50.007 Machine Learning

2026 Spring

2. Perceptron

Na Zhao
Assistant Professor, ISTD

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Recap – Steps to solving a supervised learning problem

1. Decide what the input-output pairs are.
2. Decide how to encode inputs and outputs.
   - This defines the input space $\mathcal{X}$, and the output space $\mathcal{Y}$.
3. Choose a *class of hypotheses* $\mathcal{H}$, *a.k.a.* the *model* or *set of classifiers.*
4. Choose an *error function* (*cost function*) to define the best hypothesis h.
5. Choose a *learning algorithm/criterion* for searching efficiently through the space of hypotheses - *optimization.*

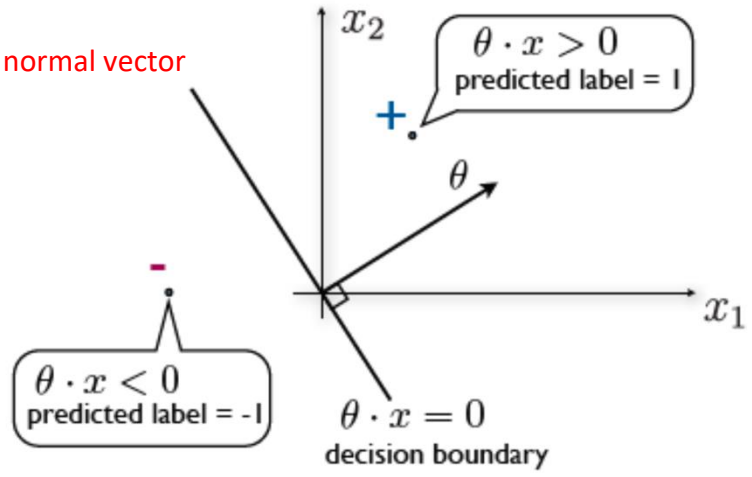
Recap – Linear Classifier

Linear classifier is a highly constrained set of classifiers (through origin):

$$h(x;\theta) = \text{sign}(\theta_1 x_1 + \ldots + \theta_d x_d) = \text{sign}(\theta \cdot x) = \begin{cases} +1, & \theta \cdot x \geq 0 \\ -1, & \theta \cdot x < 0 \end{cases}$$

- $\theta = [\theta_1, \ldots, \theta_d]^T$ is a column vector of real valued parameters or weights
- Different settings of the **weights** give rise to **different classifiers**.

theta here is normal vector



- A **decision boundary** is a line (or a surface, in higher dimensions) that separates the two classes, resulting in two half-spaces:

$$\mathcal{X}^+(\theta) = \{x \in \mathcal{R}^d : h(x;\theta) = +1\}$$
$$\mathcal{X}^-(\theta) = \{x \in \mathcal{R}^d : h(x;\theta) = -1\}$$

- Note that these half spaces as well as the decision boundary, depend on how we set θ.

Learning Objectives

You should be able to know:

1. What is a linearly separable set of examples?
2. How does the Perceptron algorithm work?
3. What is the guarantee of the Perceptron algorithm when the dataset is linearly separable?

Linear Classifier

$$h(x;\theta) = \text{sign}(\theta_1 x_1 + \ldots + \theta_d x_d) = \text{sign}(\theta \cdot x) = \begin{cases} +1, & \theta \cdot x \geq 0 \\ -1, & \theta \cdot x < 0 \end{cases}$$

h - prediction
y - true value

- How should we pick θ? - choose theta chooses final model

  Error minimization!

  - Intuitively, θ should make the predictions of h close to the true values y on the data we have (training data set)
  - Hence, we will define an error function or cost function to measure how much our prediction differs from the "true" answer:

0.x(t) = 0, 0 lies on the decision boundary

$$\mathcal{E}_n(\theta) = \frac{1}{n}\sum_{t=1}^{n} [\![y^{(t)} \neq h(x^{(t)};\theta)]\!] = \frac{1}{n}\sum_{t=1}^{n} [\![y^{(t)}(\theta \cdot x^{(t)}) \leq 0]\!]$$

← **zero-one Loss**

y can only have 2 valurs since binary classificaition, true label is 1/-1

$[\![\cdot]\!]$ returns 1 if the logical expression in the argument is true, and zero otherwise.

