
# 10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

# Perceptron

Matt Gormley
Lecture 6
Feb. 17, 2021

# Reminders

- Homework 2: Decision Trees
  - Out: Wed, Feb. 10
  - Due: Mon, Feb. 22 at 11:59pm
- Homework 3: KNN, Perceptron, Lin.Reg.
  - Out: Mon, Feb. 22
  - Due: Mon, Mar. 01 at 11:59pm

# THE PERCEPTRON ALGORITHM

# Perceptron: History

Imagine you are trying to build a new machine learning technique… your name is Frank Rosenblatt… and the year is 1957

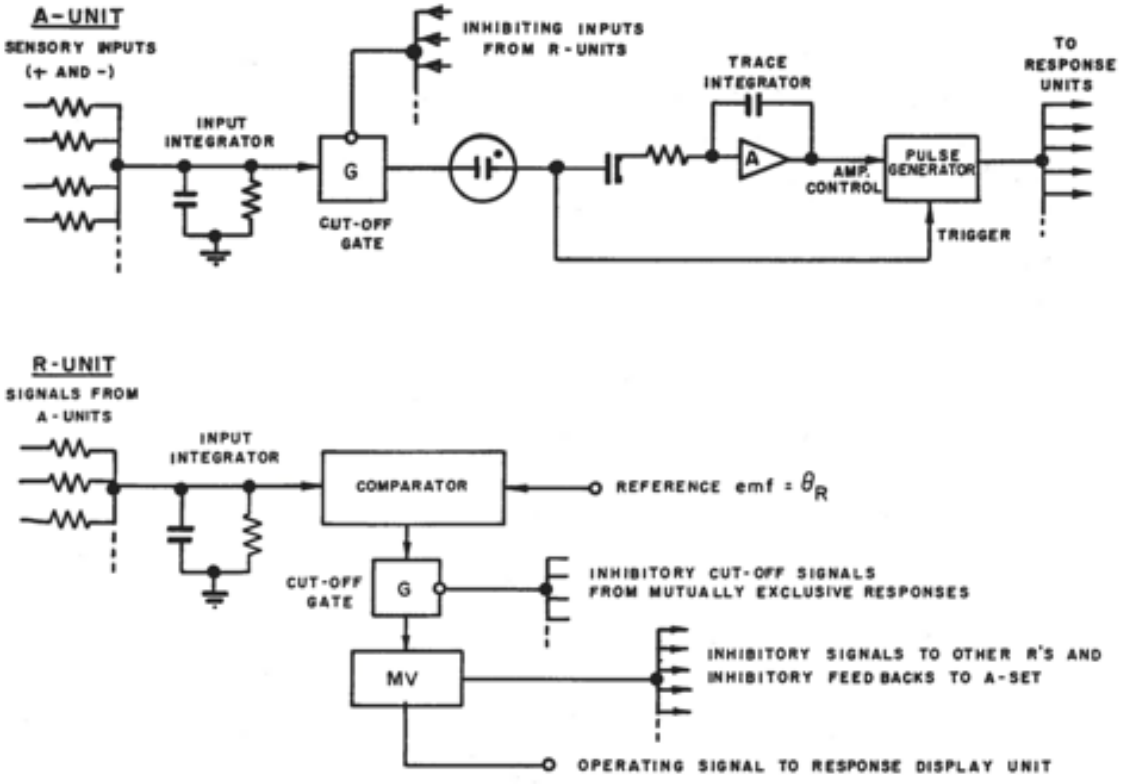
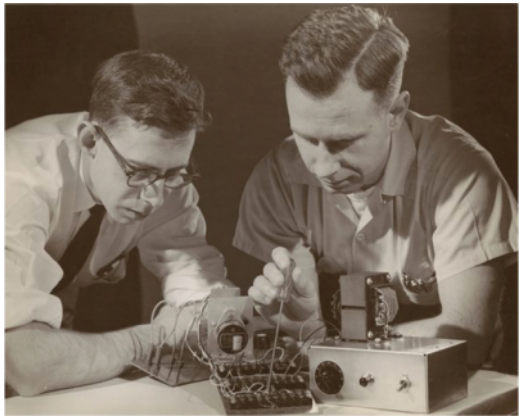
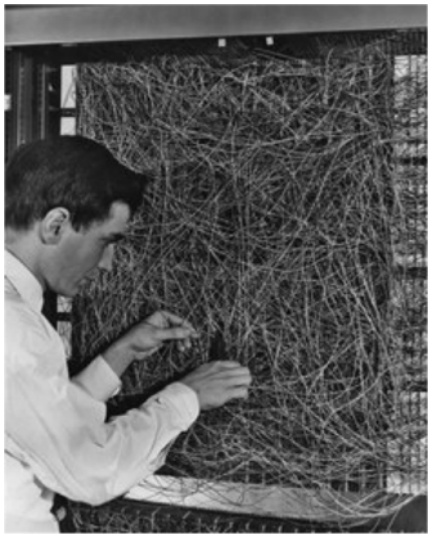



FIGURE 5
DESIGN OF TYPICAL UNITS

# Perceptron: History

Imagine you are trying to build a new machine learning technique… your name is Frank Rosenblatt… and the year is 1957

