Lecture 11: Perceptrons

COMP 411, Fall 2021

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) and Dan Roth (Penn)

Today's Topics

Perceptron

- Overview
- Perceptron learning algorithm
- Geometry of algorithm update
- Learnability

vumanfredi@wesleyan.edu

Perceptron

OVERVIEW

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 to the following classification rule

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

$$\text{if } \mathbf{w}^T\mathbf{x} + b \geq 0 \Rightarrow y = +1$$

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

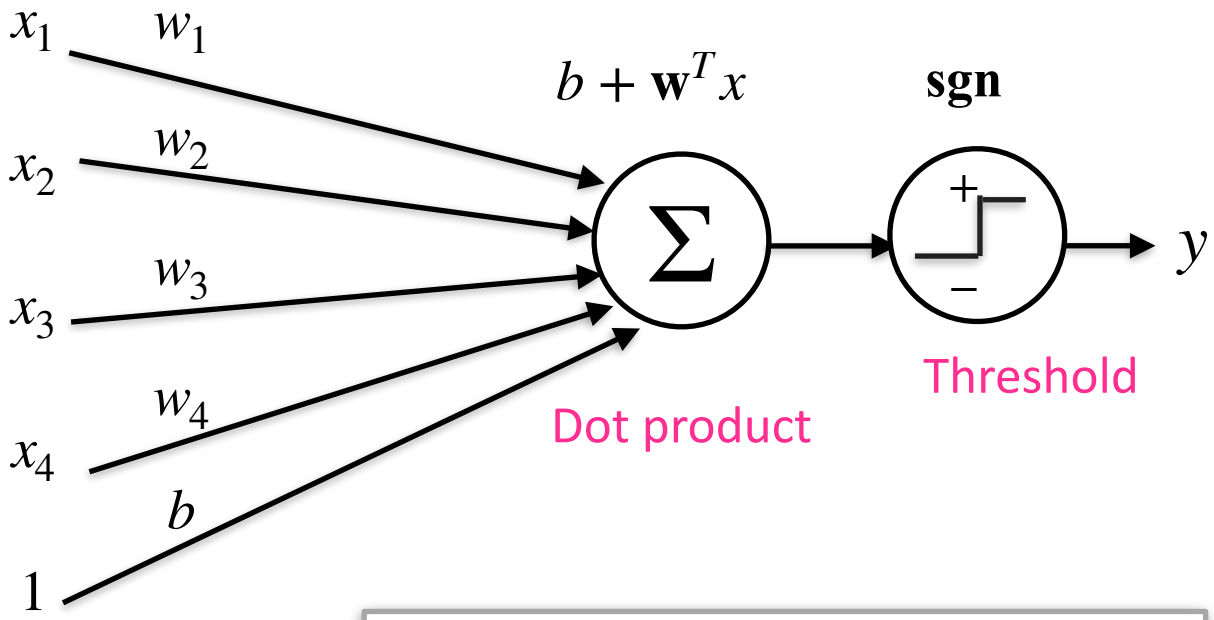
b is called the *bias* term

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 to the following classification rule



Perceptron = Linear Threshold Unit

Linear Threshold Functions

Many functions are Linear

- Conjunctions:
  - $y = x_1 \wedge x_3 \wedge x_5$
  - $y = \text{sgn}\{1 \cdot x_1 + 1 \cdot x_3 + 1 \cdot x_5 - 3\}; \quad \mathbf{w} = (1,0,1,0,1) \quad b = 3$
- At least m of n:
  - $y =$ at least 2 of $\{x_1, x_3, x_5\}$
  - $y = \text{sgn}\{1 \cdot x_1 + 1 \cdot x_3 + 1 \cdot x_5 - 2\}; \quad \mathbf{w} = (1,0,1,0,1) \quad b = 2$

Many functions are not

- Xor: $y = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$
- Non trivial DNF: $y = (x_1 \wedge x_2) \vee (x_3 \wedge x_4)$

But can be made linear

Note: all the variables above are Boolean variables

Dot product

The dot product of two vectors is written as $\mathbf{m}^T\mathbf{x}$ or $\mathbf{m} \cdot \mathbf{x}$, which is defined as: $\mathbf{m}^T\mathbf{x} = \sum_{i=1}^{k} m_i x_i$

Example

$$\mathbf{m} = < 5.13, 1.08, -0.03, 7.29 >$$

$$\mathbf{x} = < x_1, x_2, x_3, x_4 >$$

$$\mathbf{m}^T\mathbf{x} = 5.13x_1 + 1.08x_2 - 0.03x_3 + 7.29x_4$$

If dot product of two vectors is zero: means the two vectors are perpendicular (90° angle)