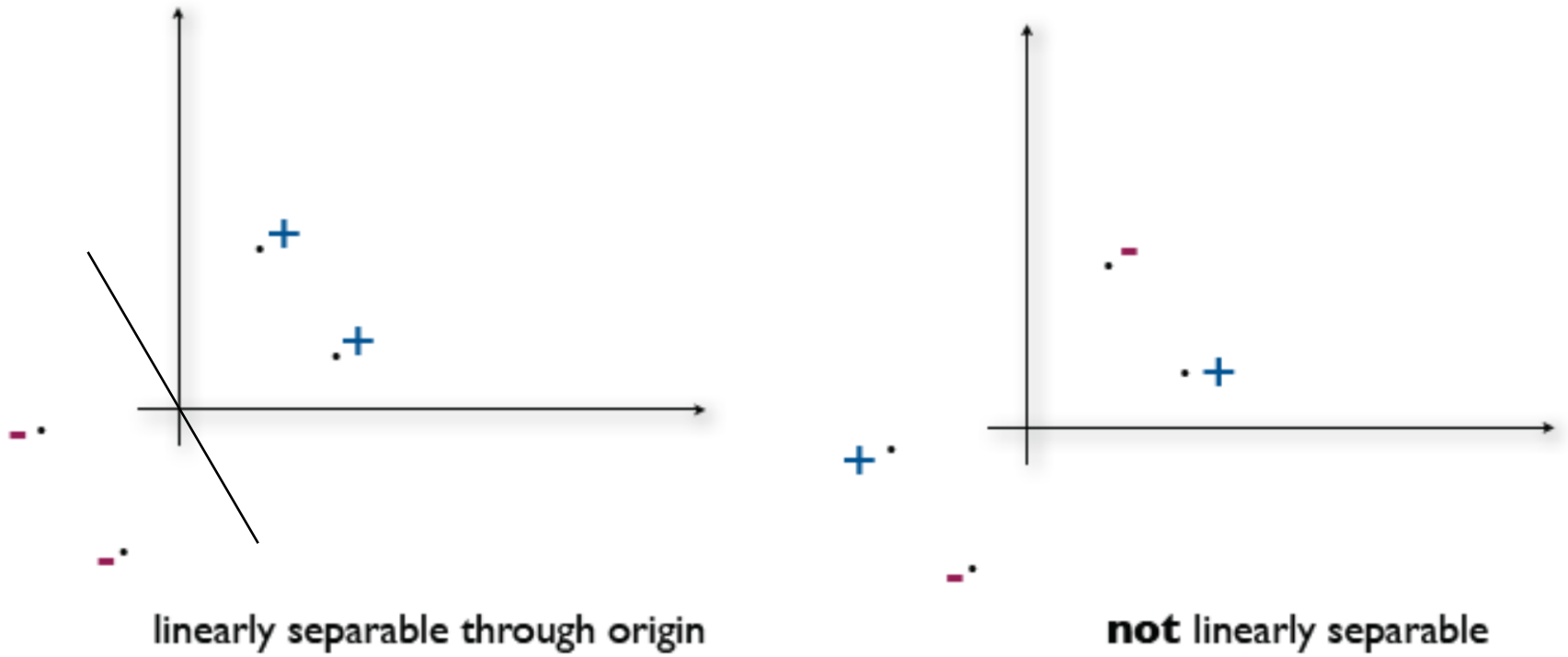
n training sample and input x training data, h will be prediction data, y will be actual data

Error Minimization

- What would a reasonable algorithm be for finding the optimal θ that minimizes the training error?
  - Generally, this is not an easy problem to solve…
- But in this lecture, we consider a special case where there exists a linear classifier that achieves **zero** training error, known as the ***realizable*** case.
- "Realizability" depends on both the training examples as well as the set of classifiers we have adopted.
- Hence, we make <u>two assumptions</u>:
  1. The training examples are *linearly separable through origin.*
  2. The hypothesis is a *linear classifier through origin*.

