The Perceptron Algorithm

Machine Learning

THE UNIVERSITY OF UTAH

Some slides based on lectures from Dan Roth, Avrim Blum and others

Outline

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

Where are we?

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

Recall: Linear Classifiers

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 the following classification rule

$$\text{Output} = \text{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b) = \text{sign}(\textstyle\sum_i w_i x_i + b)$$

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

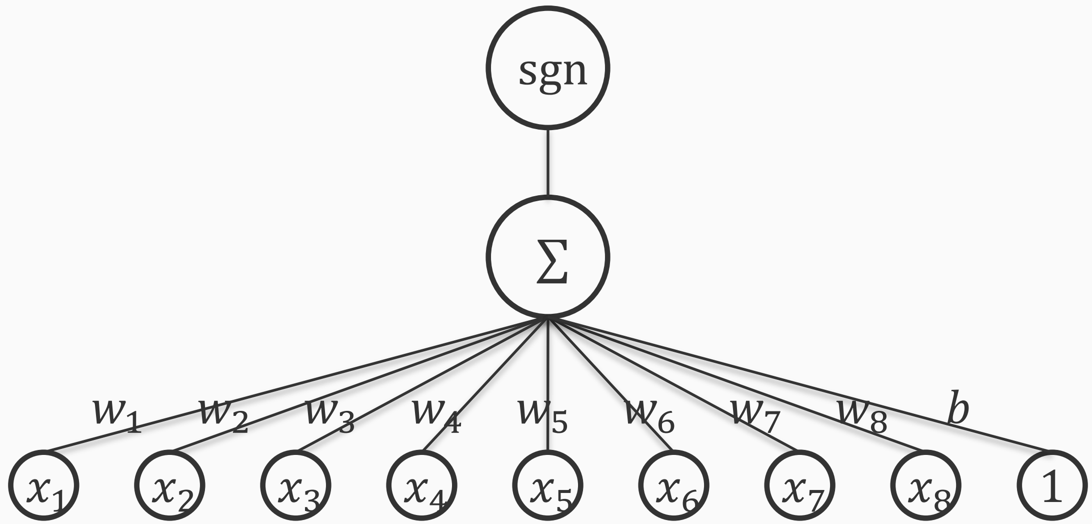
b is called the bias term

Recall: Linear Classifiers

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

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

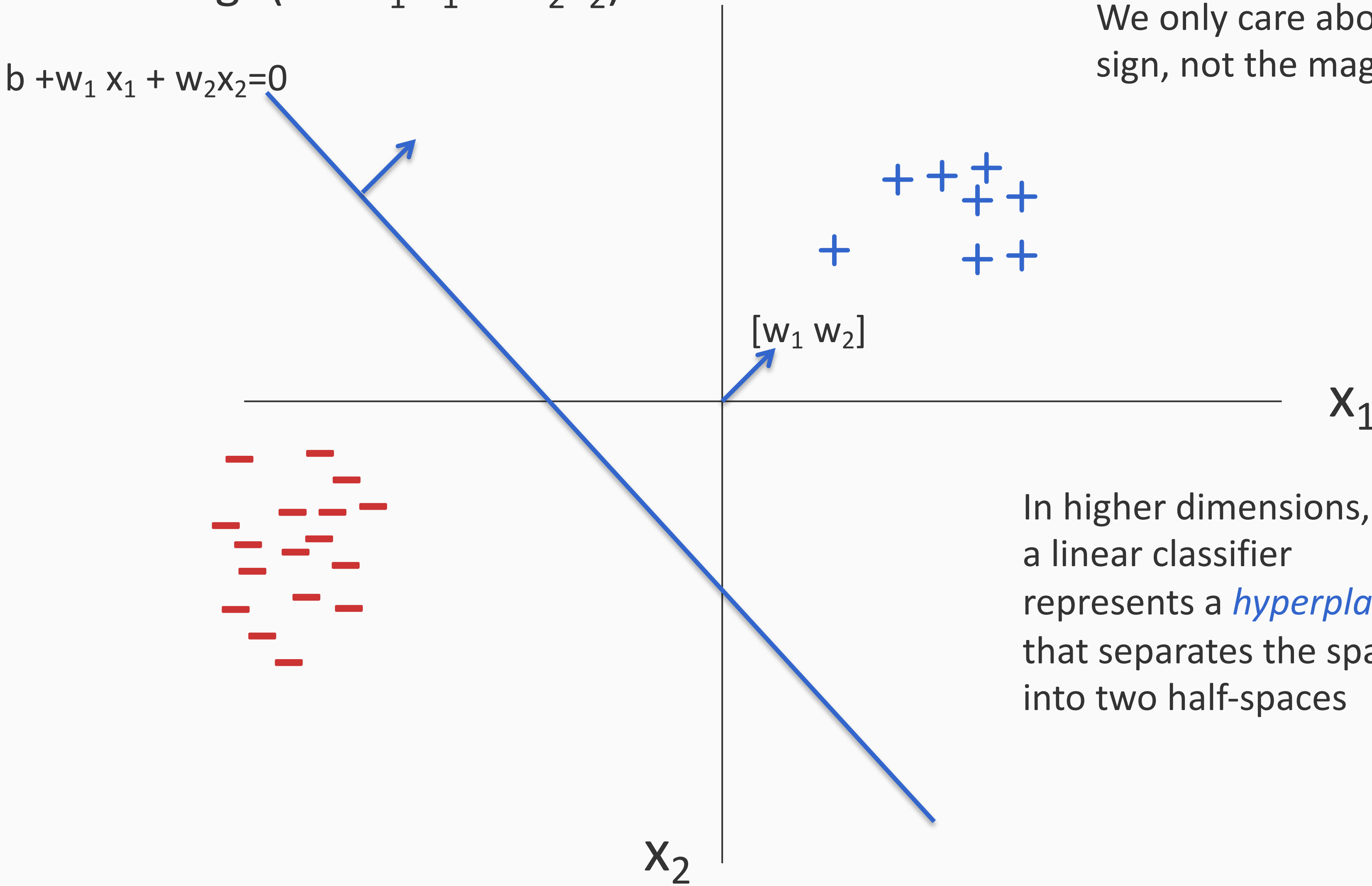
Linear Threshold Units classify an example $\mathbf{x}$ using parameters $\mathbf{w}$ (a d dimensional vector) and b (a real number) according the following classification rule



