Lecture 11: Practical Use of Perceptron and Variants

COMP 343, Spring 2022

Victoria Manfredi

WESLEYAN UNIVERSITY

Acknowledgements: These slides are based primarily on content from the book "Machine Learning" by Tom Mitchell and slides created by Vivek Srikumar (Utah), Dan Roth (Penn), Julia Hockenmaier (Illinois Urbana-Champaign), and Kai-Wei Chang (UCLA)

Today's Topics

Homework 5

- Due Friday, March 11 by 5p

Perceptron

- Recap
- Weight update
- Practical use and variants

Homework 5 discussion

Debugging

Dataframes: how to access subset of columns? (aka, remove the label)

Cross-validation

Office hours Thursday?

Installing and updating python libraries using pip

```
> pip3 install numpy==1.19.5
Requirement already satisfied: numpy==1.19.5 in /usr/local/lib/python3.9/site-packages (1.19.5)
vmanfredi@ ~ () $
>
```

Vector and its transpose

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

Transpose of weight vector: $\mathbf{w}^T = [w_0, w_1, w_2, \ldots, w_d]$

Dot product: $\mathbf{w}^T\mathbf{x}$

$$[w_0, w_1, w_2, \ldots, w_d] \times \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} = w_0 1 + w_1 x_1 + w_2 x_2 + \ldots w_d x_d$$

How to represent vectors in python?

Use numpy library:

https://numpy.org/doc/stable/user/absolute_beginners.html

```
>>>
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> w = np.array([4, 2, 1])
```

How to implement dot product in python?

Use numpy library:

https://numpy.org/doc/stable/reference/generated/numpy.dot.html

```
>>>
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> w = np.array([4, 2, 1])
>>> wt = w.transpose()
>>> np.dot(wt, x)
11
>>> wt.dot(a)
11
>>>
```

Conceptually, need to transpose to change column vector to row vector (though, not clear whether library cares …)

Perceptron

RECAP

REPORT NO. 85-460-1

THE PERCEPTRON

A PERCEIVING AND RECOGNIZING AUTOMATON

(PROJECT PARA)

January, 1957

Prepared by: Frank Rosenblatt

Frank Rosenblatt,
Project Engineer

Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

F. ROSENBLATT

Cornell Aeronautical Laboratory

The hype

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

The New York Times, July 8 1958

HAVING told you about the giant digital computer known as I.B.M. 704 and how it has been taught to play a fairly creditable game of chess, we'd like to tell you about an even more remarkable machine, the perceptron, which, as its name implies, is capable of what amounts to original thought. The first perceptron has yet to be built,

The New Yorker, December 6, 1958 P. 44

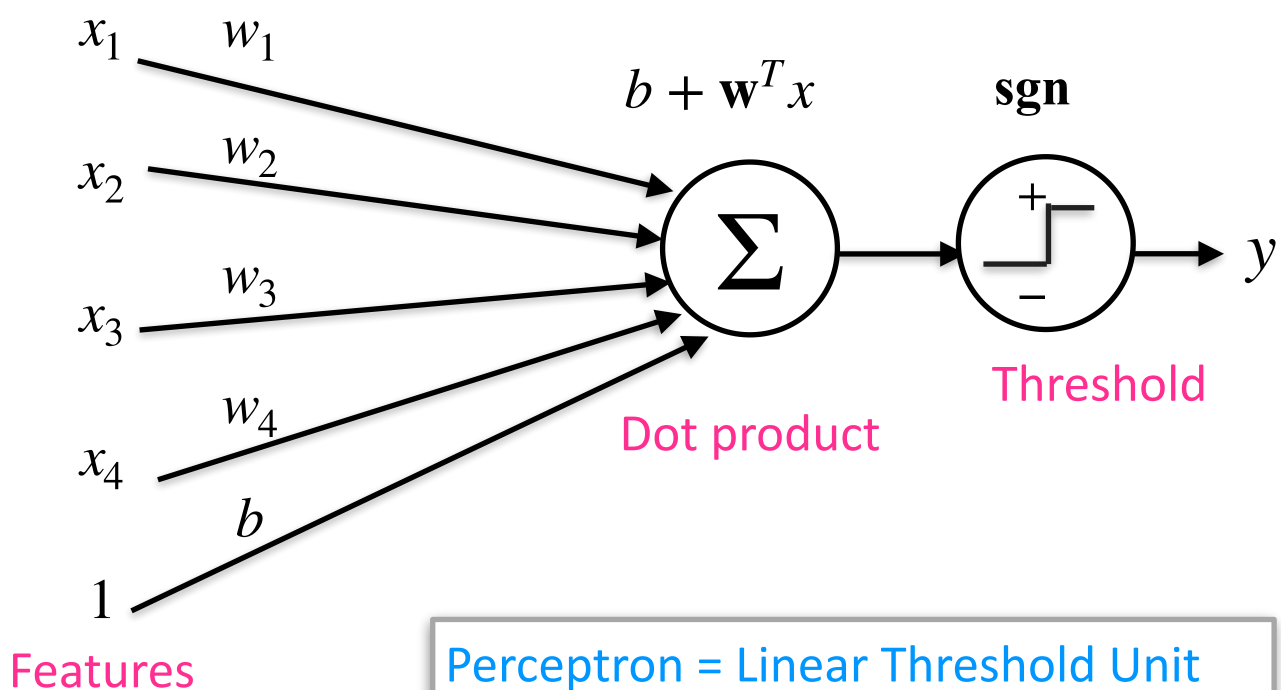
The IBM 704 computer

Perceptron is a linear threshold unit

Inputs are d dimensional vectors, denoted by $\mathbf{x}$

Output is a label $y \in \{-1,1\}$

Linear Threshold Units classify an example $\mathbf{x}$ using parameters $\mathbf{w}$ (a d dimensional vector) and b (a real number) according to the following classification rule



Perceptron = Linear Threshold Unit

The perceptron algorithm

Input: A sequence of training examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots$ where all $\mathbf{x}_i \in \Re^d$, $y_i \in \{-1, 1\}$

1. Initialize $\mathbf{w_0} = 0 \in \Re^{d+1}$ → Algorithm first chooses an initial weight vector. Here all weights initialized to 0

$d+1$-dimensional: one weight per feature plus one weight for bias

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \quad \mathbf{w}_0 = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

The perceptron algorithm

Input: A sequence of training examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots$ where all $\mathbf{x}_i \in \Re^d$, $y_i \in \{-1,1\}$

1. Initialize $\mathbf{w_0} = 0 \in \Re^{d+1}$
2. For each training example $(\mathbf{x}_i, y_i)$:
   - Predict $y' = \text{sgn}(\mathbf{w}_t^T \mathbf{x}_i)$
   - if $y' \neq y_i$:

     Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$ → This is called the perceptron update

This update has to produce a new set of weights $\mathbf{w}_{t+1}$ taking error into account. How?

The perceptron algorithm

Input: A sequence of training examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots$ where all $\mathbf{x}_i \in \Re^d$, $y_i \in \{-1,1\}$

1. Initialize $\mathbf{w_0} = 0 \in \Re^{d+1}$
2. For each training example $(\mathbf{x}_i, y_i)$:
   - Predict $y' = \text{sgn}(\mathbf{w}_t^T \mathbf{x}_i)$
   - if $y' \neq y_i$:

     Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$

r is the learning rate, a hyperparameter, typically a number between 0 and 1

If r is too small: slow too converge

if r is too big: may not converge

The perceptron algorithm

Input: A sequence of training examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots$ where all $\mathbf{x}_i \in \Re^d$, $y_i \in \{-1,1\}$

1. Initialize $\mathbf{w_0} = 0 \in \Re^{d+1}$
2. For each training example $(\mathbf{x}_i, y_i)$:
   - Predict $y' = \text{sgn}(\mathbf{w}_t^T \mathbf{x}_i)$
   - if $y' \neq y_i$:

     Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$

$\boxed{\begin{array}{l} \text{Mistake on positive: } \mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i \\ \text{Mistake on negative: } \mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i \end{array}}$ $\begin{array}{l} y_i = +1 \\ y_i = -1 \end{array}$

y_i used here to combine update into 1 equation

y_i is either -1 or +1 so $y_i\mathbf{x}_i$ is either $-\mathbf{x}_i$ or $+\mathbf{x}_i$

The perceptron algorithm

Input: A sequence of training examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots$ where all $\mathbf{x}_i \in \Re^d$, $y_i \in \{-1, 1\}$

1. Initialize $\mathbf{w_0} = 0 \in \Re^{d+1}$
2. For each training example $(\mathbf{x}_i, y_i)$:
   - Predict $y' = \text{sgn}(\mathbf{w}_t^T \mathbf{x}_i)$
   - if $y' \neq y_i$:

     Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$
3. Return final weight vector

Stop when you run out of examples and return latest weight vector

The perceptron algorithm

Input: A sequence of training examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots$ where all $\mathbf{x}_i \in \Re^d$, $y_i \in \{-1,1\}$

1. Initialize $\mathbf{w_0} = 0 \in \Re^{d+1}$
2. For each training example $(\mathbf{x}_i, y_i)$:
   - Predict $y' = \text{sgn}(\mathbf{w}_t^T \mathbf{x}_i)$
   - if $y' \neq y_i$:

     Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$
3. Return final weight vector

Mistake can be written as $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$

The perceptron algorithm

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Original way of writing mistake

$$y' = sgn(\mathbf{w}_t^T \mathbf{x}_i)$$

$y' \neq y_i$ is a mistake

Can alternatively write

$$sgn(\mathbf{w}_t^T \mathbf{x}_i) \neq y_i$$

-1	+1
+1	-1

Mistake can thus be written as

$$y_i sgn(\mathbf{w}_t^T \mathbf{x}_i) \leq 0$$

$$y_i \mathbf{w}_t^T \mathbf{x}_i \leq 0$$

Perceptron

THE WEIGHT UPDATE

Intuition behind the update

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Suppose we have made a mistake on a positive example

That is, $y = +1$ and $\mathbf{w}_t^T\mathbf{x} \leq 0$

Call the new weight vector

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{x} \qquad \text{(say } r = 1\text{)}$$

Intuition behind the update

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Suppose we have made a mistake on a positive example

That is, $y = +1$ and $\mathbf{w}_t^T\mathbf{x} \leq 0$

Call the new weight vector

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{x} \qquad \text{(say } r = 1\text{)}$$

The new dot product is

$$\mathbf{w}_{t+1}^T = (\mathbf{w}_t + \mathbf{x})^T\mathbf{x} = \mathbf{w}_\mathbf{t}^\mathbf{T}\mathbf{x} + \mathbf{x}^\mathbf{T}\mathbf{x} \geq \mathbf{w}_\mathbf{t}^\mathbf{T}\mathbf{x}$$

For a positive example, the Perceptron update will increase the score assigned to the same input if score was negative

Intuition behind the update

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Suppose we have made a mistake on a positive example

That is, $y = +1$ and $\mathbf{w}_t^T \mathbf{x} \leq 0$

Call the new weight vector

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{x} \qquad \text{(say } r = 1\text{)}$$

The new dot product is

$$\mathbf{w}_{t+1}^T = (\mathbf{w}_t + \mathbf{x})^T \mathbf{x} = \mathbf{w}_\mathbf{t}^\mathbf{T} \mathbf{x} + \mathbf{x}^\mathbf{T} \mathbf{x} \geq \mathbf{w}_\mathbf{t}^\mathbf{T} \mathbf{x}$$

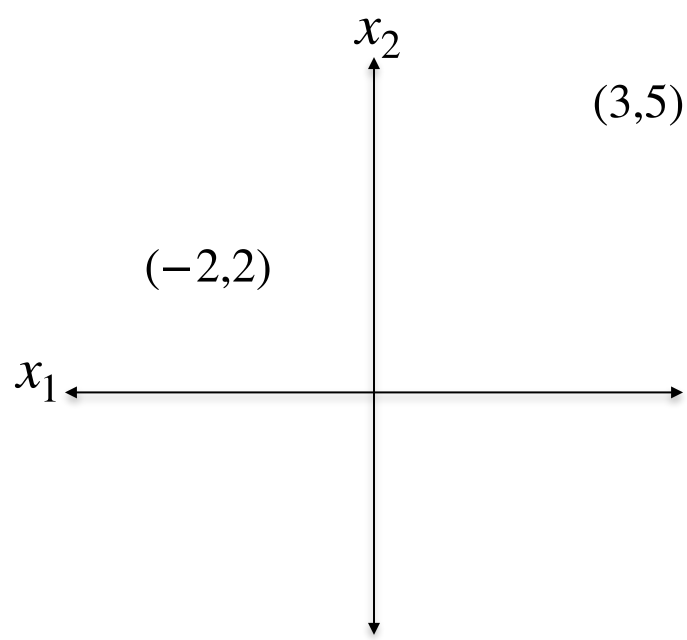
For a positive example, the Perceptron update will increase the score assigned to the same input if score was negative

Similar reasoning for negative examples: if score was positive it will decrease the score

A simple example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

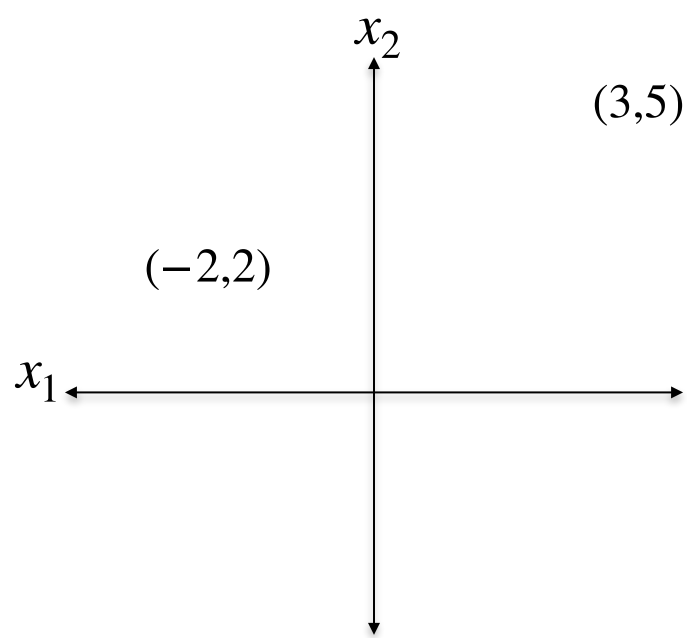
$$\mathbf{w} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$$