Linearly Separable

Definition 1.1 *Training examples* $S_n = \{(x^{(t)}, y^{(t)}), t = 1, \ldots, n\}$ *are linearly separable through origin if there exists a parameter vector* $\hat{\theta}$ *such that* $y^{(t)}(\hat{\theta} \cdot x^{(t)}) > 0$ *for all* $t = 1, \ldots, n$.



The Perceptron Algorithm

Random initialize θ
while TRUE **do**
    $m = 0$ m - indicator to see how many run cases
    for $(x^{(t)}, y^{(t)}) \in D$ **do**
        if $y^{(t)}(\theta \cdot x^{(t)}) \leq 0$ **then**
            $\theta \leftarrow \theta + y^{(t)} x^{(t)}$
            $m \leftarrow m + 1$
        end if
    end for
    if $m = 0$ **then**
        break
    end if
end while

Perceptron update rule

// Initialize θ. $\theta = \vec{0}$ misclassifies everything.
// Keep looping
// Count the number of misclassifications, m
// Loop over each (data, label) pair in the dataset, D
// If the pair $(x^{(t)}, y^{(t)})$ is misclassified
// Update the weight vector θ
// Counter the number of misclassification

// If the most recent θ gave 0 misclassifications
// Break out of the while-loop

// Otherwise, keep looping!

Adpted from Kilian Q. Weinberger's slide (Cornell University CS4780)

Perceptron Update Rule

How should we understand the perceptron update rule?

$$\text{if } y^{(t)} \neq h(x^{(t)}; \theta^{(k)}) \text{ then}$$
$$\theta^{(k+1)} = \theta^{(k)} + y^{(t)} x^{(t)}$$

- The weights θ are updated only if we make a mistake, and these updates do tend to correct mistakes.
- If the classifier predicted an example that was negative but it should have been positive...
  - Currently: $\theta \cdot x < 0$
  - Want: $\theta \cdot x > 0$
- Adjust the weights θ so that this function values becomes more positive.