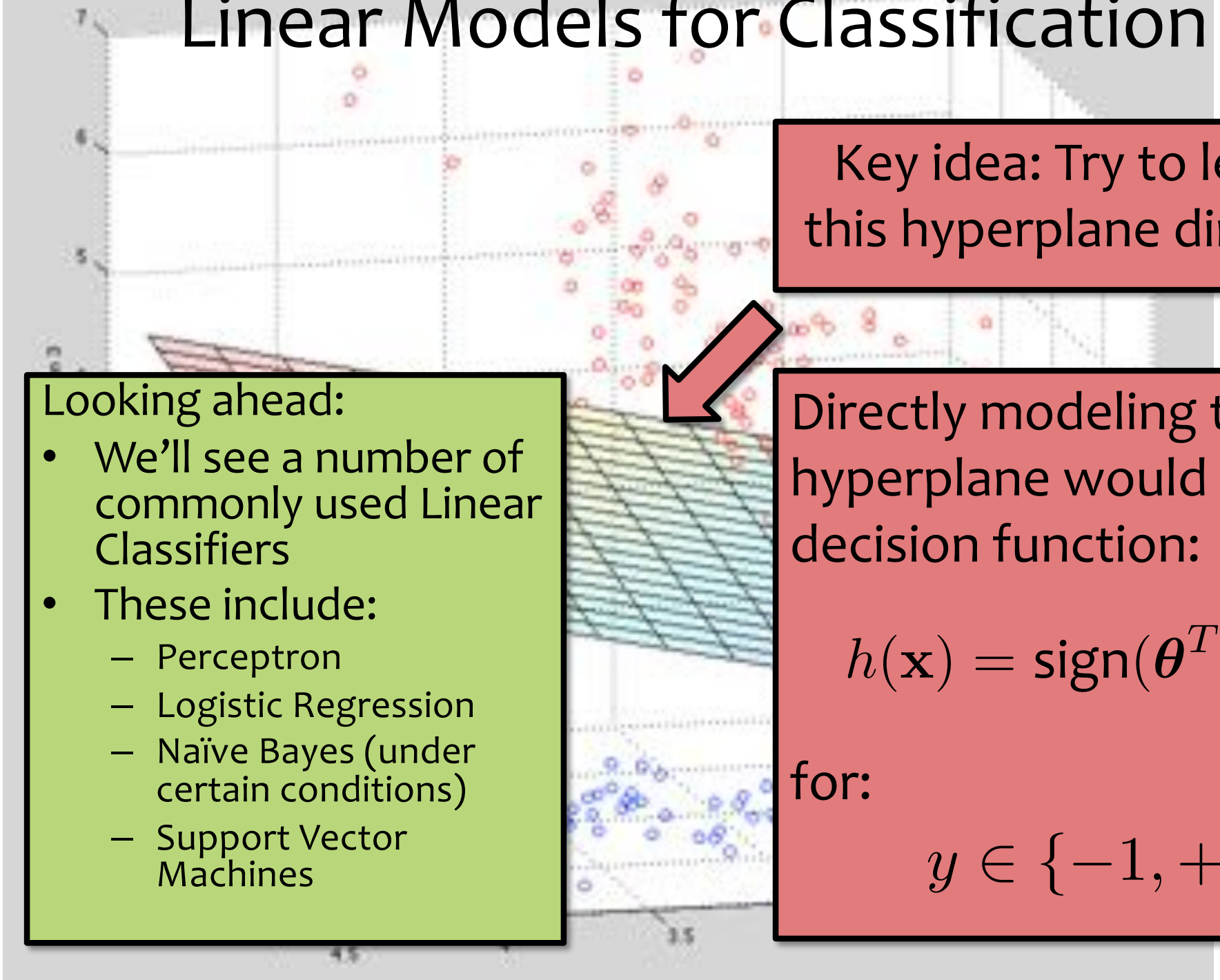




*The New Yorker*, December 6, 1958 P. 44

Talk story about the perceptron, a new electronic brain which hasn't been built, but which has been successfully simulated on the I.B.M. 704. Talk with Dr. Frank Rosenblatt, of the Cornell Aeronautical Laboratory, who is one of the two men who developed the prodigy; the other man is Dr. Marshall C. Yovits, of the Office of Naval Research, in Washington. Dr. Rosenblatt defined the perceptron as the first non-biological object which will achieve an organization o its external environment in a meaningful way. It interacts with its environment, forming concepts that have not been made ready for it by a human agent. If a triangle is held up, the perceptron's eye picks up the image & conveys it along a random succession of lines to the response units, where the image is registered. It can tell the difference betw. a cat and a dog, although it wouldn't be able to tell whether the dog was to theleft or right of the cat. Right now it is of no practical use, Dr. Rosenblatt conceded, but he said that one day it might be useful to send one into outer space to take in impressions for us.

# Linear Models for Classification



Key idea: Try to learn this hyperplane directly

Looking ahead:

- We'll see a number of commonly used Linear Classifiers
- These include:
  - Perceptron
  - Logistic Regression
  - Naïve Bayes (under certain conditions)
  - Support Vector Machines

Directly modeling the hyperplane would use a decision function:

$$h(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$

for:

$$y \in \{-1, +1\}$$

# Geometry

**In-Class Exercise**

Draw a picture of the region corresponding to:

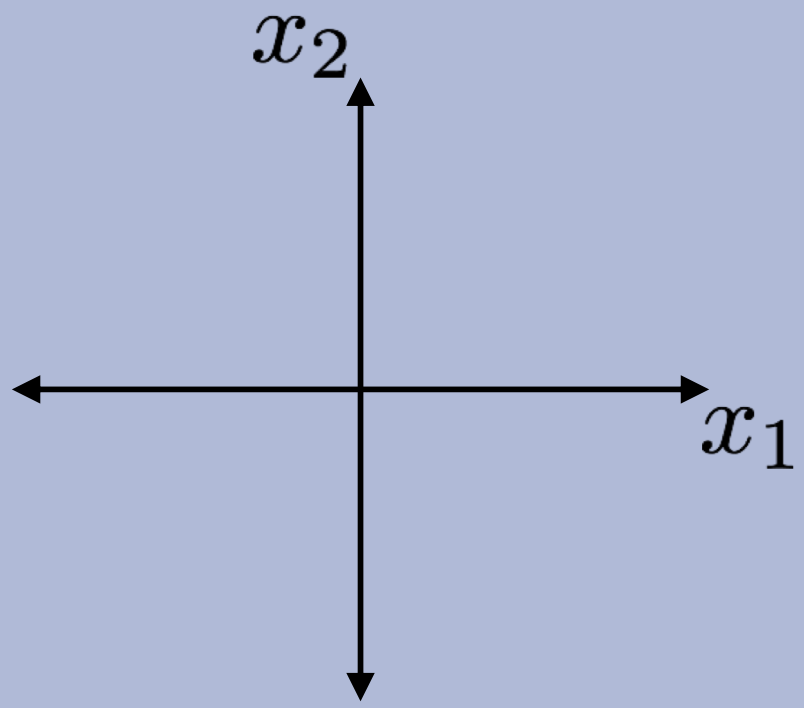
$$w_1 x_1 + w_2 x_2 + b > 0$$

where $w_1 = 2, w_2 = 3, b = 6$

Draw the vector

$\boldsymbol{w} = [w_1, w_2]$

**Answer Here:**



# Visualizing Dot-Products

*Whiteboard:*

- definition of dot product
- definition of L2 norm
- definition of orthogonality

