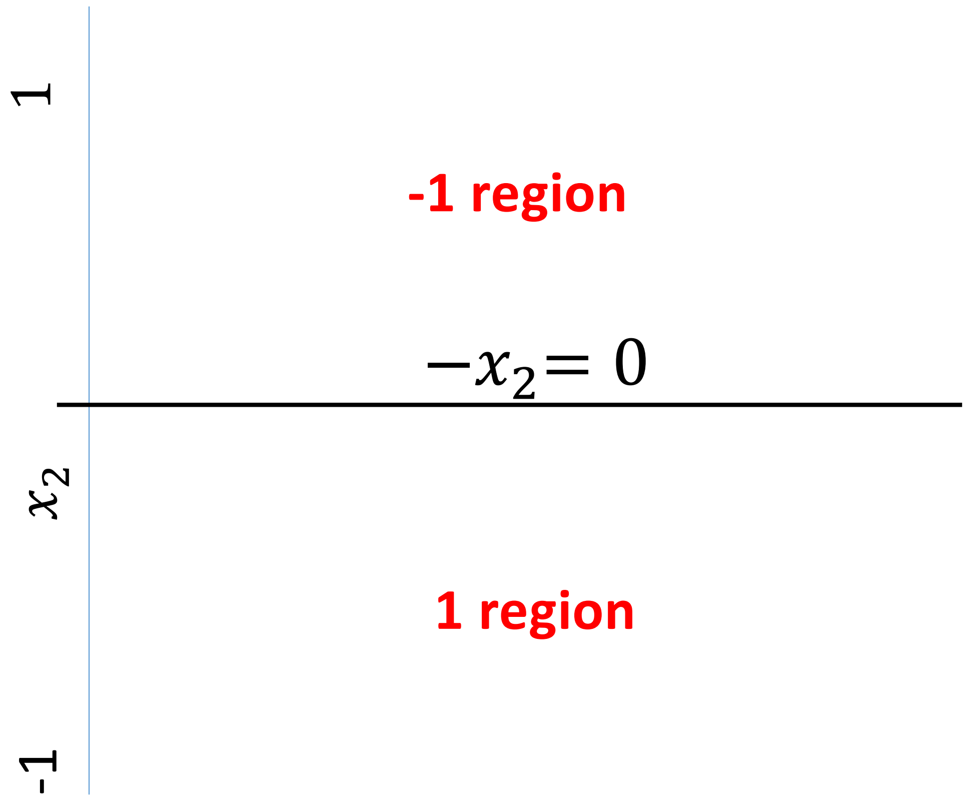
OR



If we say that the input data is all binary (0 or 1) then this line works as an AND….if, when the function outputs -1, we map that to 0 instead.