A simple example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\,\mathbf{x}_i$

$$\mathbf{w} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$$

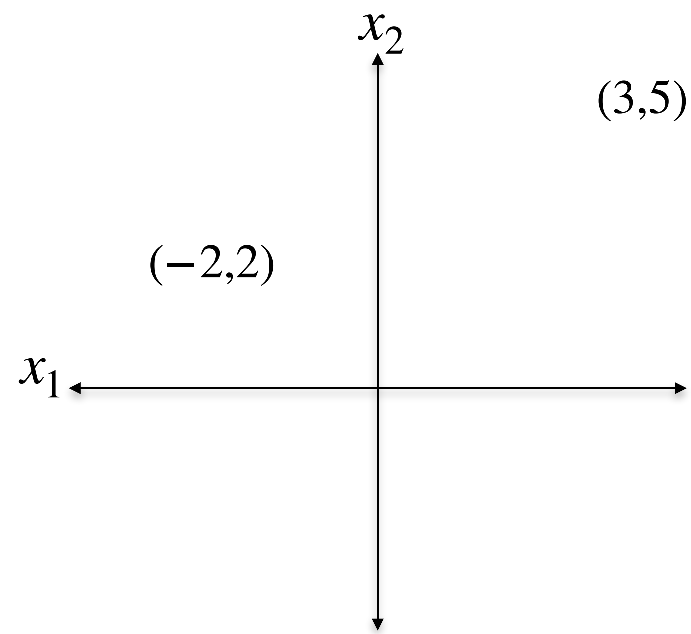


$$\mathbf{w}^T\mathbf{x}_1 = 0 \times 1 + 0 \times (-2) + 0 \times 2 = 0$$

A simple example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

$$\mathbf{w} = \begin{bmatrix}0\\0\\0\end{bmatrix} \quad \mathbf{x}_1 = \begin{bmatrix}1\\-2\\2\end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix}1\\3\\5\end{bmatrix}$$



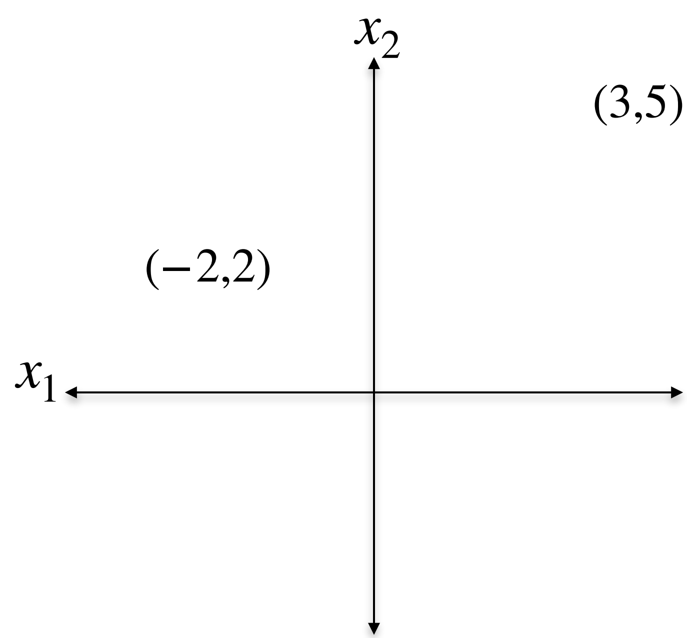
$$\mathbf{w}^T\mathbf{x}_1 = 0 \times 1 + 0 \times (-2) + 0 \times 2 = 0$$

$$\mathbf{w} \rightarrow \mathbf{w} - \mathbf{x}_1$$

A simple example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

$$\mathbf{w} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$$



$$\mathbf{w}^T\mathbf{x}_1 = 0 \times 1 + 0 \times (-2) + 0 \times 2 = 0$$

$$\mathbf{w} \rightarrow \mathbf{w} - \mathbf{x}_1$$

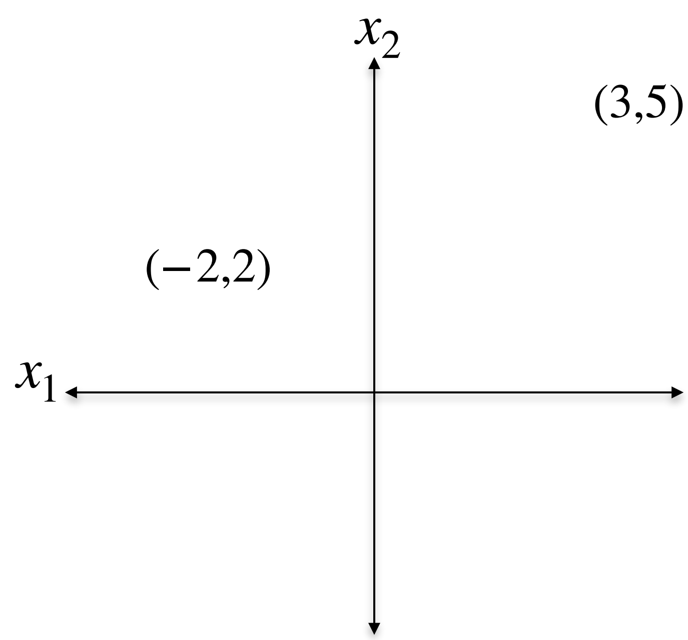
$$\mathbf{w} = \begin{bmatrix} 0-(1) \\ 0-(-2) \\ 0-(2) \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \\ -2 \end{bmatrix}$$

A simple example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$

Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

$$\mathbf{w} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$$



$$\mathbf{w}^T\mathbf{x}_1 = 0 \times 1 + 0 \times (-2) + 0 \times 2 = 0$$

$$\mathbf{w} \rightarrow \mathbf{w} - \mathbf{x}_1$$

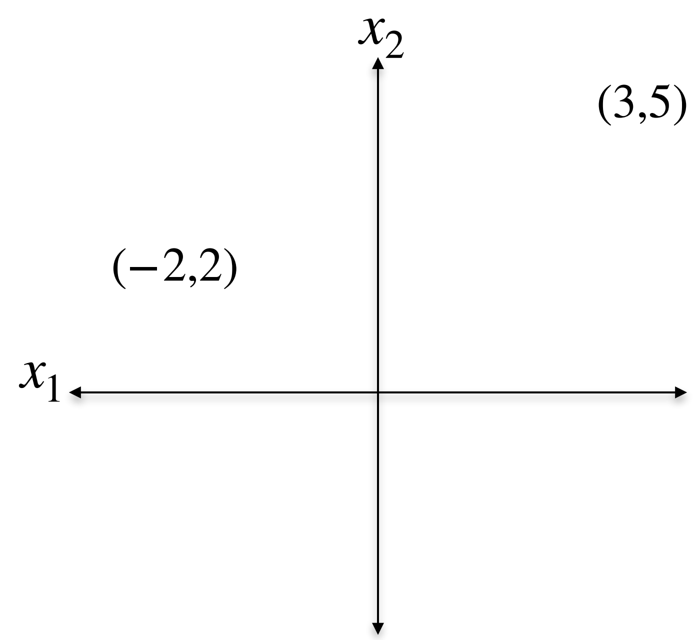
$$\mathbf{w} = \begin{bmatrix} 0 - (1) \\ 0 - (-2) \\ 0 - (2) \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \\ -2 \end{bmatrix}$$

$$\mathbf{w}^T\mathbf{x}_1 = -1 \times 1 + 2 \times (-2) + (-2) \times 2 = -9$$

A simple example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\,\mathbf{x}_i$

$$\mathbf{w} = \begin{bmatrix} -1 \\ 2 \\ -2 \end{bmatrix} \quad \mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$$

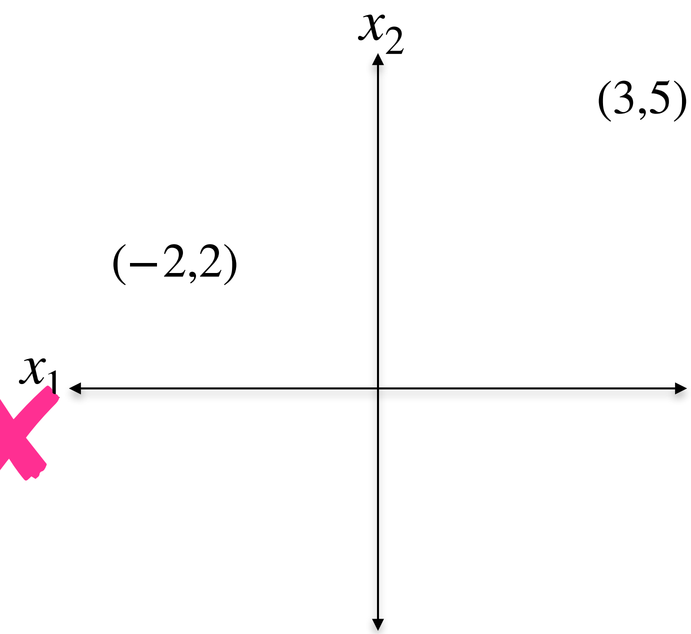


A simple example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

$$\mathbf{w} = \begin{bmatrix} -1 \\ 2 \\ -2 \end{bmatrix} \quad \mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$$

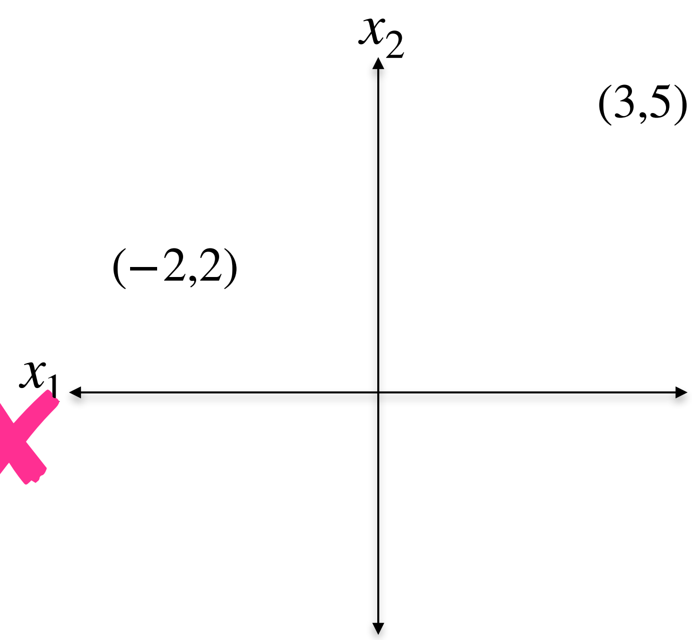
$$\mathbf{w}^T\mathbf{x}_2 = -1 \times 1 + 2 \times 3 + (-2) \times 5 = -5$$



A simple example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

$$\mathbf{w} = \begin{bmatrix} -1 \\ 2 \\ -2 \end{bmatrix} \quad \mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$$



$$\mathbf{w}^T\mathbf{x}_2 = -1 \times 1 + 2 \times 3 + (-2) \times 5 = -5$$

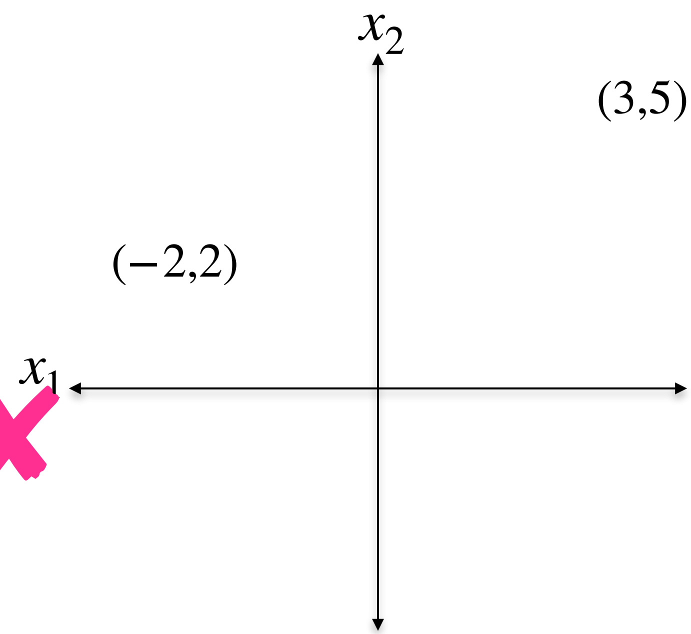
$$\mathbf{w} \rightarrow \mathbf{w} + \mathbf{x}_2$$

A simple example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$

Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

$$\mathbf{w} = \begin{bmatrix} -1 \\ 2 \\ -2 \end{bmatrix} \quad \mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$$



$$\mathbf{w}^T\mathbf{x}_2 = -1 \times 1 + 2 \times 3 + (-2) \times 5 = -5$$

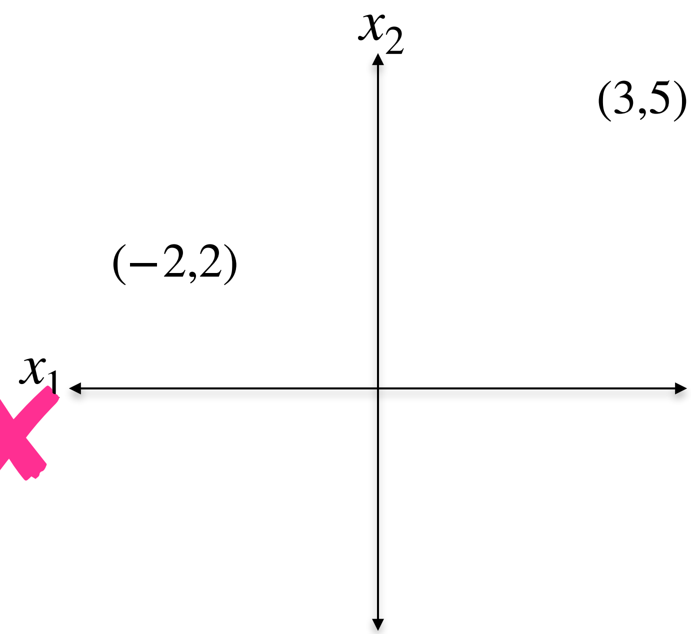
$$\mathbf{w} \to \mathbf{w} + \mathbf{x}_2$$

$$\mathbf{w} = \begin{bmatrix} -1 + (1) \\ 2 + (3) \\ -2 + (5) \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \\ 3 \end{bmatrix}$$

