Perceptrons

CS530
Chapman

Spring 2020

Table of Contents

Topics to be discussed

Fixed-weight neural models

- Lateral inhibition
- Winner-take-all networks

Learning feed-forward neural-networks

- Perceptron
- Gradient descent & stochastic gradient descent
  - Widrow-Hoff (delta) rule
  - Backpropagation

Topics to be discussed

Recurrent neural networks

- Hopfield models
- Hopfield network extensions

Multi-layer neural networks & deep learning

Learning in feed-forward neural-networks

Learning feed-forward neural-networks

Perceptron

Gradient descent & stochastic gradient descent

- Widrow-Hoff (delta) rule
- Backpropagation

The simplest neural network

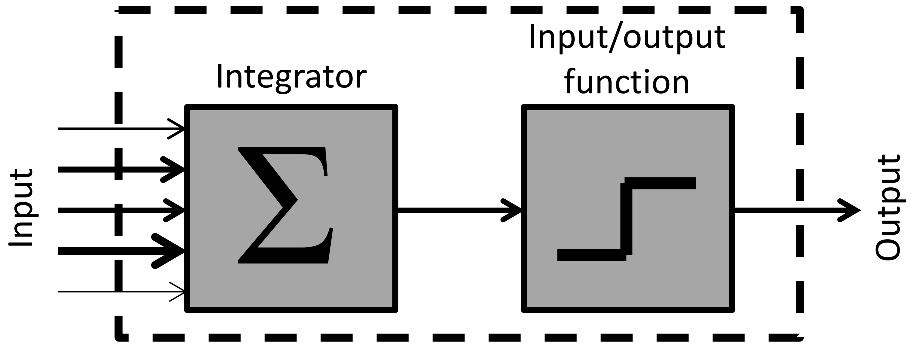
PERCEPTRON

Constructing the perceptron

The perceptron is a single-layer neural network—or, actually, a single neuron—that sums its inputs (including a bias, b) and outputs the sign of that sum.

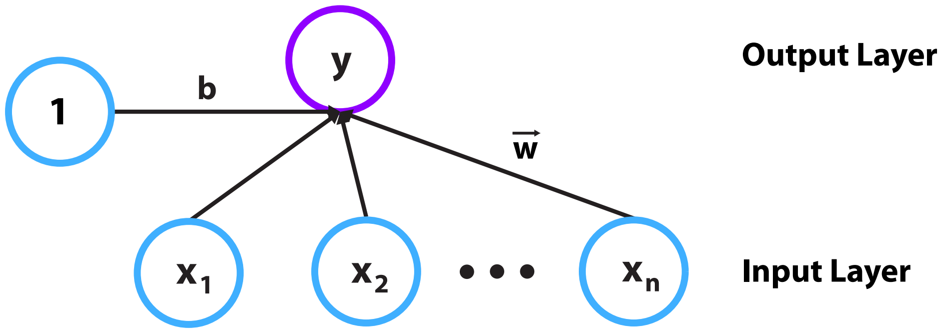
The goal of the perceptron is to learn to correctly classify a set of inputs into one of two classes. It therefore outputs 1 for Class 1 and -1 for Class 2.

It was the first and remains the simplest learning neural network (Rosenblatt, 1957).



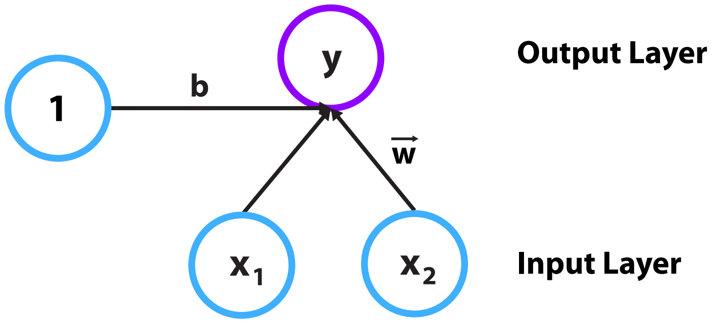
$$\vec{x},\vec{w},b \longrightarrow \vec{x}\cdot\vec{w}+b \longrightarrow f(\ ) \longrightarrow y$$

$$y = \operatorname{sign}\left(\vec{x}\cdot\vec{w}+b\right)$$



Perceptron decision boundaries

To develop insight into the behavior of the perceptron, let us examine the decision boundaries of the simplest, 2-element perceptron. We will find the boundary, ℓ.



