The Perceptron Algorithm

Machine Learning
Spring 2018

THE UNIVERSITY OF UTAH

The slides are mainly from Vivek Srikumar

Outline

- The Perceptron Algorithm
- Perceptron Mistake Bound
- Variants of Perceptron

Where are we?

- The Perceptron Algorithm
- Perceptron Mistake Bound
- Variants of Perceptron

Recall: Linear Classifiers

- Input is a n dimensional vector **x**
- Output is a label $y \in \{-1, 1\}$

- *Linear Threshold Units* (LTUs) classify an example **x** using the following classification rule

  - Output = $\mathrm{sgn}(\mathbf{w}^T\mathbf{x} + b) = \mathrm{sgn}(b + \sum w_i x_i)$

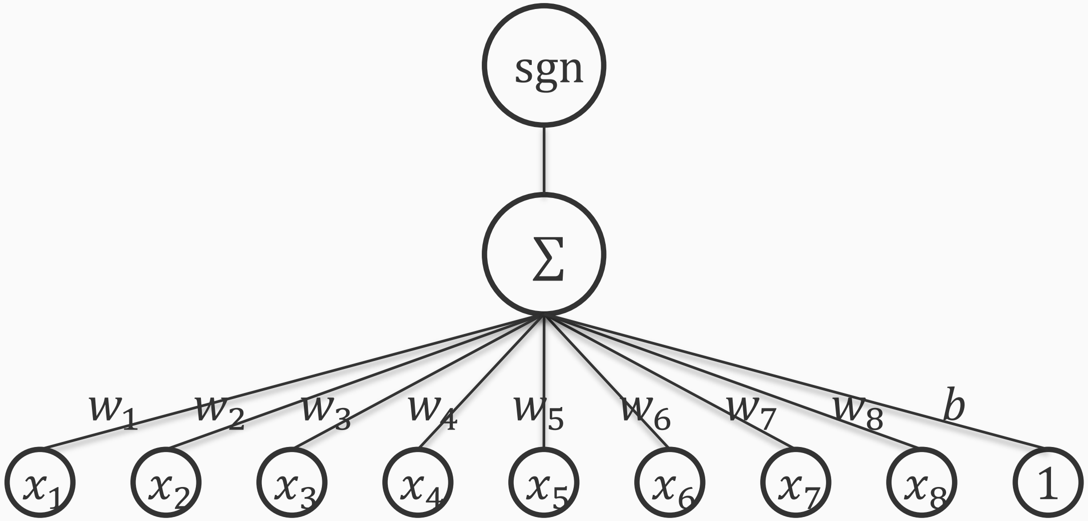
  - $\mathbf{w}^T\mathbf{x} + b \geq 0 \rightarrow$ Predict y = 1
  - $\mathbf{w}^T\mathbf{x} + b < 0 \rightarrow$ Predict y = -1

    b is called the bias term

Recall: Linear Classifiers

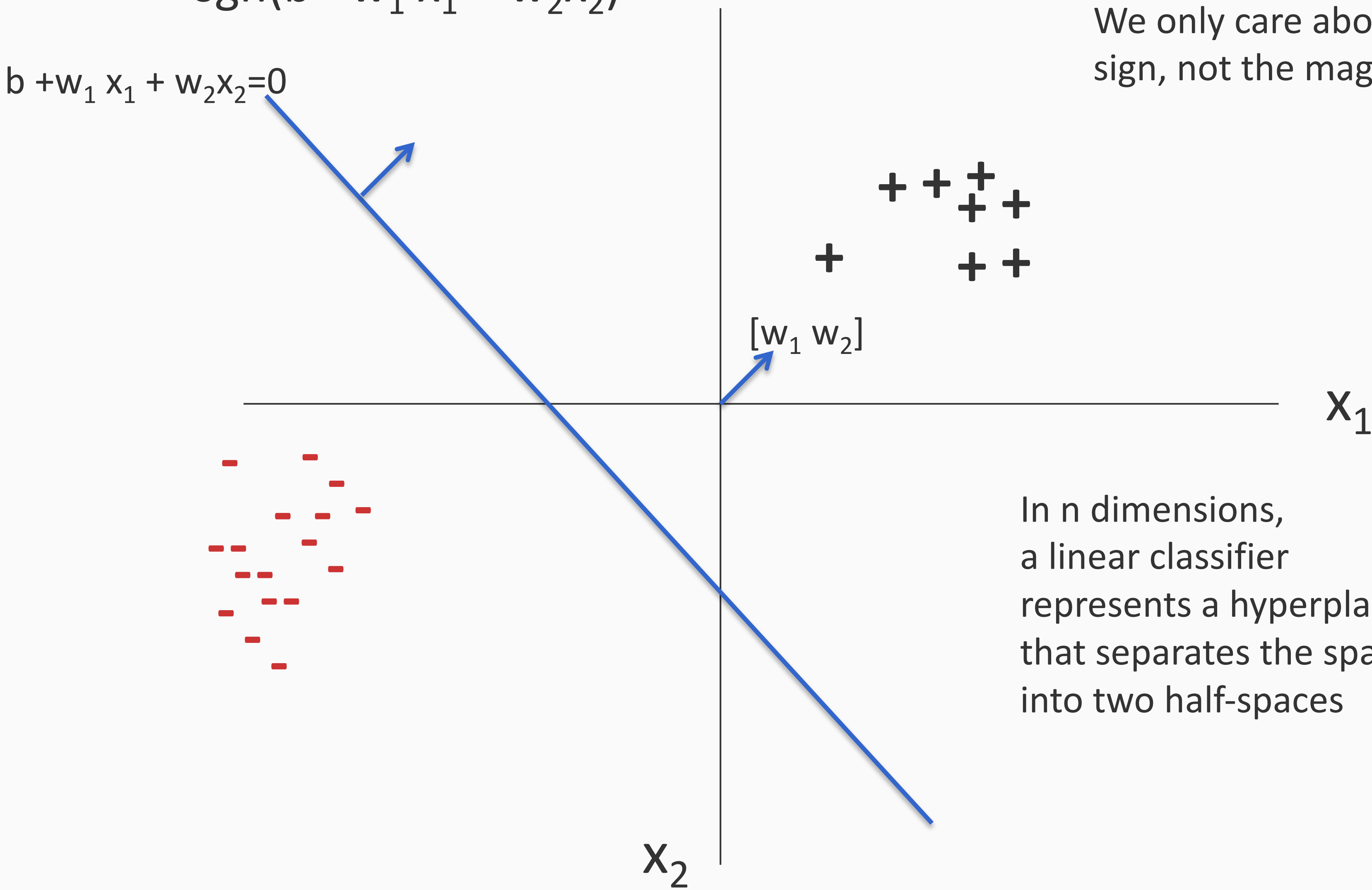
- Input is a n dimensional vector **x**
- Output is a label y ∈ {-1, 1}

- *Linear Threshold Units* (LTUs) classify an example **x** using the following classification rule



The geometry of a linear classifier

$\mathrm{sgn}(b + w_1 x_1 + w_2 x_2)$



We only care about the sign, not the magnitude

In n dimensions, a linear classifier represents a hyperplane that separates the space into two half-spaces

The Perceptron

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.

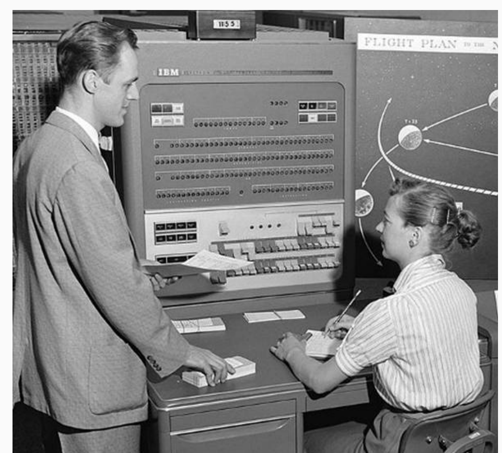
The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

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

The Perceptron algorithm

- Rosenblatt 1958

- The goal is to find a separating hyperplane
  - For separable data, guaranteed to find one

- An online algorithm
  - Processes one example at a time

- Several variants exist (will discuss briefly at towards the end)

The Perceptron algorithm

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

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example $(\mathbf{x}_i, y_i)$:
  - Predict $y' = \text{sgn}(\mathbf{w}_t^T \mathbf{x}_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\,\mathbf{x}_i)$
- Return final weight vector

The Perceptron algorithm

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

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example $(\mathbf{x}_i, y_i)$:
  - Predict $y' = \text{sgn}(\mathbf{w}_t^T \mathbf{x}_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\,\mathbf{x}_i)$
- Return final weight vector

Remember:
Prediction = $\text{sgn}(\mathbf{w}^T\mathbf{x})$

There is typically a bias term also $(\mathbf{w}^T\mathbf{x} + b)$, but the bias may be treated as a constant feature and folded into $\mathbf{w}$

The Perceptron algorithm

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

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example $(\mathbf{x}_i, y_i)$:
  - Predict $y' = \text{sgn}(\mathbf{w}_t^T \mathbf{x}_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\, \mathbf{x}_i)$
- Return final weight vector

Remember:
Prediction = $\text{sgn}(\mathbf{w}^T\mathbf{x})$

There is typically a bias term also $(\mathbf{w}^T\mathbf{x} + b)$, but the bias may be treated as a constant feature and folded into $\mathbf{w}$

Footnote: For some algorithms it is mathematically easier to represent False as -1, and at other times, as 0. For the Perceptron algorithm, treat -1 as false and +1 as true.

The Perceptron algorithm

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

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

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example $(\mathbf{x}_i, y_i)$:
  - Predict $y' = \mathrm{sgn}(\mathbf{w}_t^T \mathbf{x}_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\,\mathbf{x}_i)$
- Return final weight vector

The Perceptron algorithm

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

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

r is the learning rate, a small positive number less than 1

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example $(\mathbf{x}_i, y_i)$:
  - Predict $y' = \text{sgn}(\mathbf{w}_t^T \mathbf{x}_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\, \mathbf{x}_i)$
- Return final weight vector

The Perceptron algorithm

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

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example $(\mathbf{x}_i, y_i)$:
  - Predict $y' = \text{sgn}(\mathbf{w}_t^T \mathbf{x}_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\, \mathbf{x}_i)$
- Return final weight vector

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

r is the learning rate, a small positive number less than 1

Update only on error. A mistake-driven algorithm

The Perceptron algorithm

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

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example $(\mathbf{x}_i, y_i)$:
  - Predict $y' = \text{sgn}(\mathbf{w}_t^T \mathbf{x}_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\, \mathbf{x}_i)$
- Return final weight vector

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

r is the learning rate, a small positive number less than 1

Update only on error. A mistake-driven algorithm

This is the simplest version. We will see more robust versions at the end

The Perceptron algorithm

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

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example $(\mathbf{x}_i, y_i)$:
  - Predict $y' = \text{sgn}(\mathbf{w}_t^T \mathbf{x}_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\, \mathbf{x}_i)$
- Return final weight vector

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

r is the learning rate, a small positive number less than 1

Update only on error. A mistake-driven algorithm

This is the simplest version. We will see more robust versions at the end

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

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

Intuition behind the update

Suppose we have made a mistake on a positive example

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

Call the new weight vector $\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{x}$ (say $r = 1$)

The new dot product will be $\mathbf{w}_{t+1}^T\mathbf{x} = (\mathbf{w}_t + \mathbf{x})^T\mathbf{x} = \mathbf{w}_t^T\mathbf{x} + \mathbf{x}^T\mathbf{x} > \mathbf{w}_t^T\mathbf{x}$

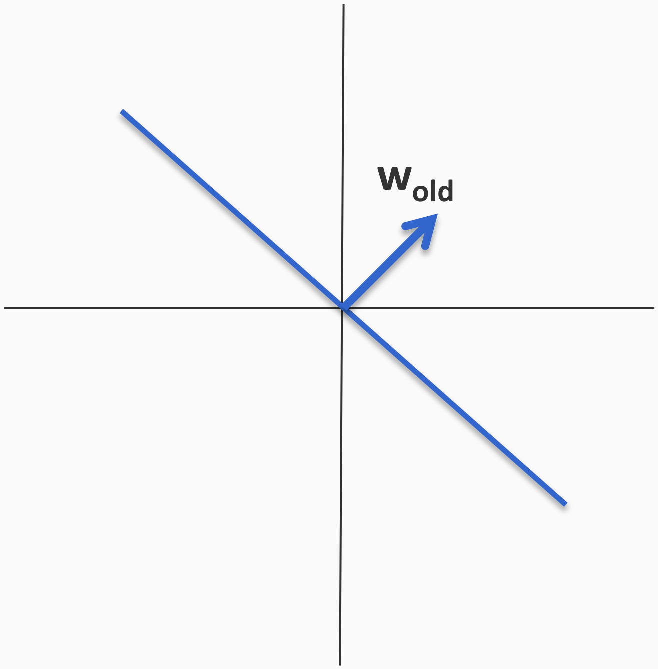
For a positive example, the Perceptron update will increase the score assigned to the same input