Length or norm of a vector

The length or norm of a vector $\mathbf{v}$ is the square root of length $= ||\mathbf{v}|| = \sqrt{\mathbf{v} \cdot \mathbf{v}}$

Dot product here is not zero ($\mathbf{v}$ is not perpendicular to itself), so now have 0° angle: do
product of $\mathbf{v} \cdot \mathbf{v}$ gives length of $\mathbf{v}$ squared

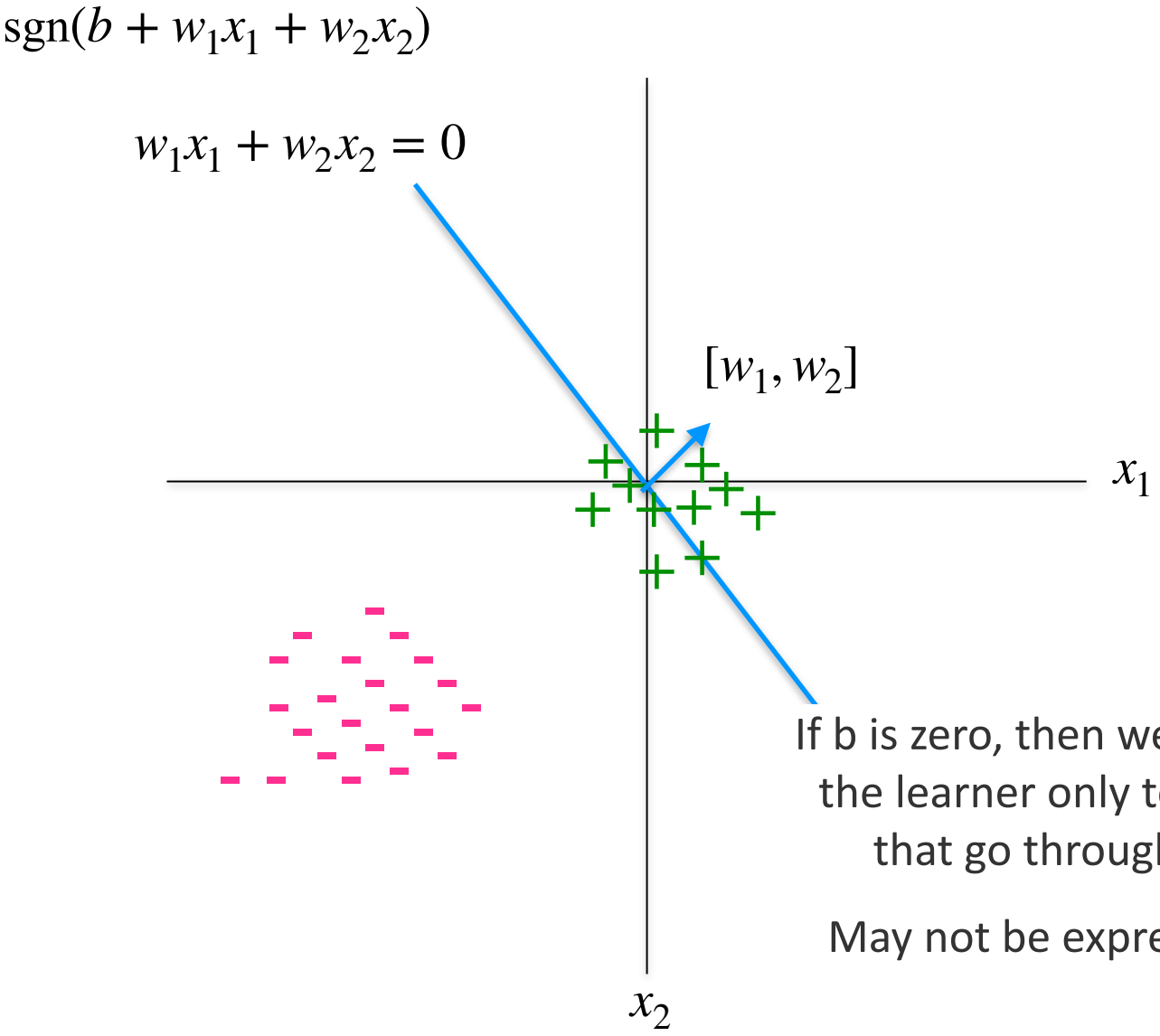
In 2 dimensions: length $= ||\mathbf{v}|| = \sqrt{v_1^2 + v_2^2}$

In 3 dimensions: length $= ||\mathbf{v}|| = \sqrt{v_1^2 + v_2^2 + v_e^2}$

See Introduction to Linear Algebra by Gilbert Strang

Why is the bias term needed?