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

The Perceptron

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 Perceptron algorithm

- Introduced by Rosenblatt (1958)
  - Though there were some hints of a similar idea earlier
    - Agmon (1954), Motzkin and Schonberg (1954)
- The goal is to find a separating hyperplane
  - For separable data, guaranteed to find one
- An *online* algorithm
  - That is, it processes one example at a time
- A mistake-driven algorithm
  - That is, it makes updates if, and only if, it makes a mistake on an example
- Several variants exist
  - We will see these 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 $\mathbf{x}_i \in \Re^d$, $y_i \in \{-1, 1\}$

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

The Perceptron algorithm

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

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

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

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

Remember:
Prediction = 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 **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 $\mathbf{x}_i \in \Re^d, y_i \in \{-1, 1\}$

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

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

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

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

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

The Perceptron algorithm

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

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

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

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 $\mathbf{x}_i \in \Re^d, y_i \in \{-1, 1\}$

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

Mistake 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

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

The Perceptron algorithm

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

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

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

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

Update only on error. A mistake-driven algorithm

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

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

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

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

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

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

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

The new dot product is $\mathbf{w}_{\mathrm{t+1}}^{\mathrm{T}}\mathbf{x} = (\mathbf{w}_t + \mathbf{x})^{\mathrm{T}}\mathbf{x} = \mathbf{w}_t^{\mathrm{T}}\mathbf{x} + \mathbf{x}^{\mathbf{T}}\mathbf{x} \geq \mathbf{w}_{\mathbf{t}}^{\mathbf{T}}\mathbf{x}$

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

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

The new dot product is $\mathbf{w}_{\mathrm{t+1}}^{\mathrm{T}}\mathbf{x} = (\mathbf{w}_t + \mathbf{x})^{\mathrm{T}}\mathbf{x} = \mathbf{w}_t^{\mathrm{T}}\mathbf{x} + \mathbf{x}^{\mathbf{T}}\mathbf{x} \geq \mathbf{w}_{\mathbf{t}}^{\mathbf{T}}\mathbf{x}$

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

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

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

The new dot product is $\mathbf{w}_{\mathrm{t+1}}^{\mathrm{T}}\mathbf{x} = (\mathbf{w}_t + \mathbf{x})^{\mathrm{T}}\mathbf{x} = \mathbf{w}_t^{\mathrm{T}}\mathbf{x} + \mathbf{x}^{\mathbf{T}}\mathbf{x} \geq \mathbf{w}_{\mathbf{t}}^{\mathbf{T}}\mathbf{x}$

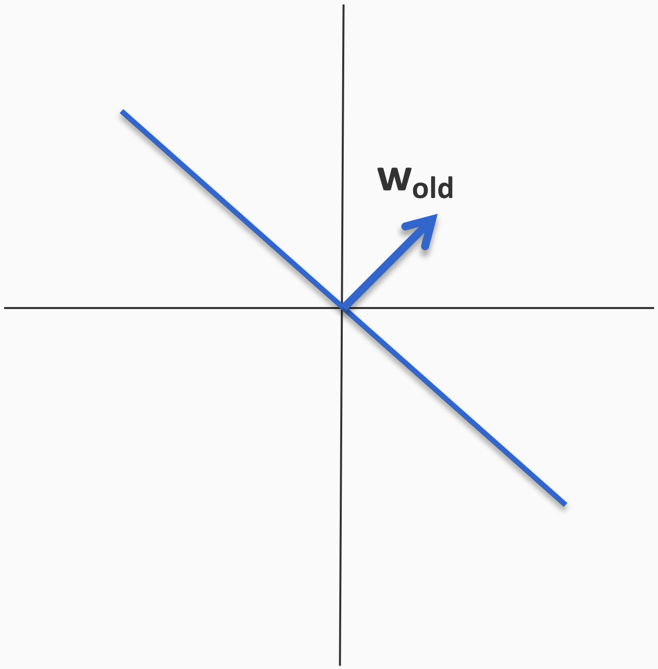
For a positive example, the Perceptron update will increase the score assigned to the same input