Similar reasoning for negative 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

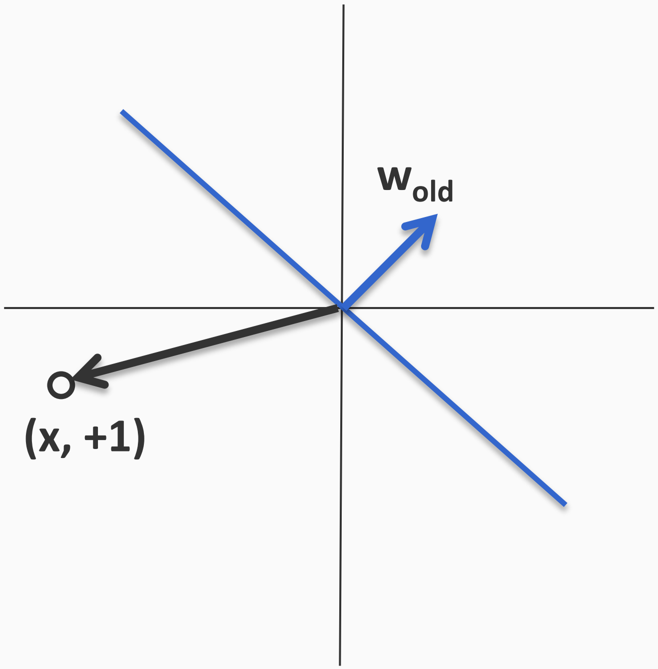
Predict



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

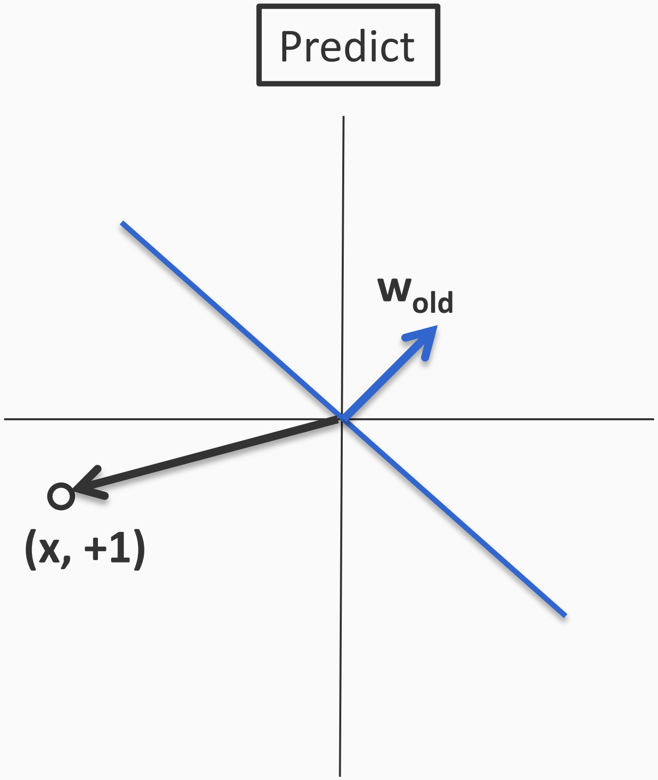
Geometry of the perceptron update

Predict



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

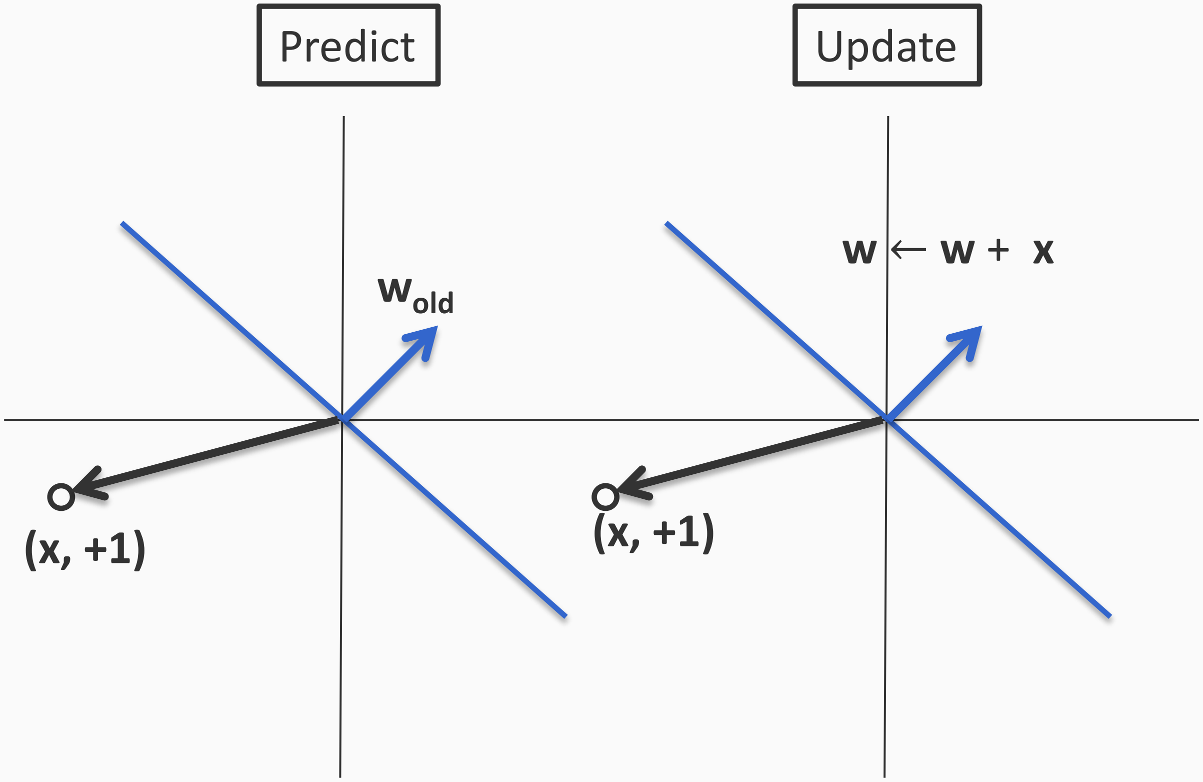
Geometry of the perceptron update



For a mistake on a positive example

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

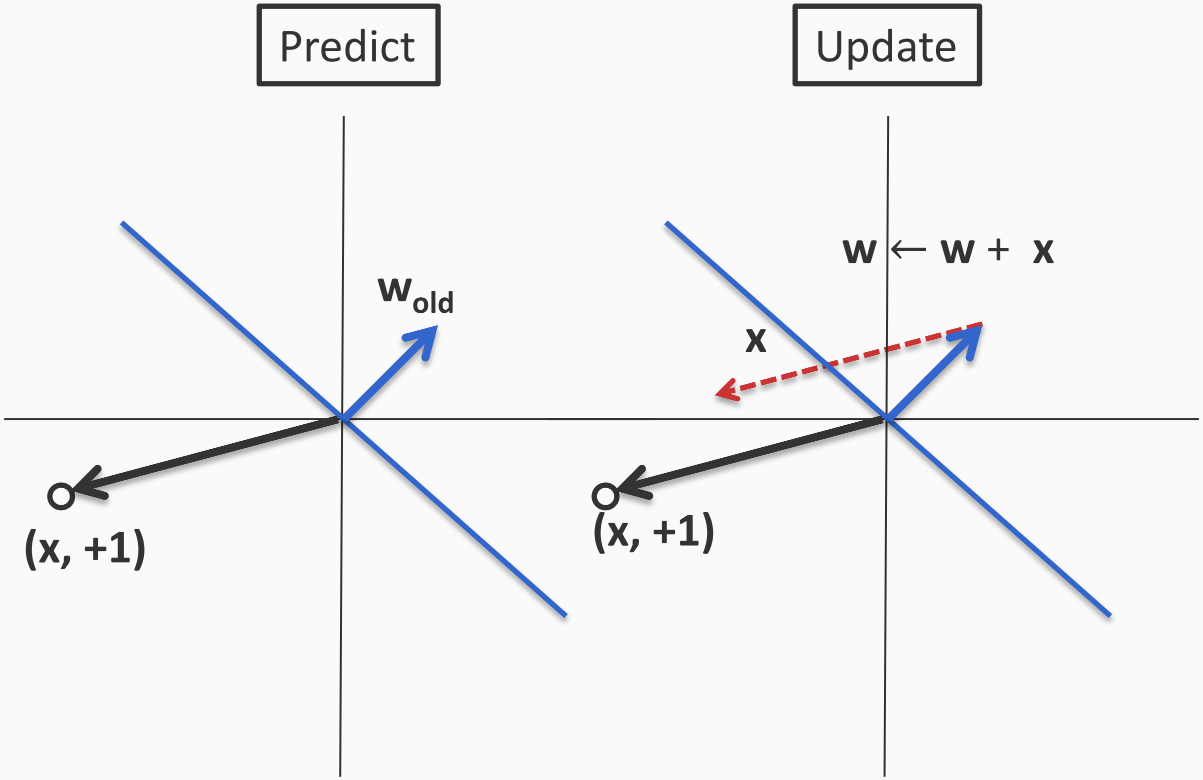
Geometry of the perceptron update



For a mistake on a positive example

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

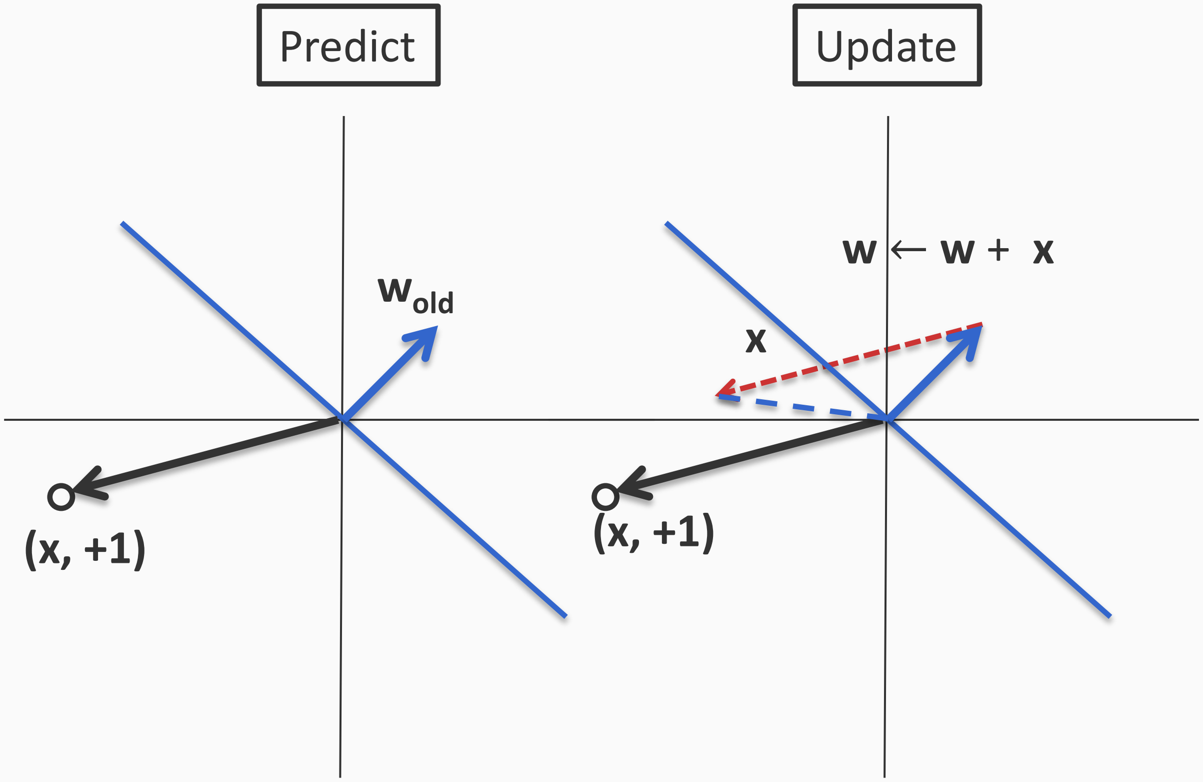
Geometry of the perceptron update



For a mistake on a positive example

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

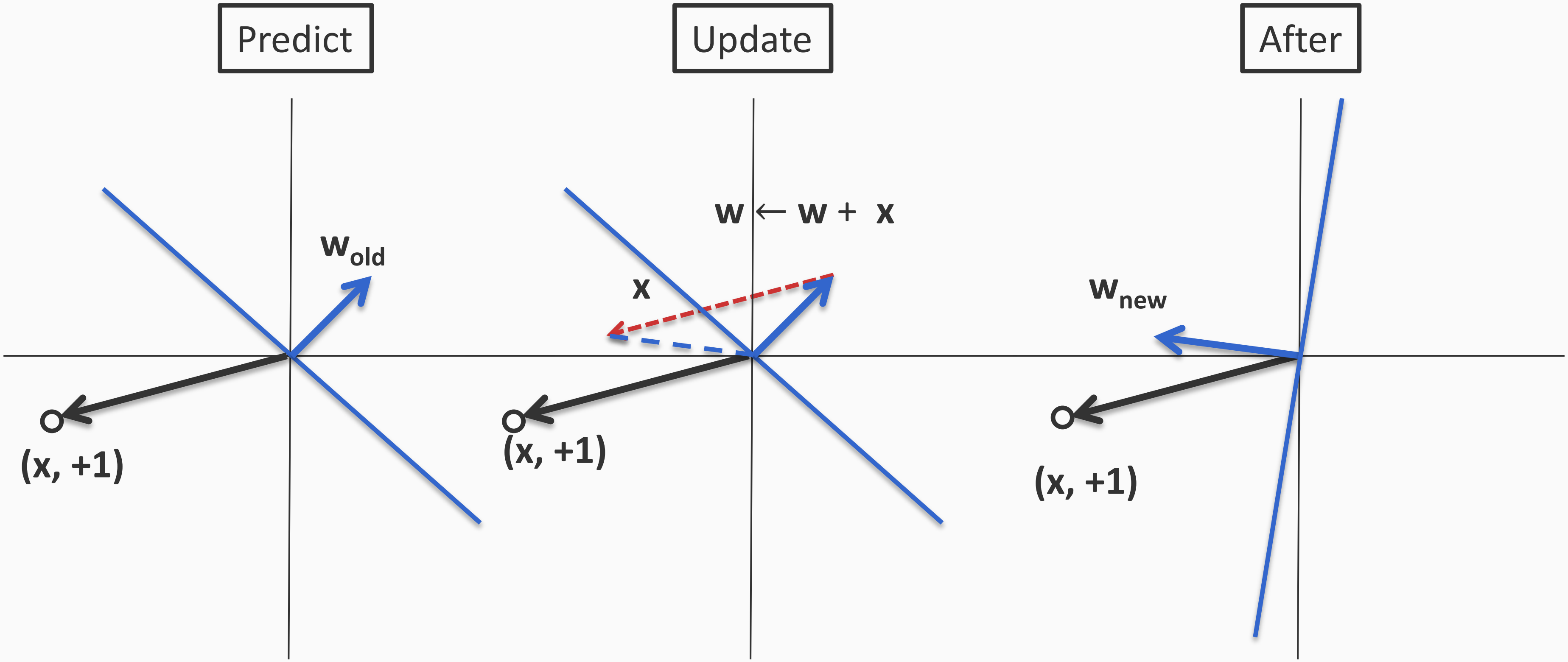
Geometry of the perceptron update



For a mistake on a positive example

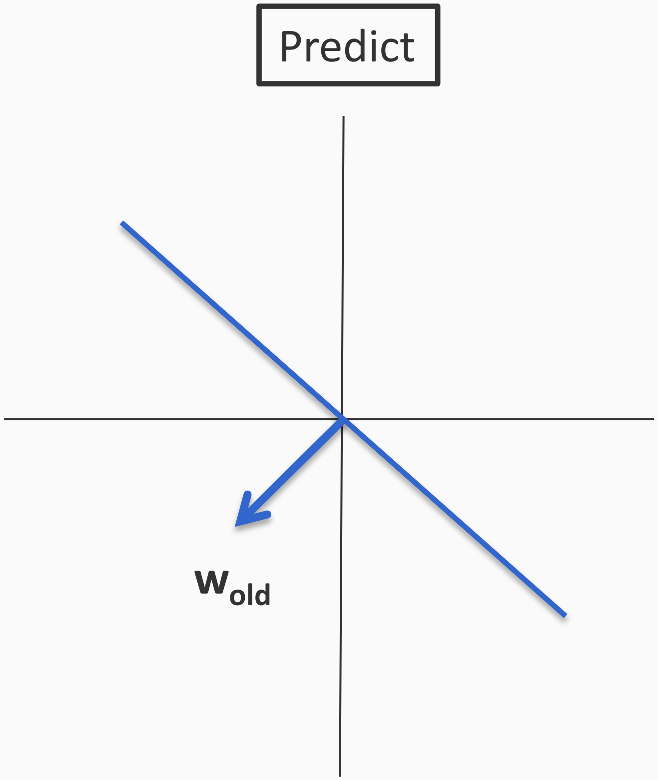
Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\,\mathbf{x}_i$