A simple example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

$$\mathbf{w} = \begin{bmatrix} -1 \\ 2 \\ -2 \end{bmatrix} \quad \mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$$



$$\mathbf{w}^T\mathbf{x}_2 = -1 \times 1 + 2 \times 3 + (-2) \times 5 = -5$$

$$\mathbf{w} \rightarrow \mathbf{w} + \mathbf{x}_2$$

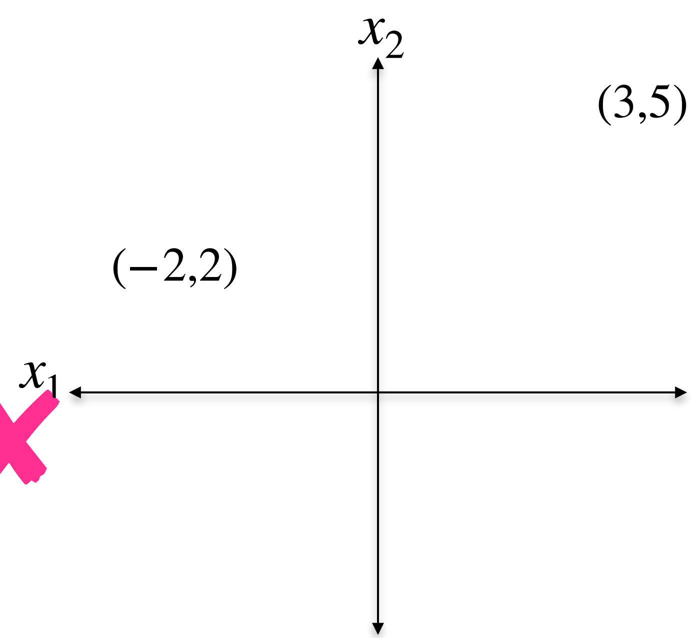
$$\mathbf{w} = \begin{bmatrix} -1 + (1) \\ 2 + (3) \\ -2 + (5) \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \\ 3 \end{bmatrix}$$

$$\mathbf{w}^T\mathbf{x}_2 = 0 \times 1 + 5 \times 3 + 3 \times 5 = 30$$

A simple example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

$$\mathbf{w} = \begin{bmatrix} -1 \\ 2 \\ -2 \end{bmatrix} \quad \mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$$



$$\mathbf{w}^T\mathbf{x}_2 = -1 \times 1 + 2 \times 3 + (-2) \times 5 = -5$$

$$\mathbf{w} \rightarrow \mathbf{w} + \mathbf{x}_2$$

$$\mathbf{w} = \begin{bmatrix} -1 + (1) \\ 2 + (3) \\ -2 + (5) \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \\ 3 \end{bmatrix}$$

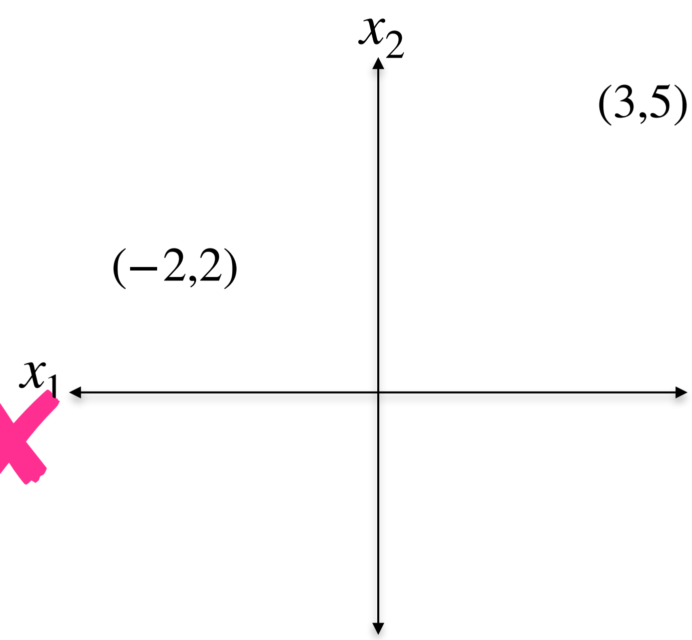
$$\mathbf{w}^T\mathbf{x}_2 = 0 \times 1 + 5 \times 3 + 3 \times 5 = 30$$

$$\mathbf{w}^T\mathbf{x}_1 = 0 \times 1 + 5 \times (-2) + 3 \times 2 = -4$$

A simple example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

$$\mathbf{w} = \begin{bmatrix} -1 \\ 2 \\ -2 \end{bmatrix} \quad \mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$$



$$\mathbf{w}^T\mathbf{x}_2 = -1 \times 1 + 2 \times 3 + (-2) \times 5 = -5$$

$$\mathbf{w} \rightarrow \mathbf{w} + \mathbf{x}_2$$

$$\mathbf{w} = \begin{bmatrix} -1 + (1) \\ 2 + (3) \\ -2 + (5) \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \\ 3 \end{bmatrix}$$

$$\mathbf{w}^T\mathbf{x} = 5x_1 + 3x_2 = 0$$

$$5(-3) + 3(5) = 0$$

$$5(3) + 3(-5) = 0$$

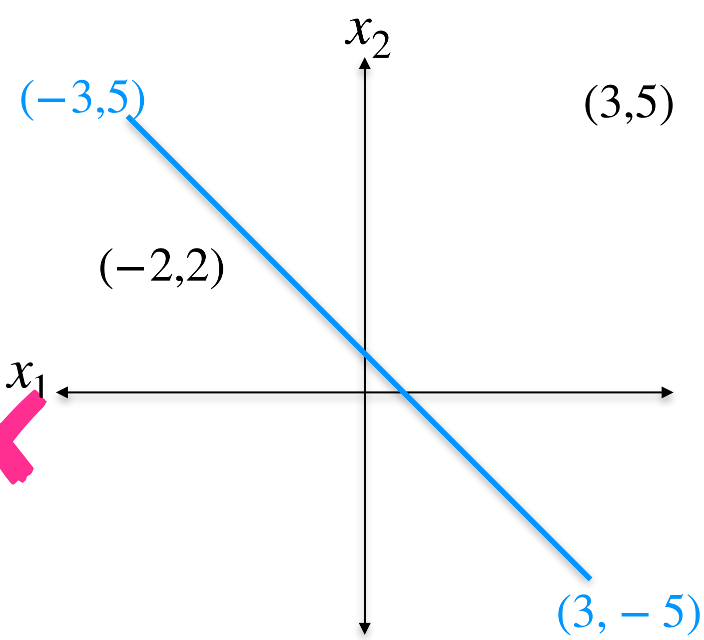
$$\mathbf{w}^T\mathbf{x}_2 = 0 \times 1 + 5 \times 3 + 3 \times 5 = 30$$

$$\mathbf{w}^T\mathbf{x}_1 = 0 \times 1 + 5 \times (-2) + 3 \times 2 = -4$$

A simple example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

$$\mathbf{w} = \begin{bmatrix} -1 \\ 2 \\ -2 \end{bmatrix} \quad \mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$$



$$\mathbf{w}^T\mathbf{x}_2 = -1 \times 1 + 2 \times 3 + (-2) \times 5 = -5$$

$$\mathbf{w} \rightarrow \mathbf{w} + \mathbf{x}_2$$

$$\mathbf{w} = \begin{bmatrix} -1 + (1) \\ 2 + (3) \\ -2 + (5) \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \\ 3 \end{bmatrix}$$

$$\mathbf{w}^T\mathbf{x} = 5x_1 + 3x_2 = 0$$

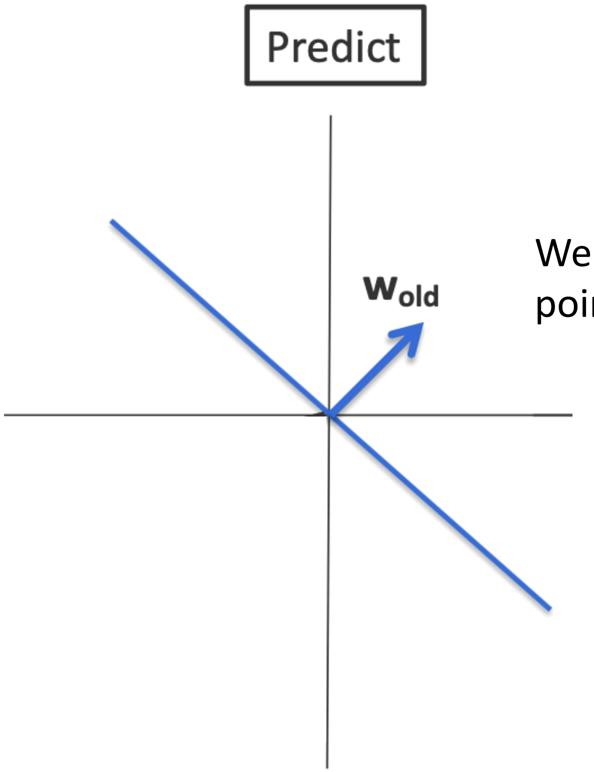
$$5(-3) + 3(5) = 0$$

$$5(3) + 3(-5) = 0$$

$$\mathbf{w}^T\mathbf{x}_2 = 0 \times 1 + 5 \times 3 + 3 \times 5 = 30$$

$$\mathbf{w}^T\mathbf{x}_1 = 0 \times 1 + 5 \times (-2) + 3 \times 2 = -4$$

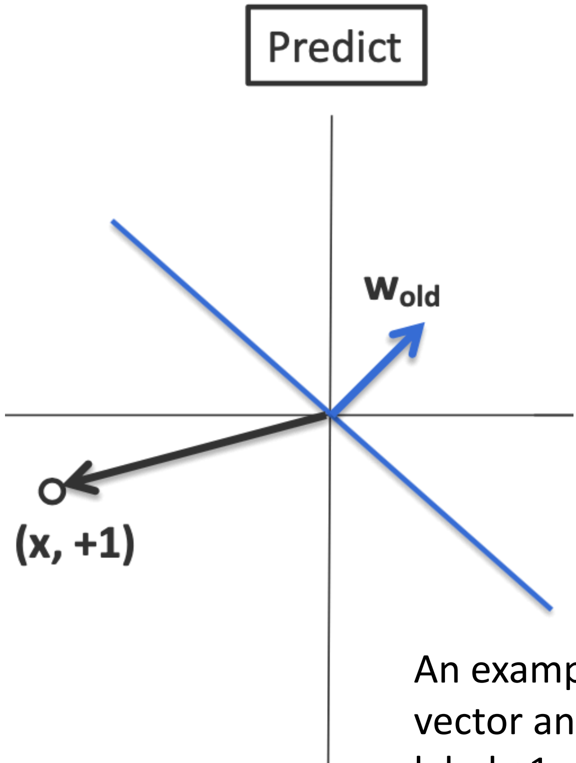
Geometry of the perceptron update



Weight vector is normal to hyperplane, points in direction of positive examples

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

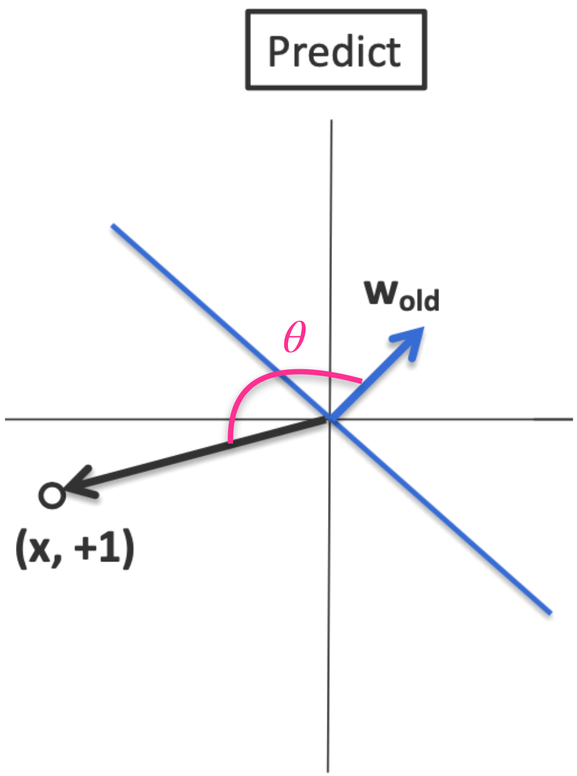
Geometry of the perceptron update



An example $\mathbf{x}$ is also a vector and here has label +1

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Geometry of the perceptron update



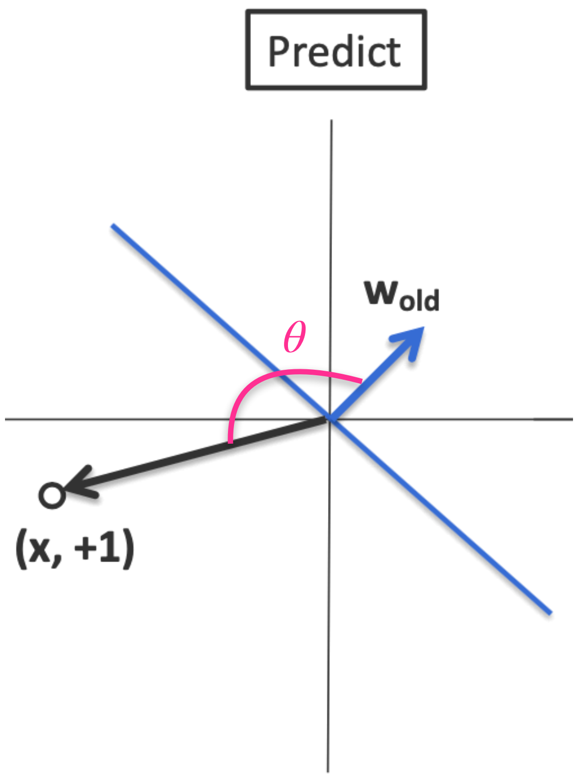
Geometric interpretation of dot product of $\mathbf{w}$ and $\mathbf{x}$