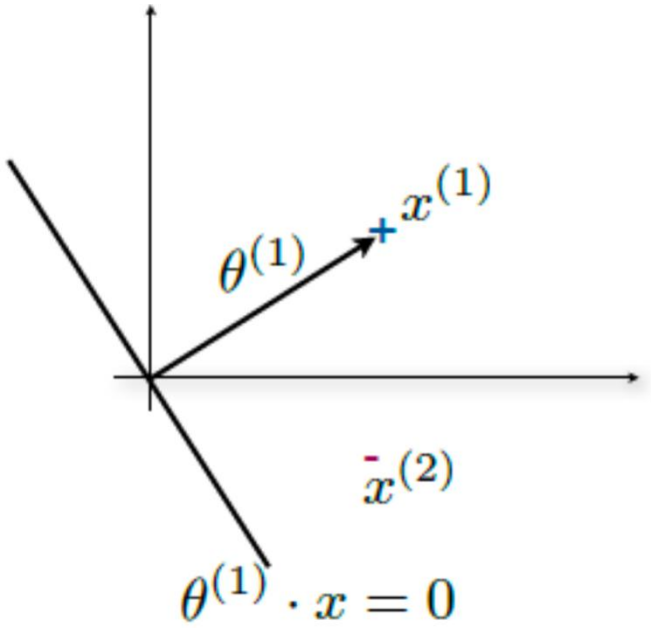
Perceptron Update Rule – An Example

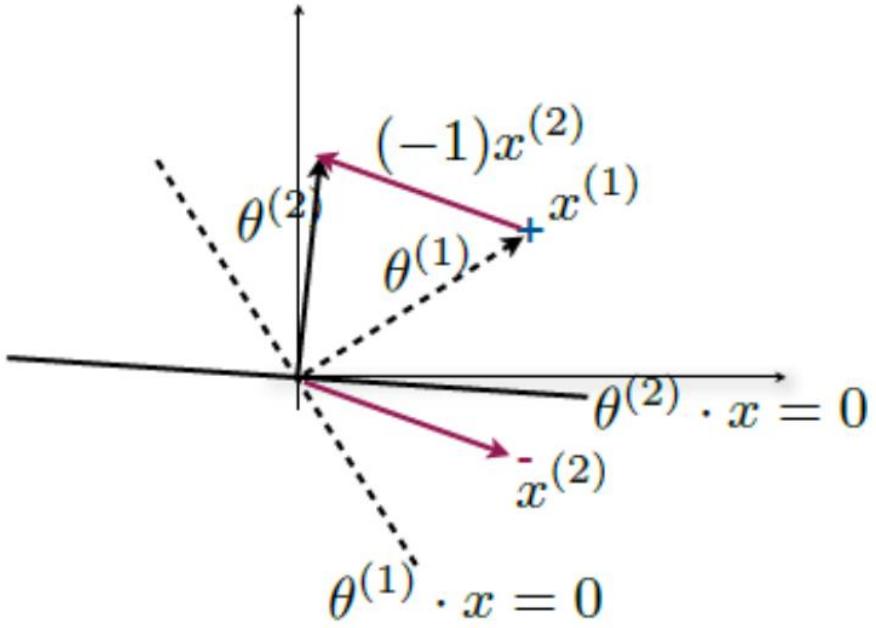
- The points $x^{(1)}$ and $x^{(2)}$ in the figure are chosen such that the algorithm makes a mistake on both of them during it's first pass.
- $\theta^{(0)} = 0 \longrightarrow \theta^{(1)} = \theta^{(0)} + x^{(1)} \longrightarrow \theta^{(2)} = \theta^{(1)} + (-1)x^{(2)}$

Perceptron update rule:

if $y^{(t)} \neq h(x^{(t)}; \theta^{(k)})$ then

$$\theta^{(k+1)} = \theta^{(k)} + y^{(t)}x^{(t)}$$





Perceptron Update Rule - Algebraical View

- Suppose we make a mistake on $x^{(t)}$
- Then the updated weights are given by $\theta^{(k+1)} = \theta^{(k)} + y^{(t)}x^{(t)}$
- If we consider classifying the same example $x^{(t)}$ after the update, using the new weights $\theta^{(k+1)}$

$$\begin{aligned} y^{(t)}(\theta^{(k+1)} \cdot x^{(t)}) &= y^{(t)}(\theta^{(k)} + y^{(t)}x^{(t)}) \cdot x^{(t)} \\ &= y^{(t)}(\theta^{(k)} \cdot x^{(t)}) + (y^{(t)})^2(x^{(t)} \cdot x^{(t)}) \\ &= y^{(t)}(\theta^{(k)} \cdot x^{(t)}) + \left\|x^{(t)}\right\|^2 \end{aligned}$$

- As a result of the update, the value of $y^{(t)}(\theta \cdot x^{(t)})$ increases (becomes more positive)
- If we consider the same example repeatedly, then we will necessarily change the weights such that the example will be classified correctly.

😕 *Mistakes on other examples may steer the parameters in different directions*, so will the algorithm converge to something useful if we repeatedly cycle through the training examples?

The Perceptron Algorithm - Convergence

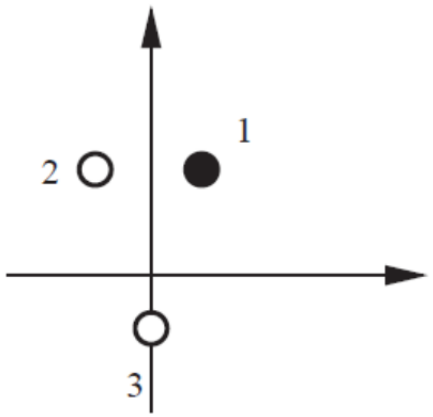
😎 Yes, it will converge in the ***realizable*** case.