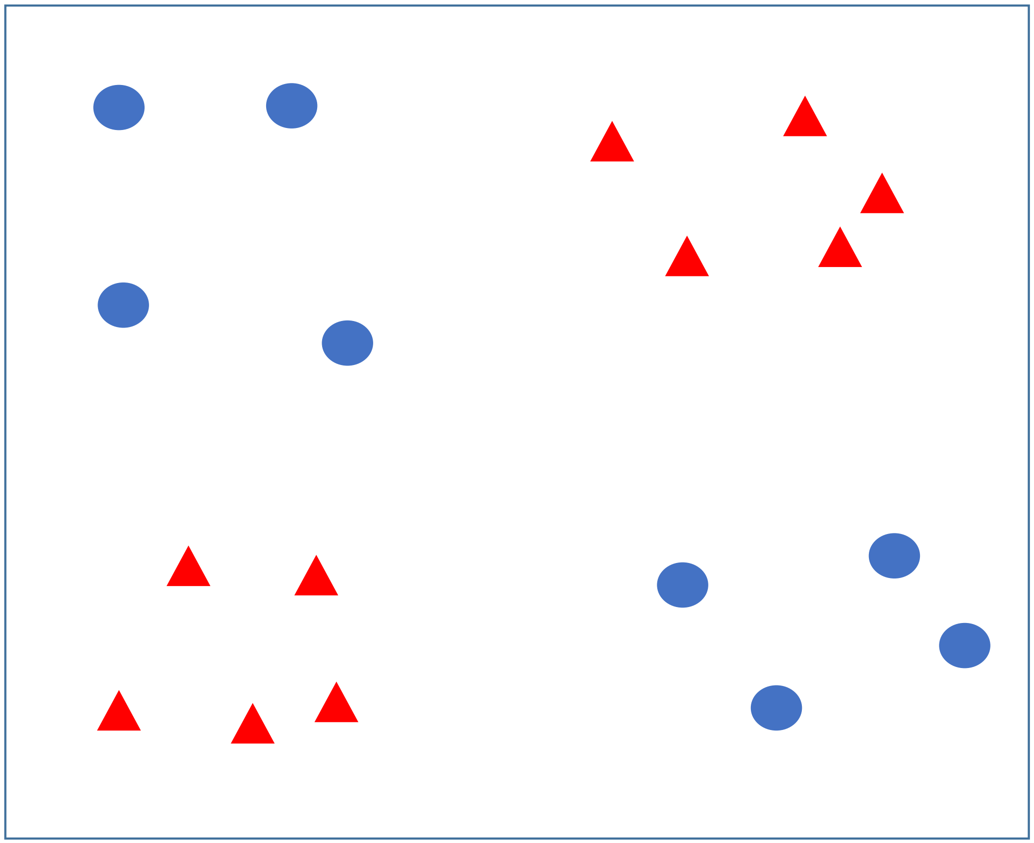
And, Or, Not are easy to define





If we say that the input data is all binary (0 or 1) then this line works as an AND….if, when the function outputs -1, we map that to 0 instead.

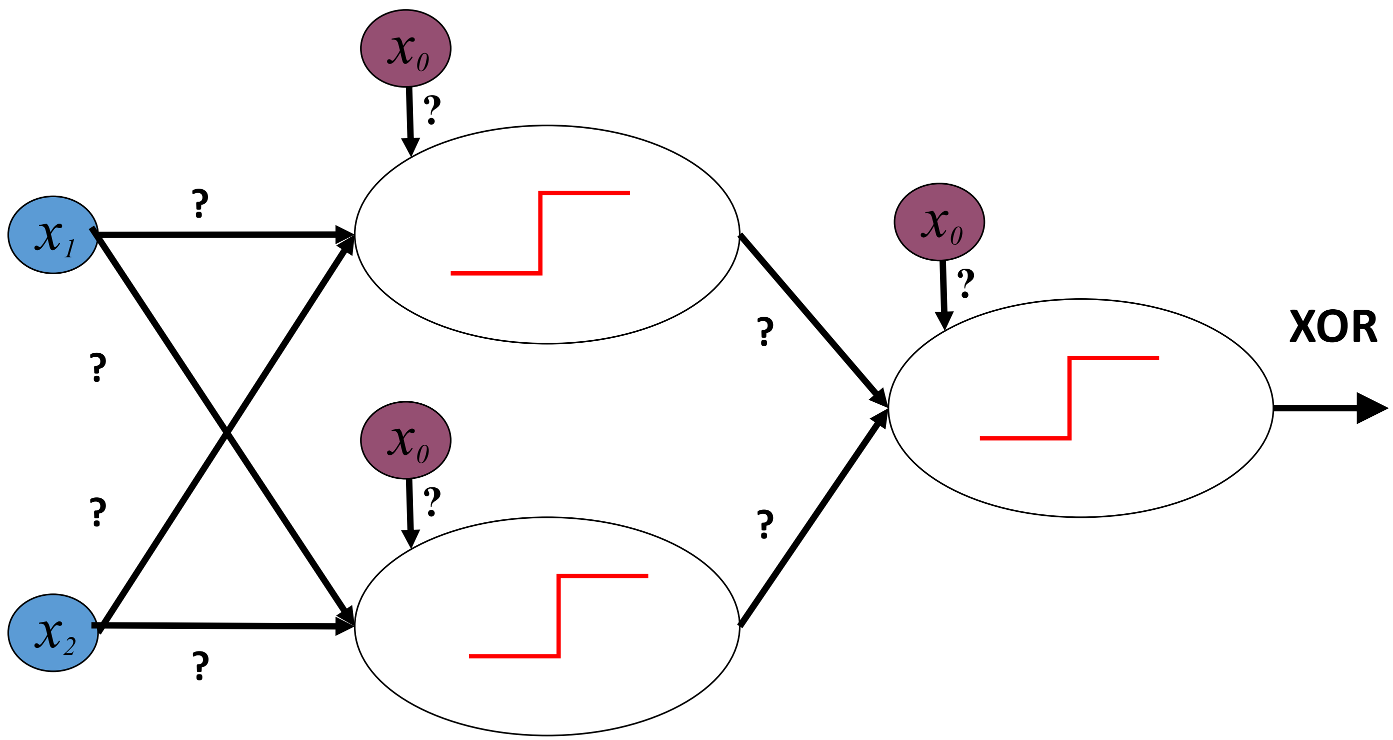
One perceptron: Only linear decisions



This is XOR.