$\mathbf{w}^T\mathbf{x} = |\,\mathbf{w}\,|\,|\,\mathbf{x}\,|\,cos\theta$ where magnitude of vector is given by $|\,\mathbf{w}\,| = \sqrt{\sum_i w_i^2}$

If angle θ is more than 90 degrees, dot product is negative

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Geometry of the perceptron update



Geometric interpretation of dot product of $\mathbf{w}$ and $\mathbf{x}$

$\mathbf{w}^T\mathbf{x} = |\,\mathbf{w}\,|\,|\,\mathbf{x}\,|\, cos\theta$ where magnitude of vector is given by $|\,\mathbf{w}\,| = \sqrt{\sum_i w_i^2}$

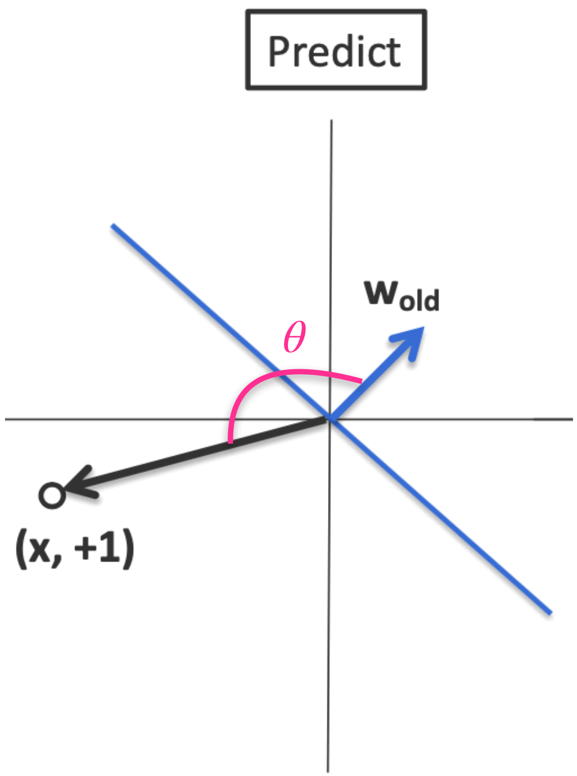
If angle θ is more than 90 degrees, dot product is negative

So what is happening here? What kind of mistake?

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$

Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Geometry of the perceptron update



Geometric interpretation of dot product of $\mathbf{w}$ and $\mathbf{x}$

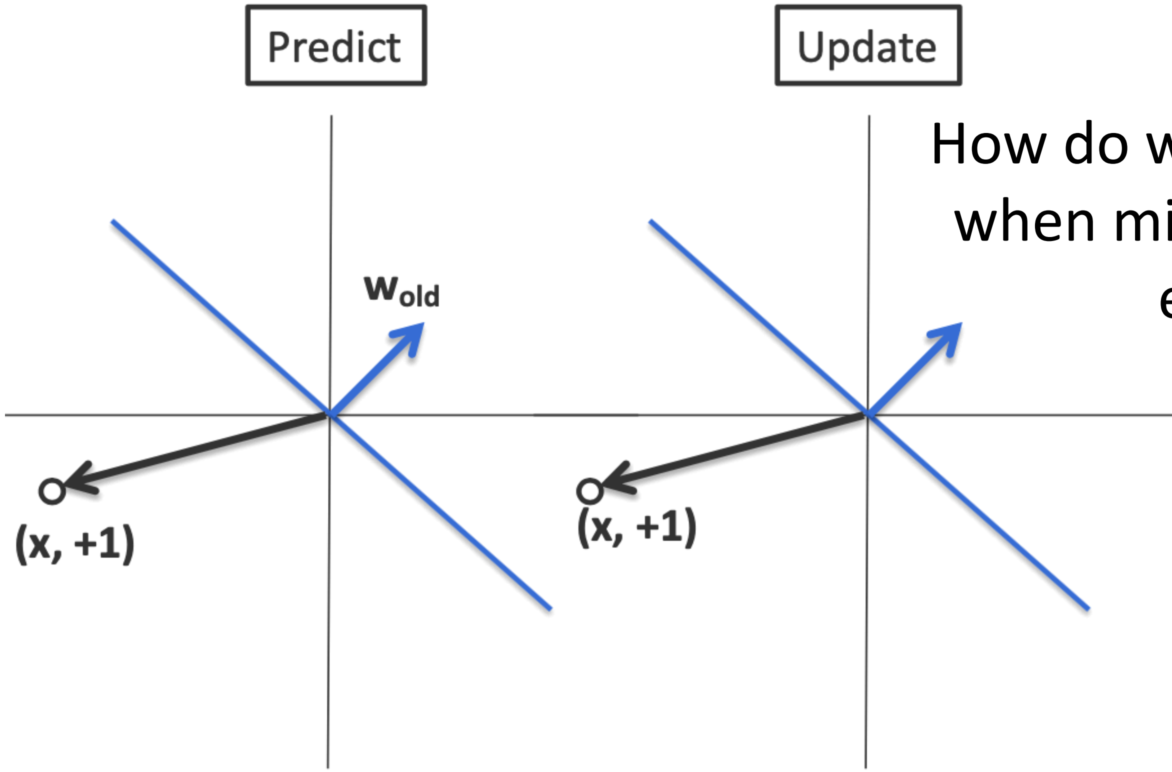
$\mathbf{w}^T\mathbf{x} = |\,\mathbf{w}\,|\,|\,\mathbf{x}\,|\,cos\theta$ where magnitude of vector is given by $|\,\mathbf{w}\,| = \sqrt{\sum_i w_i^2}$

If angle θ is more than 90 degrees, dot product is negative

Mistake on positive example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Geometry of the perceptron update

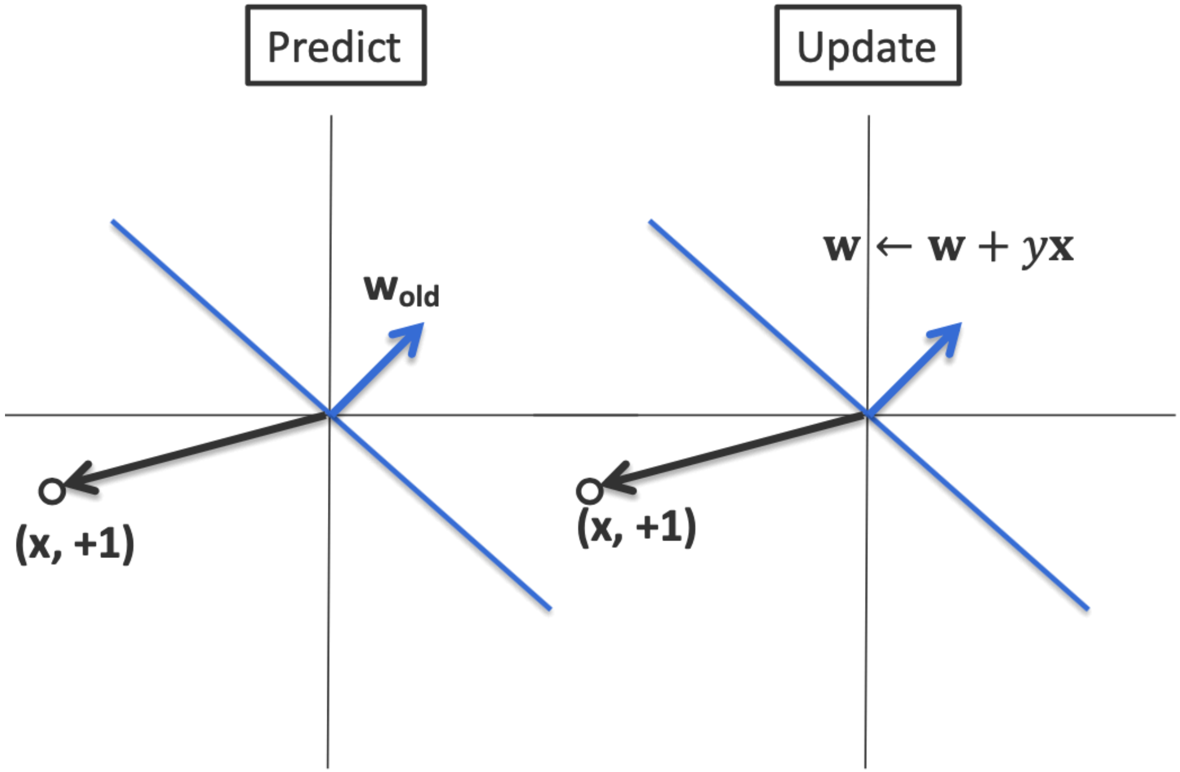


How do we update weights when mistake on positive example?

Mistake on positive example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

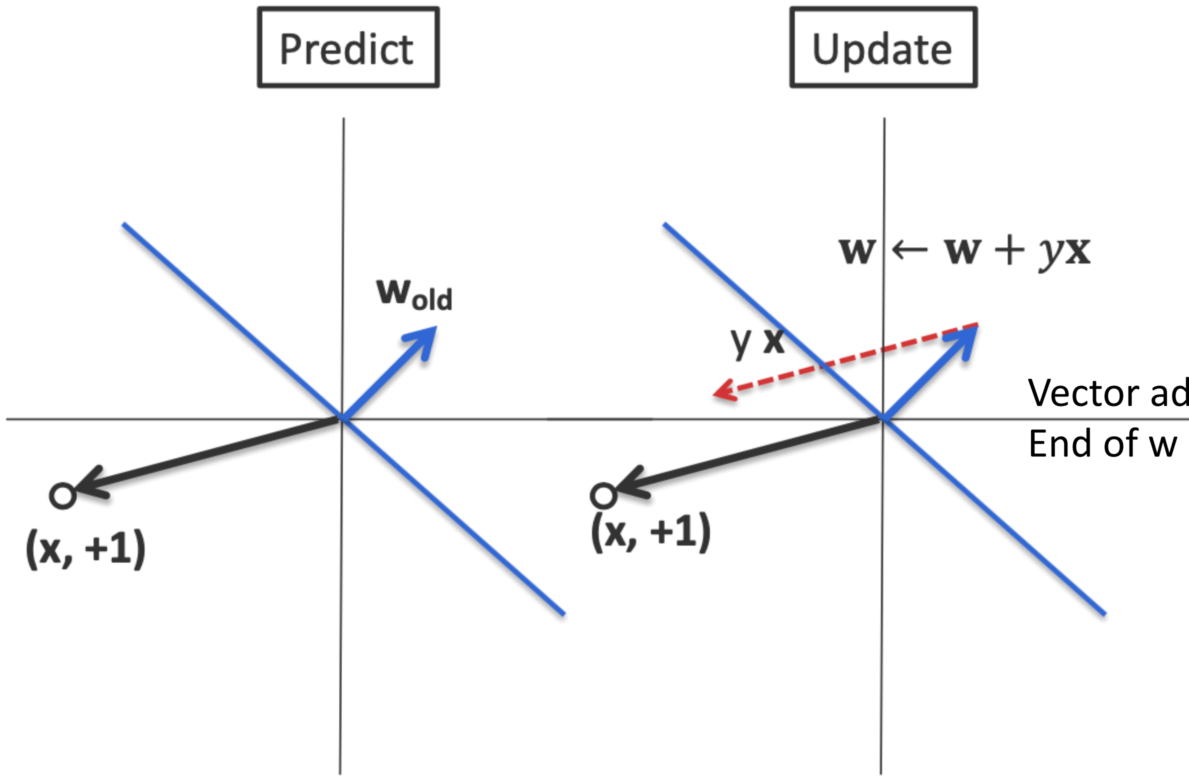
Geometry of the perceptron update



Mistake on positive example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Geometry of the perceptron update

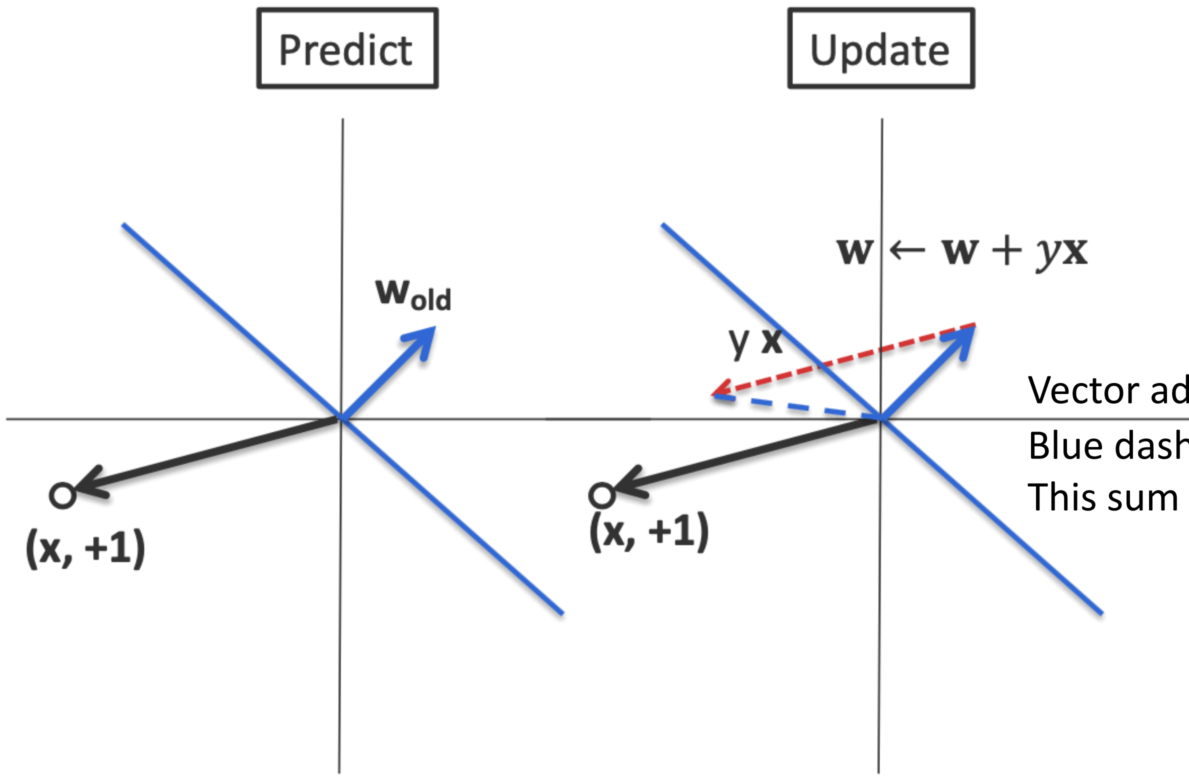


Vector addition:
End of w arrow is start of x arrow

Mistake on positive example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Geometry of the perceptron update