# Vector Projection

**Question:**

*Which of the following is the projection of a vector **a** onto a vector **b**?*

A. $\frac{\mathbf{a}^T\mathbf{b}}{\mathbf{b}}\mathbf{a}$

B. $\frac{\mathbf{a}\cdot\mathbf{b}}{\mathbf{a}^T\mathbf{b}}$

C. $\frac{(\mathbf{a}^T\mathbf{b})}{||\mathbf{b}||_2}\mathbf{b}$

D. $\frac{(\mathbf{a}\cdot\mathbf{b})}{||\mathbf{b}||_2}\mathbf{b}$

E. $\frac{(\mathbf{a}^T\mathbf{b})}{||\mathbf{b}||_2^2}\mathbf{b}$

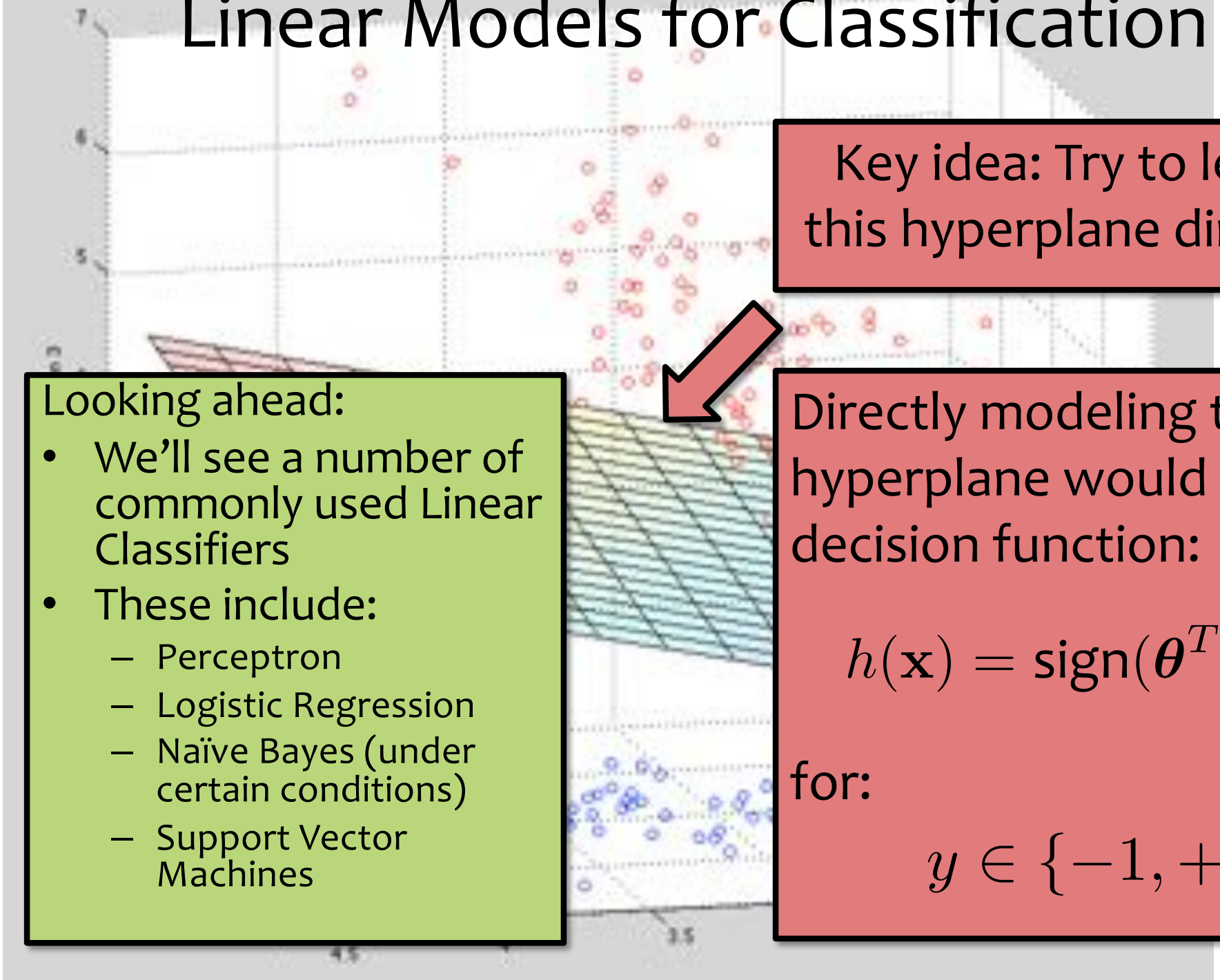
F. $\frac{(\mathbf{a}^T\mathbf{b})^2}{||\mathbf{b}||_2}\mathbf{b}$

# Visualizing Dot-Products

*Whiteboard:*

- vector projection
- hyperplane definition
- half-space definitions

# Linear Models for Classification



Key idea: Try to learn this hyperplane directly

Looking ahead:

- We'll see a number of commonly used Linear Classifiers
- These include:
  - Perceptron
  - Logistic Regression
  - Naïve Bayes (under certain conditions)
  - Support Vector Machines

Directly modeling the hyperplane would use a decision function:

$$h(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$

for:

$$y \in \{-1, +1\}$$

# Online vs. Batch Learning

**Batch Learning**

Learn from all the examples at once

**Online Learning**

Gradually learn as each example is received

# Online Learning

**Examples**

1. **Stock market** prediction (what will the value of Alphabet Inc. be tomorrow?)
2. **Email** classification (distribution of both spam and regular mail changes over time, but the target function stays fixed - last year's spam still looks like spam)
3. **Recommendation** systems. Examples: recommending movies; predicting whether a user will be interested in a new news article
4. **Ad placement** in a new market



Slide adapted from Nina Balcan

# Online Learning

**For** i = 1, 2, 3, … **:**

- **Receive** an unlabeled instance $\mathbf{x}^{(i)}$
- **Predict** y' = $h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$
- **Receive** true label $y^{(i)}$
- **Suffer loss** if a mistake was made, y' ≠ $y^{(i)}$
- **Update** parameters $\boldsymbol{\theta}$