Theorem 2.1 *The perceptron update rule converges after a finite number of mistakes when the training examples are linearly separable through origin.* ← Perceptron convergence theorem

Note:

- The finite number of steps could still be many
- There can be more than one solution
  – sensitive to initialization
- If the dataset is not linearly separable, the training will never converge

Exercise: Perceptron Algorithm



Input data

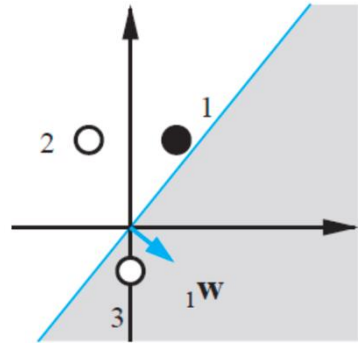
$x^{(1)} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, y^{(1)} = 1$

$x^{(2)} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, y^{(2)} = -1$

$x^{(3)} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, y^{(3)} = -1$

→ **Random weight initialization**

$$\theta^{(0)} = \begin{bmatrix} 1 \\ -0.8 \end{bmatrix}$$

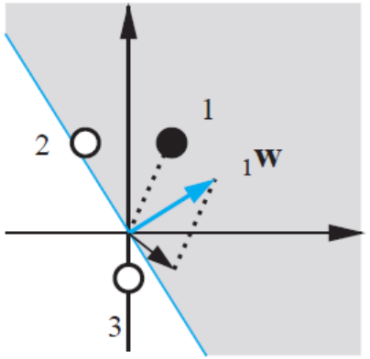


→ **Check if the point is classified correctly:**

$$y^{(1)}\left(\theta^{(0)}.\mathrm{x}^{(1)}\right) = (+1)((1*1) + (-0.8*2))$$
$$= (1 - 1.6) = -0.6 < 0$$

→ **Update $\theta^{(0)}$:**

$$\theta^{(1)} = \theta^{(0)} + y^{(1)}x^{(1)} = \begin{bmatrix} 1 \\ -0.8 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1.2 \end{bmatrix}$$

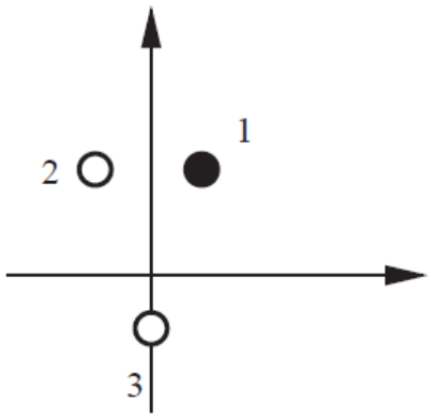


Perceptron update rule:

if $y^{(t)} \neq h(x^{(t)}; \theta^{(k)})$ then

$$\theta^{(k+1)} = \theta^{(k)} + y^{(t)}x^{(t)}$$

Exercise: Perceptron Algorithm



Input data

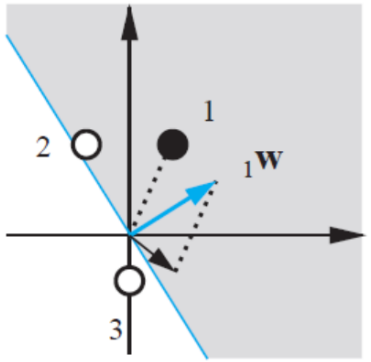
$x^{(1)} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, y^{(1)} = 1$