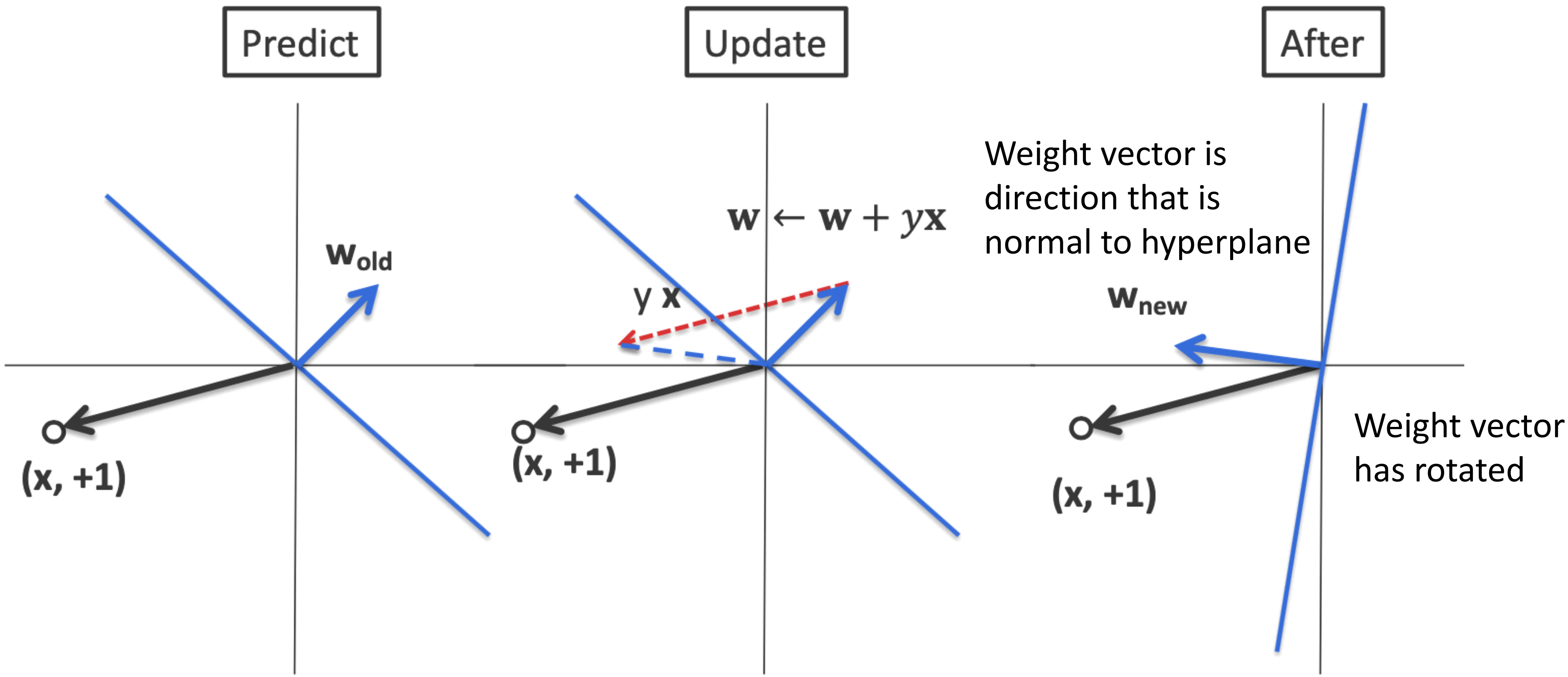


Vector addition:
Blue dashed line is sum, $\mathbf{w} + \mathbf{x}$.
This sum is the new weight vector

Mistake on positive example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

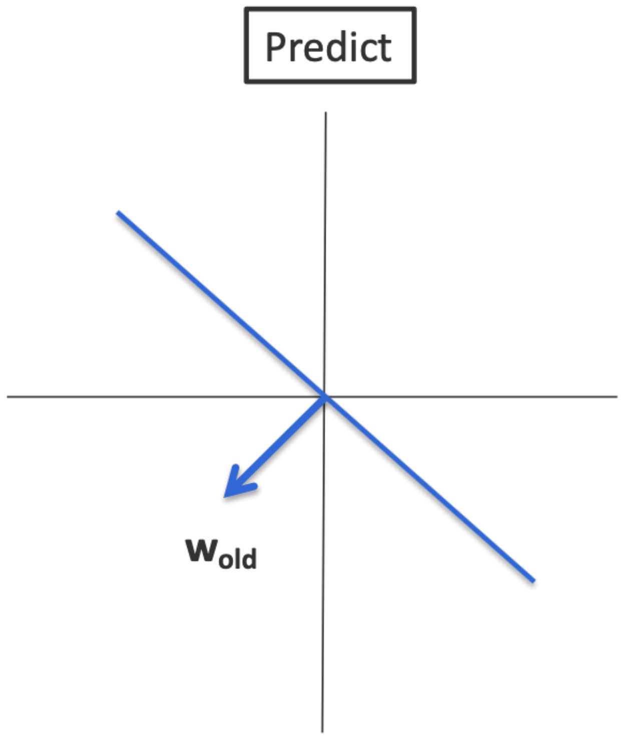
Geometry of the perceptron update



Mistake on positive example

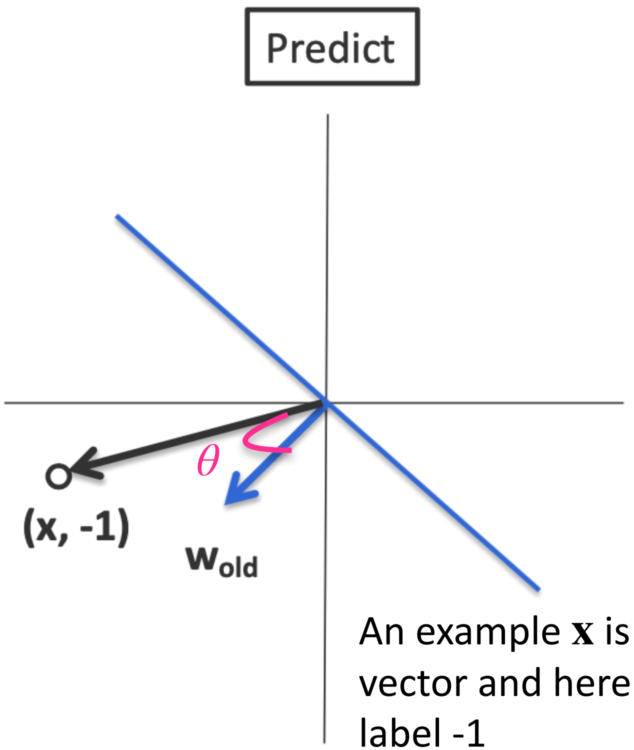
Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Geometry of the perceptron update



Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\,\mathbf{x}_i$

Geometry of the perceptron update



An example $\mathbf{x}$ is also a vector and here has label -1

Geometric interpretation of dot product of $\mathbf{w}$ and $\mathbf{x}$

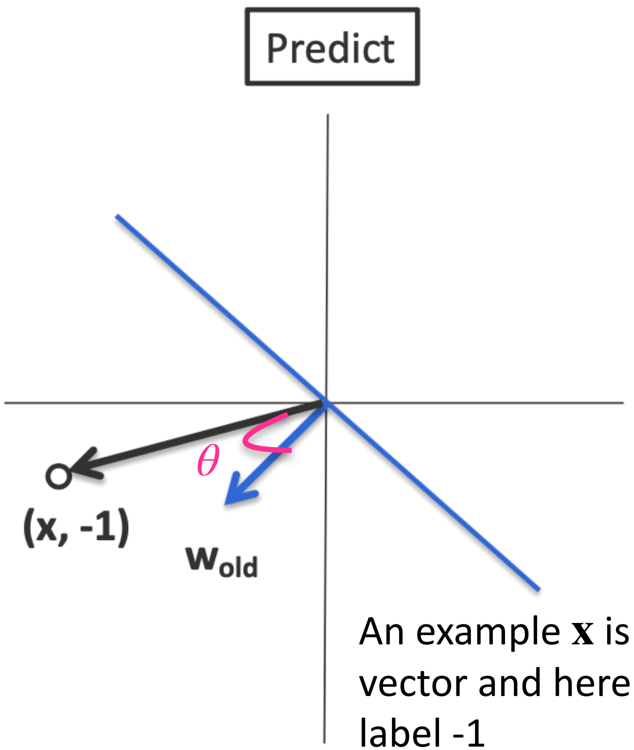
$\mathbf{w}^T\mathbf{x} = |\,\mathbf{w}\,|\,|\,\mathbf{x}\,|\,cos\theta$ where magnitude of vector is given by $|\,\mathbf{w}\,| = \sqrt{\sum_i w_i^2}$

If angle θ is less than 90 degrees, dot product is positive

So what is happening here? What kind of mistake?

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Geometry of the perceptron update



An example $\mathbf{x}$ is also a vector and here has label -1

Geometric interpretation of dot product of $\mathbf{w}$ and $\mathbf{x}$

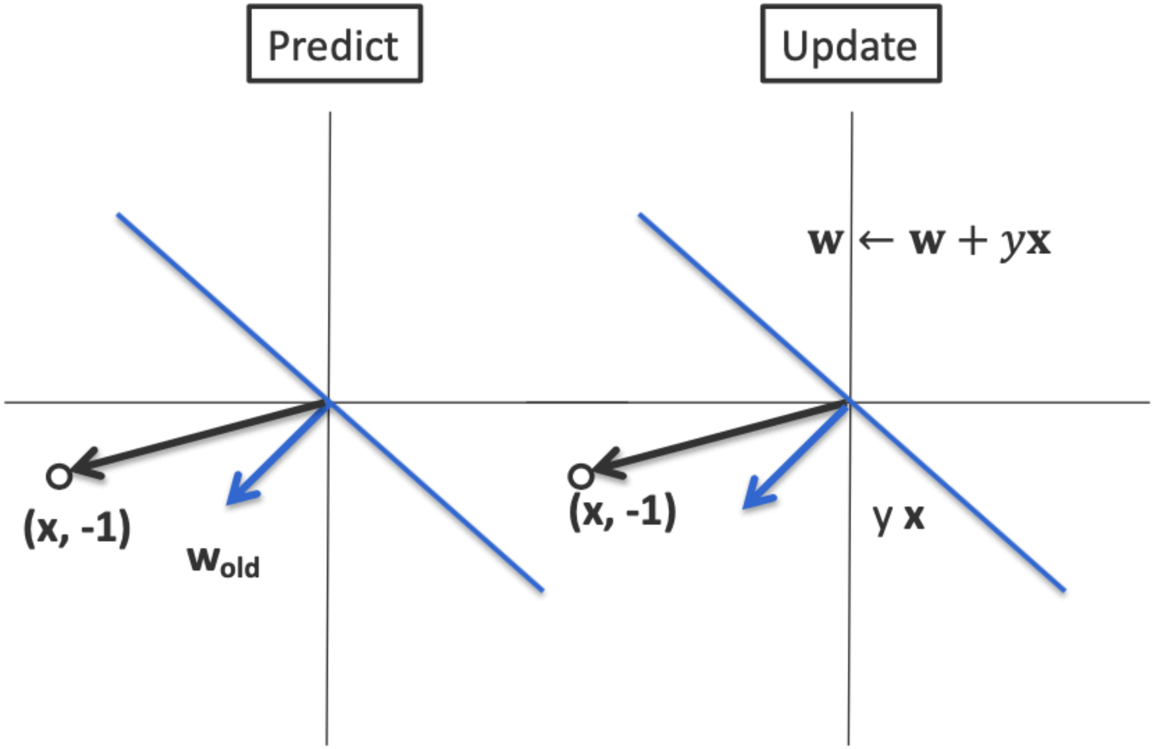
$\mathbf{w}^T\mathbf{x} = |\,\mathbf{w}\,|\,|\,\mathbf{x}\,|\,cos\theta$ where magnitude of vector is given by $|\,\mathbf{w}\,| = \sqrt{\sum_i w_i^2}$