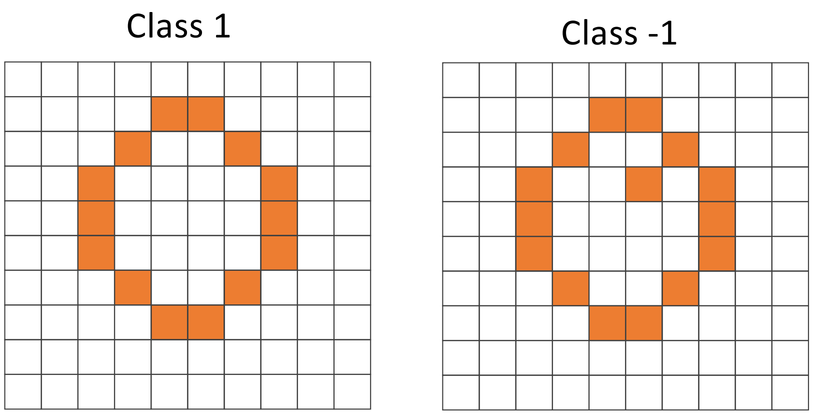
It can't learn XOR.

Combining perceptrons can make any Boolean function
...if you can set the weights & connections right



How would you set the weights and connections to make XOR?

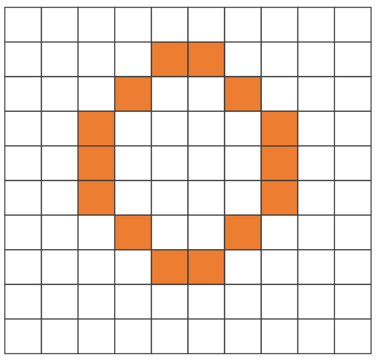
Classifying image data with a single perceptron



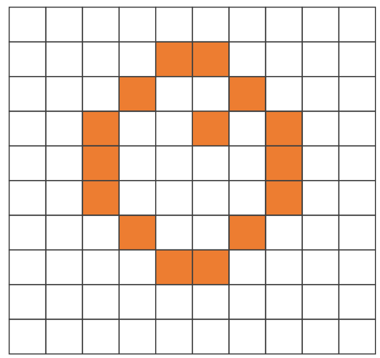
- Each image is an *m* by *n* array of pixel values.
- Assume each pixel is set to 0 or 1.
- Define a set of weights that would separate class 1 from class -1.

Classifying image data with a single perceptron

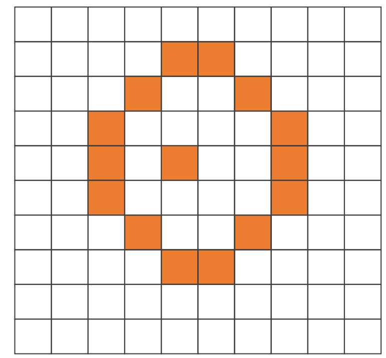
Class 1



Class -1

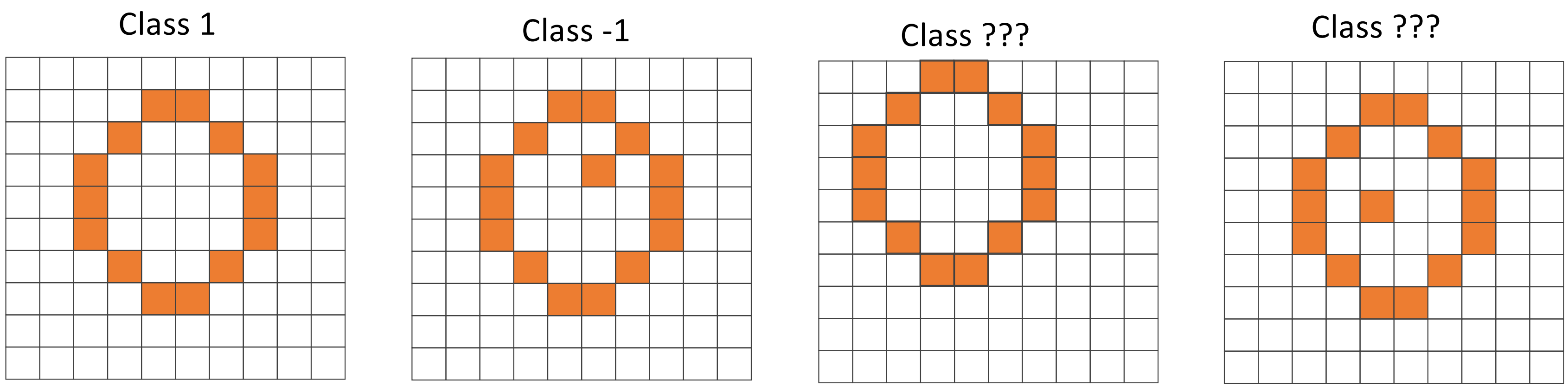


Class ???