$$y = \text{sign}(x_1 w_1 + x_2 w_2 + b)$$

$$\ell : y = \text{sign}(\vec{x} \cdot \vec{w} + b) = 0$$

$$\Rightarrow \vec{x} \cdot \vec{w} + b = 0$$

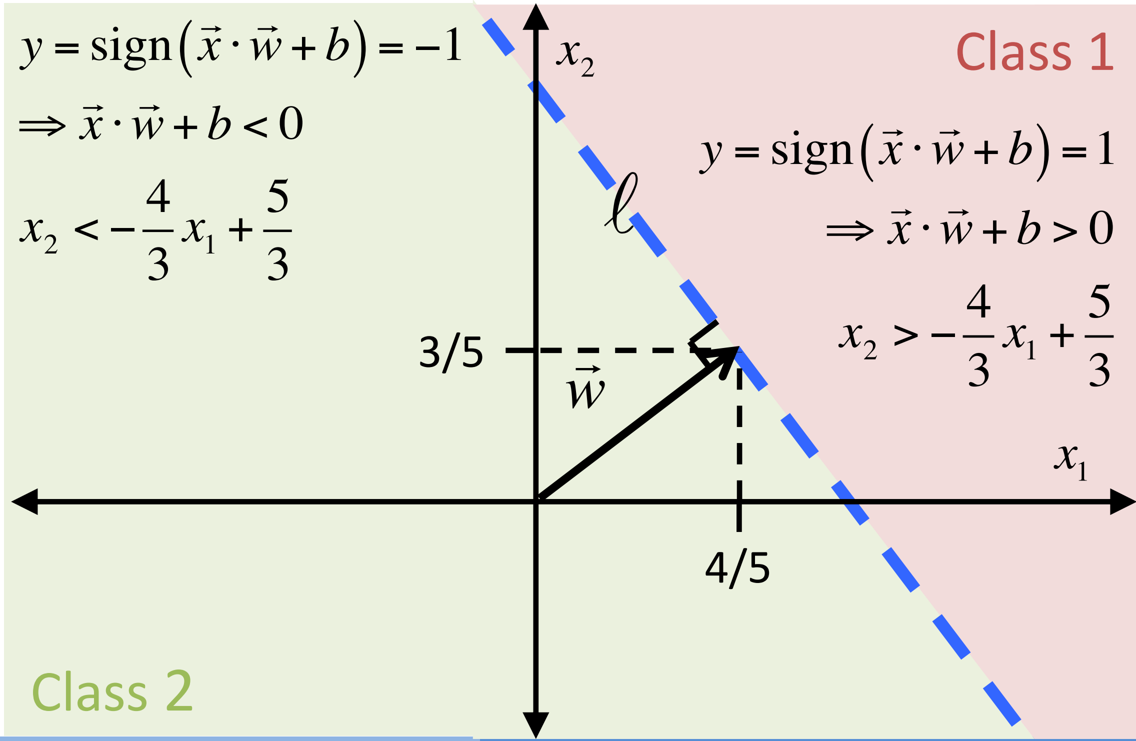
$$\Rightarrow \vec{x} \cdot \vec{w} = -b$$

$$\vec{w} = \left(\frac{4}{5}, \frac{3}{5}\right) \Rightarrow \|\vec{w}\| = 1 \quad \wedge \quad b = -1$$

$$\Rightarrow (x_1, x_2) \cdot \left(\frac{4}{5}, \frac{3}{5}\right) = 1$$

$$\Rightarrow \frac{4}{5} x_1 + \frac{3}{5} x_2 = 1$$

$$\ell : x_2 = -\frac{4}{3} x_1 + \frac{5}{3}$$



$$y = \text{sign}(\vec{x} \cdot \vec{w} + b) = -1$$

$$\Rightarrow \vec{x} \cdot \vec{w} + b < 0$$

$$x_2 < -\frac{4}{3} x_1 + \frac{5}{3}$$

$$y = \text{sign}(\vec{x} \cdot \vec{w} + b) = 1$$

$$\Rightarrow \vec{x} \cdot \vec{w} + b > 0$$

$$x_2 > -\frac{4}{3} x_1 + \frac{5}{3}$$

Perceptron convergence theorem

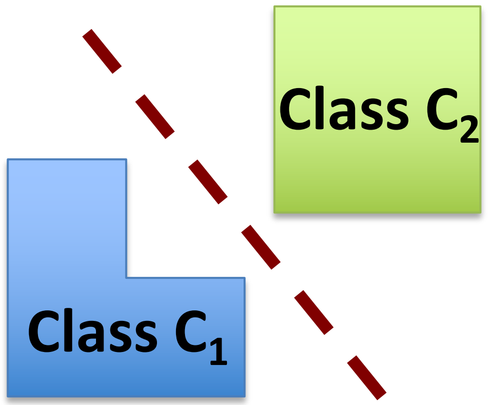
Instead of carrying a separate bias term, *b*, let us incorporate it into the perceptron:

$$\vec{x} = (x_1, x_2, \cdots, x_n, 1)$$
$$\vec{w} = (w_1, w_2, \cdots, w_n, b)$$
$$y = \text{sign}\left(\vec{x} \cdot \vec{w}\right)$$

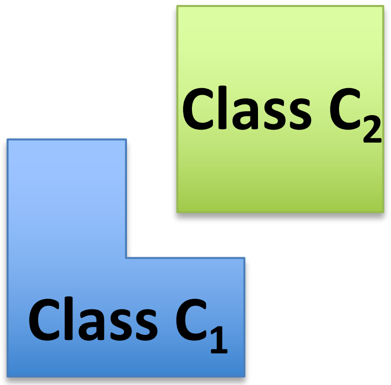
Now, if *C_1* and *C_2* are linearly separable, there exists a $\vec{w}$ such that:

$$\vec{x} \cdot \vec{w} > 0 \quad \forall \vec{x} \in C_1$$

$$\vec{x} \cdot \vec{w} < 0 \quad \forall \vec{x} \in C_2$$



Linearly separable classes



Linearly inseparable classes

Perceptron learning rule

The perceptron algorithm:

For inputs $\vec{x}, \vec{w}, y, d,$ and η,

where d are the desired outputs

and η is the learning rate:

$\vec{w}(0) = \vec{0}$

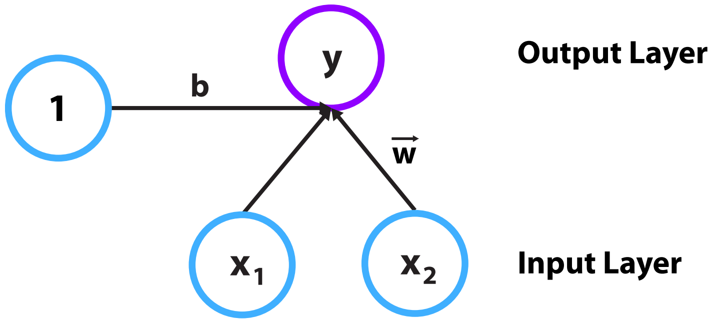
For $n = 1, 2, \cdots$

$$y(n) = \text{sign}\left(\vec{x}(n) \cdot \vec{w}(n)\right)$$

$$\vec{w}(n+1) = \vec{w}(n) + \frac{\eta}{2}\left[d(n) - y(n)\right]\vec{x}(n)$$

Note that if $d(n) = y(n)$, $\vec{w}(n+1) = \vec{w}(n)$

Hence, for incorrect assignments, the weights move toward the missed sample, making it more likely that samples like that would be correctly classified, if encountered again.



$$y = \text{sign}\left(x_1 w_1 + x_2 w_2 + b\right)$$

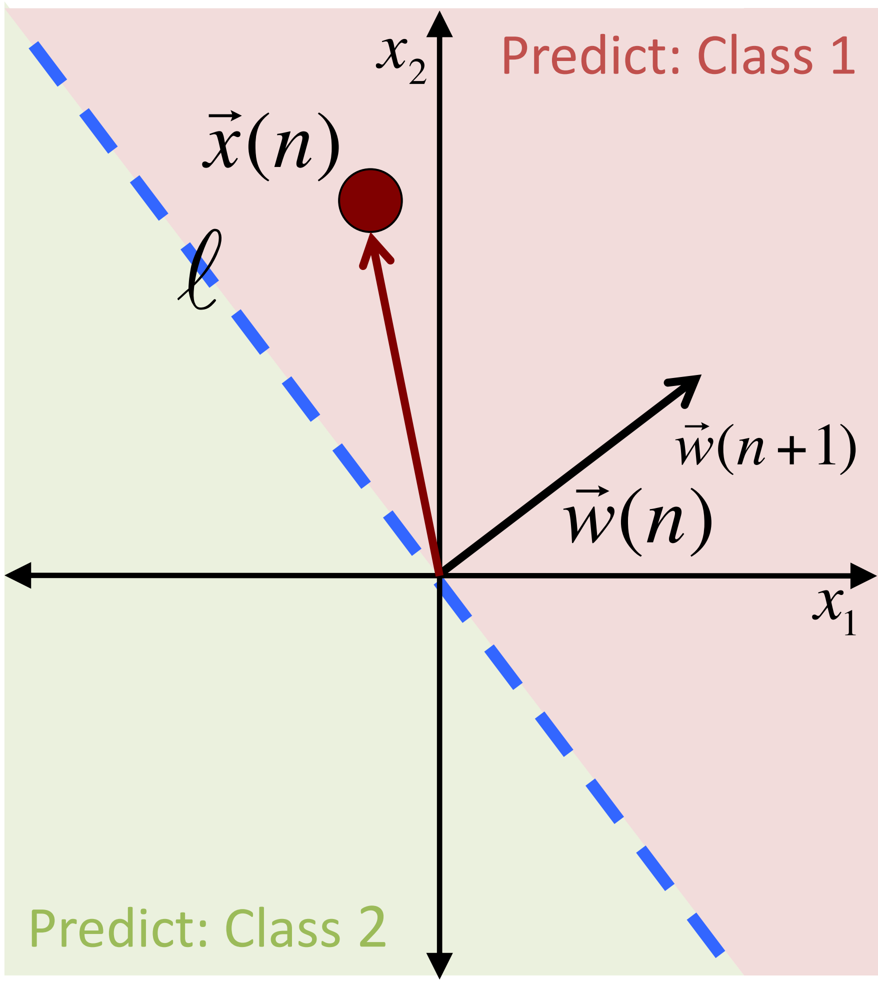
Perceptron learning rule

Situation #1:

$$y(n) = \operatorname{sign}\left(\vec{x}(n)\cdot\vec{w}\right) = 1$$

$$d(n) = 1$$

$$\Rightarrow \vec{w}(n+1) = \vec{w}(n)$$



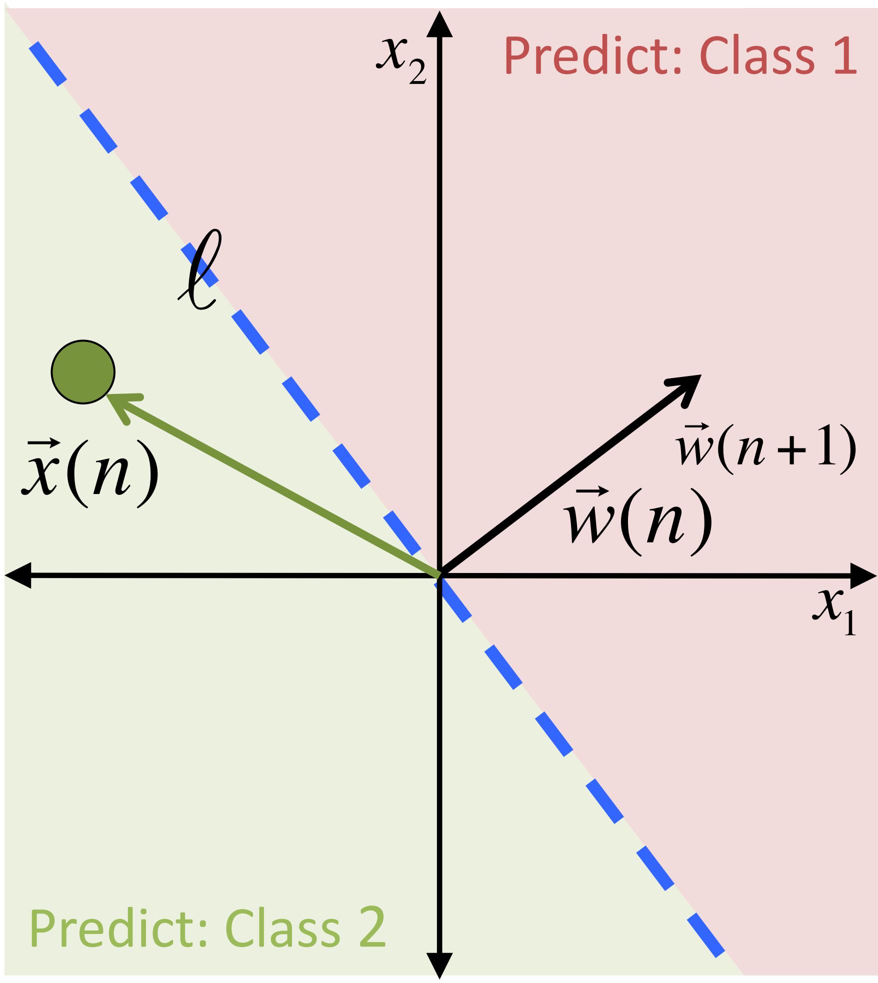
Perceptron learning rule

Situation #2:

$$y(n) = \text{sign}\left(\vec{x}(n)\cdot\vec{w}\right) = -1$$

$$d(n) = -1$$

$$\Rightarrow \vec{w}(n+1) = \vec{w}(n)$$



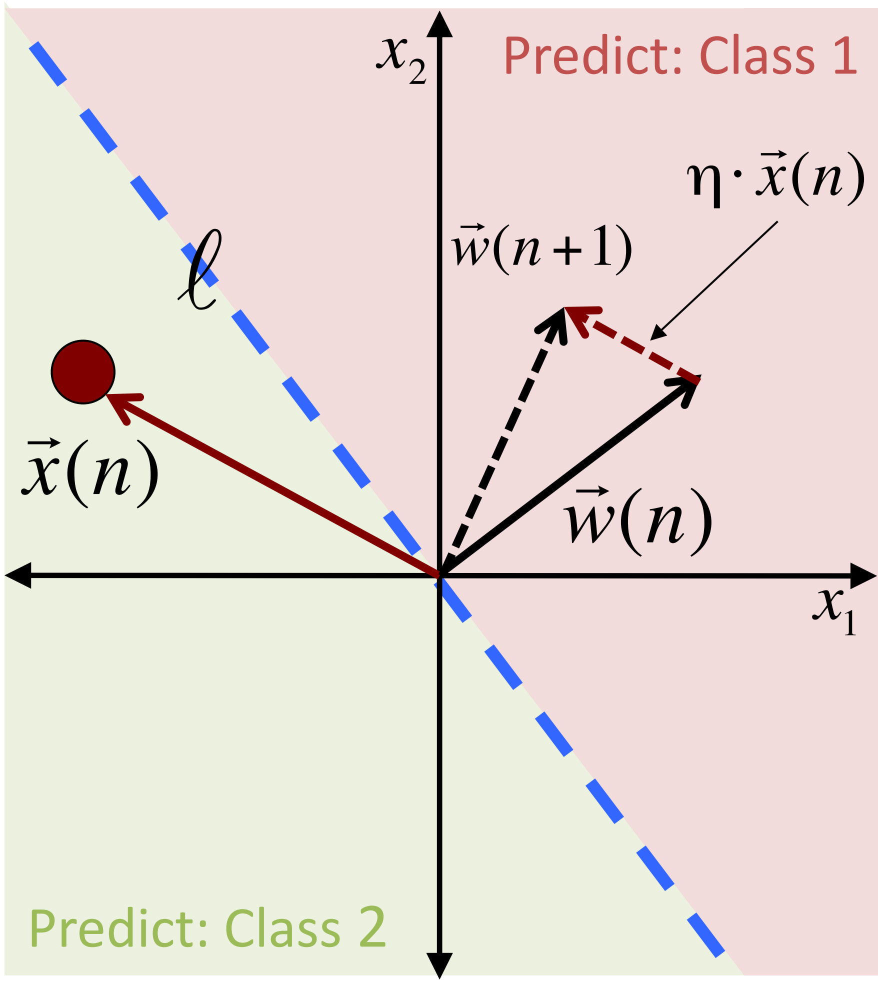
Perceptron learning rule

Situation #3:

$$y(n) = \text{sign}\left(\vec{x}(n) \cdot \vec{w}\right) = -1$$

$$d(n) = 1$$

$$\Rightarrow \vec{w}(n+1) = \vec{w}(n) + \eta \cdot \vec{x}(n)$$



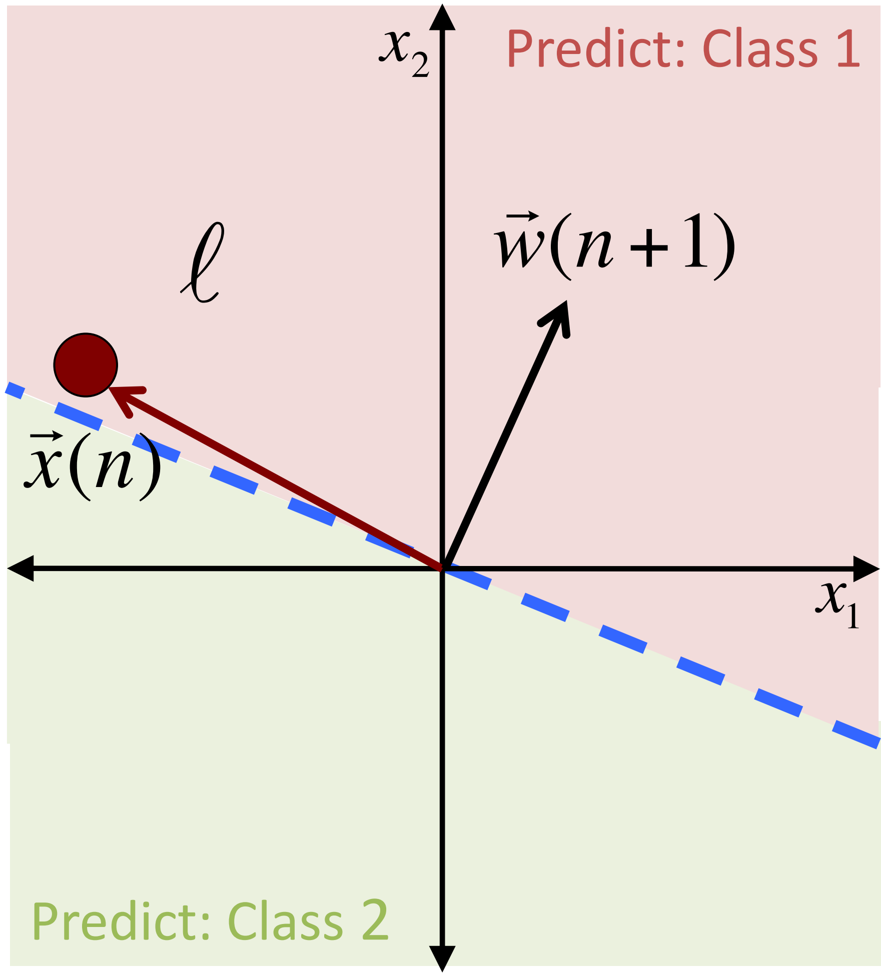
Perceptron learning rule

Situation #3:

$$y(n) = \text{sign}\left(\vec{x}(n) \cdot \vec{w}\right) = -1$$

$$d(n) = 1$$

$$\Rightarrow \vec{w}(n+1) = \vec{w}(n) + \eta \cdot \vec{x}(n)$$



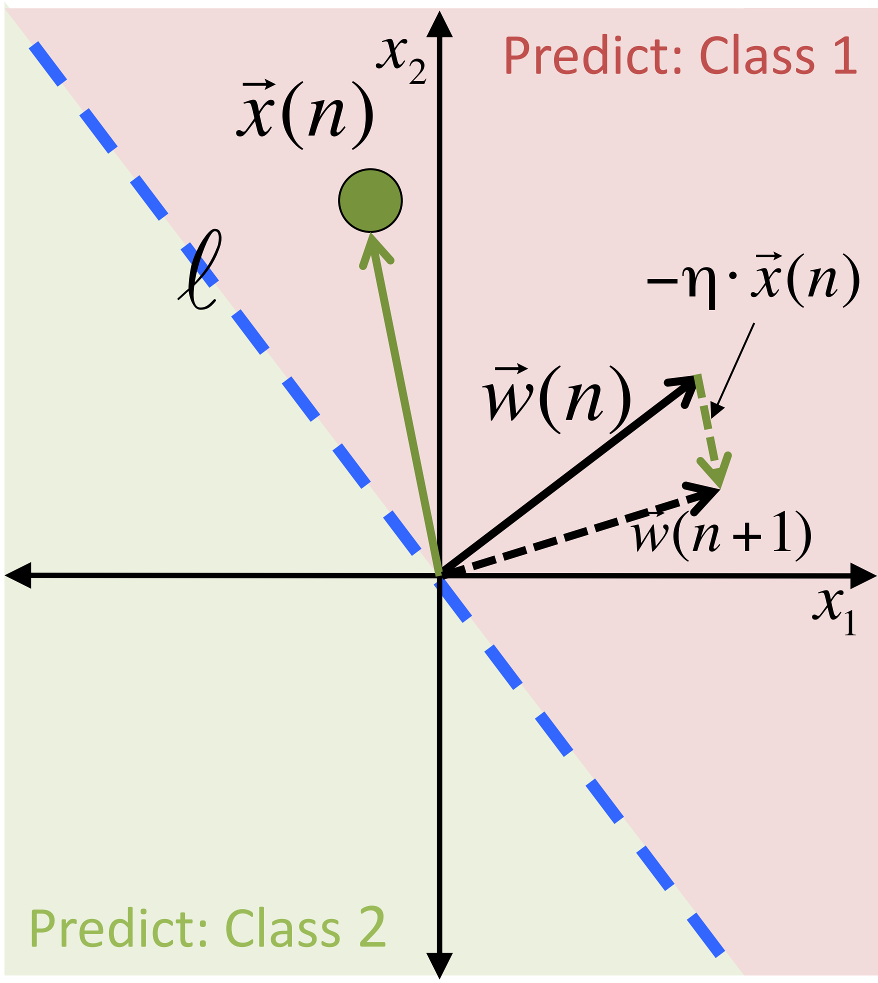
Perceptron learning rule

Situation #4:

$$y(n) = \operatorname{sign}\left(\vec{x}(n)\cdot\vec{w}\right) = 1$$

$$d(n) = -1$$

$$\Rightarrow \vec{w}(n+1) = \vec{w}(n) - \eta\cdot\vec{x}(n)$$



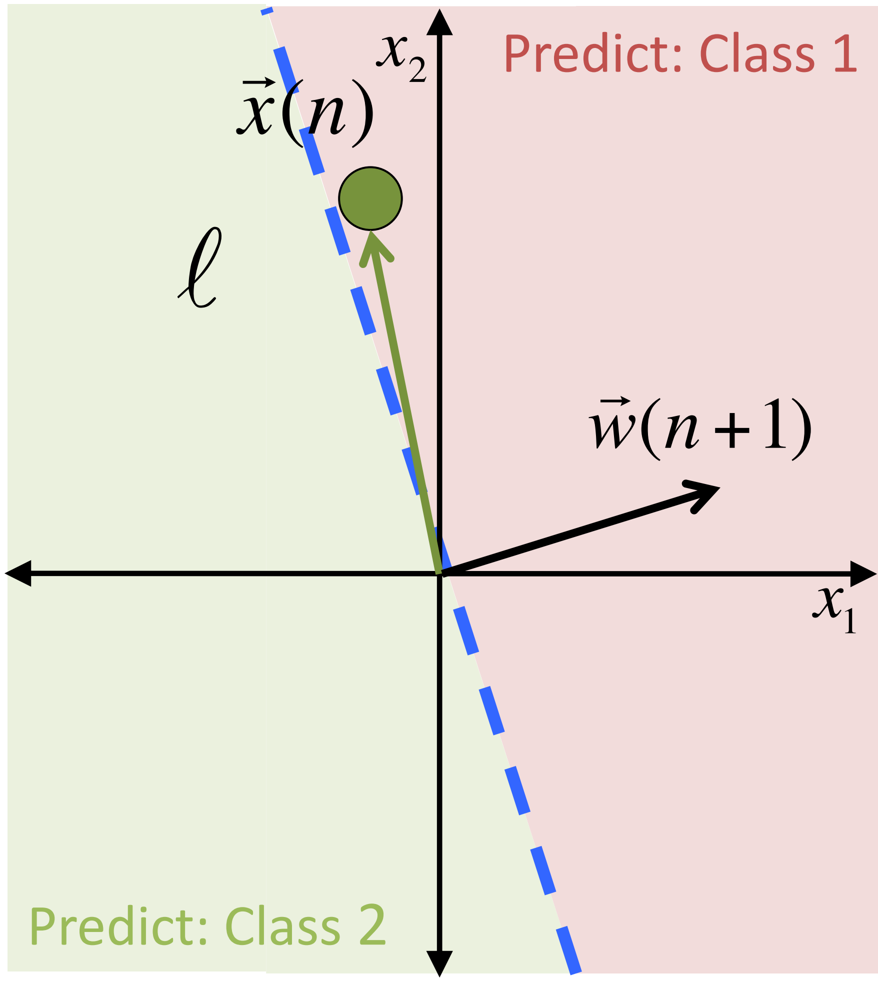
Perceptron learning rule

Situation #4:

$$y(n) = \operatorname{sign}\left(\vec{x}(n)\cdot\vec{w}\right) = 1$$

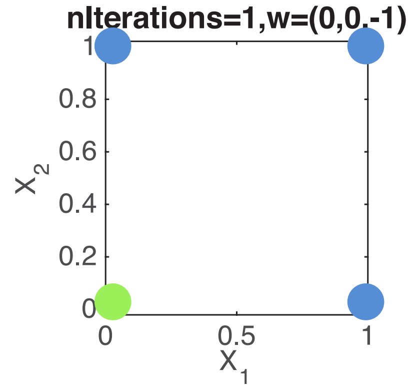
$$d(n) = -1$$

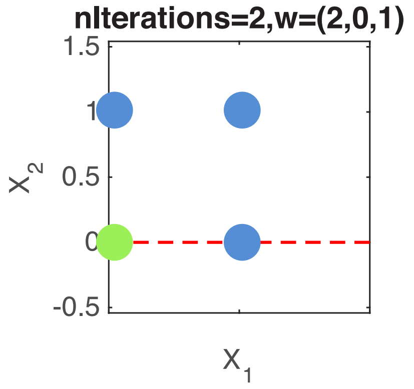
$$\Rightarrow \vec{w}(n+1) = \vec{w}(n) - \eta\cdot\vec{x}(n)$$

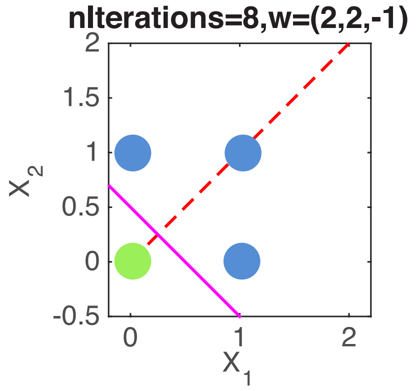


Training a perceptron on logical OR

If we want to fit a perceptron to the logical OR operation, we give it four inputs: (0,0), (1,0), (0,1), and (1,1). We give them the labels (-1,1,1,1). The classes are linearly separable. So let the perceptron learn.