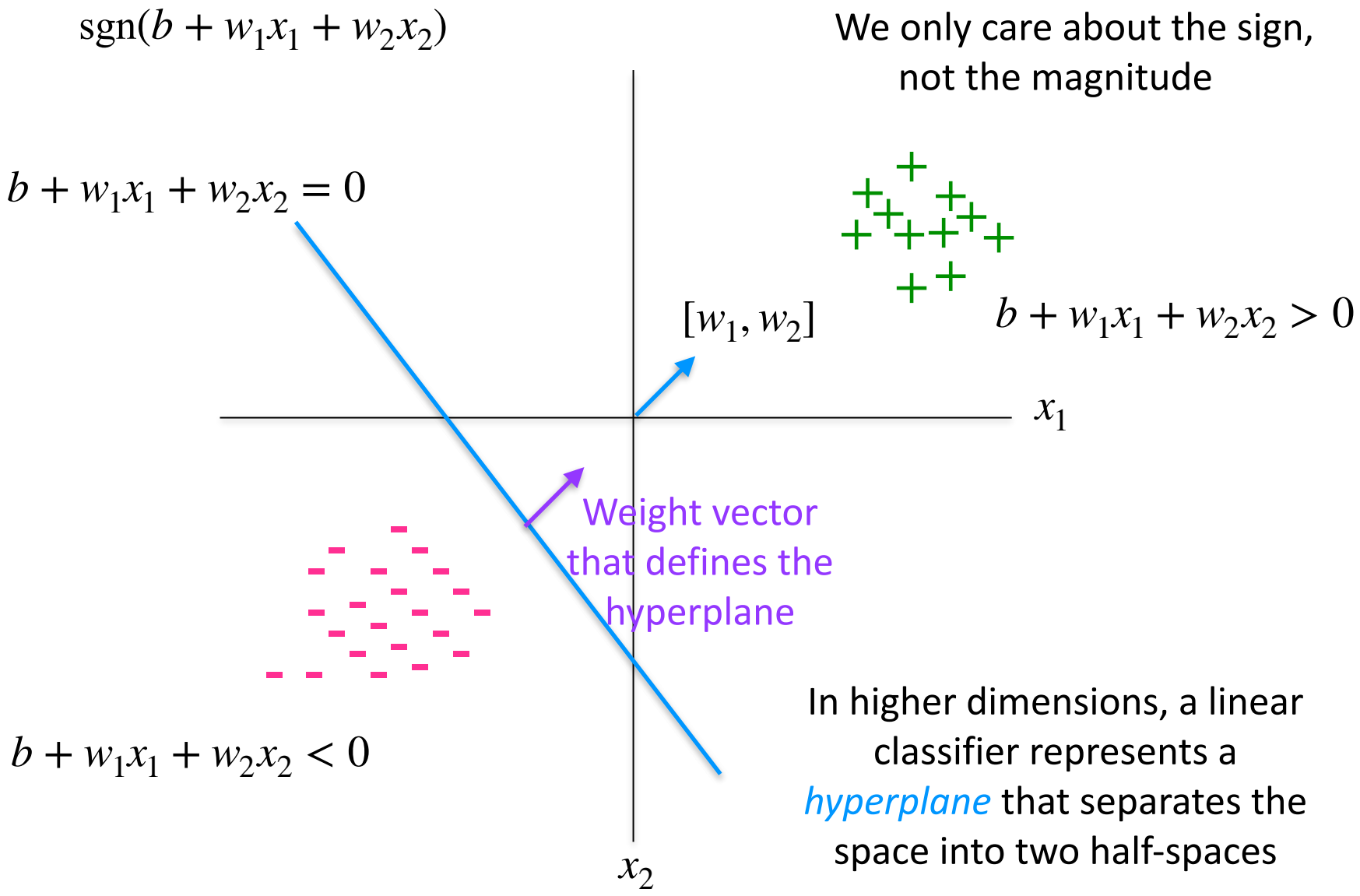
An illustration in two dimensions



If b is zero, then we are restricting the learner only to hyperplanes that go through the origin