Similar reasoning for negative examples

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

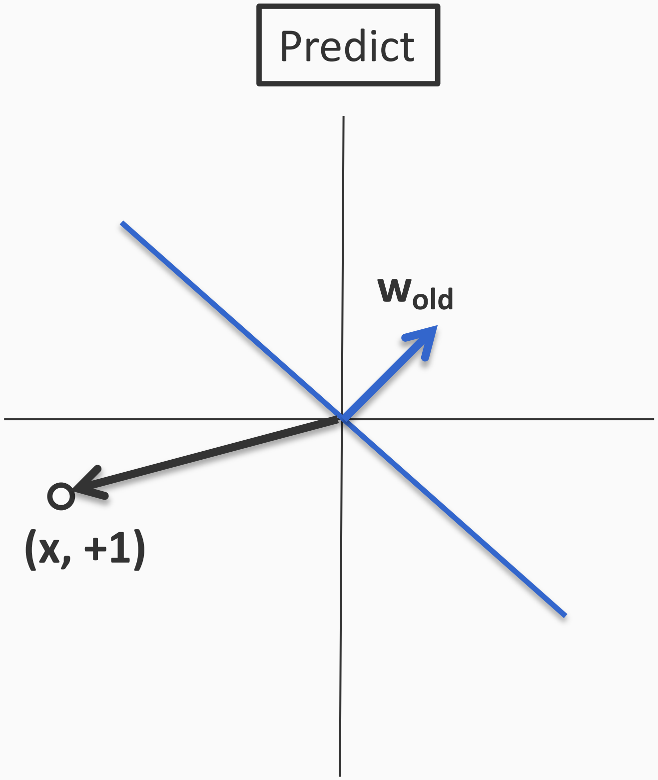
Geometry of the perceptron update

Predict



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

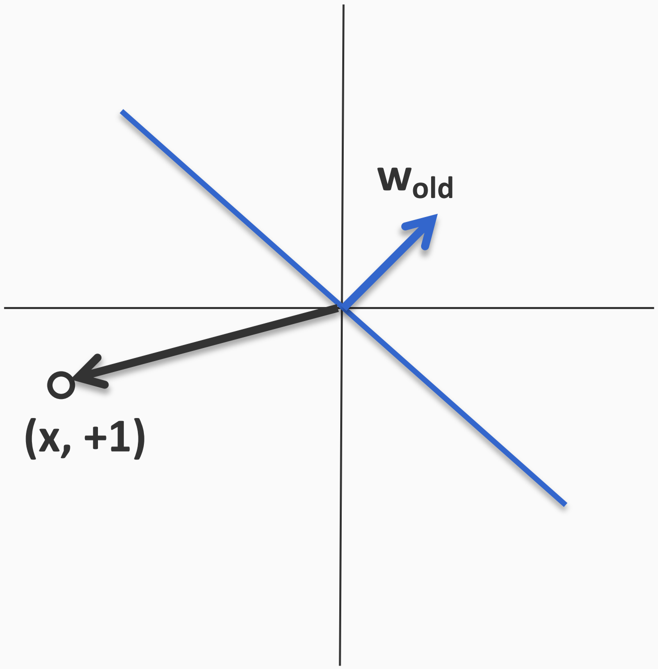
Geometry of the perceptron update