$x^{(2)} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, y^{(2)} = -1$

$x^{(3)} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, y^{(3)} = -1$

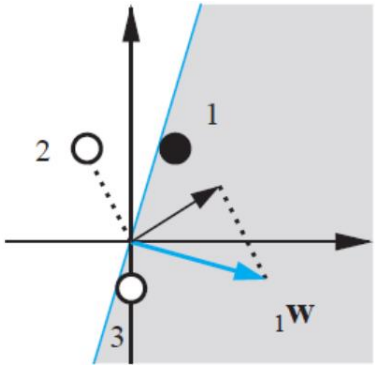
→ **1st Update**

$\theta^{(1)} = \begin{bmatrix} 2 \\ 1.2 \end{bmatrix}$



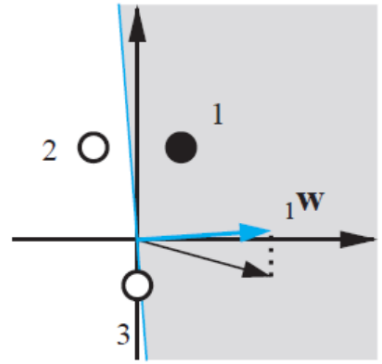
→ **2nd Update**

$\theta^{(2)} = \begin{bmatrix} 3 \\ -0.8 \end{bmatrix}$



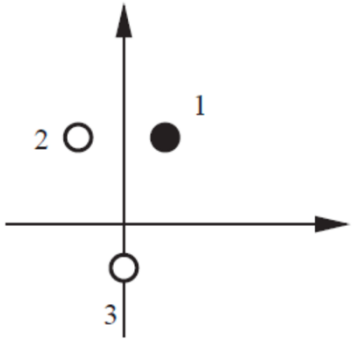
→ **3rd Update**

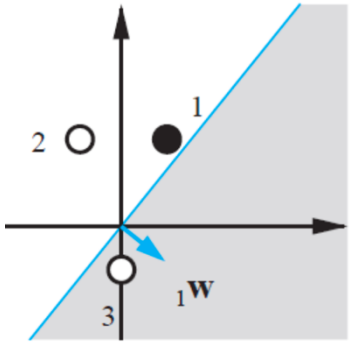
$\theta^{(3)} = \begin{bmatrix} 3 \\ 0.2 \end{bmatrix}$



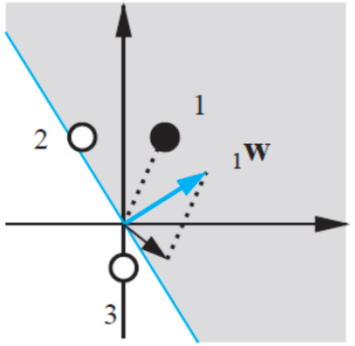
Exercise: Perceptron Algorithm

- Perceptron algorithm oscillates and terminates with zero error in linearly separable case.

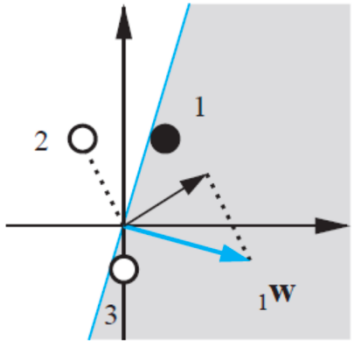




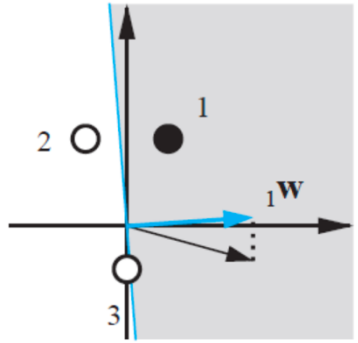
initialisation



1st update

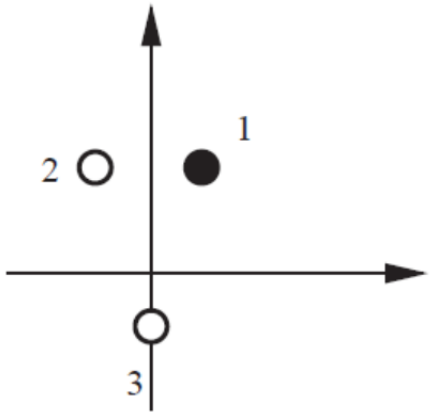


2nd update



3rd update

Exercise: Perceptron Algorithm



Input data

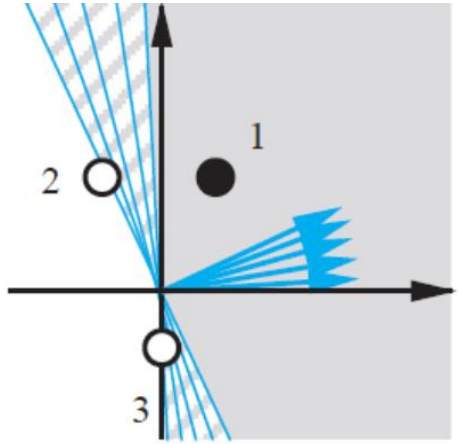
$$x^{(1)} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, y^{(1)} = 1$$