Geometry of the perceptron update

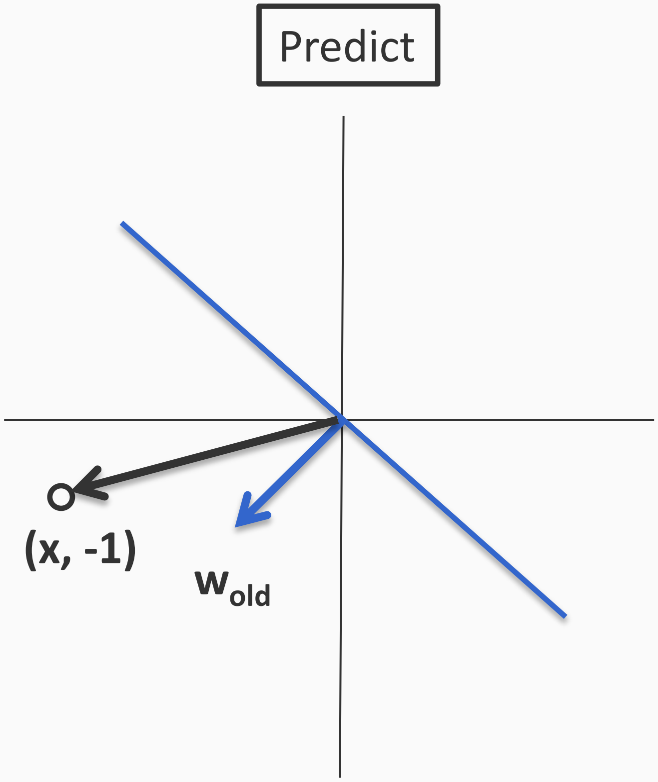


For a mistake on a positive example

Geometry of the perceptron update

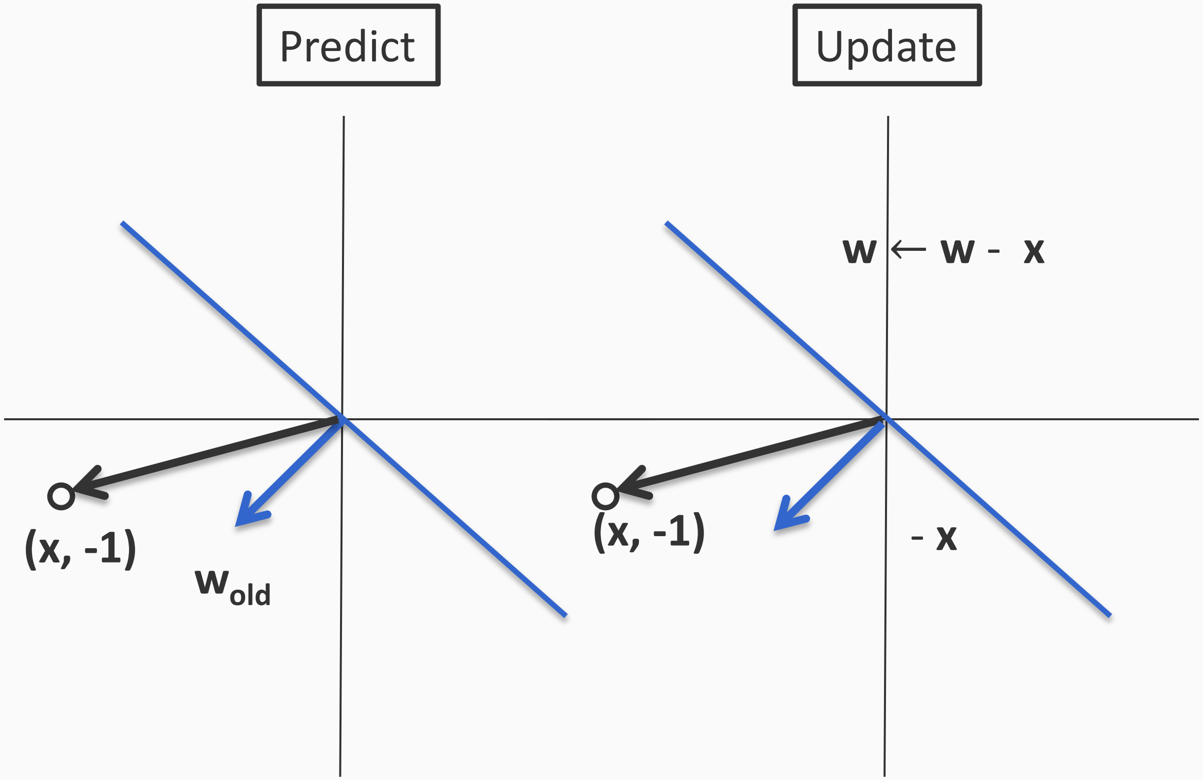


Geometry of the perceptron update



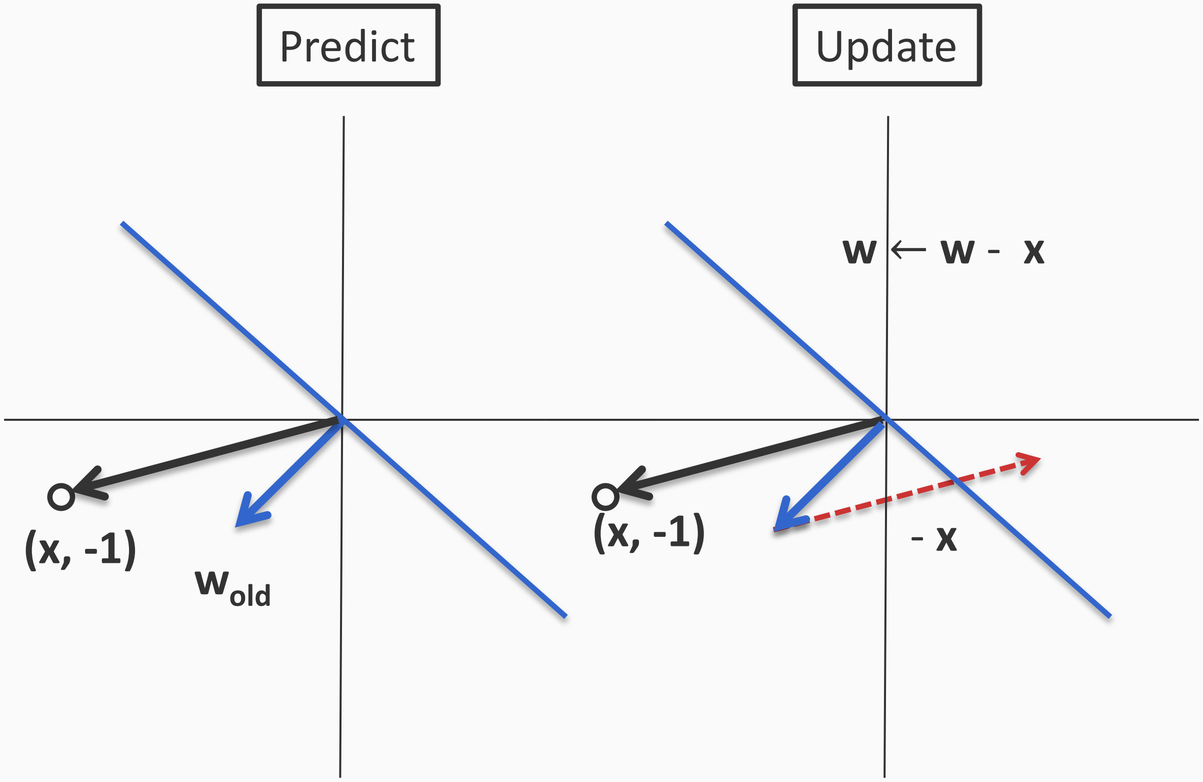
For a mistake on a negative example

Geometry of the perceptron update



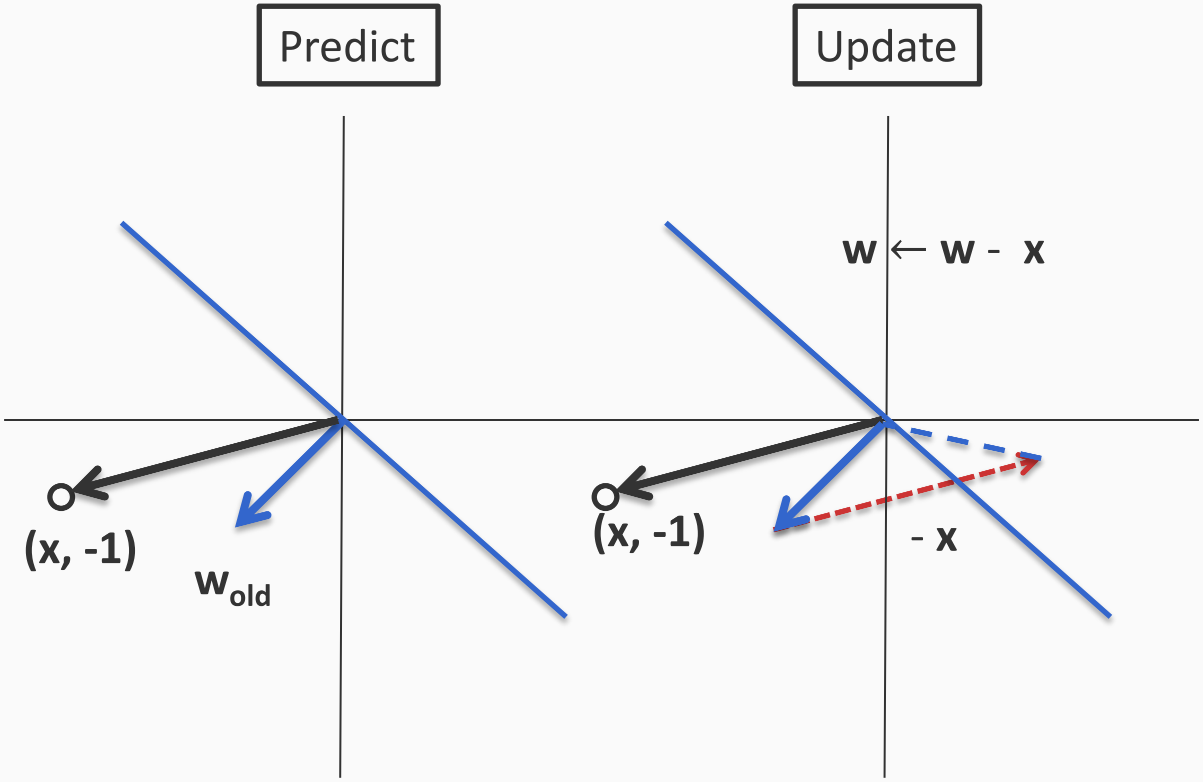
For a mistake on a negative example

Geometry of the perceptron update



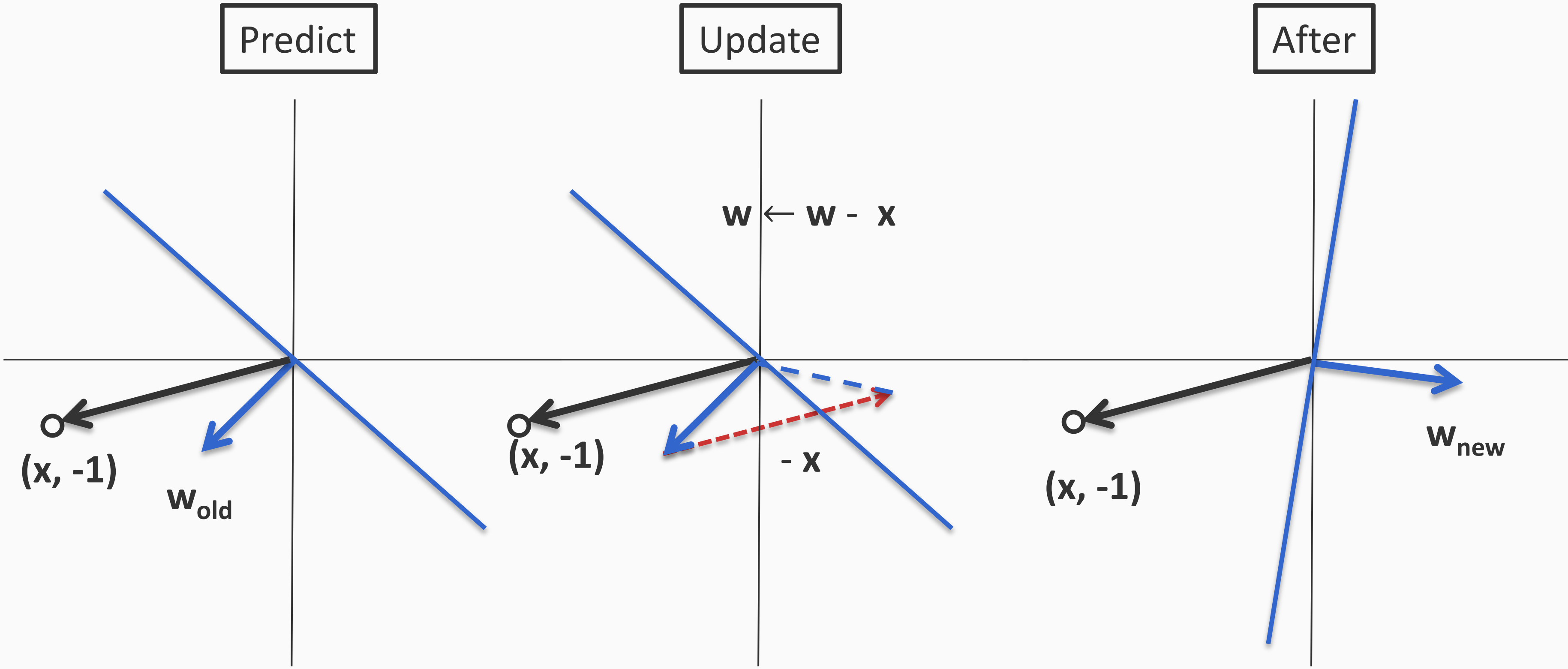
For a mistake on a negative example

Geometry of the perceptron update



For a mistake on a negative example

Geometry of the perceptron update



For a mistake on a negative example

Perceptron Learnability

- Obviously Perceptron cannot learn what it cannot represent
  - Only linearly separable functions

- Minsky and Papert (1969) wrote an influential book demonstrating Perceptron's representational limitations
  - Parity functions can't be learned (XOR)