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

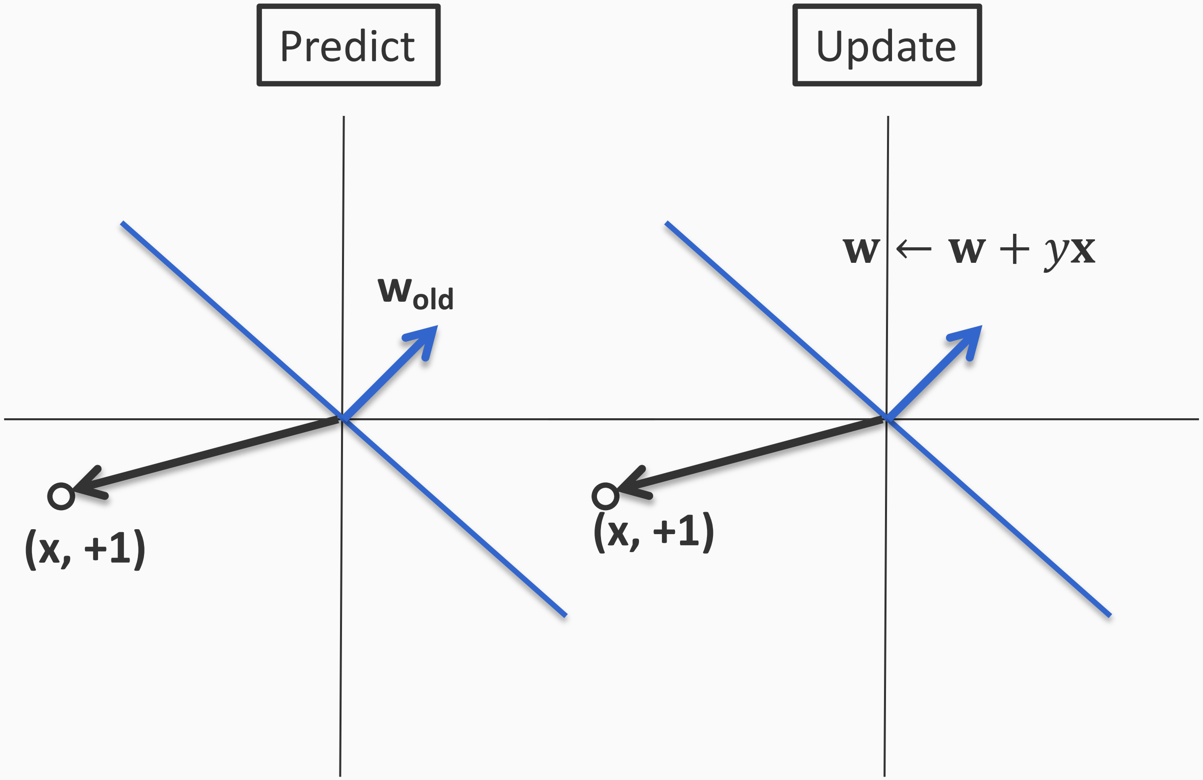
Predict



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

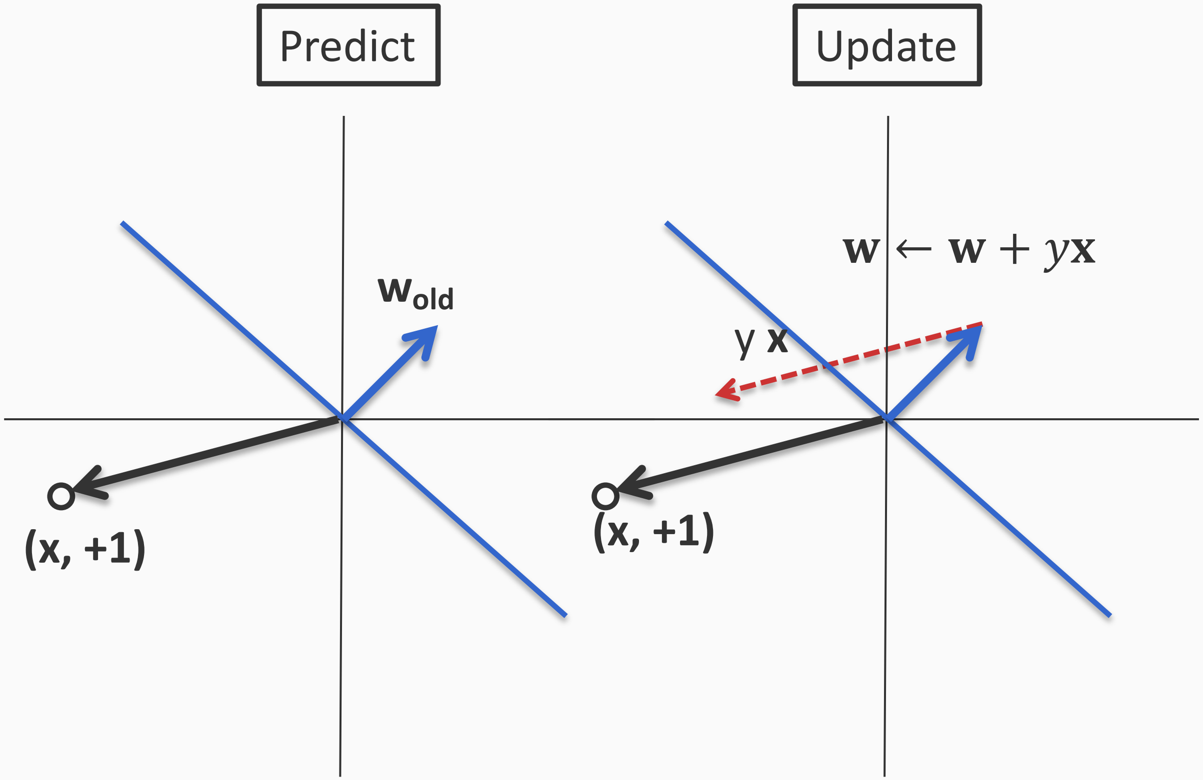
Geometry of the perceptron update



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

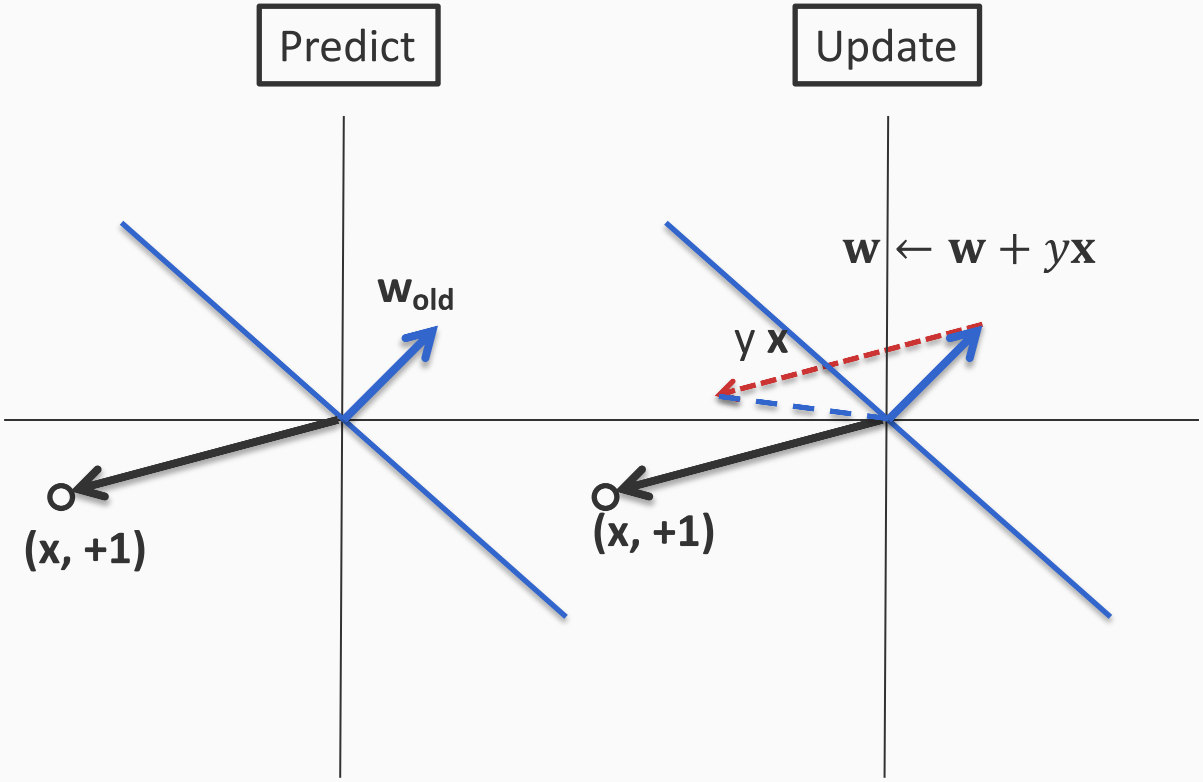