- Now…given your weights, which class would the new image be?
- Can you define a new set of weights that groups this new image in with the opposite class?

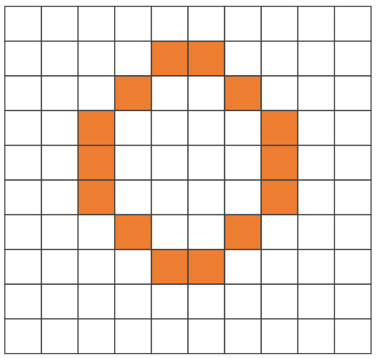
Classifying image data with a single perceptron



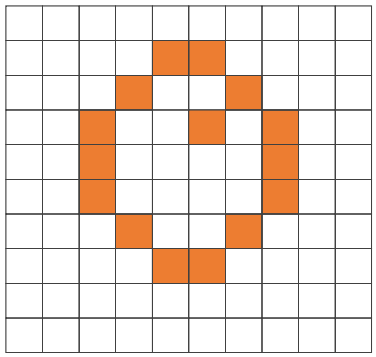
- What is a solution that generalizes so that all the circles with no dot in the middle go in the same class?
- Would your solution generalize to a situation with a bigger or smaller circle with no dot?

Classifying image data with a single perceptron

Class 1



Class -1



- Now…given your weights, which class would the new image be?
- Can you define a new set of weights that groups this new image in with the opposite class?

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

Perceptrons are linear models.

- $\mathbf{x}$ is a vector of attributes $<x_1, x_2, \ldots x_k>$
- $\mathbf{w}$ is a vector of weights $<w_1, w_2, \ldots w_k>$ **THIS IS WHAT IS LEARNED**
- Given this…

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

- We can notate it with linear algebra as

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

It is more convenient if…

- $g(x) = w_0 + \mathbf{w}^{\mathbf{T}}\mathbf{x}$ is ALMOST what we want…
- Define $\mathbf{w}$ to include w_0 and $\mathbf{x}$ to include an x_0 that is always 1

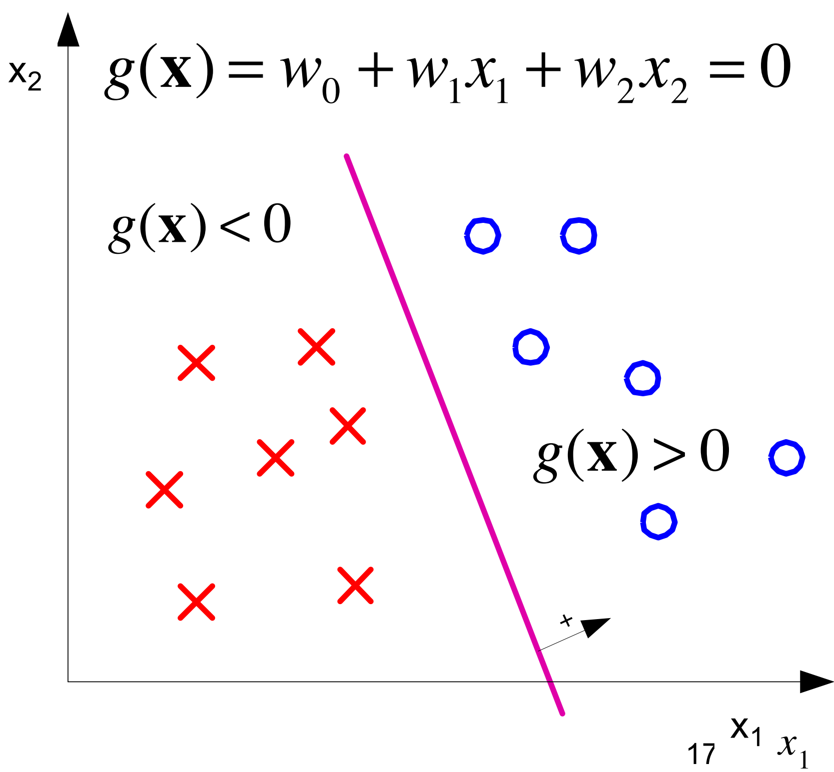
  $\mathbf{x}$ is a vector of attributes $<1, x_1, x_2, \ldots x_k>$

  $\mathbf{w}$ is a vector of weights $<w_0, w_1, w_2, \ldots w_k>$
- This lets us notate things as…

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

Two-Class Classification

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



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

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

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.