If angle θ is less than 90 degrees, dot product is positive

Mistake on negative example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

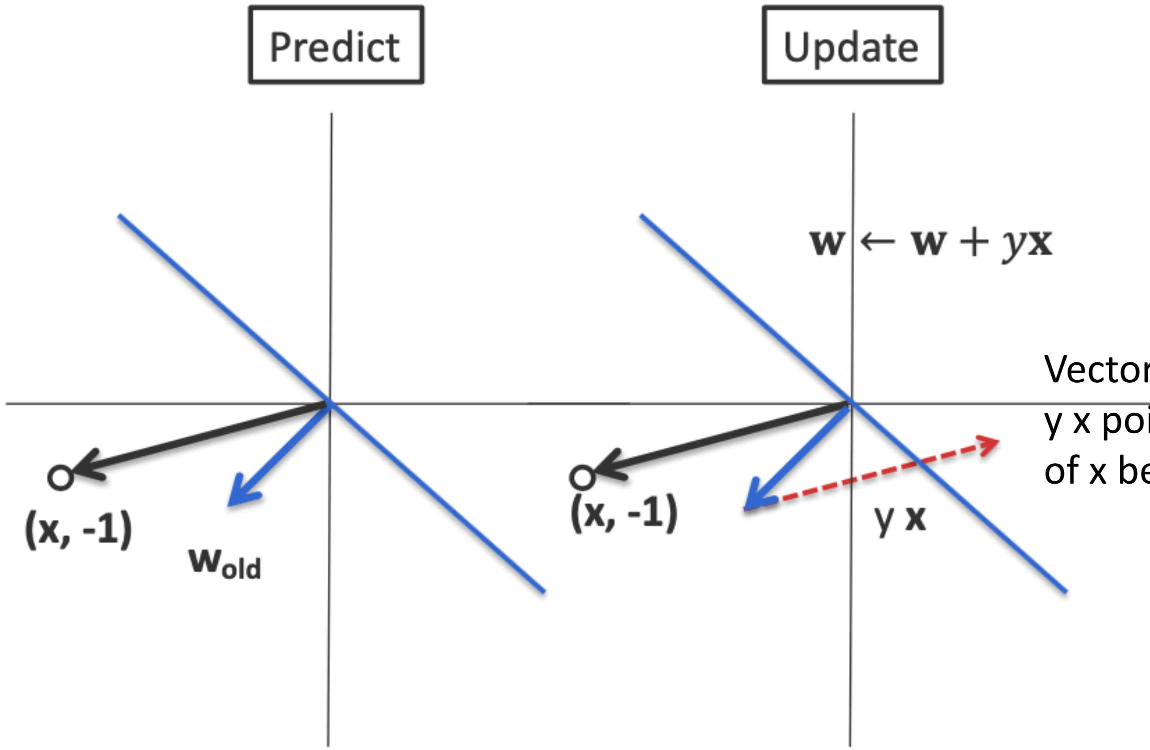
Geometry of the perceptron update



Mistake on negative example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\,\mathbf{x}_i$

Geometry of the perceptron update

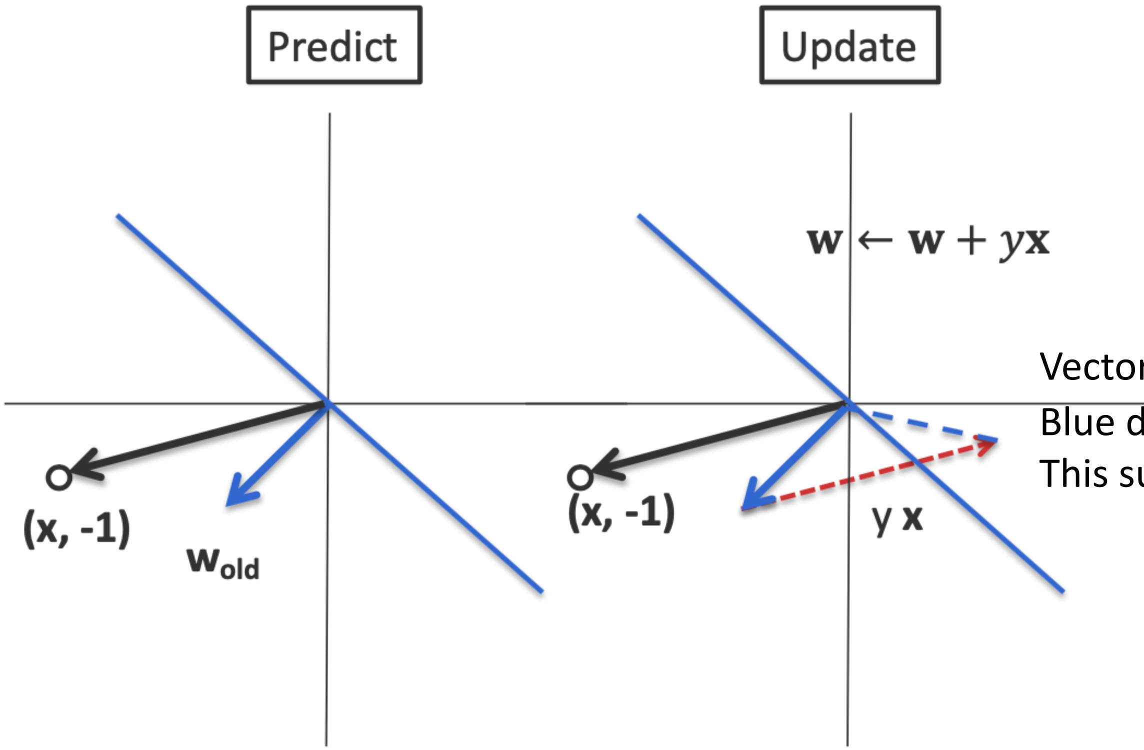


Vector addition:
y x points in opposite direction
of x because negative

Mistake on negative example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Geometry of the perceptron update

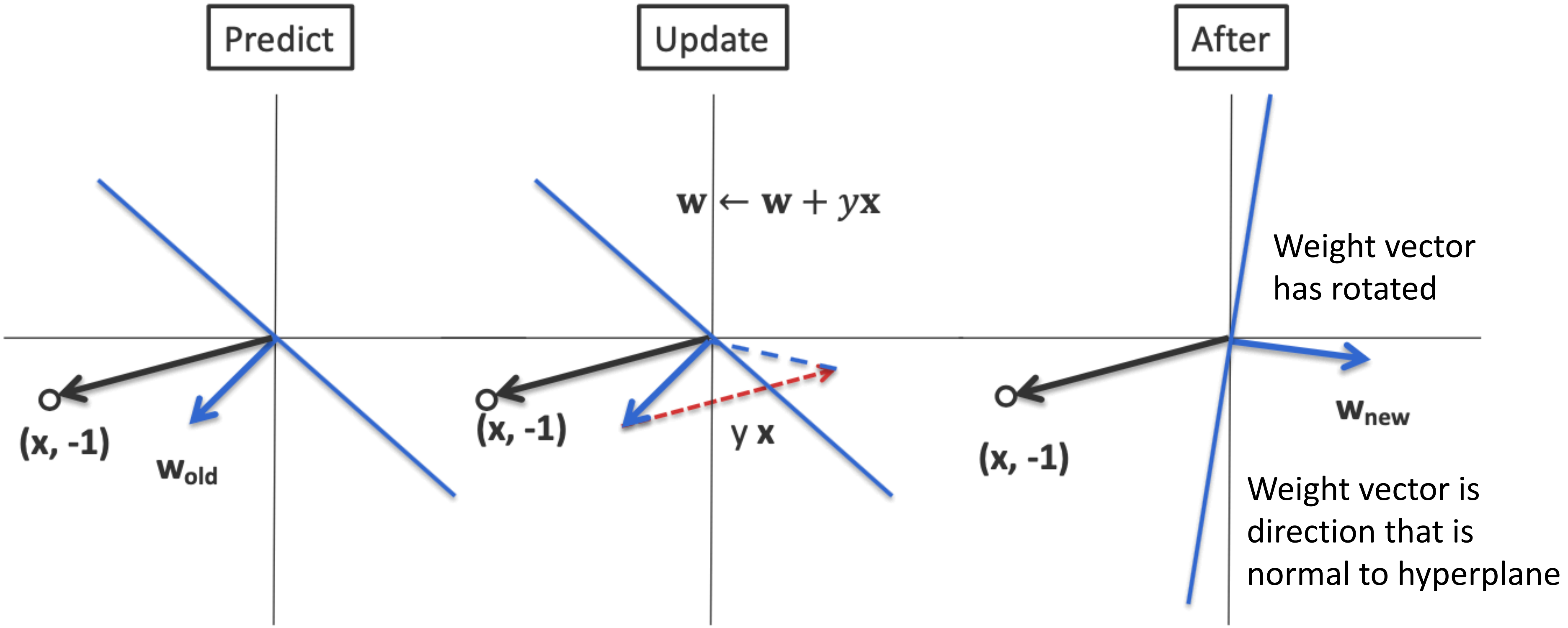


Vector addition:
Blue dashed line is sum, $\mathbf{w} + \mathbf{x}$.
This sum is the new weight vector

Mistake on negative example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Geometry of the perceptron update



Mistake on negative example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Perceptron Convergence

Perceptron Convergence Theorem: If there exist a set of weights that are consistent with the data (i.e., the data is linearly separable), the perceptron learning algorithm will converge. Further, the number of times the perceptron must adjust weights before convergence is upper bounded

Perceptron Convergence

Perceptron Convergence Theorem: If there exist a set of weights that are consistent with the data (i.e., the data is linearly separable), the perceptron learning algorithm will converge. Further, the number of times the perceptron must adjust weights before convergence is upper bounded

Perceptron Cycling Theorem: If the training data is ***not*** linearly separable the perceptron learning algorithm will eventually repeat the same set of weights and therefore enter an infinite loop

Perceptron

PRACTICAL USE AND VARIANTS

Practical use

Randomize order of examples

- Avoid presenting examples in fixed order: re-permute examples every iteration
- Improves convergence speed

Practical use

Randomize order of examples

- Avoid presenting examples in fixed order: re-permute examples every iteration
- Improves convergence speed

Number of iterations to convergence

- Will converge if possible to converge, but how many iterations?
- While the convergence theorem gives a bound, you will not get an exact number
- Number of iterations depends on how much distance between hyperplane and nearest point

The perceptron algorithm

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^{d+1}, y_i \in \{-1,1\}$

1. Initialize $\mathbf{w_0} = 0 \in \Re^{d+1}$
2. For each training example $(\mathbf{x}_i, y_i)$:
   - Predict $y' = \text{sgn}(\mathbf{w}_t^T \mathbf{x}_i)$
   - if $y' \neq y_i$:

     Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$
3. Return final weight vector

The perceptron algorithm

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^{d+1}, y_i \in \{-1,1\}$

1. Initialize $\mathbf{w_0} = 0 \in \Re^{d+1}$
2. For each training example $(\mathbf{x}_i, y_i)$:
   - Predict $y' = \text{sgn}(\mathbf{w}_t^T \mathbf{x}_i)$
   - if $y' \neq y_i$:

     Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$
3. Return final weight vector

Mistake can be written as $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$

The standard algorithm

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^{d+1}$, $y_i \in \{-1, 1\}$

1. Initialize $\mathbf{w} = 0 \in \Re^{d+1}$
2. For epoch in $1 \ldots T$:
   - Shuffle the data
   - For each training example $(\mathbf{x}_i, y_i)$:

     if $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$: update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$
3. Return $\mathbf{w}$

The standard algorithm

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^{d+1}, y_i \in \{-1,1\}$

1. Initialize $\mathbf{w} = 0 \in \Re^{d+1}$
2. For epoch in $1 \ldots T$: → T is a hyper-parameter
   - Shuffle the data
   - For each training example $(\mathbf{x}_i, y_i)$:

     if $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$: update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$
3. Return $\mathbf{w}$

Mimic infinite stream of examples by going over data again and again. Each pass over data is called an epoch

The standard algorithm

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^{d+1}, y_i \in \{-1,1\}$

1. Initialize $\mathbf{w} = 0 \in \Re^{d+1}$
2. For epoch in $1 \dots T$:
   - Shuffle the data $\longrightarrow$ Every epoch go through examples in different random order
   - For each training example $(\mathbf{x}_i, y_i)$:

     if $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$: update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$
3. Return $\mathbf{w}$

The standard algorithm

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^{d+1}, y_i \in \{-1,1\}$

1. Initialize $\mathbf{w} = 0 \in \Re^{d+1}$
2. For epoch in $1 \ldots T$:
   - Shuffle the data
   - For each training example $(\mathbf{x}_i, y_i)$:

     if $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$: update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$
3. Return $\mathbf{w}$

Prediction on a new example with features $\mathbf{x}$: $sgn(\mathbf{w}^T \mathbf{x})$

Learning rate

Recall the update rule

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r(y_i \mathbf{x}_i)$$

r is called the learning rate or step size

- When you update w_j to be more positive or negative, this controls the size of the change you make (or, how large a "step" you take)

Learning rate

Recall the update rule

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r(y_i \mathbf{x}_i)$$

r is called the learning rate or step size

- When you update w_j to be more positive or negative, this controls the size of the change you make (or, how large a "step" you take)

How to choose the step size?