Geometry of the perceptron update



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

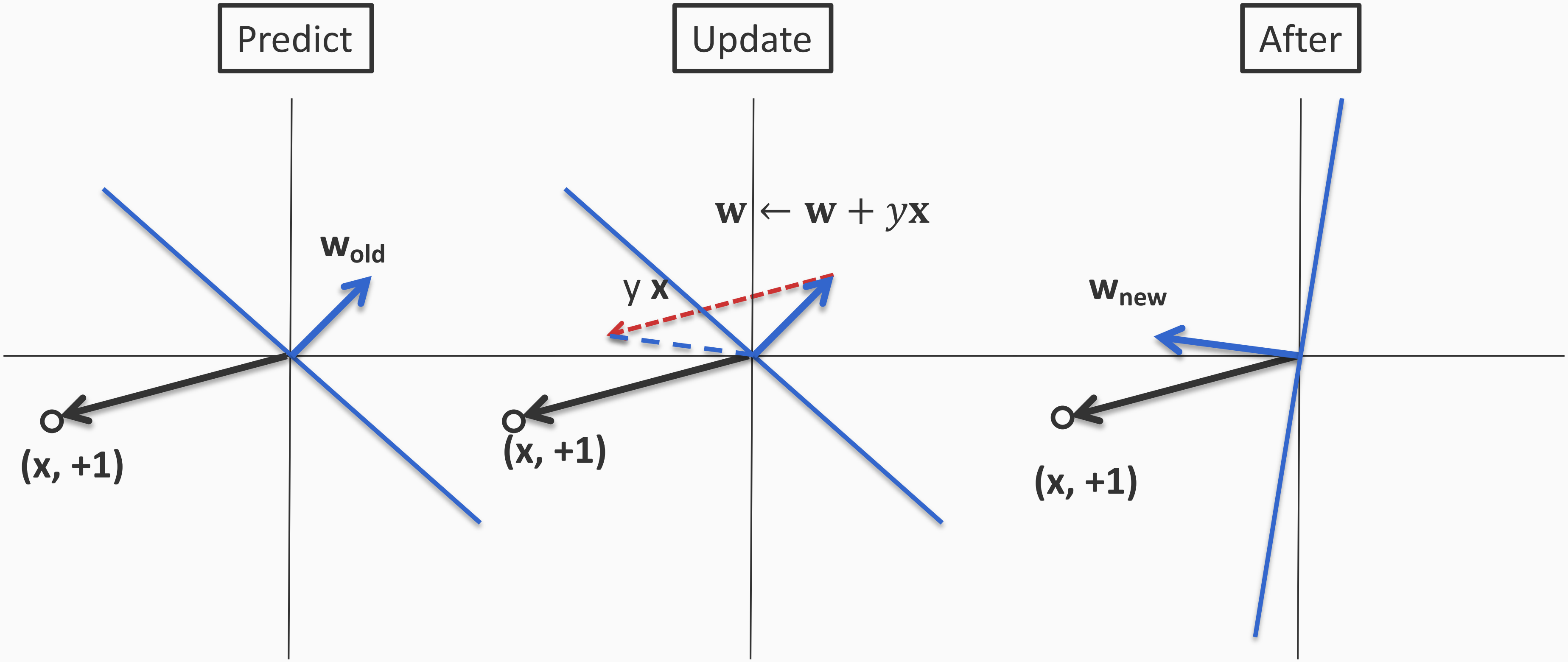
Geometry of the perceptron update



For a mistake on a positive example

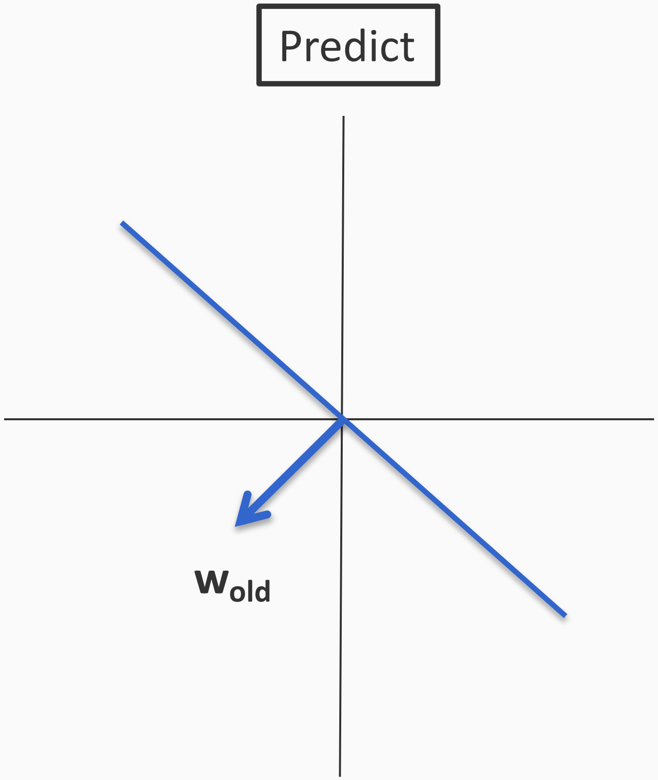
Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Geometry of the perceptron update