    - But we already know that XOR is not linearly separable
  - Feature transformation trick

What you need to know

- The Perceptron algorithm
- The geometry of the update
- What can it represent

Where are we?

- The Perceptron Algorithm
- Perceptron Mistake Bound
- Variants of Perceptron

Convergence

Convergence theorem

- If there exist a set of weights that are consistent with the data (i.e. the data is linearly separable), the perceptron algorithm will converge.

Convergence

Convergence theorem

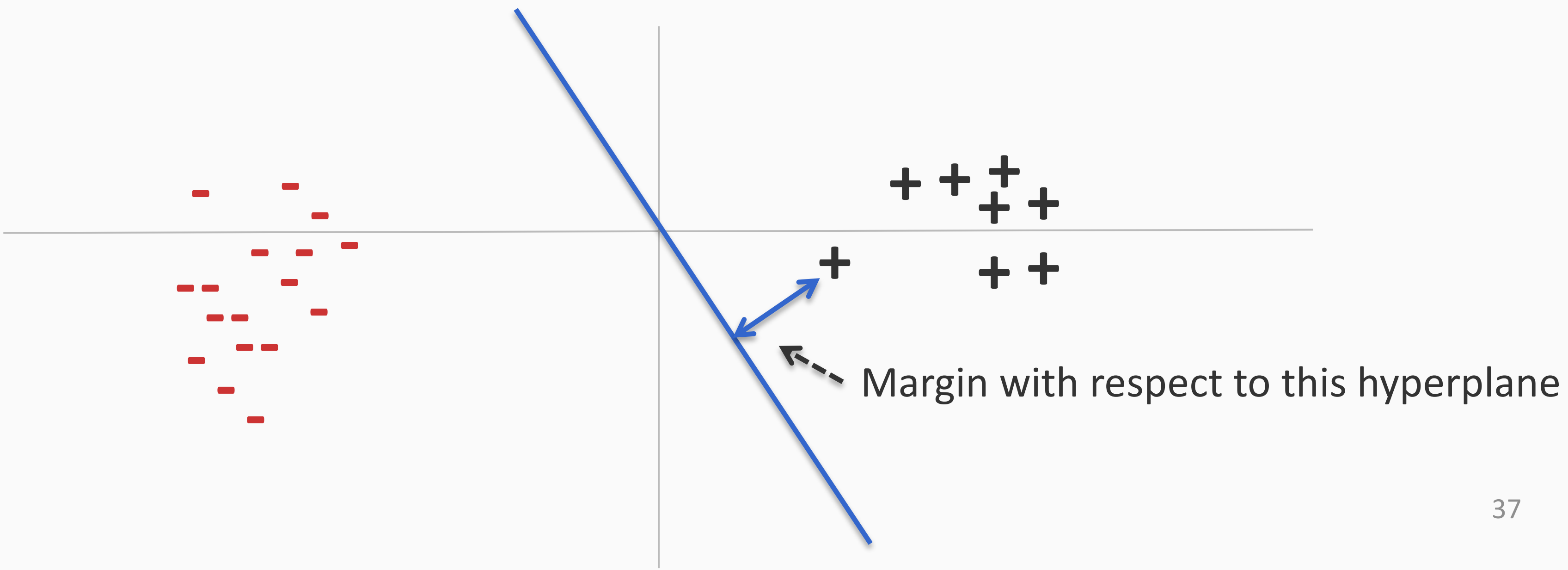
– If there exist a set of weights that are consistent with the data (i.e. the data is linearly separable), the perceptron algorithm will converge.

Cycling theorem

– If the training data is *not* linearly separable, then the learning algorithm will eventually repeat the same set of weights and enter an infinite loop

Margin

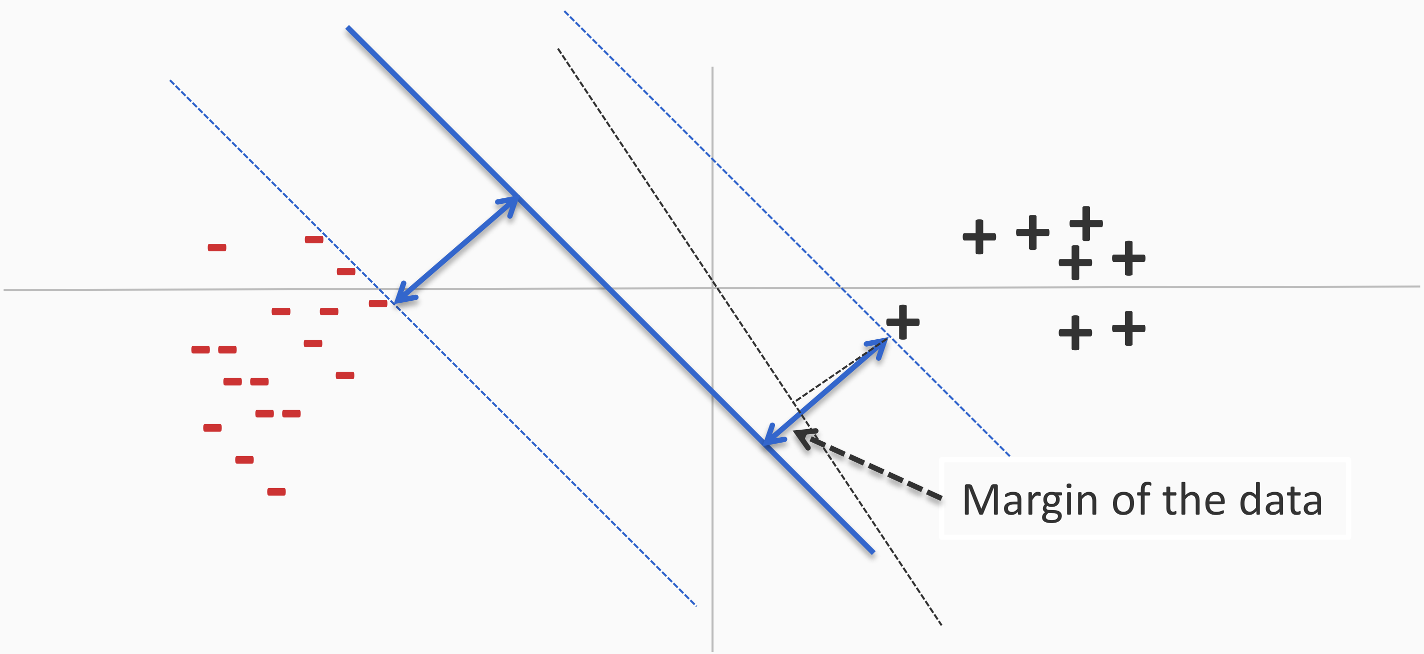
The margin of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it.