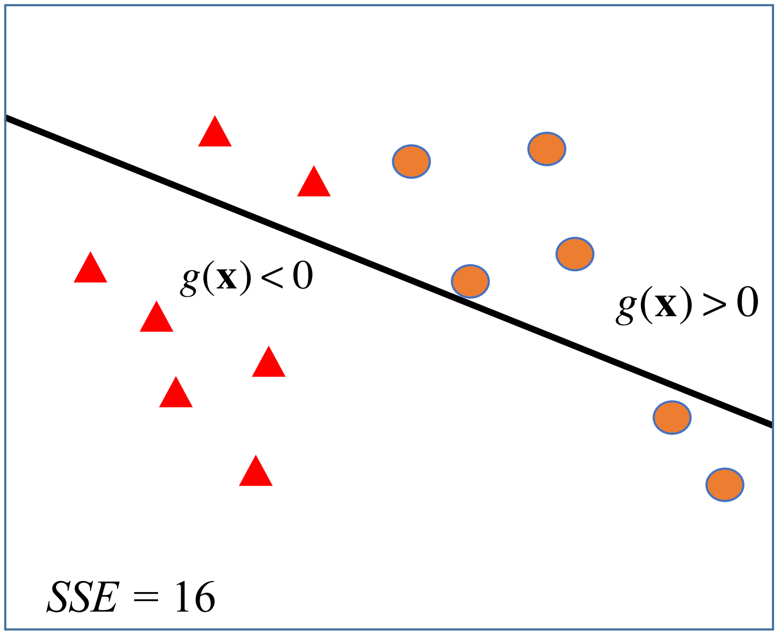
Loss/Objective function

- Let's define an objective (aka "loss") function that directly measures the thing we want to get right

- Then let's try and find the line that minimizes the loss.

- How about basing our loss function on the number of misclassifications?

sum of squared errors (SSE)

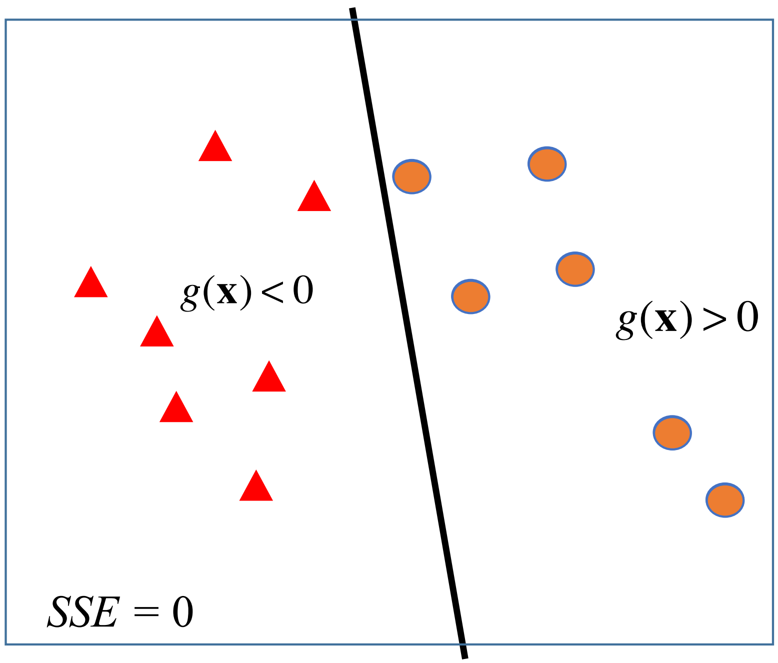


$$g(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 = 0$$
$$= \mathbf{w}^T \mathbf{x}$$

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

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

sum of squared errors (SSE)



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

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

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

No closed form solution!