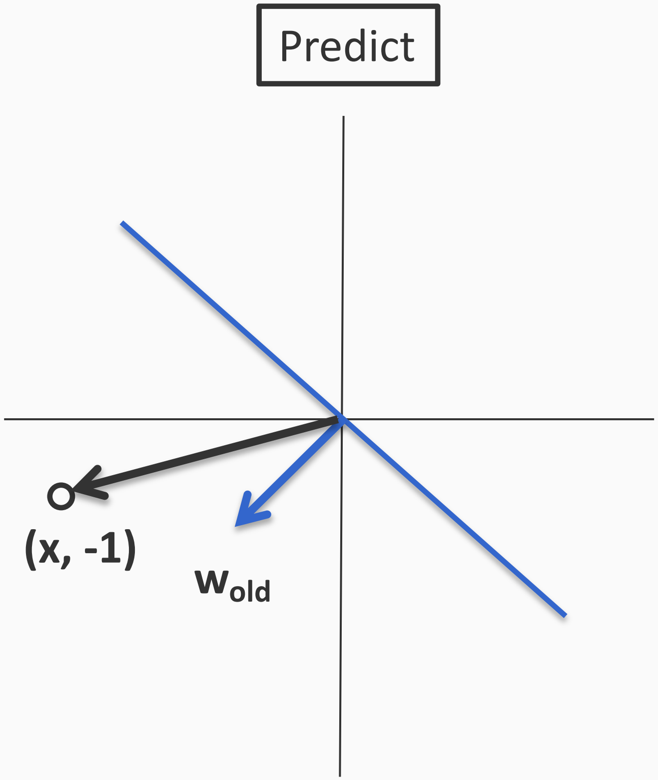


For a mistake on a positive example

Geometry of the perceptron update

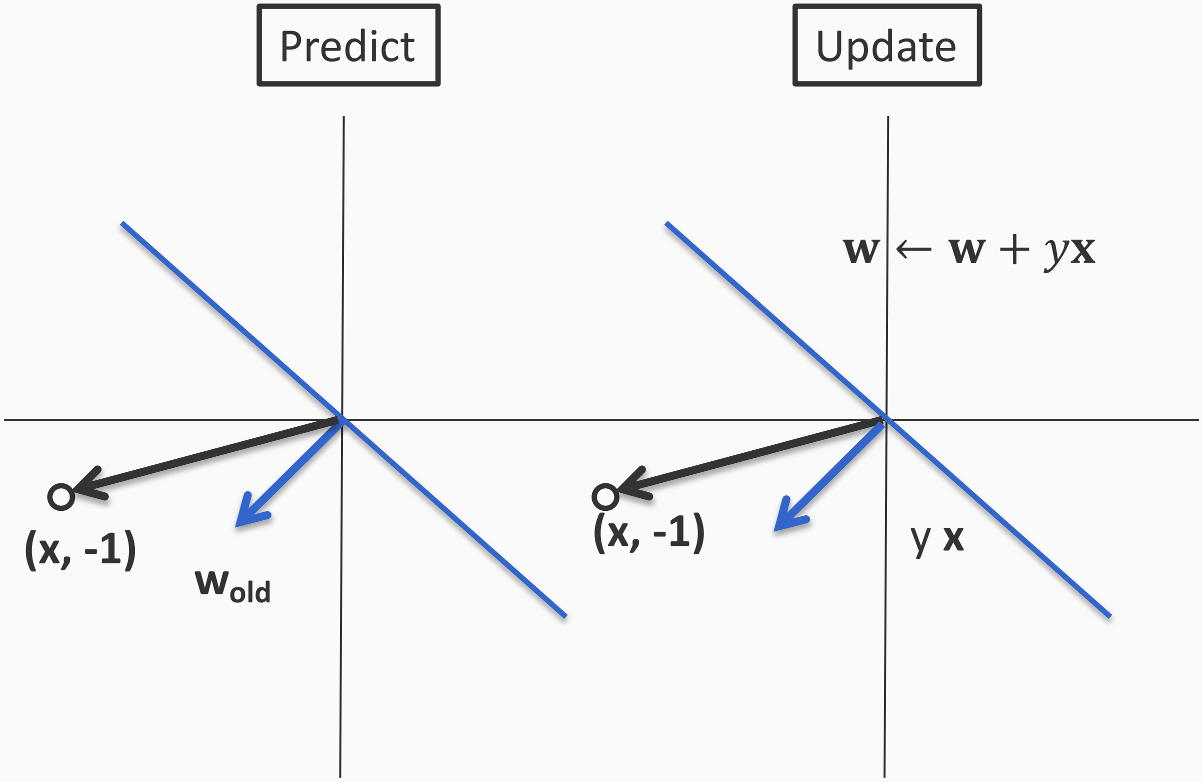


Geometry of the perceptron update



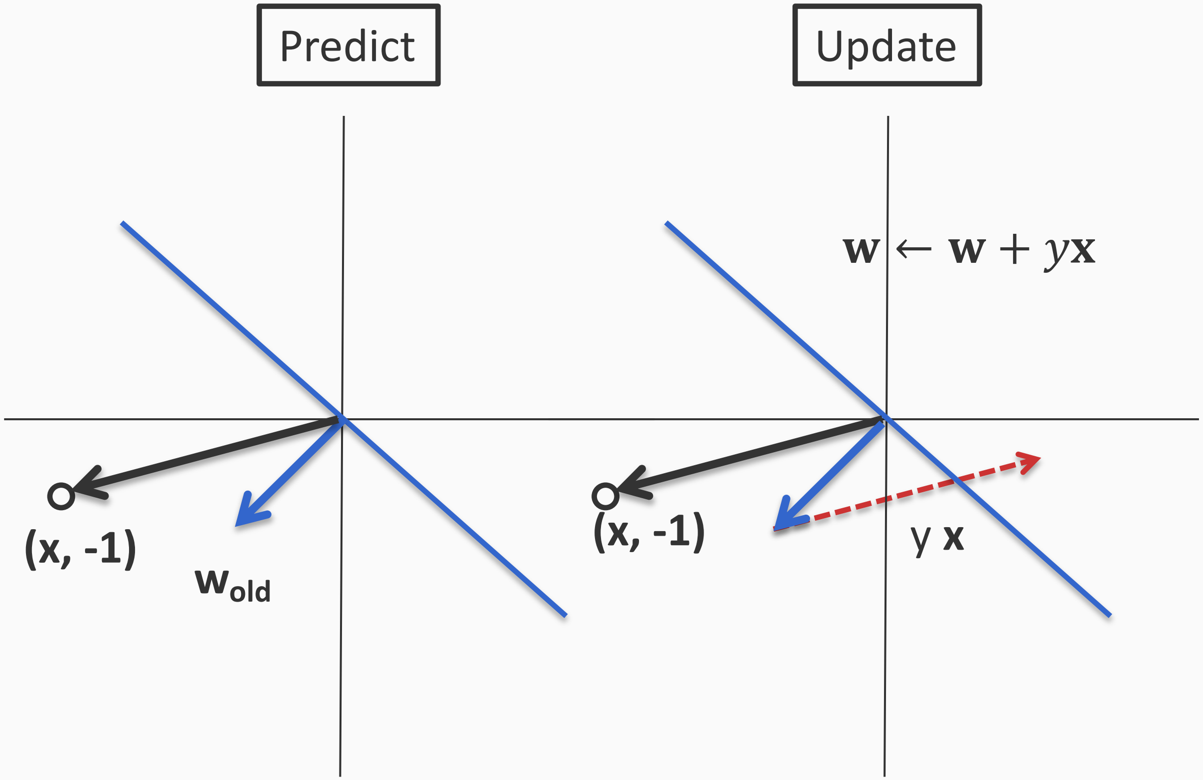
For a mistake on a negative example

Geometry of the perceptron update



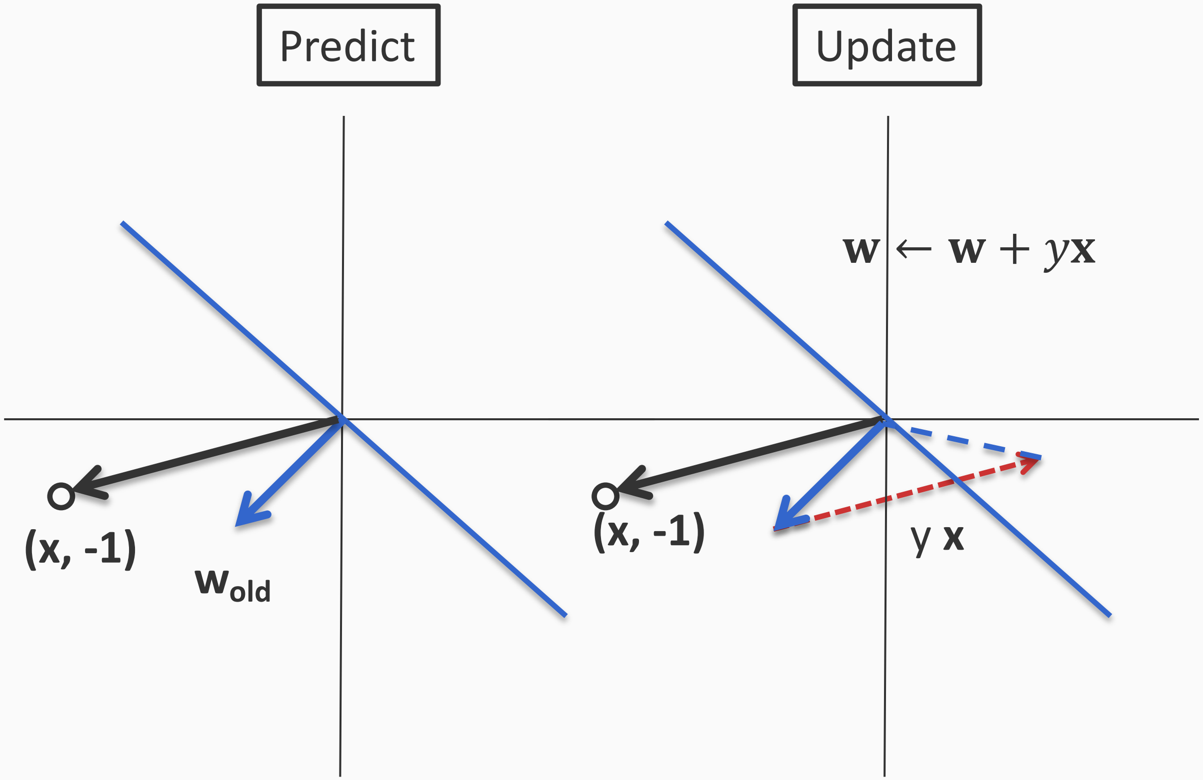
For a mistake on a negative example

Geometry of the perceptron update



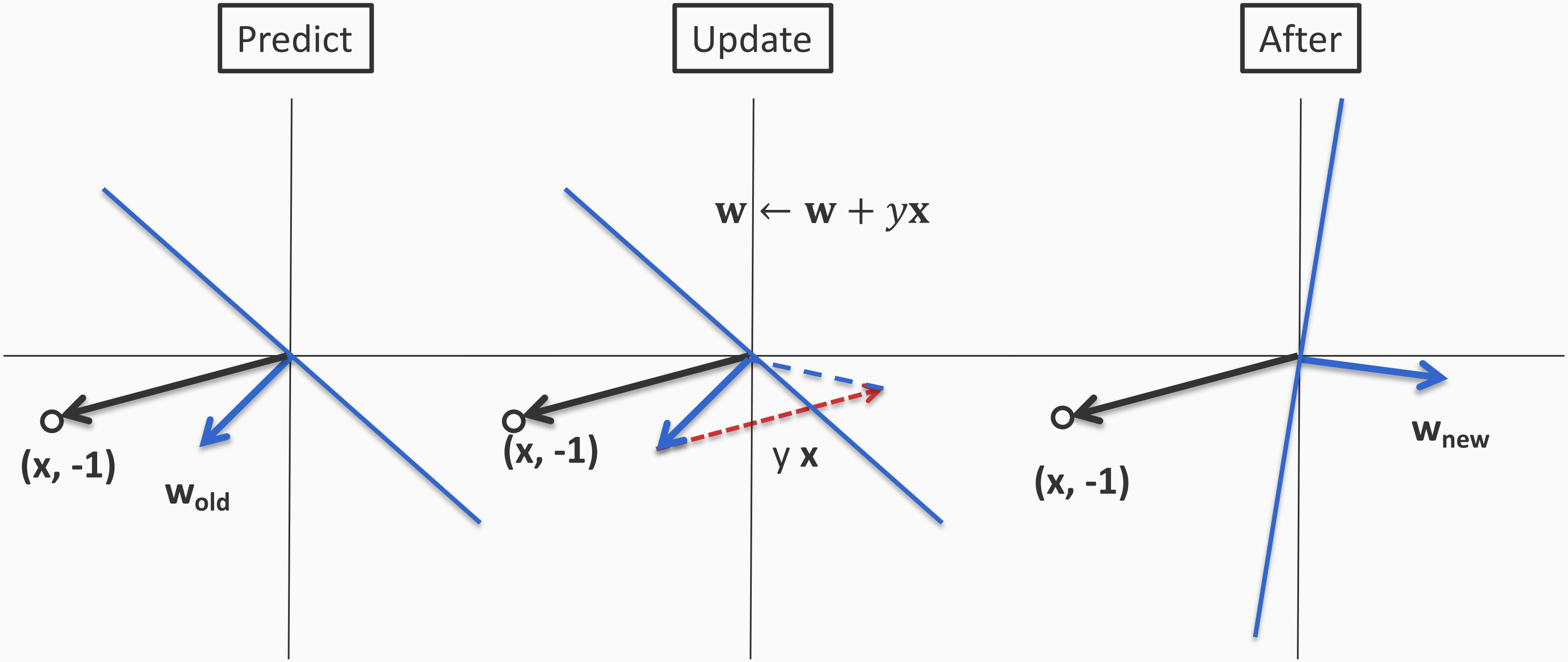
For a mistake on a negative example

Geometry of the perceptron update



For a mistake on a negative example

Geometry of the perceptron update



For a mistake on a negative example

Where are we?

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

Practical use of the Perceptron algorithm

1. Using the Perceptron algorithm with a finite dataset

2. Voting and Averaging

3. Margin Perceptron

1. The "standard" algorithm

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

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

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

1. The "standard" algorithm

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

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

T is a hyper-parameter to the algorithm