**Goal:**

- **Minimize** the number of **mistakes**

# Perceptron

*Whiteboard:*

- (Online) Perceptron Algorithm
- Hypothesis class for Perceptron
- 2D Example of Perceptron

# Perceptron Algorithm: Example

Example: $(-1,2)$ − ✗

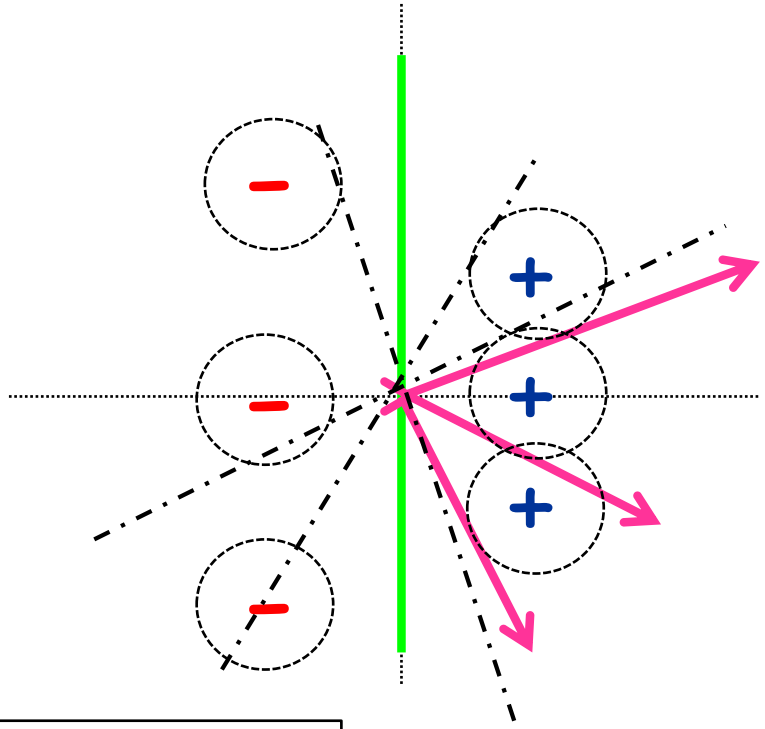
$(1,0)$ + ✓

$(1,1)$ + ✗

$(-1,0)$ − ✓

$(-1,-2)$ − ✗

$(1,-1)$ + ✓



**Perceptron Algorithm: (without the bias term)**

- Set t=1, start with all-zeroes weight vector $w_1$.
- Given example $x$, predict positive iff $w_t \cdot x \geq 0$.
- On a mistake, update as follows:
  - Mistake on positive, update $w_{t+1} \leftarrow w_t + x$
  - Mistake on negative, update $w_{t+1} \leftarrow w_t - x$