$$x^{(2)} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, y^{(2)} = -1$$

$$x^{(3)} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, y^{(3)} = -1$$

→ **Random weight initialization** →

(*different initialized weights*)



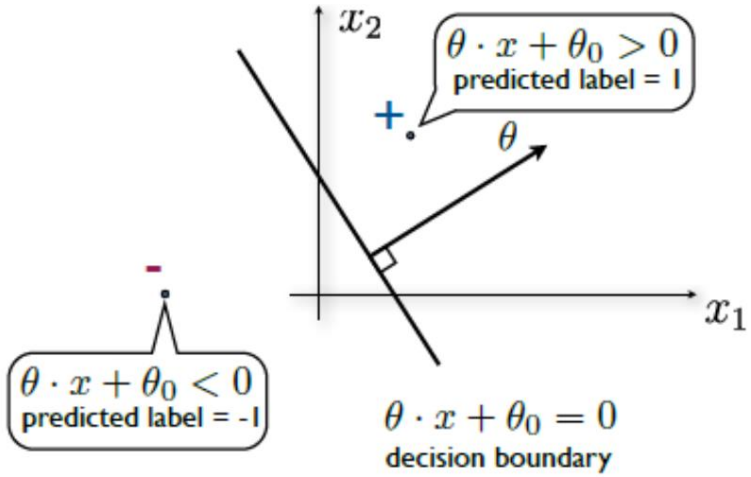
different decision boundaries

Linear Classifier (with offset)

Linear Classifier with offset

- We extend the set of linear classifiers slightly by including a scalar offset parameter θ_0.
  - This parameter will enable us to place the decision boundary anywhere in $\mathbb{R}^d$, not only through the origin.
- **Linear classifier with offset** is defined as:

$$h(x;\theta,\theta_0) = \text{sign}(\theta \cdot x + \theta_0) = \begin{cases} +1, & \theta \cdot x + \theta_0 \geq 0 \\ -1, & \theta \cdot x + \theta_0 < 0 \end{cases}$$



  - The hyper-plane (line in 2d) $\theta \cdot x + \theta_0 = 0$ is **oriented parallel** to $\theta \cdot x = 0$.
  - Vector θ is still **orthogonal** to the decision boundary.
  - And θ also still defines the **positive direction** in the sense that if we move x in this direction, the value of $\theta \cdot x + \theta_0$ increases.

Linear Classifier with offset

- Both linear separability and the perceptron algorithm for learning linear classifiers generalize easily to the case of linear classifiers with offset.
- **Linearly Separable**

Definition 3.1 *Training examples* $S_n = \{(x^{(t)}, y^{(t)}), t = 1; \ldots, n\}$ *are linearly separable if there exists a parameter vector* $\hat{\theta}$ *and offset parameter* $\hat{\theta}_0$ *such that* $y^{(t)}(\hat{\theta} \cdot x^{(t)} + \hat{\theta}_0) > 0$ *for all* $t = 1, \ldots, n$.