- If r is too small, the algorithm will be slow because the updates won't make much progress
- If r is too large, the algorithm will be slow because the updates will "overshoot" and may cause previous correct classification to become incorrect

Learning rate

Perceptron often just uses $r = 1$

- When we see gradient descent, setting learning rate will be more important

Learning rate

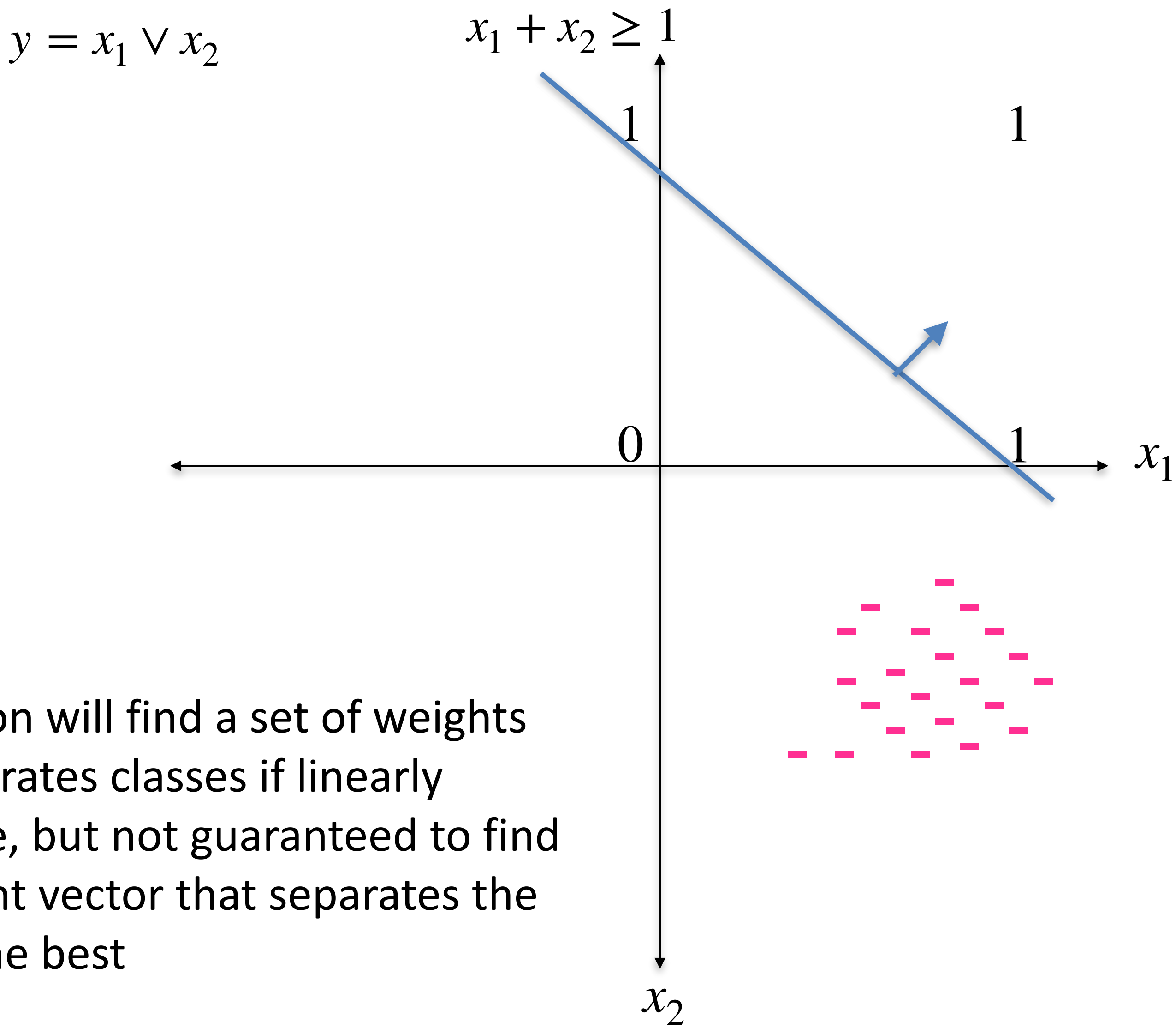
Perceptron often just uses $r = 1$

- When we see gradient descent, setting learning rate will be more important

Why?

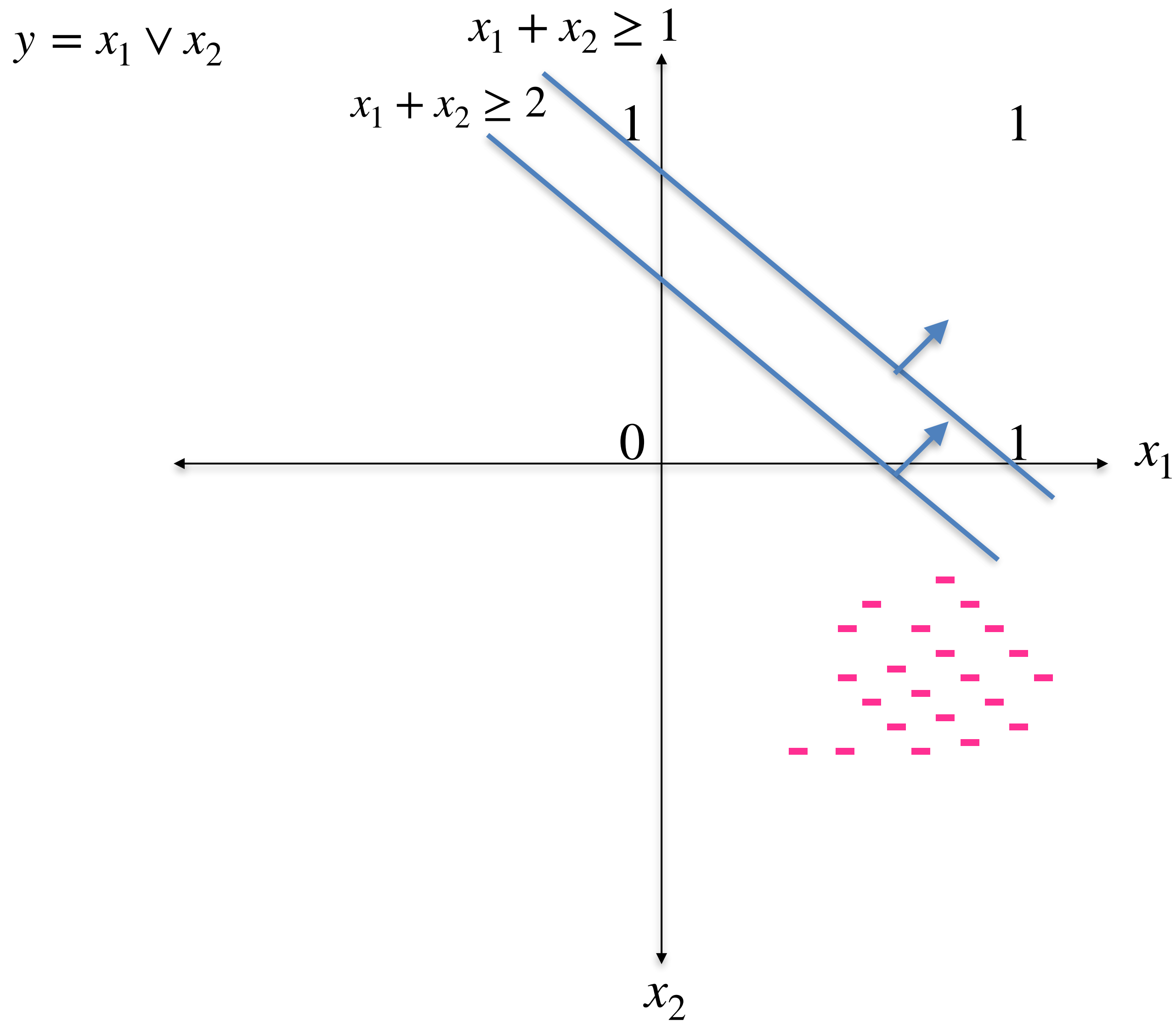
- Learning rate in case of perceptron just scales the weights and hence the dot product $\mathbf{w}^T\mathbf{x}$, but we only care about the sign
- Same number of mistakes will be made regardless of learning rate, and we know that some number of mistakes must be made

How do you know which line is best?

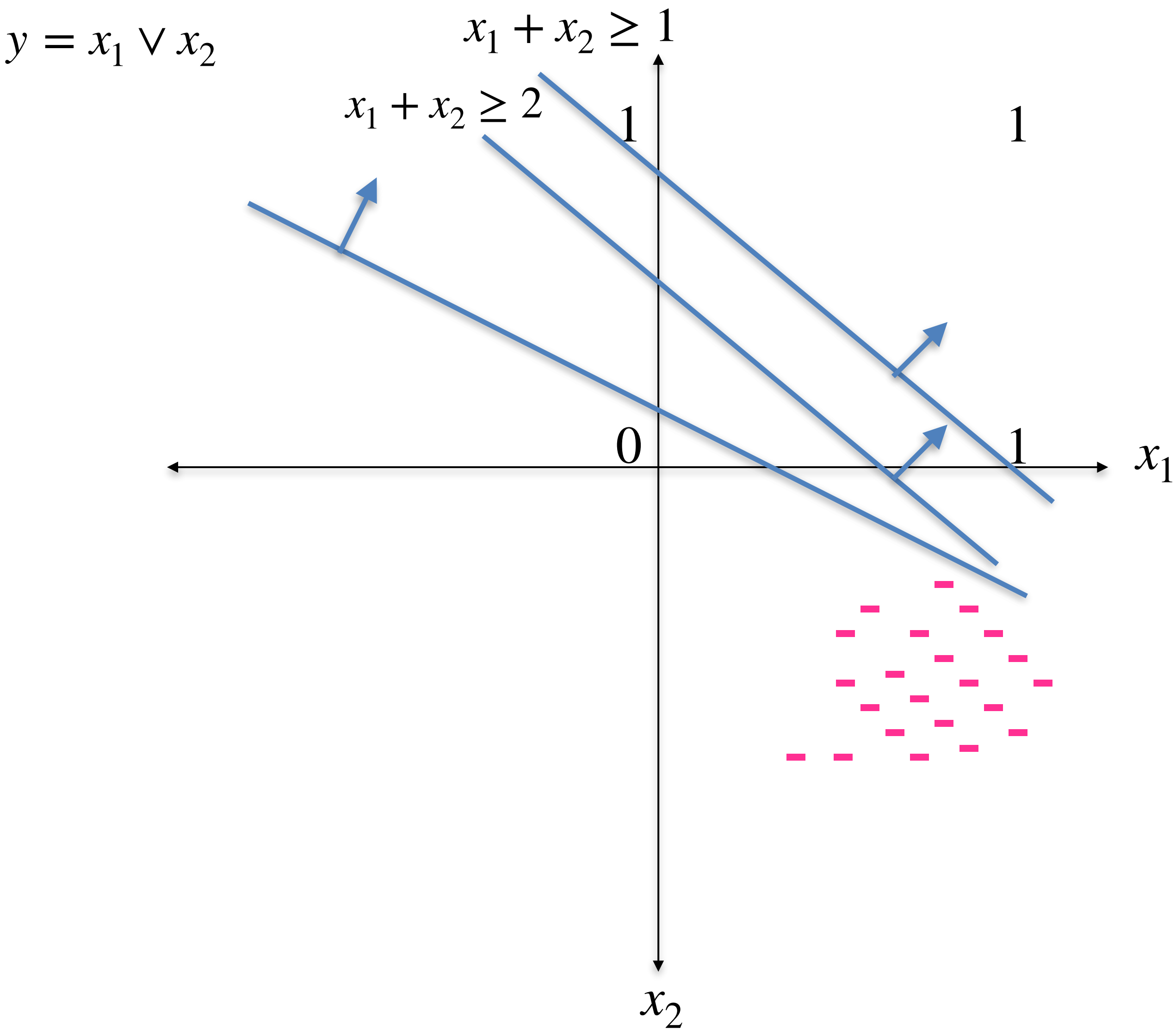


Perceptron will find a set of weights that separates classes if linearly separable, but not guaranteed to find the weight vector that separates the classes the best

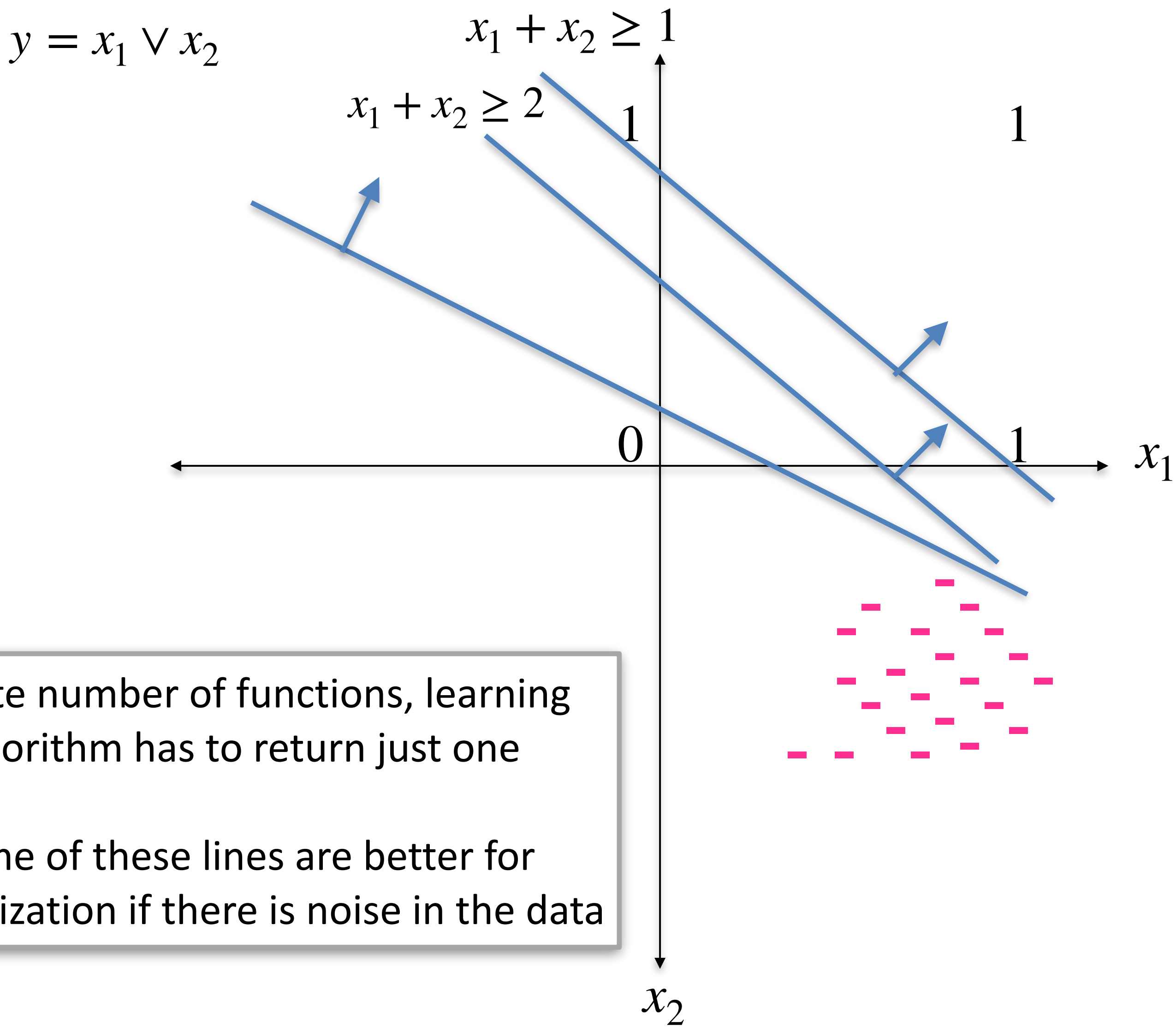
How do you know which line is best?