- For many objective (aka loss) functions we can’t find a formula to to get the best model parameters, like one can with regression.
- The objective function from the previous slide is one of those ”no closed form solution” functions.
- This means we must try various guesses for what the weights should be.
- 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

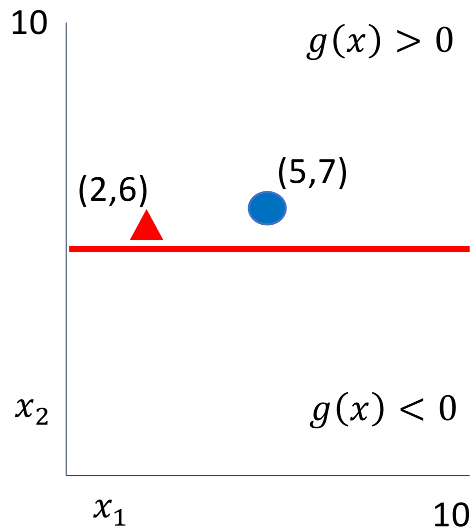
  $$0 = g(x) = \mathbf{w}^{\mathbf{T}}\mathbf{x}$$

- 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.

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.

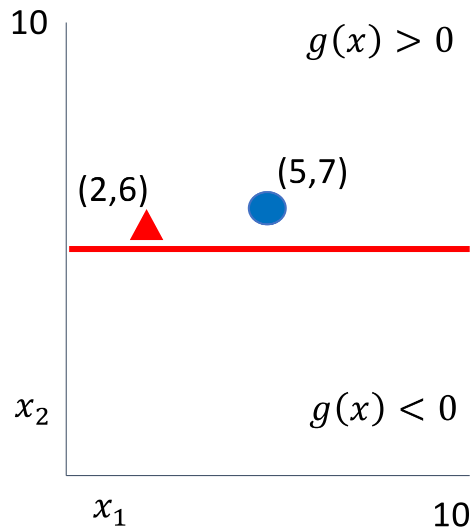
$$\mathbf{w} = [w_0, w_1, w_2] = [-5, 0, 1]$$

Measure the objective for each point.

Move the line if the objective isn't met.

An example

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



Our objective function is…

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

Start with a randomly placed line.

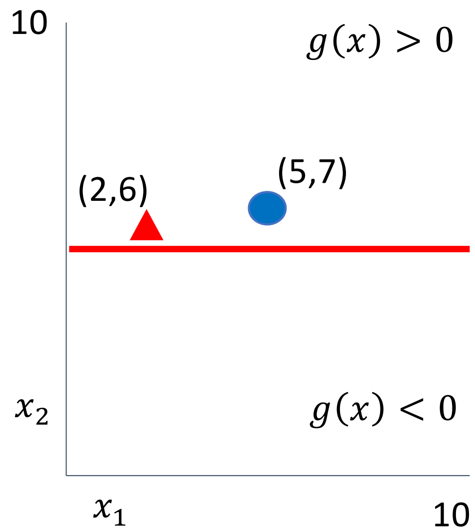
$$\mathbf{w} = [w_0, w_1, w_2] = [-5, 0, 1]$$

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

Objective met. Don't move the line.

An example

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



Our objective function is…

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

Start with a randomly placed line.

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

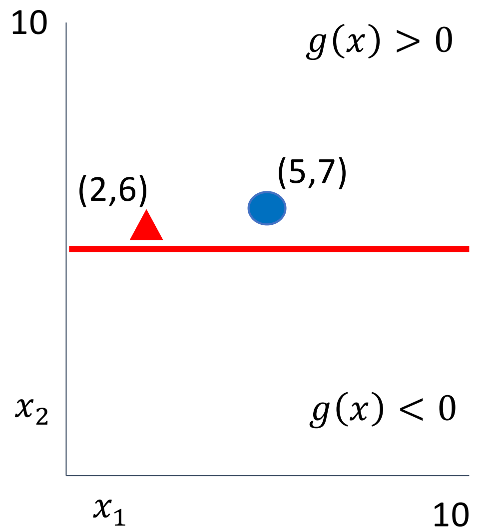
▲ $(\mathbf{w}^{\boldsymbol{T}}\mathbf{x})y = [-5{,}0{,}1]^T[1{,}2{,}6](-1)$

$= (-5 + 6)(-1)$

$= -1$

Objective not met. Move the line. < 0

An example



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

Our objective function is…

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

Start with a randomly placed line.

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

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

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

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 find a good separation line for a lot of data?