$w_1 = (0,0)$

$w_2 = w_1 - (-1,2) = (1,-2)$

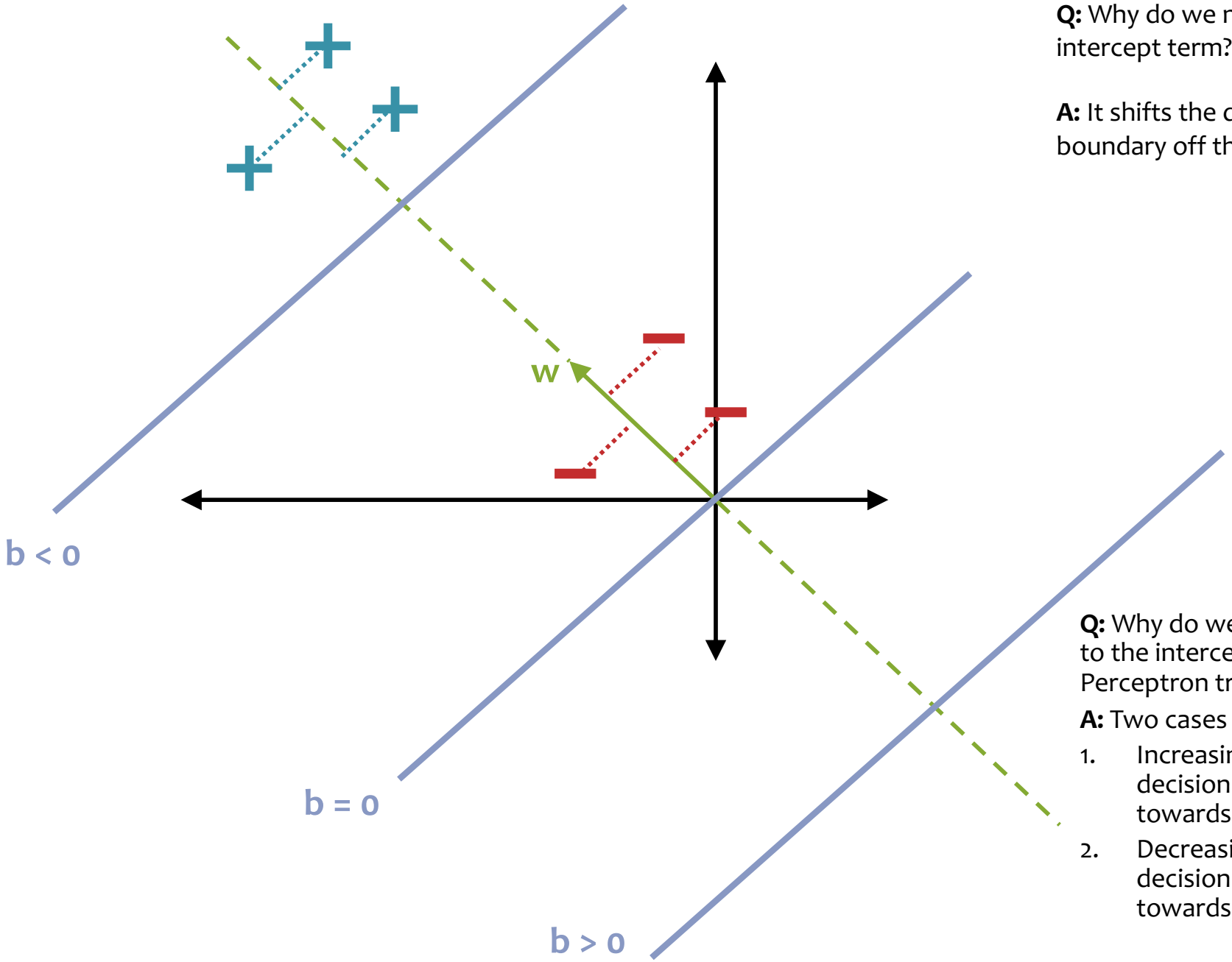
$w_3 = w_2 + (1,1) = (2,-1)$

$w_4 = w_3 - (-1,-2) = (3,1)$

Slide adapted from Nina Balcan

# THE PERCEPTRON ALGORITHM

# Intercept Term




**Q:** Why do we need an intercept term?

**A:** It shifts the decision boundary off the origin

**Q:** Why do we add / subtract 1.0 to the intercept term during Perceptron training?

**A:** Two cases

1. Increasing b shifts the decision boundary towards the negative side
2. Decreasing b shifts the decision boundary towards the positive side

# Perceptron Inductive Bias

1. Decision boundary should be linear
2. Most recent mistakes are most important (and should be corrected)

# Background: Hyperplanes

*Notation Trick*: fold the bias *b* and the weights **w** into a single vector **θ** by prepending a constant to **x** and increasing dimensionality by one to get **x'**!

Hyperplane (Definition 1):

$$\mathcal{H} = \{\mathbf{x} : \mathbf{w}^T\mathbf{x} = b\}$$

Hyperplane (Definition 2):

$$\mathcal{H} = \{\mathbf{x}' : \boldsymbol{\theta}^T\mathbf{x}' = 0 \text{ and } x'_1 = 1\}$$

$$\boldsymbol{\theta} = [b, w_1, \ldots, w_M]^T$$

Half-spaces:

$$\mathcal{H}^+ = \{\mathbf{x} : \boldsymbol{\theta}^T\mathbf{x} > 0 \text{ and } x_1 = 1\}$$

$$\mathcal{H}^- = \{\mathbf{x} : \boldsymbol{\theta}^T\mathbf{x} < 0 \text{ and } x_1 = 1\}$$

# (Online) Perceptron Algorithm

**Data:** Inputs are continuous vectors of length *M*. Outputs are discrete.

$$(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots$$

where $\mathbf{x} \in \mathbb{R}^M$ and $y \in \{+1, -1\}$

**Prediction:** Output determined by hyperplane.

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$

$$\text{sign}(a) = \begin{cases} 1, & \text{if } a \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

Assume $\boldsymbol{\theta} = [b, w_1, \ldots, w_M]^T$ and $x_1 = 1$

**Learning:** Iterative procedure:

- **initialize** parameters to vector of all zeroes
- **while** not converged
  - **receive** next example ($\mathbf{x}^{(i)}$, $y^{(i)}$)
  - **predict** y' = h($\mathbf{x}^{(i)}$)
  - **if** positive mistake: **add** $\mathbf{x}^{(i)}$ to parameters
  - **if** negative mistake: **subtract** $\mathbf{x}^{(i)}$ from parameters

# (Online) Perceptron Algorithm

**Data:** Inputs are continuous vectors of length *M*. Outputs are discrete.

$$(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots$$

where $\mathbf{x} \in \mathbb{R}^M$ and $y \in \{+1, -1\}$

**Prediction:** Output determined by hyperplane.

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$

$$\text{sign}(a) = \begin{cases} 1, & \text{if } a \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

Assume $\boldsymbol{\theta} = [b, w_1, \ldots, w_M]^T$ and $x_1 = 1$

**Learning:**

**Algorithm 1** Perceptron Learning Algorithm (Online)

1: **procedure** PERCEPTRON($\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots\}$)
2: $\quad \boldsymbol{\theta} \leftarrow \mathbf{0}$ ▷ Initialize parameters
3: $\quad$ **for** $i \in \{1, 2, \ldots\}$ **do** ▷ For each example
4: $\quad\quad \hat{y} \leftarrow \text{sign}(\boldsymbol{\theta}^T \mathbf{x}^{(i)})$ ▷ Predict
5: $\quad\quad$ **if** $\hat{y} \neq y^{(i)}$ **then** ▷ If mistake
6: $\quad\quad\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)} \mathbf{x}^{(i)}$ ▷ Update parameters
7: $\quad$ **return** $\boldsymbol{\theta}$

# (Online) Perceptron Algorithm

**Data:** Inputs are continuous vectors of length *M*. Outputs are discrete.

$$(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots$$

where $\mathbf{x} \in \mathbb{R}^M$ and $y \in \{+1, -1\}$

**Prediction:** Output determine

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$

Assume $\boldsymbol{\theta} = [b, w_1, \ldots, w_M]$

*Implementation Trick*: same behavior as our "*add on positive mistake and subtract on negative mistake*" version, because $y^{(i)}$ takes care of the sign

**Learning:**

**Algorithm 1** Perceptron Learning Alg

1: **procedure** PERCEPTRON($\mathcal{D} = \{(\mathbf{x}$
2: $\quad \boldsymbol{\theta} \leftarrow \mathbf{0}$ ▷ Initialize parameters
3: $\quad$ **for** $i \in \{1, 2, \ldots\}$ **do** ▷ For each example
4: $\quad\quad \hat{y} \leftarrow \text{sign}(\boldsymbol{\theta}^T \mathbf{x}^{(i)})$ ▷ Predict
5: $\quad\quad$ **if** $\hat{y} \neq y^{(i)}$ **then** ▷ If mistake
6: $\quad\quad\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)} \mathbf{x}^{(i)}$ ▷ Update parameters
7: $\quad$ **return** $\boldsymbol{\theta}$

# (Batch) Perceptron Algorithm

Learning for Perceptron also works if we have a fixed training dataset, D. We call this the "batch" setting in contrast to the "online" setting that we've discussed so far.