How do you know which line is best?



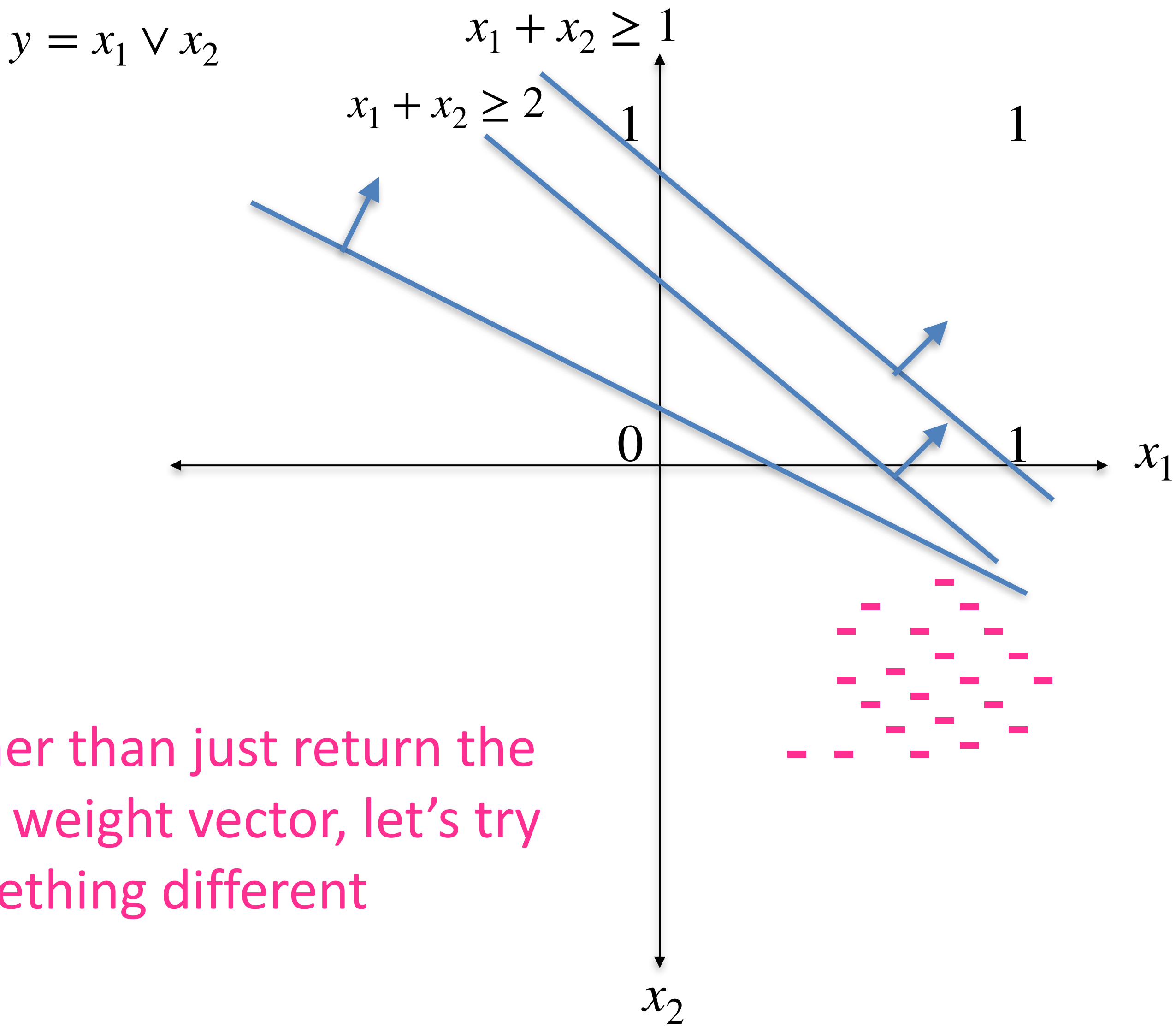
How do you know which line is best?



Infinite number of functions, learning algorithm has to return just one

Some of these lines are better for generalization if there is noise in the data

How do you know which line is best?



Rather than just return the final weight vector, let's try something different

Perceptron variants

Voted perceptron

- Remember every weight vector in your sequence of updates
- At final prediction time, each weight vector gets to vote on the label. The number of votes it gets is the number of iterations it survived before being updated

Return sequence of (weights, # of examples survived)

Every one of those weight vectors votes on final prediction, gets as many votes as # of examples survived

What's the problem?

Perceptron variants

Voted perceptron

- Remember every weight vector in your sequence of updates
- At final prediction time, each weight vector gets to vote on the label. The number of votes it gets is the number of iterations it survived before being updated

Return sequence of (weights, # of examples survived)

Every one of those weight vectors votes on final prediction, gets as many votes as # of examples survived

What's the problem? Too many things to remember. What if 1 million features so weight vectors of 1 million?

Perceptron variants

Voted perceptron

- Remember every weight vector in your sequence of updates
- At final prediction time, each weight vector gets to vote on the label. The number of votes it gets is the number of iterations it survived before being updated
- Comes with strong theoretical guarantees about generalization, impractical because of storage issues

Averaged perceptron

- Instead of using all weight vectors, use the average weight vector (i.e., longer surviving weight vectors get more say)
- More practical alternative and widely used

Perceptron variants

Voted perceptron

- Remember every weight vector in your sequence of updates
- At final prediction time, each weight vector gets to vote on the label. The number of votes it gets is the number of iterations it survived before being updated
- Comes with strong theoretical guarantees about generalization, impractical because of storage issues

Averaged perceptron

- Instead of using all weight vectors, use the average weight vector (i.e., longer surviving weight vectors get more say)
- More practical alternative and widely used

Weight vector that survives longer should dominate the average

Averaged perceptron

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \mathfrak{R}^{d+1}, y_i \in \{-1,1\}$

1. Initialize $\mathbf{w} = 0 \in \mathfrak{R}^{d+1}$ and $\mathbf{a} = \mathbf{0} \in \mathfrak{R}^{\mathbf{d+1}}$

Average vector

2. For epoch in $1 \ldots T$:
   - Shuffle the data
   - For each training example $(\mathbf{x}_i, y_i)$:

     if $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$: update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$

     $\mathbf{a} \rightarrow \mathbf{a} + \mathbf{w}$

Remember every weight vector in your sequence of updates

3. Return $\mathbf{a}$

Averaged perceptron

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^{d+1}, y_i \in \{-1,1\}$

1. Initialize $\mathbf{w} = 0 \in \Re^{d+1}$ and $\mathbf{a} = \mathbf{0} \in \Re^{\mathbf{d+1}}$

   Average vector

2. For epoch in $1 \ldots T$:
   - Shuffle the data
   - For each training example $(\mathbf{x}_i, y_i)$:

     if $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$: update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$

     $\mathbf{a} \rightarrow \mathbf{a} + \mathbf{w}$

     Remember every weight vector in your sequence of updates

3. Return $\mathbf{a}$

Prediction on a new example with features $\mathbf{x}$: $sgn(\mathbf{a}^T \mathbf{x})$

Averaged perceptron

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^{d+1}, y_i \in \{-1,1\}$

1. Initialize $\mathbf{w} = 0 \in \Re^{d+1}$ and $\mathbf{a} = \mathbf{0} \in \Re^{\mathbf{d+1}}$
2. For epoch in $1 \ldots T$:
   - Shuffle the data
   - For each training example $(\mathbf{x}_i, y_i)$:

     if $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$: update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$

     $\mathbf{a} \rightarrow \mathbf{a} + \mathbf{w}$
3. Return $\mathbf{a}$

This is the simplest version of the averaged perceptron

Extra vector addition step: there are easy programming tricks to make sure that $\mathbf{a}$ is also updated only when there is an error

Averaged perceptron

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^{d+1}, y_i \in \{-1,1\}$

1. Initialize $\mathbf{w} = 0 \in \Re^{d+1}$ and $\mathbf{a} = \mathbf{0} \in \Re^{\mathbf{d+1}}$
2. For epoch in $1 \ldots T$:
   - Shuffle the data
   - For each training example $(\mathbf{x}_i, y_i)$:

     if $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$: update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r y_i \mathbf{x}_i$

     $\mathbf{a} \rightarrow \mathbf{a} + \mathbf{w}$
3. Return $\mathbf{a}$

If you want to use the Perceptron algorithm, use averaging

Margin perceptron

Perceptron makes updates only when the prediction is incorrect

$$y_i \mathbf{w}^T \mathbf{x}_i \leq 0$$

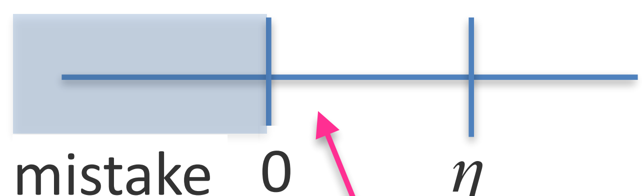
Margin perceptron

Perceptron makes updates only when the prediction is incorrect

$$y_i \mathbf{w}^T \mathbf{x}_i \leq 0$$

What if the prediction is close to being incorrect? Pick a small positive η and update when

$$y_i \mathbf{w}^T \mathbf{x}_i \leq \eta$$



If the current $\mathbf{w}$ gives labels very close to hyperplane that is bad even if correct

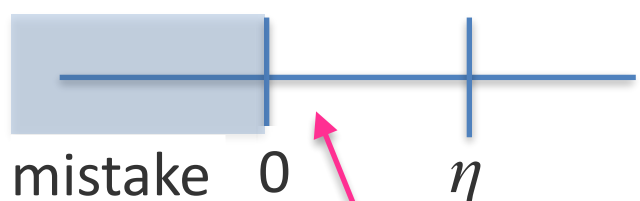
Margin perceptron

Perceptron makes updates only when the prediction is incorrect

$$y_i \mathbf{w}^T \mathbf{x}_i \leq 0$$

What if the prediction is close to being incorrect? Pick a small positive η and update when

$$y_i \mathbf{w}^T \mathbf{x}_i \leq \eta$$

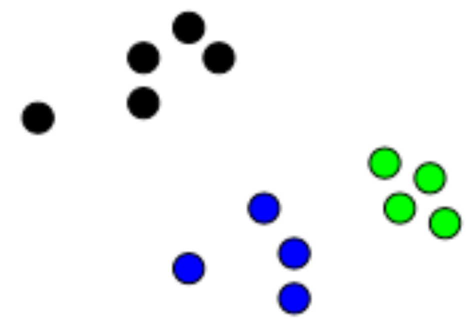


If the current $\mathbf{w}$ gives labels very close to hyperplane that is bad even if correct

Can generalize better, but need to choose η

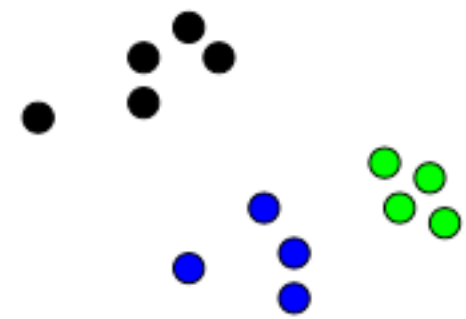
Multi-class perceptron

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^{d+1}, y_i \in \{1,2,3,\ldots,k\}$



Multi-class perceptron

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^{d+1}, y_i \in \{1,2,3,\ldots,k\}$

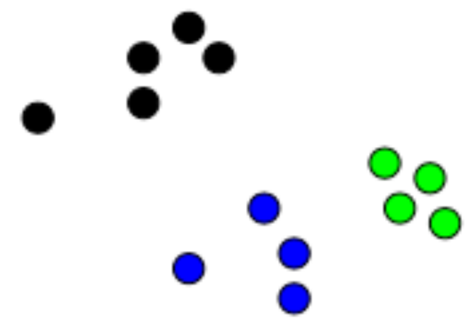


One approach: reduce multi-class problem to binary problems

3 min: How might you do that?

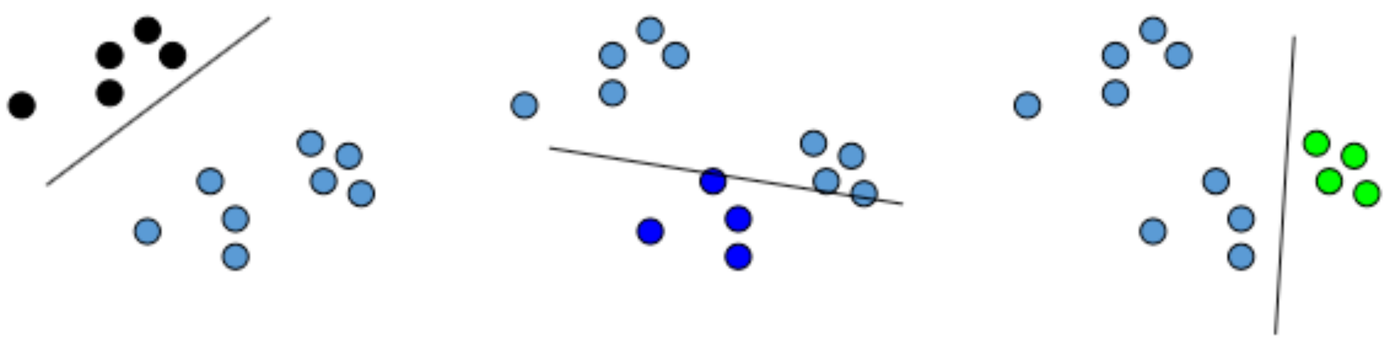
Multi-class perceptron

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^{d+1}, y_i \in \{1,2,3,\ldots,k\}$



One approach: reduce multi-class problem to binary problems

Ideally: only correct label has positive score



$w_{black}^T x > 0$ $w_{blue}^T x > 0$ $w_{green}^T x > 0$