May not be expressive enough

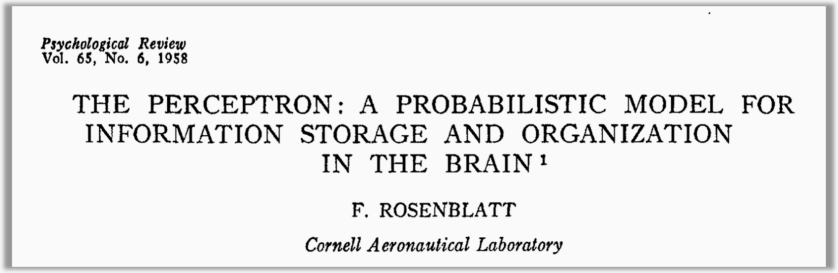
The geometry of a linear classifier



Perceptron

LEARNING ALGORITHM

The perceptron algorithm



Rosenblatt 1958

- Suggested that when a target output value is provided for a single neuron with fixed input, it can incrementally change weights and learn to produce the output using the Perceptron learning rule (algorithm)

The goal is to find a **separating hyperplane**

- For linearly separable data, guaranteed to find one

Online mistake-driven algorithm

- Processes one example at a time
- Update your hypothesis only when you make mistakes
- Now, we can only reason about # of mistakes, not # of examples

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 $(\mathbf{x}_i, y_i)$:
   - 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

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 other times as 0. For the perceptron algorithm, treat -1 as false and +1 as true

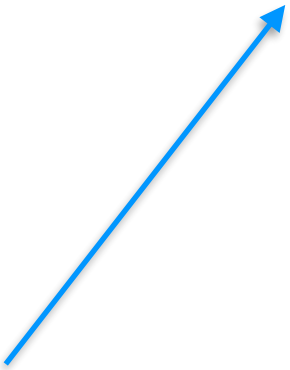
The perceptron algorithm

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 $(\mathbf{x}_i, y_i)$:
   - 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

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 $(\mathbf{x}_i, y_i)$:
   - 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

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

1. Initialize $\mathbf{w_0} = 0 \in \Re^d$
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 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 $\mathbf{x}_i \in \Re^d$, $y_i \in \{-1, 1\}$

1. Initialize $\mathbf{w_0} = 0 \in \Re^d$
2. For each training example $(\mathbf{x}_i, y_i)$:
   - 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 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 $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$

The perceptron algorithm

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

1. Initialize $\mathbf{w_0} = 0 \in \Re^d$
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 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 $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$

Perceptron

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

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

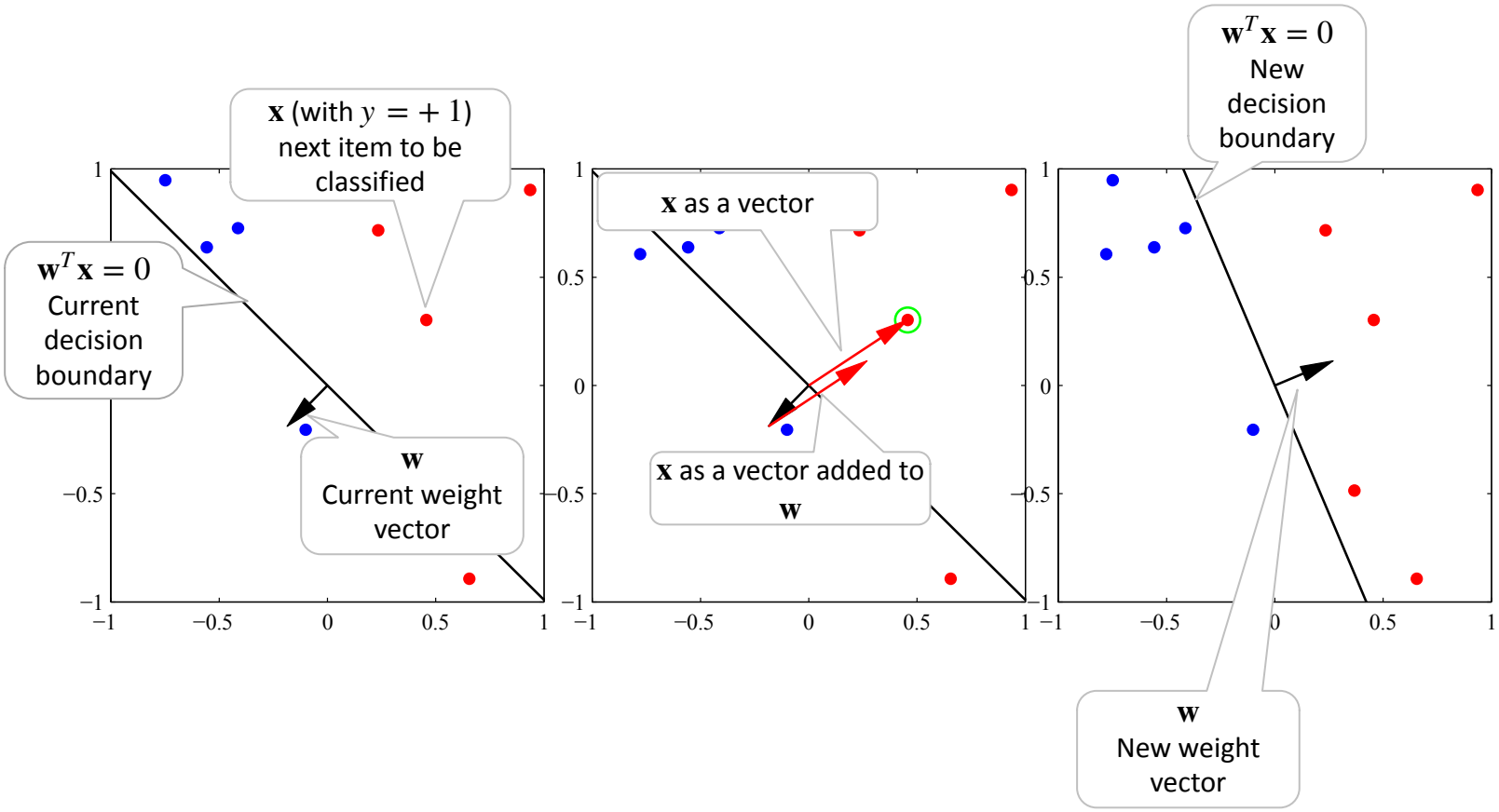
The new dot product $\mathbf{w}_{t+1}^T \mathbf{x} = (\mathbf{w}_t + \mathbf{x})^T \mathbf{x} = \mathbf{w}_t^T \mathbf{x} + \mathbf{x}^T \mathbf{x} \geq w_t^T \mathbf{x}$