**Algorithm 1** Perceptron Learning Algorithm (Batch)

**procedure** PERCEPTRON($\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$)
  $\boldsymbol{\theta} \leftarrow \mathbf{0}$ ▷ Initialize parameters
  **while** not converged **do**
    **for** $i \in \{1, 2, \ldots, N\}$ **do** ▷ For each example
      $\hat{y} \leftarrow \text{sign}(\boldsymbol{\theta}^T \mathbf{x}^{(i)})$ ▷ Predict
      **if** $\hat{y} \neq y^{(i)}$ **then** ▷ If mistake
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)} \mathbf{x}^{(i)}$ ▷ Update parameters
  **return** $\boldsymbol{\theta}$

# (Batch) Perceptron Algorithm

Learning for Perceptron also works if we have a fixed training dataset, D. We call this the "batch" setting in contrast to the "online" setting that we've discussed so far.

**Discussion:**

The Batch Perceptron Algorithm can be derived in two ways.

1. By extending the online Perceptron algorithm to the batch setting (as mentioned above)
2. By applying **Stochastic Gradient Descent (SGD)** to minimize a so-called **Hinge Loss** on a linear separator

# Extensions of Perceptron

- **Voted Perceptron**
  - generalizes better than (standard) perceptron
  - memory intensive (keeps around every weight vector seen during training, so each one can vote)
- **Averaged Perceptron**
  - empirically similar performance to voted perceptron
  - can be implemented in a memory efficient way (running averages are efficient)
- **Kernel Perceptron**
  - Choose a kernel K(x', x)
  - Apply the **kernel trick** to Perceptron
  - Resulting algorithm is **still very simple**
- **Structured Perceptron**
  - Basic idea can also be applied when **y** ranges over an exponentially large set
  - Mistake bound **does not** depend on the size of that set

# Perceptron Exercises

**Question:**

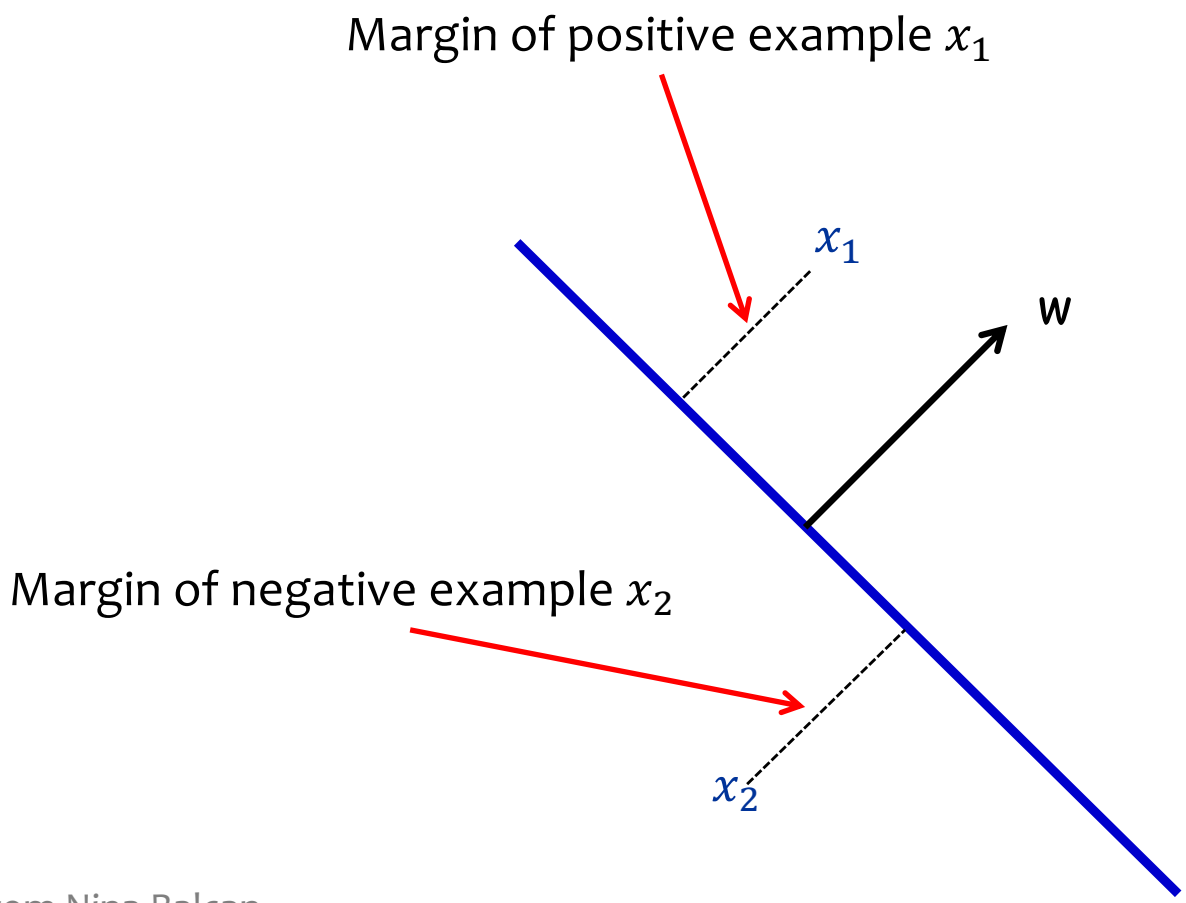
*The parameter vector* ***w*** *learned by the Perceptron algorithm can be* ***written as a linear combination*** *of the feature vectors* $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}$.

A. *True, if you replace "linear" with "polynomial" above*
B. *True, for all datasets*
C. *False, for all datasets*
D. *True, but only for certain datasets*
E. *False, but only for certain datasets*

# ANALYSIS OF PERCEPTRON

# Geometric Margin

**Definition:** The margin of example $x$ w.r.t. a linear sep. $w$ is the distance from $x$ to the plane $w \cdot x = 0$ (or the negative if on wrong side)