Margin

- The margin of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it.

- The margin of a data set (γ) is the maximum margin possible for that dataset using any weight vector.



Mistake Bound Theorem [Novikoff 1962, Block 1962]

Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}$ be a sequence of training examples such that for all i, the feature vector $\mathbf{x}_i \in \Re^n$, $\|\mathbf{x}_i\| \leq R$ and the label $y_i \in \{-1, +1\}$.

Mistake Bound Theorem [Novikoff 1962, Block 1962]

Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}$ be a sequence of training examples such that for all i, the feature vector $\mathbf{x}_i \in \Re^n$, $\|\mathbf{x}_i\| \leq R$ and the label $y_i \in \{-1, +1\}$.

Suppose there exists a unit vector $\mathbf{u} \in \Re^n$ (i.e $\|\mathbf{u}\| = 1$) such that for some $\gamma \in \Re$ and $\gamma > 0$ we have $y_i (\mathbf{u}^T \mathbf{x}_i) \geq \gamma$.

Mistake Bound Theorem [Novikoff 1962, Block 1962]

Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}$ be a sequence of training examples such that for all i, the feature vector $\mathbf{x}_i \in \Re^n$, $\|\mathbf{x}_i\| \leq R$ and the label $y_i \in \{-1, +1\}$.

Suppose there exists a unit vector $\mathbf{u} \in \Re^n$ (i.e $\|\mathbf{u}\| = 1$) such that for some $\gamma \in \Re$ and $\gamma > 0$ we have $y_i (\mathbf{u}^T \mathbf{x}_i) \geq \gamma$.

Then, the perceptron algorithm will make at most $(R/\gamma)^2$ mistakes on the training sequence.

Mistake Bound Theorem [Novikoff 1962, Block 1962]

Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}$ be a sequence of training examples such that for all i, the feature vector $\mathbf{x}_i \in \Re^n$, $||\mathbf{x}_i|| \leq R$ and the label $y_i \in \{-1, +1\}$.

We can always find such an R. Just look for the farthest data point from the origin.

Suppose there exists a unit vector $\mathbf{u} \in \Re^n$ (i.e $||\mathbf{u}|| = 1$) such that for some $\gamma \in \Re$ and $\gamma > 0$ we have $y_i (\mathbf{u}^T \mathbf{x}_i) \geq \gamma$.

Then, the perceptron algorithm will make at most $(R/\gamma)^2$ mistakes on the training sequence.

Mistake Bound Theorem [Novikoff 1962, Block 1962]

Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}$ be a sequence of training examples such that for all i, the feature vector $\mathbf{x}_i \in \Re^n$, $||\mathbf{x}_i|| \leq R$ and the label $y_i \in \{-1, +1\}$.

Suppose there exists a unit vector $\mathbf{u} \in \Re^n$ (i.e $||\mathbf{u}|| = 1$) such that for some $\gamma \in \Re$ and $\gamma > 0$ we have $y_i (\mathbf{u}^T \mathbf{x}_i) \geq \gamma$.

Then, the perceptron algorithm will make at most $(R/\gamma)^2$ mistakes on the training sequence.

The data and **u** have a margin γ.
Importantly, the data is *separable*.
γ is the complexity parameter that defines the separability of data.

Mistake Bound Theorem [Novikoff 1962, Block 1962]

Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}$ be a sequence of training examples such that for all i, the feature vector $\mathbf{x}_i \in \Re^n$, $||\mathbf{x}_i|| \leq R$ and the label $y_i \in \{-1, +1\}$.

Suppose there exists a unit vector $\mathbf{u} \in \Re^n$ (i.e $||\mathbf{u}|| = 1$) such that for some $\gamma \in \Re$ and $\gamma > 0$ we have $y_i (\mathbf{u}^T \mathbf{x}_i) \geq \gamma$.

Then, the perceptron algorithm will make at most $(R/\gamma)^2$ mistakes on the training sequence.

If **u** hadn't been a unit vector, then we could scale γ in the mistake bound. This will change the final mistake bound to $(||\mathbf{u}||R/\gamma)^2$

Mistake Bound Theorem [Novikoff 1962, Block 1962]

Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}$ be a sequence of training examples such that for all i, the feature vector $\mathbf{x}_i \in \Re^n$, $\|\mathbf{x}_i\| \leq R$ and the label $y_i \in \{-1, +1\}$.

Suppose we have a binary classification dataset with n dimensional inputs.

Suppose there exists a unit vector $\mathbf{u} \in \Re^n$ (i.e $\|\mathbf{u}\| = 1$) such that for some $\gamma \in \Re$ and $\gamma > 0$ we have $y_i (u^T x_i) \geq \gamma$.

If the data is separable,…

Then, the perceptron algorithm will make at most $(R/\gamma)^2$ mistakes on the training sequence.

…then the Perceptron algorithm will find a separating hyperplane after making a finite number of mistakes

Proof (preliminaries)

- Receive an input $(\mathbf{x}_i, y_i)$
- if $\text{sgn}(\mathbf{w}_t^T \mathbf{x}_i) \neq y_i$:
  Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_i\, \mathbf{x}_i$

The setting

- Initial weight vector **w** is all zeros
- Learning rate = 1
  - Effectively scales inputs, but does not change the behavior
- All training examples are contained in a ball of size R
  - $\|\mathbf{x}_i\| \leq R$
- The training data is separable by margin γ using a unit vector **u**
  - $y_i\,(\mathbf{u}^T \mathbf{x}_i) \geq \gamma$

Proof (1/3)

- Receive an input ($\mathbf{x}_i$, y_i)
- if $\text{sgn}(\mathbf{w}_t^T\mathbf{x}_i) \neq y_i$:
  Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_i\,\mathbf{x}_i$

1. Claim: After t mistakes, $\mathbf{u}^T\mathbf{w}_t \geq t\,\gamma$

$$\mathbf{u}^T\mathbf{w}_{t+1} \quad = \quad \mathbf{u}^T\mathbf{w}_t + y_i\mathbf{u}^T\mathbf{x}_i$$

Proof (1/3)

- Receive an input ($\mathbf{x}_i$, y_i)
- if sgn($\mathbf{w}_t^T\mathbf{x}_i$) ≠ y_i:
  Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_i\,\mathbf{x}_i$

1. Claim: After t mistakes, $\mathbf{u}^T\mathbf{w}_t \geq t\,\gamma$

$$\begin{aligned}\mathbf{u}^T\mathbf{w}_{t+1} &= \mathbf{u}^T\mathbf{w}_t + y_i\mathbf{u}^T\mathbf{x}_i \\ &\geq \mathbf{u}^T\mathbf{w}_t + \gamma\end{aligned}$$

Because the data is separable by a margin γ

Proof (1/3)

- Receive an input ($\mathbf{x}_i$, y_i)
- if $\text{sgn}(\mathbf{w}_t^T \mathbf{x}_i) \neq y_i$:
  Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_i \mathbf{x}_i$

1. Claim: After t mistakes, $\mathbf{u}^T\mathbf{w}_t \geq t\,\gamma$

$$\begin{aligned} \mathbf{u}^T\mathbf{w}_{t+1} &= \mathbf{u}^T\mathbf{w}_t + y_i\mathbf{u}^T\mathbf{x}_i \\ &\geq \mathbf{u}^T\mathbf{w}_t + \gamma \end{aligned}$$

Because the data is separable by a margin γ

Because $\mathbf{w}_0 = \mathbf{0}$ (i.e $\mathbf{u}^T\mathbf{w}_0 = 0$), straightforward induction gives us $\mathbf{u}^T\mathbf{w}_t \geq t\,\gamma$

Proof (2/3)

- Receive an input ($\mathbf{x}_i$, y_i)
- if $\text{sgn}(\mathbf{w}_t^T \mathbf{x}_i) \neq y_i$:
  Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_i \mathbf{x}_i$

2. Claim: After t mistakes, $||\mathbf{w}_t||^2 \leq tR^2$

$$
\begin{aligned}
\left\|\mathbf{w}_{t+1}\right\|^2 &= \left\|\mathbf{w}_t + y_i \mathbf{x}_i\right\|^2 \\
&= \left\|\mathbf{w}_t\right\|^2 + 2y_i(\mathbf{w}_t^T \mathbf{x}_i) + \left\|\mathbf{x}_i\right\|^2
\end{aligned}
$$

Proof (2/3)

- Receive an input ($\mathbf{x}_i$, y_i)
- if $\text{sgn}(\mathbf{w}_t^T \mathbf{x}_i) \neq y_i$:
  Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_i \mathbf{x}_i$

2. Claim: After t mistakes, $||\mathbf{w}_t||^2 \leq tR^2$

$$
\begin{aligned}
||\mathbf{w}_{t+1}||^2 &= ||\mathbf{w}_t + y_i \mathbf{x}_i||^2 \\
&= ||\mathbf{w}_t||^2 + 2y_i(\mathbf{w}_t^T \mathbf{x}_i) + ||\mathbf{x}_i||^2
\end{aligned}
$$

The weight is updated only when there is a mistake. That is when $y_i \mathbf{w}_t^T \mathbf{x}_i < 0$.

$||\mathbf{x}_i|| \leq R$, by definition of R

Proof (2/3)

- Receive an input ($\mathbf{x}_i$, y_i)
- if $\text{sgn}(\mathbf{w}_t^T\mathbf{x}_i) \neq y_i$:
  Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_i\,\mathbf{x}_i$

2. Claim: After t mistakes, $||\mathbf{w}_t||^2 \leq tR^2$

$$
\begin{aligned}
||\mathbf{w}_{t+1}||^2 &= ||\mathbf{w}_t + y_i\mathbf{x}_i||^2 \\
&= ||\mathbf{w}_t||^2 + 2y_i(\mathbf{w}_t^T\mathbf{x}_i) + ||\mathbf{x}_i||^2 \\
&\leq ||\mathbf{w}_t||^2 + R^2
\end{aligned}
$$

Because $\mathbf{w}_0 = \mathbf{0}$ (i.e $\mathbf{u}^T\mathbf{w}_0 = 0$), straightforward induction gives us $||\mathbf{w}_t||^2 \leq tR^2$

Proof (3/3)

What we know:

1. After t mistakes, $\mathbf{u}^T\mathbf{w}_t \geq t\gamma$
2. After t mistakes, $\|\mathbf{w}_t\|^2 \leq tR^2$

Proof (3/3)

What we know:

1. After t mistakes, $\mathbf{u}^T\mathbf{w}_t \geq t\gamma$
2. After t mistakes, $\|\mathbf{w}_t\|^2 \leq tR^2$

$$R\sqrt{t} \geq \|\mathbf{w}_t\|$$

From (2)

Proof (3/3)

What we know:

1. After t mistakes, $\mathbf{u}^T\mathbf{w}_t \geq t\gamma$
2. After t mistakes, $||\mathbf{w}_t||^2 \leq tR^2$

$$R\sqrt{t} \geq \|\mathbf{w}_t\| \geq \mathbf{u}^T\mathbf{w}_t$$

From (2)

$\mathbf{u}^T \mathbf{w}_t = ||\mathbf{u}||\ ||\mathbf{w}_t||$ cos(<angle between them>)