Inputs (time t)		Output (time t+1)
a	b	
0	0	0
0	1	1
1	0	1
1	1	1

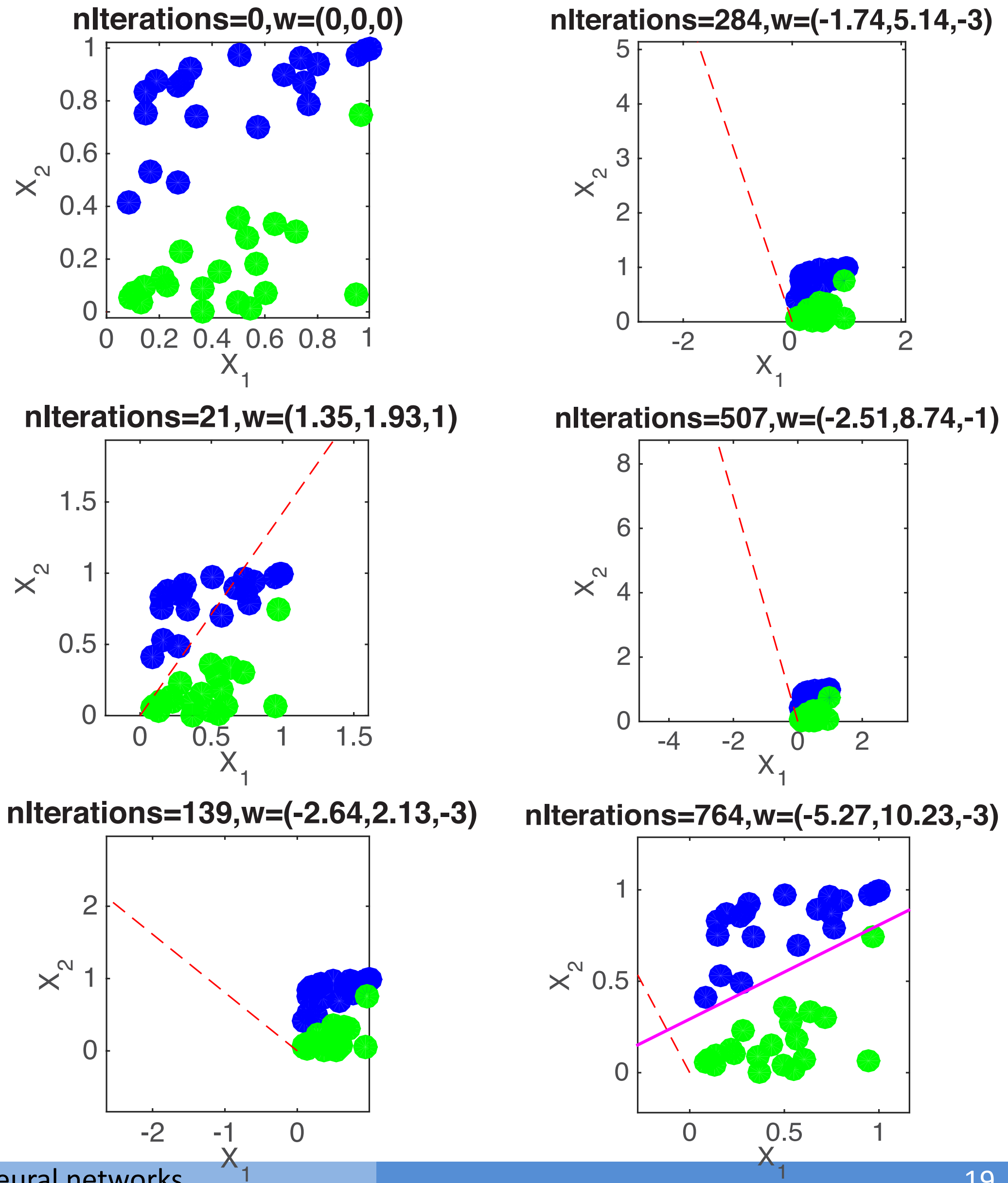
Another perceptron demo

We now generate 20 random points in the upper-left half of the unit square and 20 others in the lower-right half. We make sure to leave a margin between the two classes so that they would be linearly separable.

We train a perceptron on this set (top-left panel). It takes it 764 iterations to converge. But it does find a separator (bottom-right panel).

The perceptron updates its weights only when erring, so, as it gets closer to the correct separator, it converges increasingly slower.

It is also very sensitive to noise. A single misclassified sample could abolish its convergence, and keep it making large weight changes forever.



Perceptron convergence theorem

The perceptron algorithm:

For inputs $\vec{x}, \vec{w}, y, d,$ and η,

where d are the desired outputs

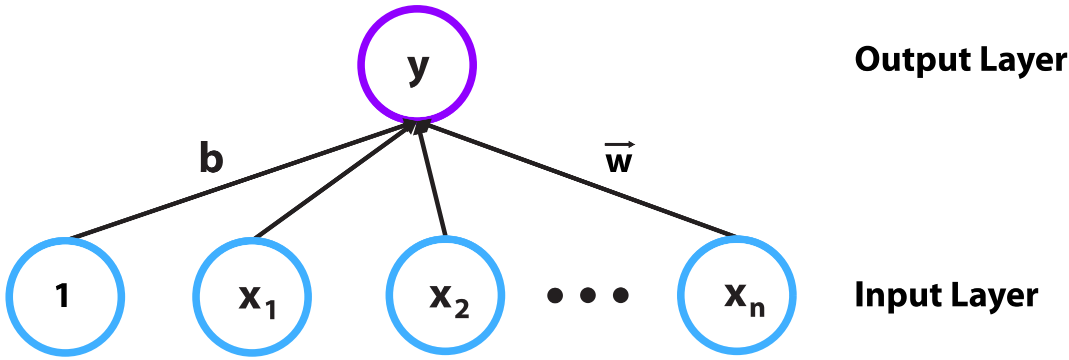
and η is the learning rate:

$\vec{w}(0) = \vec{0}$

For $n = 1, 2, \cdots$

$$y(n) = \operatorname{sign}\left(\vec{x}(n) \cdot \vec{w}(n)\right)$$

$$\vec{w}(n+1) = \vec{w}(n) + \frac{\eta}{2}\left[d(n) - y(n)\right]\vec{x}(n)$$

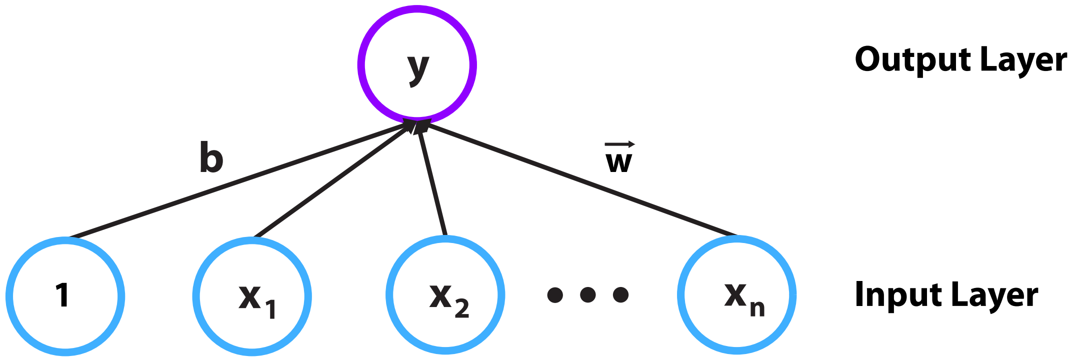


Note that if $d(n) = y(n)$, $\vec{w}(n+1) = \vec{w}(n)$

Hence, for incorrect assignments, the weights move toward the missed sample, making it more likely that samples like that would be correctly classified, if encountered again.

Perceptron as an online algorithm

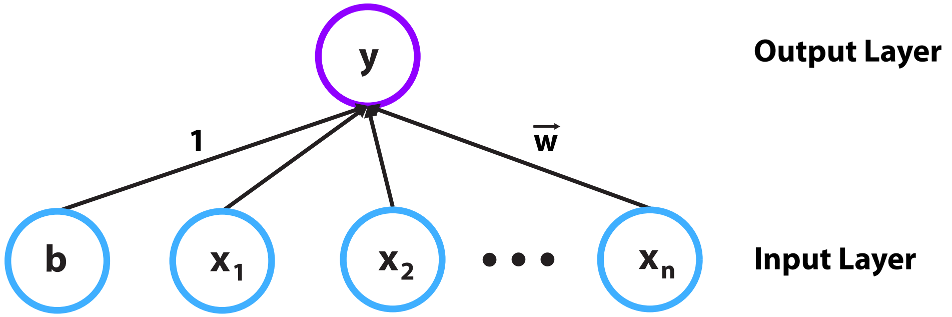
There is a sense in which the Perceptron is different from other algorithms we discussed so far. In those algorithms, the data samples were input in batched—often termed *batch processing*. The perceptron receives samples one at a time—often termed *online processing*.

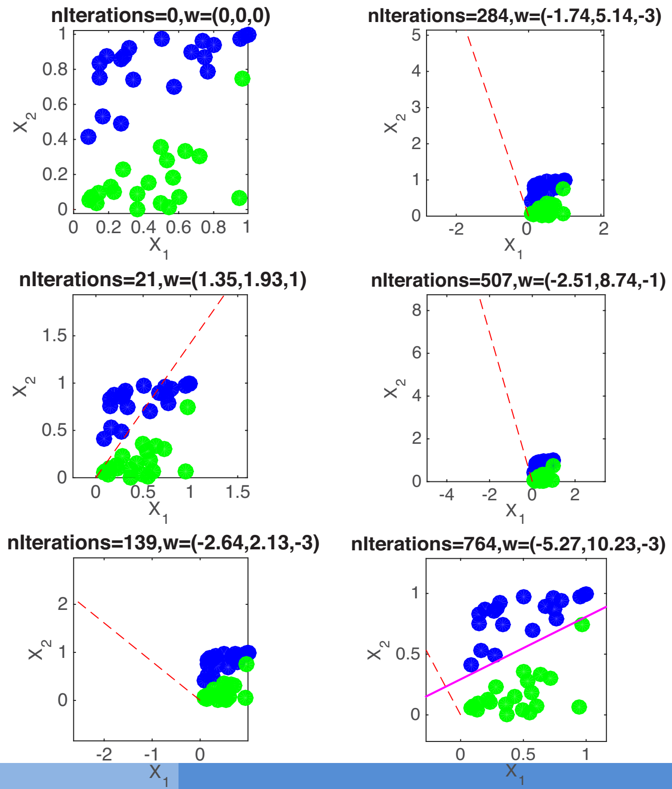


Perceptron summary

The perceptron was the first learning neural network. It was simple and elegant. And an upper bound was derived for the maximal number of iterations until convergence for linearly separable classes. So, it elicited much excitement and interest in neural networks in the 1960's.

However, its convergence was often impractically slow in practice, especially when the margin between the classes was small. And, if the classes were not linearly separable, or classification was noisy, it would never converge.


y
Output Layer
1
$\vec{w}$
b
x_1
x_2
• • •
x_n
Input Layer


nIterations=0,w=(0,0,0)
nIterations=284,w=(-1.74,5.14,-3)
nIterations=21,w=(1.35,1.93,1)
nIterations=507,w=(-2.51,8.74,-1)
nIterations=139,w=(-2.64,2.13,-3)
nIterations=764,w=(-5.27,10.23,-3)
X_1
X_2