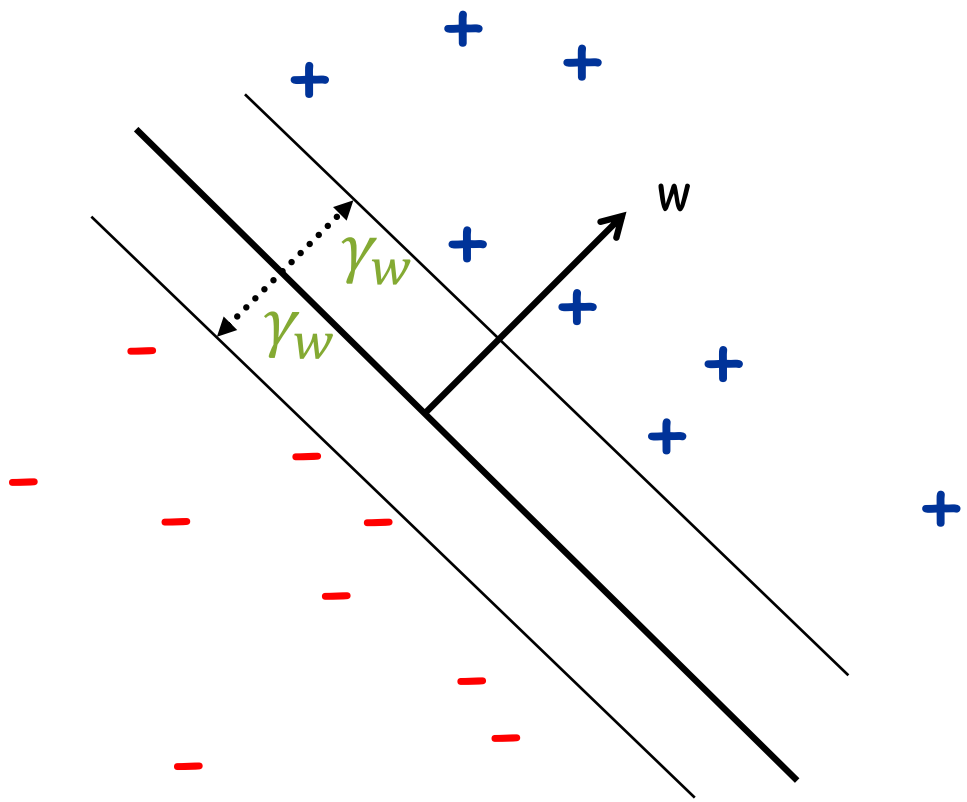



Slide from Nina Balcan

# Geometric Margin

**Definition:** The margin of example $x$ w.r.t. a linear sep. $w$ is the distance from $x$ to the plane $w \cdot x = 0$ (or the negative if on wrong side)

**Definition:** The margin $\gamma_w$ of a set of examples $S$ wrt a linear separator $w$ is the smallest margin over points $x \in S$.



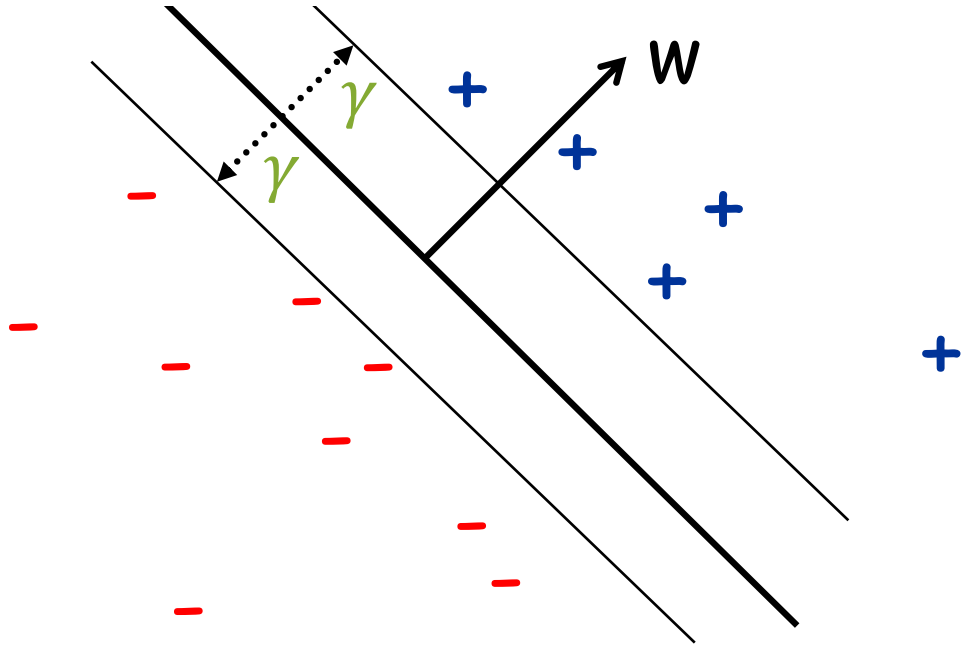

Slide from Nina Balcan

# Geometric Margin

**Definition:** The margin of example $x$ w.r.t. a linear sep. $w$ is the distance from $x$ to the plane $w \cdot x = 0$ (or the negative if on wrong side)

**Definition:** The margin $\gamma_w$ of a set of examples $S$ wrt a linear separator $w$ is the smallest margin over points $x \in S$.

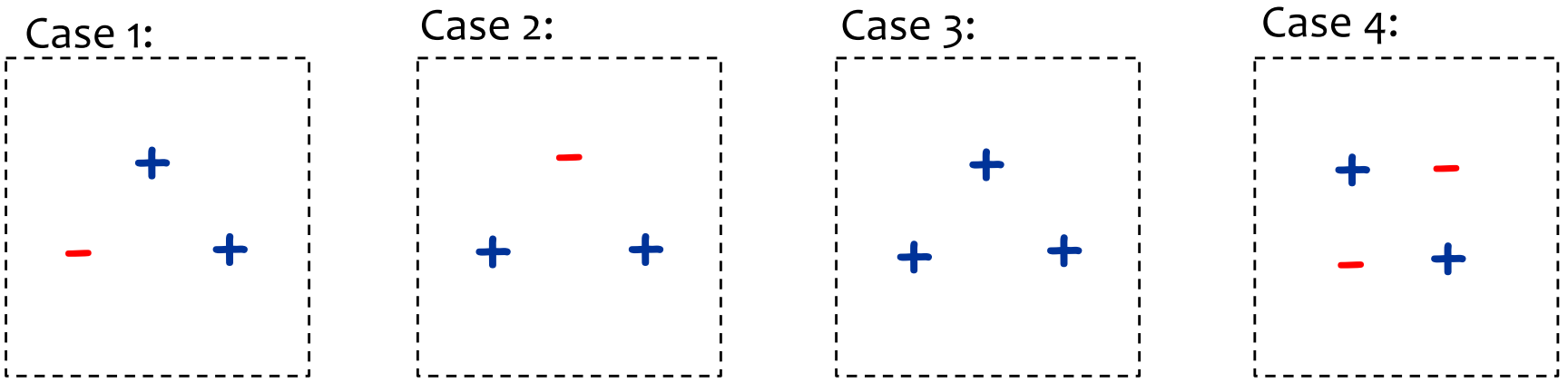
**Definition:** The margin $\gamma$ of a set of examples $S$ is the maximum $\gamma_w$ over all linear separators $w$.