But $||\mathbf{u}|| = 1$ and cosine is less than 1

So $\mathbf{u}^T \mathbf{w}_t \cdot ||\mathbf{w}_t||$

Proof (3/3)

What we know:

1. After t mistakes, $\mathbf{u}^T\mathbf{w}_t \geq t\gamma$
2. After t mistakes, $||\mathbf{w}_t||^2 \leq tR^2$

$$R\sqrt{t} \geq ||\mathbf{w}_t|| \geq \mathbf{u}^T\mathbf{w}_t$$

From (2)

$\mathbf{u}^T \mathbf{w}_t = ||\mathbf{u}|| \, ||\mathbf{w}_t||$ cos(<angle between them>)

But $||\mathbf{u}|| = 1$ and cosine is less than 1

So $\mathbf{u}^T \mathbf{w}_t \cdot ||\mathbf{w}_t||$ *(Cauchy-Schwarz inequality)*

Proof (3/3)

What we know:

1. After t mistakes, $\mathbf{u}^T\mathbf{w}_t \geq t\gamma$
2. After t mistakes, $||\mathbf{w}_t||^2 \leq tR^2$

$$R\sqrt{t} \geq \|\mathbf{w}_t\| \geq \mathbf{u}^T\mathbf{w}_t \geq t\gamma$$

From (2)

From (1)

$\mathbf{u}^T \mathbf{w}_t = ||\mathbf{u}|| \, ||\mathbf{w}_t||$ cos(<angle between them>)

But $||\mathbf{u}|| = 1$ and cosine is less than 1

So $\mathbf{u}^T \mathbf{w}_t \leq ||\mathbf{w}_t||$

Proof (3/3)

What we know:

1. After t mistakes, $\mathbf{u}^T\mathbf{w}_t \geq t\gamma$
2. After t mistakes, $||\mathbf{w}_t||^2 \leq tR^2$

$$R\sqrt{t} \geq \|\mathbf{w}_t\| \geq \mathbf{u}^T\mathbf{w}_t \geq t\gamma$$

Number of mistakes $t \leq \frac{R^2}{\gamma^2}$

Proof (3/3)

What we know:

1. After t mistakes, $\mathbf{u}^T\mathbf{w}_t \geq t\gamma$
2. After t mistakes, $\|\mathbf{w}_t\|^2 \leq tR^2$

$$R\sqrt{t} \geq \|\mathbf{w}_t\| \geq \mathbf{u}^T\mathbf{w}_t \geq t\gamma$$

Number of mistakes $t \leq \frac{R^2}{\gamma^2}$

Bounds the total number of mistakes!

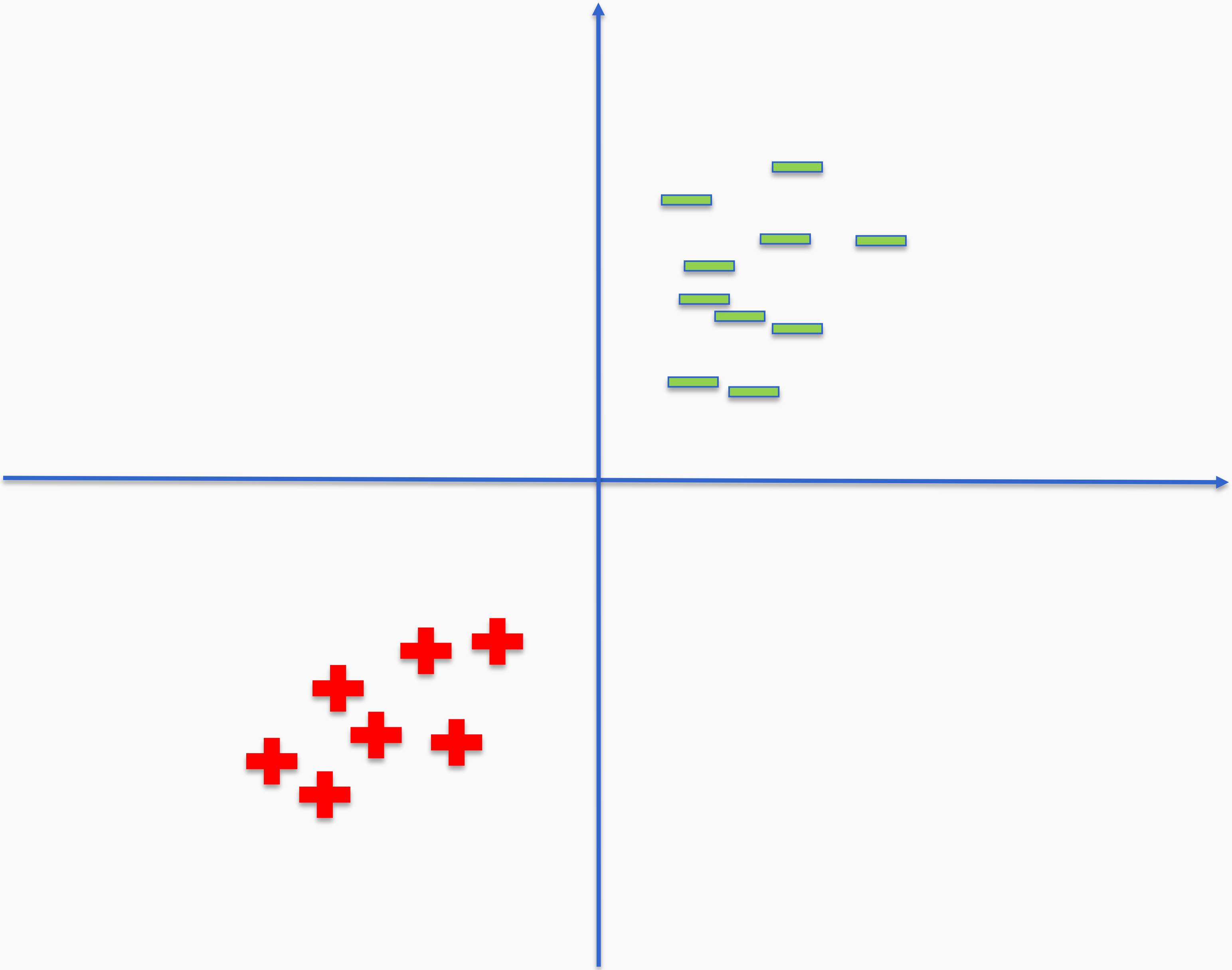
Mistake Bound Theorem [Novikoff 1962, Block 1962]

Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}$ be a sequence of training examples such that for all i, the feature vector $\mathbf{x}_i \in \Re^n$, $\|\mathbf{x}_i\| \leq R$ and the label $y_i \in \{-1, +1\}$.

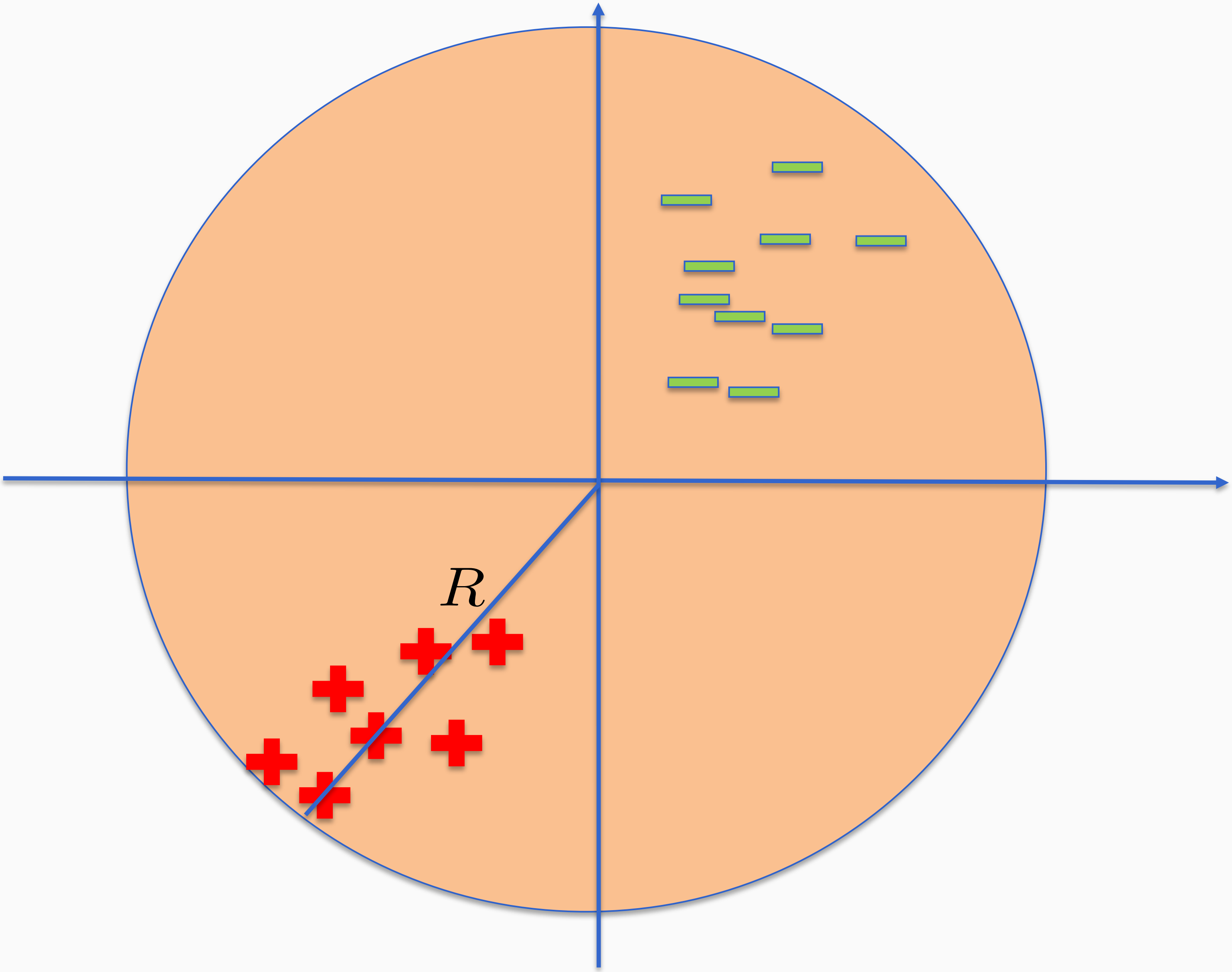
Suppose there exists a unit vector $\mathbf{u} \in \Re^n$ (i.e $\|\mathbf{u}\| = 1$) such that for some $\gamma \in \Re$ and $\gamma > 0$ we have $y_i (\mathbf{u}^T \mathbf{x}_i) \geq \gamma$.

Then, the perceptron algorithm will make at most $(R/\gamma)^2$ mistakes on the training sequence.