Perceptron Algorithm

The decision boundary

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

$$m = |D| = size\ of\ data\ set$$

The classification function

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

The weight update algorithm

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

Do

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

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

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

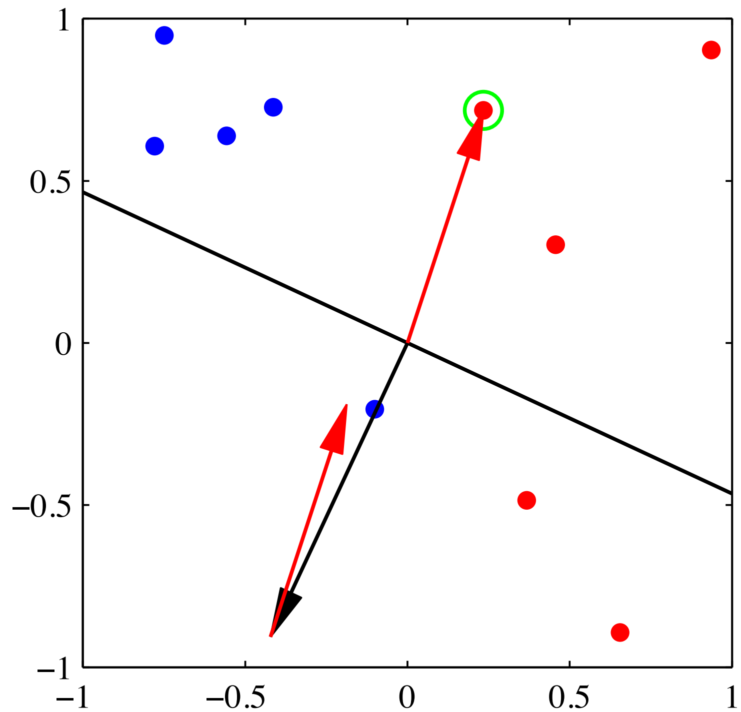
Until $\forall k,\ \ h(\mathbf{x}_k) = y_k$

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:

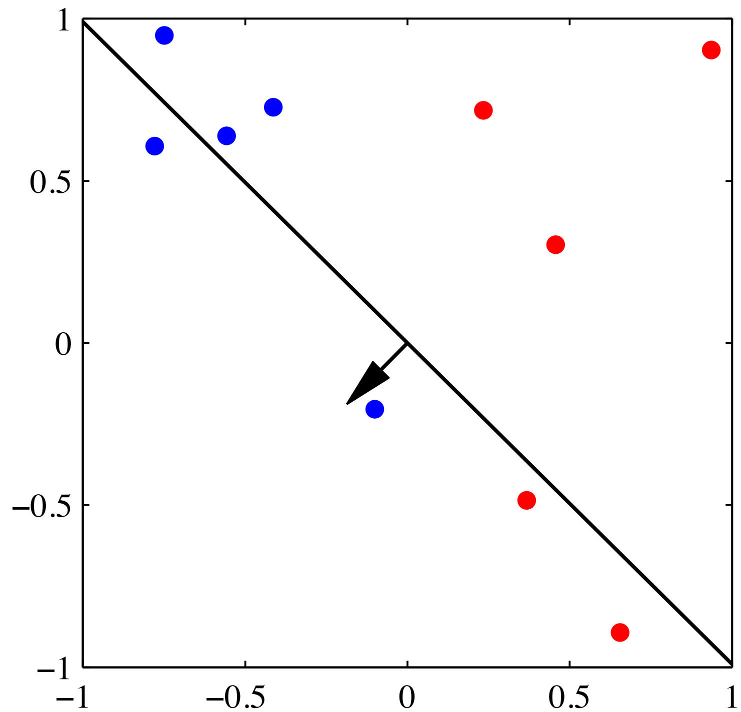


Red is the positive class

Blue is the negative class

Perceptron Algorithm

- Example (cont'd):