Another way of writing that there is an error

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

2. Voting and Averaging

- So far: We return the final weight vector

- 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

2. Voting and Averaging

- So far: We return the final weight vector

- 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

Averaged Perceptron

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

1. Initialize $\mathbf{w} = \mathbf{0} \in \Re^d$ and $\boxed{\mathbf{a} = \mathbf{0} \in \Re^d}$
2. For epoch in $1 \cdots T$:
   1. Shuffle the data
   2. For each training example $(\mathbf{x}_i, y_i) \in D$:
      - If $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$, then:
        – update $\mathbf{w} \leftarrow \mathbf{w} + r y_i \mathbf{x}_i$
      - $\boxed{\mathbf{a} \leftarrow \boldsymbol{a} + \mathbf{w}}$
3. Return $\mathbf{a}$

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

Averaged Perceptron

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

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

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

Averaged Perceptron

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

1. Initialize $\mathbf{w} = \mathbf{0} \in \Re^d$ and $\boxed{\mathbf{a} = \mathbf{0} \in \Re^d}$
2. For epoch in $1 \cdots T$:
   1. Shuffle the data
   2. For each training example $(\mathbf{x}_i, y_i) \in D$:
      - If $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$, then:
        – update $\mathbf{w} \leftarrow \mathbf{w} + r y_i \mathbf{x}_i$
      - $\boxed{\mathbf{a} \leftarrow \boldsymbol{a} + \mathbf{w}}$
3. Return $\mathbf{a}$

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

If you want to use the Perceptron algorithm, use averaging

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

3. 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? That is, Pick a small positive η and update when

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

- Can generalize better, but extra hyper-parameter η

  Exercise: Why is the margin perceptron a good idea?

The Perceptron

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 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.

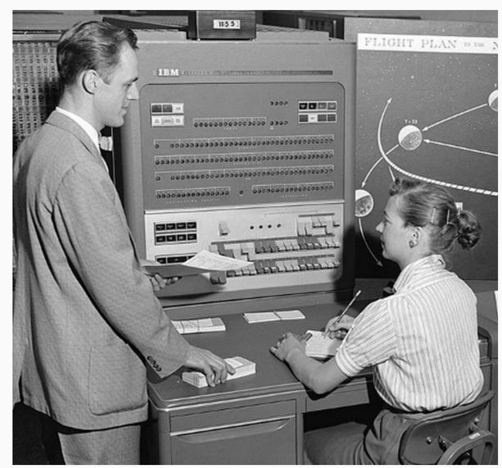
The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

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

What you need to know

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