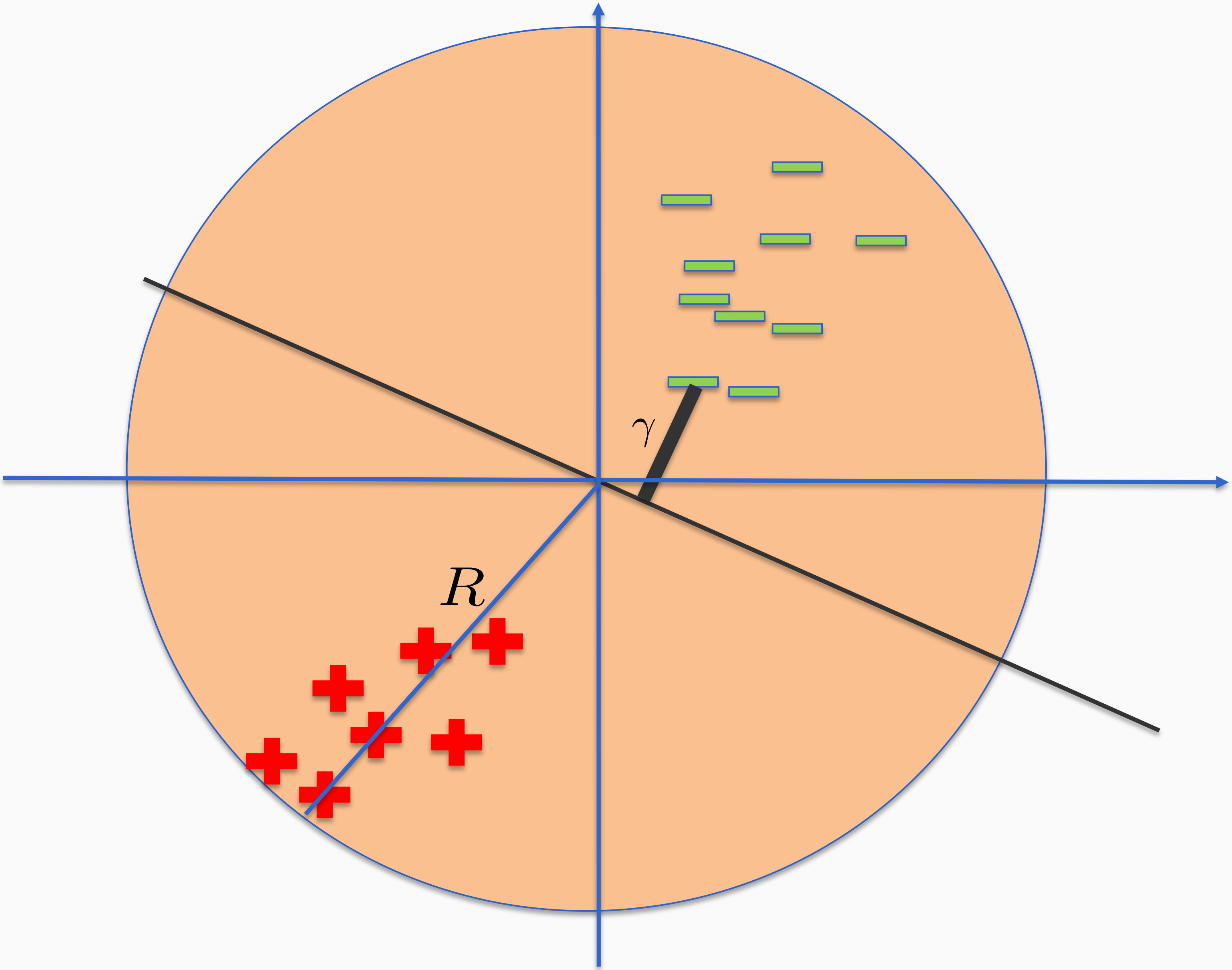
Mistake Bound Theorem [Novikoff 1962, Block 1962]



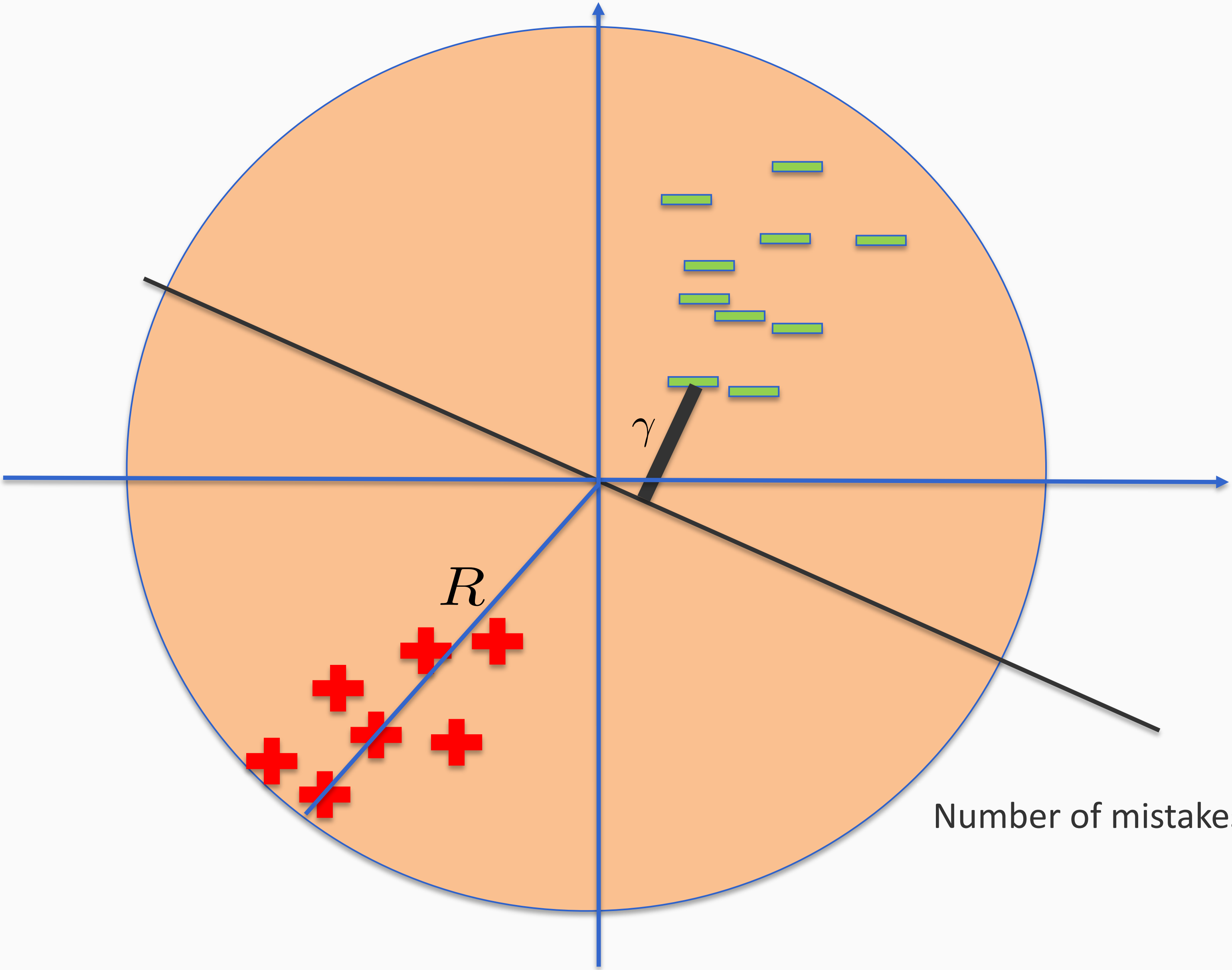
Mistake Bound Theorem [Novikoff 1962, Block 1962]



Mistake Bound Theorem [Novikoff 1962, Block 1962]



Mistake Bound Theorem [Novikoff 1962, Block 1962]



Number of mistakes $\leq \frac{R^2}{\gamma^2}$

The Perceptron Mistake bound

Number of mistakes $\leq \frac{R^2}{\gamma^2}$

- Exercises:
  - How many mistakes will the Perceptron algorithm make for disjunctions with n attributes?
    - What are R and γ?
  - How many mistakes will the Perceptron algorithm make for k-disjunctions with n attributes?

What you need to know

- What is the perceptron mistake bound?
- How to prove it

Where are we?

- The Perceptron Algorithm
- Perceptron Mistake Bound
- Variants of Perceptron

Practical use of the Perceptron algorithm

1. Using the Perceptron algorithm with a finite dataset

2. Margin Perceptron

3. Voting and Averaging

1. The "standard" algorithm

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

1. Initialize $\mathbf{w} = 0 \in \Re^n$
2. For epoch = 1 … T:
    1. Shuffle the data
    2. For each training example $(\mathbf{x}_i, y_i) \in D$:
        - If $y_i\, \mathbf{w}^T\mathbf{x}_i \leq 0$, update $\mathbf{w} \leftarrow \mathbf{w} + r\, y_i\, \mathbf{x}_i$
3. Return **w**

Prediction: $\text{sgn}(\mathbf{w}^T\mathbf{x})$

1. The "standard" algorithm

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

1. Initialize $\mathbf{w} = 0 \in \Re^n$
2. For epoch = 1 … T:
    1. Shuffle the data
    2. For each training example $(\mathbf{x}_i, y_i) \in D$:
        - If $y_i \mathbf{w}^T\mathbf{x}_i \leq 0$, update $\mathbf{w} \leftarrow \mathbf{w} + r\, y_i\, \mathbf{x}_i$
3. Return **w**