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

Similar reasoning for negative examples

Perceptron in action

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



(Figures from Bishop 2006)

- Positive
- Negative

Perceptron

LEARNABILITY

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 Learnability

Can't learn what it can't 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)
- In vision, if patterns are represented with local features, can't represent symmetry, connectivity

Research on Neural Networks stopped for years …

Rosenblatt himself (1959) asked,

- "What pattern recognition problems can be transformed so as to become linearly separable?"

Why is XOR not linearly separable?

Perceptron uses linear function to predict classes (negative or positive) for inputs: e.g., sgn$(b + w_1 x_1 + w_2 x_2)$

XOR function is non-linear:

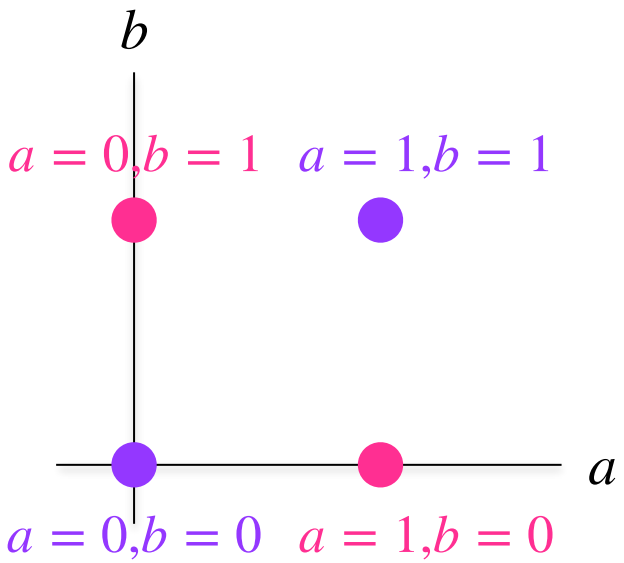
$$f(a,b) = a + b - 2ab$$

$$f(a=0,b=0) = 0 + 0 - 2 \cdot 0 \cdot 0 = 0$$

$$f(a=0,b=1) = 0 + 1 - 2 \cdot 0 \cdot 1 = 1$$

$$f(a=1,b=0) = 1 + 0 - 2 \cdot 1 \cdot 0 = 1$$

$$f(a=1,b=1) = 1 + 1 - 2 \cdot 1 \cdot 1 = 0$$



Beyond the separable case

Good news

- Perceptron makes know assumptions about data distribution, could be even adversarial
- After a fixed number of mistakes, you are done. Don't even need to see more data

Bad news: Real world is not linearly separable

- Can't expect to never make mistakes again
- What can we do: more features, try to be linearly separable if you an, use averaged/voted perceptron

Summary: perceptron

- Online learning algorithm, very widely used, easy to implement
- Additive updates to weights
- Geometric interpretation
- You should be able to implement the perceptron algorithm