Slide from Nina Balcan

# Linear Separability

***Def:*** For a **binary classification** problem, a set of examples $S$ is **linearly separable** if there exists a linear decision boundary that can separate the points

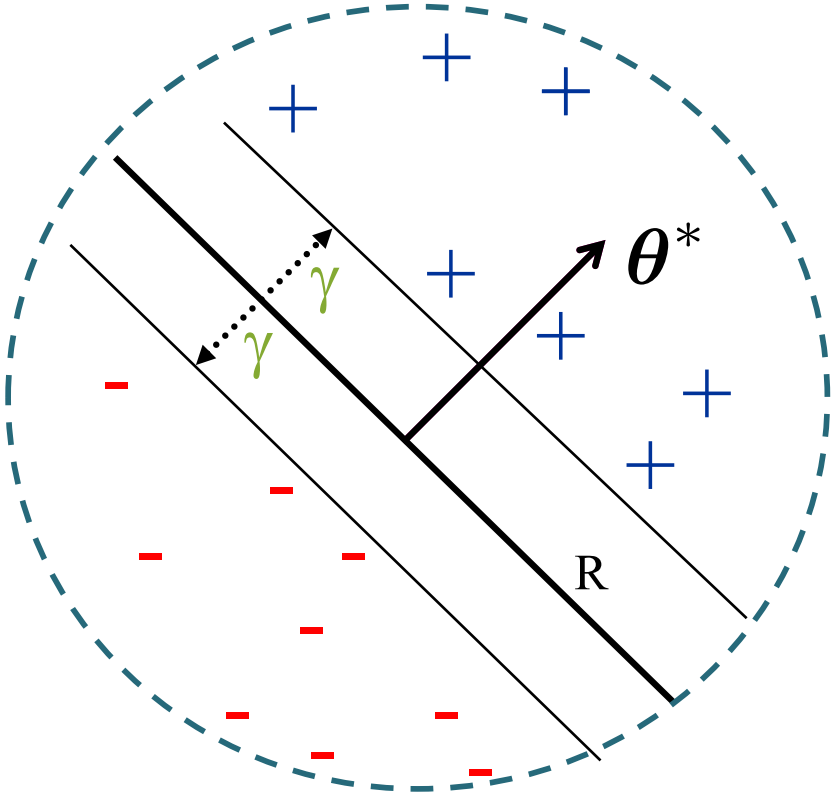


# Analysis: Perceptron

**Perceptron Mistake Bound**

**Guarantee:** If data has margin $\gamma$ and all points inside a ball of radius $R$, then Perceptron makes $\leq (R/\gamma)^2$ mistakes.

(Normalized margin: multiplying all points by 100, or dividing all points by 100, doesn't change the number of mistakes; algo is invariant to scaling.)



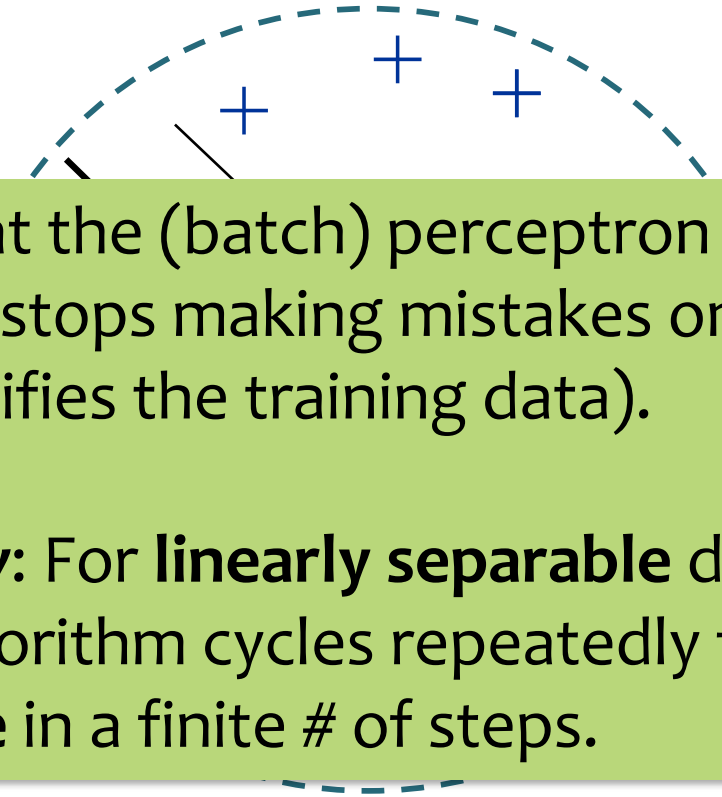

Slide adapted from Nina Balcan

# Analysis: Perceptron

**Perceptron Mistake Bound**

**Guarantee:** If data has margin $\gamma$ and all points inside a ball of radius $R$, then Perceptron makes $\leq (R/\gamma)^2$ mistakes.

(Normalized margin: multiplying all points by 100, or dividing all points by 100, doesn't change the number of mistakes; algo is invariant to scaling.)



***Def:*** We say that the (batch) perceptron algorithm has **converged** if it stops making mistakes on the training data (perfectly classifies the training data).

***Main Takeaway***: For **linearly separable** data, if the perceptron algorithm cycles repeatedly through the data, it will **converge** in a finite # of steps.

Slide adapted from Nina Balcan

# Perceptron Exercises

**Question:**

*Unlike Decision Trees and K-Nearest Neighbors, the Perceptron algorithm* ***does not suffer from overfitting*** *because it does not have any hyperparameters that could be over-tuned on the training data.*

A. *True*
B. *False*
C. *True and False*