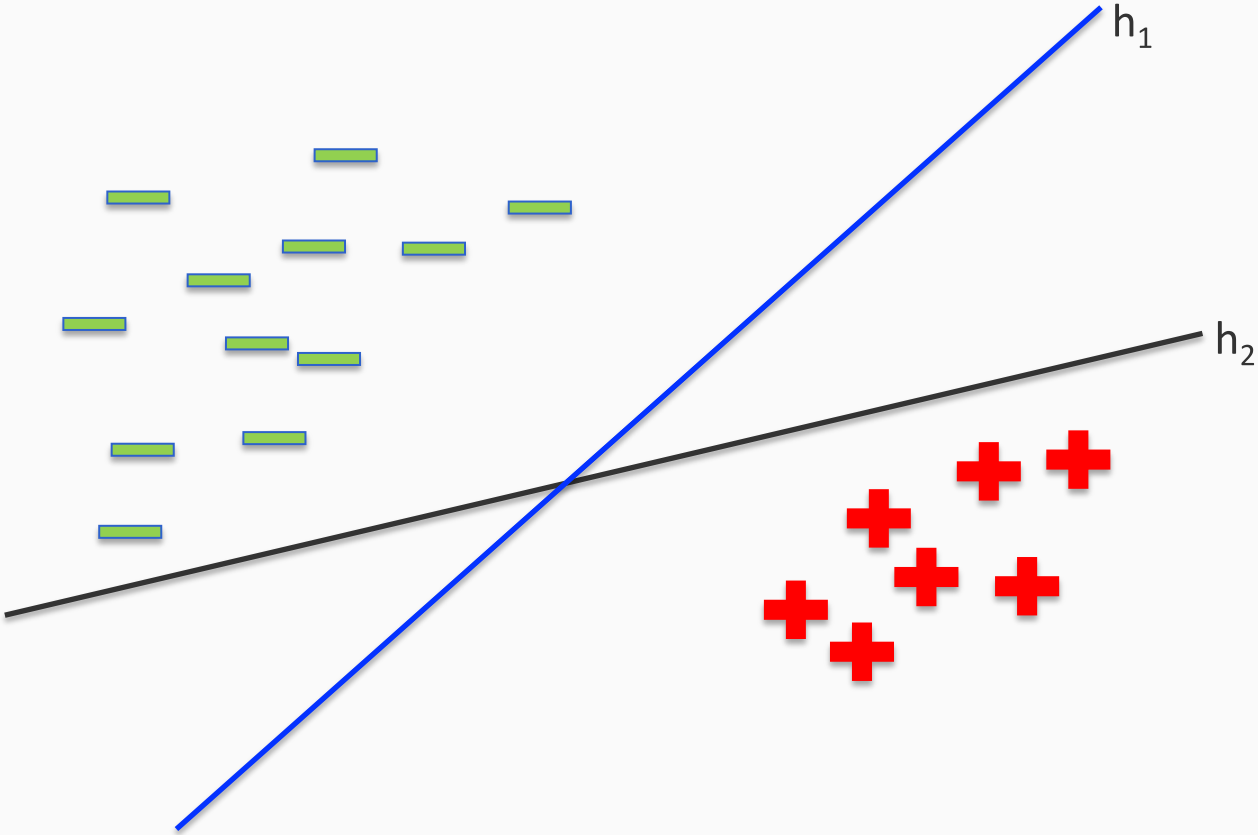
T is a hyper-parameter to the algorithm

Another way of writing that there is an error

Prediction: $\text{sgn}(\mathbf{w}^T\mathbf{x})$

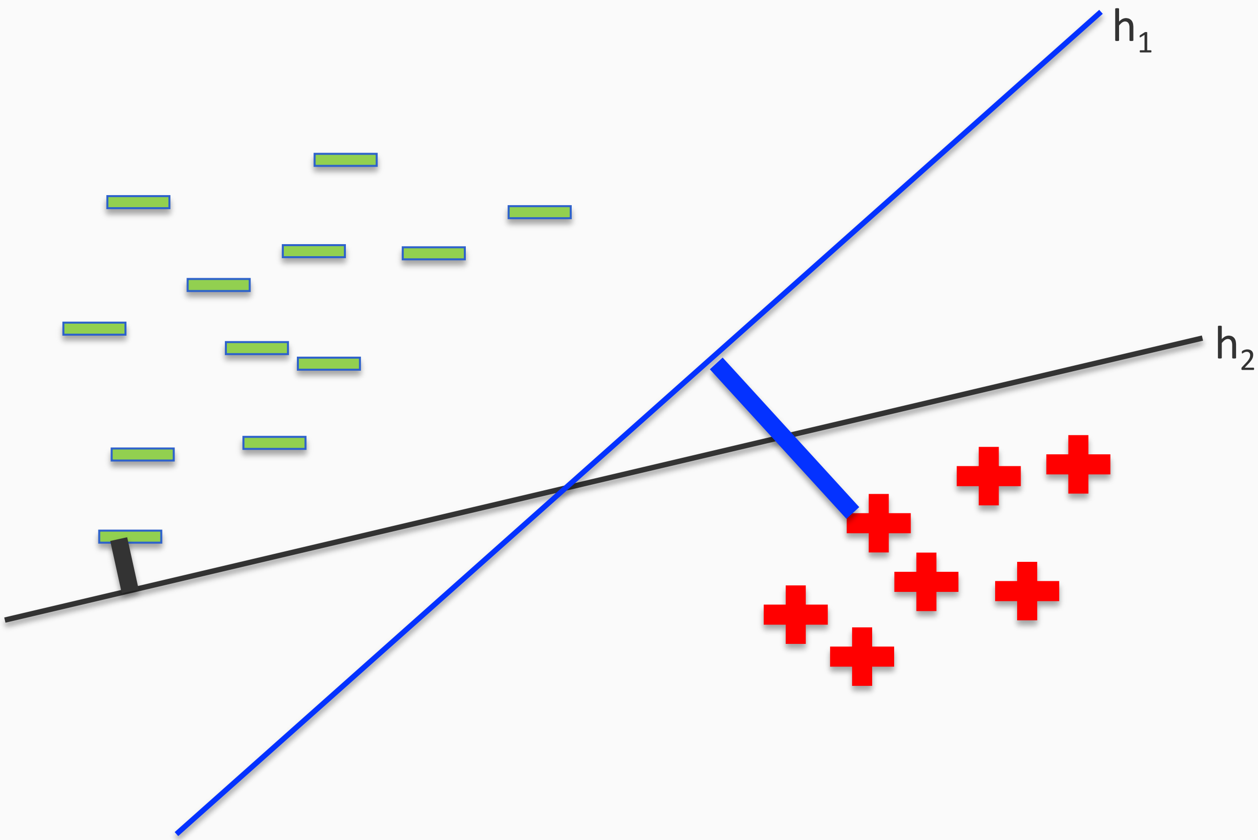
2. Margin Perceptron

Which hyperplane is better?



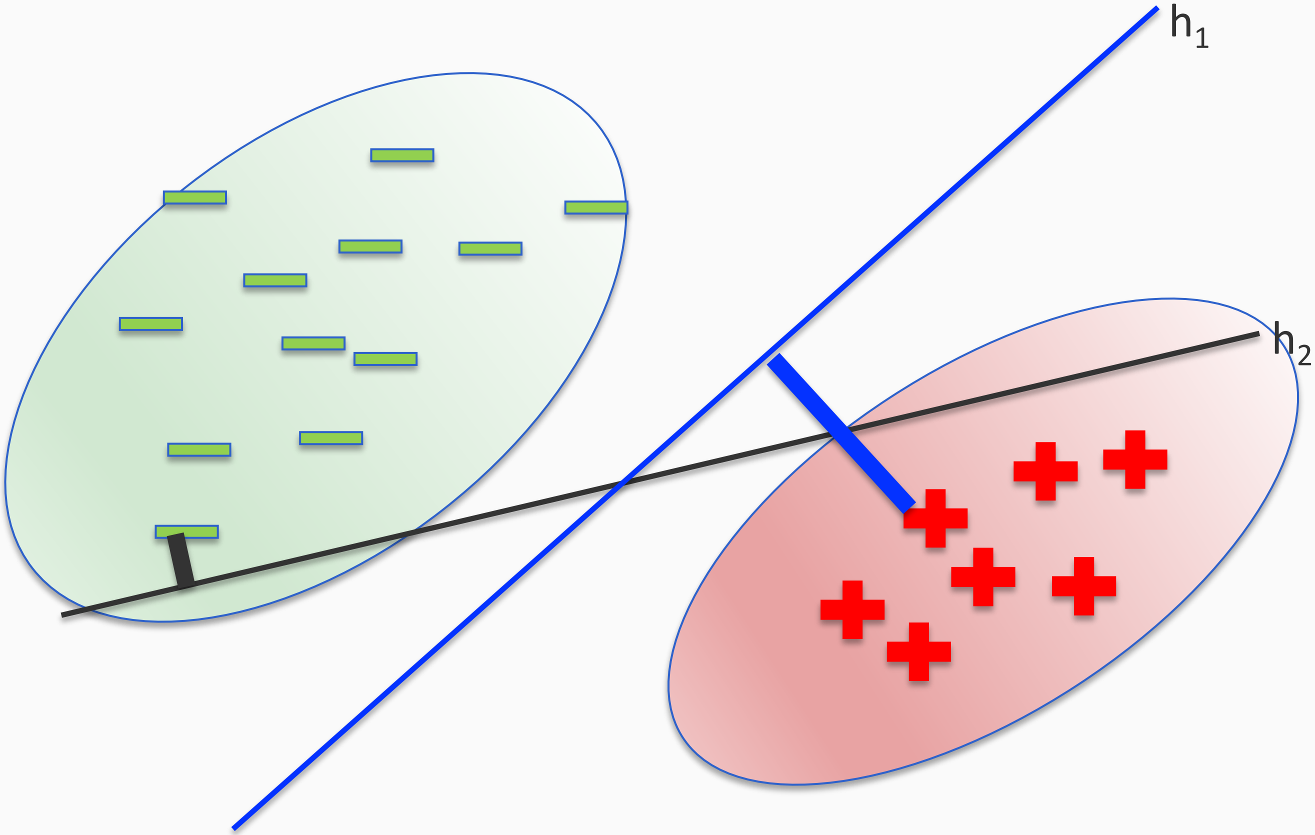
2. Margin Perceptron

Which hyperplane is better?



2. Margin Perceptron

Which hyperplane is better?



The farther from the data points, the less chance to make wrong prediction

2. Margin Perceptron

- Perceptron makes updates only when the prediction is incorrect

$$y_i \mathbf{w}^\top \mathbf{x}_i < 0$$

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

$$\frac{y_i \mathbf{w}^\top \mathbf{x}_i}{\|\mathbf{w}\|} < \eta$$

- Can generalize better, but need to choose
  - Why is this a good idea?

2. Margin Perceptron

- Perceptron makes updates only when the prediction is incorrect

  $$y_i \mathbf{w}^\top \mathbf{x}_i < 0$$

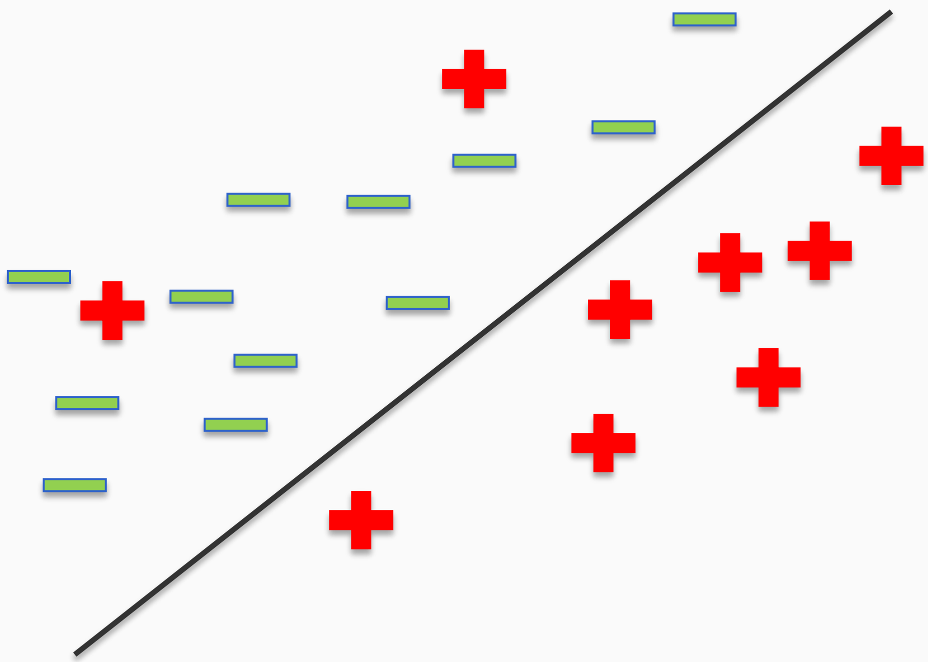
- What if the prediction is close to being incorrect? That is, Pick a positive η and update when

  $$\frac{y_i \mathbf{w}^\top \mathbf{x}_i}{\|\mathbf{w}\|} < \eta$$

- Can generalize better, but need to choose
  - Why is this a good idea? Intentionally set a large margin

3. Voting and Averaging

What if data is not linearly separable?



Finding a hyperplane with minimum mistakes is **NP hard**

Voted Perceptron

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

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

2. For epoch = 1 … T:
   - For each training example $(\mathbf{x}_i, y_i) \in D$:
     - If $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$
       - update $\mathbf{w}_{m+1} \leftarrow \mathbf{w}_m + r\, y_i\, \mathbf{x}_i$
       - m=m+1
       - $C_m = 1$
     - Else
       - $\mathbf{C}_m = C_m + 1$
3. Return $(\mathbf{w}_1, \mathbf{c}_1), (\mathbf{w}_2, \mathbf{c}_2), \ldots, (\mathbf{w}_k, \mathbf{C}_k)$

Prediction: $$\text{sgn}\Big(\sum_{i=1}^{k} c_i \cdot \text{sgn}(\mathbf{w}_i^\top \mathbf{x})\Big)$$

Averaged Perceptron

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

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

2. For epoch = 1 … T:
   - For each training example $(\mathbf{x}_i, y_i) \in D$:
     - If $y_i \mathbf{w}^T\mathbf{x}_i \leq 0$
       - update $\mathbf{w} \leftarrow \mathbf{w} + r\, y_i\, \mathbf{x}_i$
     - $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{w}$

3. Return **a**

Prediction: $\text{sgn}(\mathbf{a}^T\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^{k} c_i \mathbf{w}_i^\top \mathbf{x}\right)$

Averaged Perceptron

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

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

2. For epoch = 1 … T:
   – For each training example ($\mathbf{x}_i$, y_i) ∈ D:
     - If $y_i \mathbf{w}^T\mathbf{x}_i \leq 0$
       – update $\mathbf{w} \leftarrow \mathbf{w} + r\, y_i\, \mathbf{x}_i$
     - $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{w}$

3. Return **a**

This is the simplest version of the averaged perceptron

There are some easy programming tricks to make sure that **a** is also updated only when there is an error

Prediction: $\mathrm{sgn}(\mathbf{a}^T\mathbf{x}) = \mathrm{sgn}\left(\sum_{i=1}^{k} c_i \mathbf{w}_i^\top \mathbf{x}\right)$

Averaged Perceptron

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

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

2. For epoch = 1 … T:
   - For each training example $(\mathbf{x}_i, y_i) \in D$:
     - If $y_i \, \mathbf{w}^T\mathbf{x}_i \leq 0$
       - update $\mathbf{w} \leftarrow \mathbf{w} + r \, y_i \, \mathbf{x}_i$
     - $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{w}$

3. Return **a**

Prediction: $\text{sgn}(\mathbf{a}^T\mathbf{x})$ $= \text{sgn}\left(\sum_{i=1}^{k} c_i \mathbf{w}_i^\top \mathbf{x}\right)$

This is the simplest version of the averaged perceptron

There are some easy programming tricks to make sure that **a** is also updated only when there is an error

Extremely popular

If you want to use the Perceptron algorithm, use averaging

Question: What is the difference?

Voted: $$\operatorname{sgn}\Big(\sum_{i=1}^{k} c_i \cdot \operatorname{sgn}(\mathbf{w}_i^\top \mathbf{x})\Big)$$

Averaged: $$\operatorname{sgn}\Big(\sum_{i=1}^{k} c_i \mathbf{w}_i^\top \mathbf{x}\Big)$$

$$\mathbf{w}_1^\top \mathbf{x} = s_1, \mathbf{w}_2^\top \mathbf{x} = s_2, \mathbf{w}_3^\top \mathbf{x} = s_3 \qquad c_1 = c_2 = c_3 = 1$$

Averaged: $s_1 + s_2 + s_3 \geq 0$

Voted: Any two are positive

Summary: Perceptron

- Online learning algorithm, very widely used, easy to implement
- Additive updates to weights
- Geometric interpretation
- Mistake bound
- Practical variants abound
- You should be able to implement the Perceptron algorithm