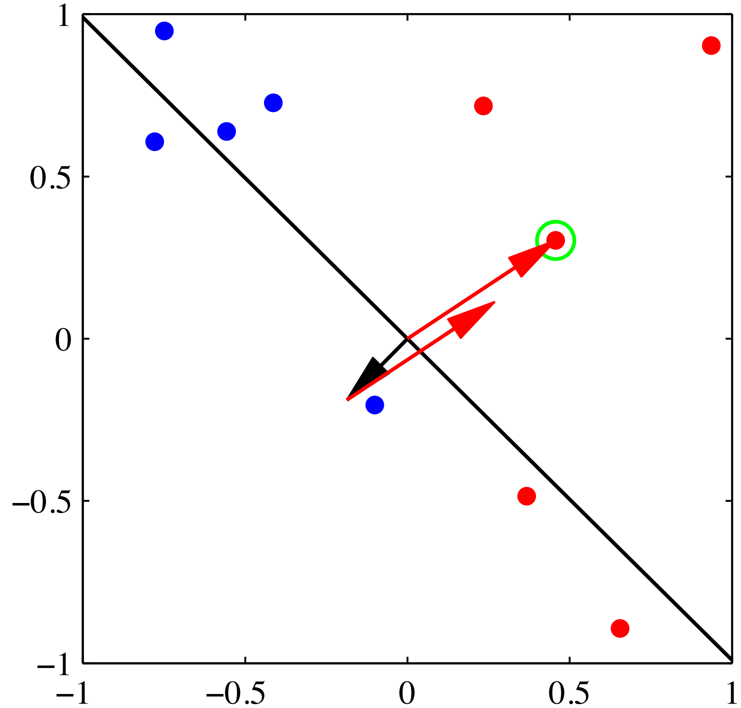


Red is the positive class

Blue is the negative class

Perceptron Algorithm

- Example (cont'd):

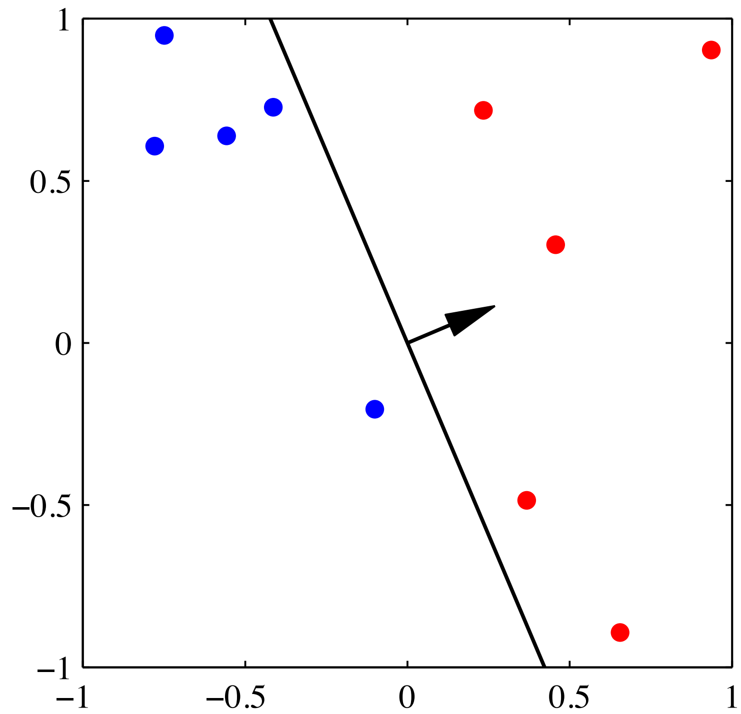


Red is the positive class

Blue is the negative class

Perceptron Algorithm

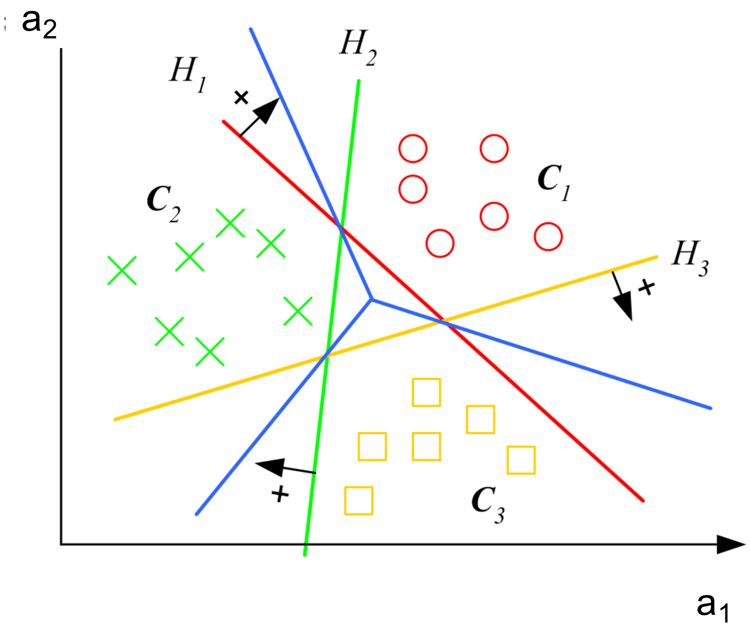
- Example (cont'd):



Red is the positive class

Blue is the negative class

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

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

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 $\boldsymbol{c}$. 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.