- **Perceptron algorithm**
  - Random initialize $\theta^{(0)}$(vector) and $\theta_0^{(0)}$(scalar)
  - Perceptron update rule: if $y^{(t)} \neq h(x^{(t)}; \theta^k, \theta_0^k)$ then

$$\theta^{(k+1)} = \theta^{(k)} + y^{(t)} x^{(t)}$$

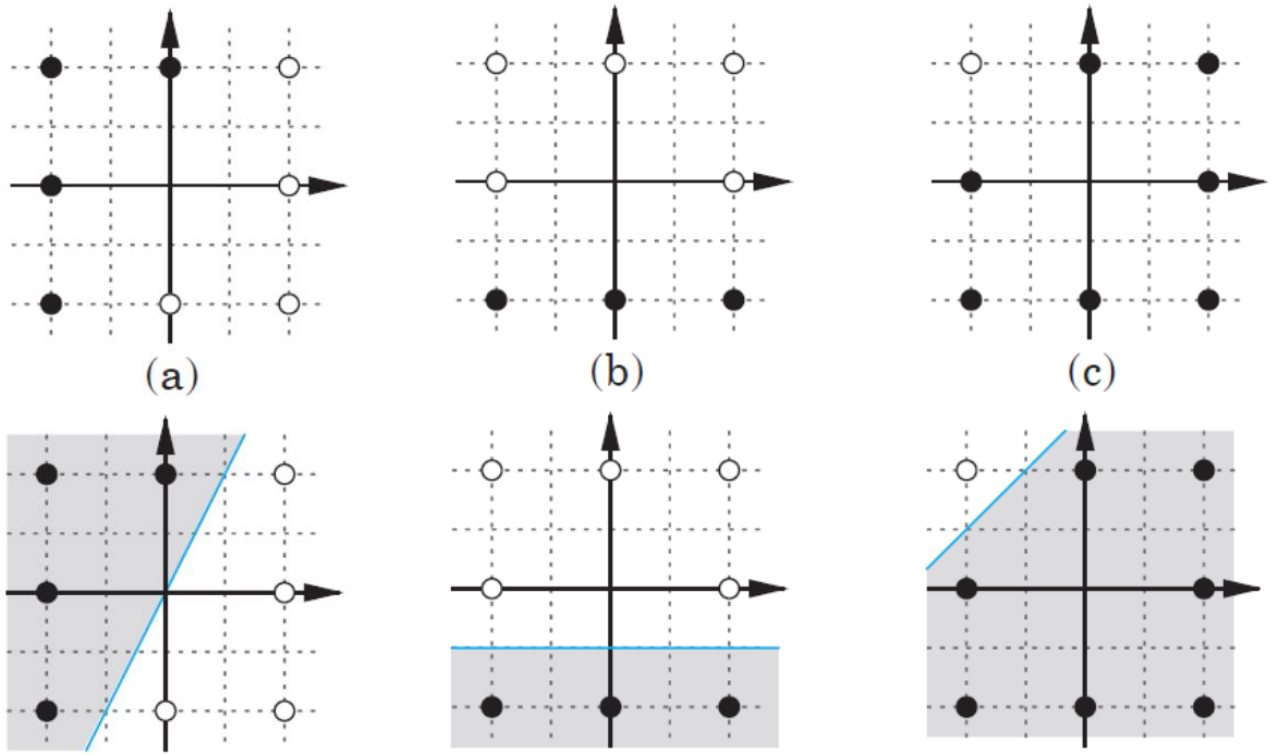
$$\theta_0^{(k+1)} = \theta_0^{(k)} + y^{(t)}$$

Perceptron convergence theorem

If training examples are linearly separable (not necessarily through the origin), then the perceptron algorithm converges after a finite number of mistakes.

Exercise: Linear Classifier with offset

- If training examples are linearly separable through origin, they are clearly linearly separable (see last definition). Is the converse true?



Summary

1. What is a linearly separable set of examples?

Definition 3.1 *Training examples* $S_n = \{(x^{(t)}, y^{(t)}), t = 1; \ldots, n\}$ *are linearly separable if there exists a parameter vector* $\hat{\theta}$ *and offset parameter* $\hat{\theta}_0$ *such that* $y^{(t)}(\hat{\theta} \cdot x^{(t)} + \hat{\theta}_0) > 0$ *for all* $t = 1, \ldots, n$.

Definition 90 (Linear separability)**.** If all the training data for class 1 lies on one side of a hyperplane, and for class 0 on the other, the data is said to be linearly separable.

From "Bayesian Reasoning and Machine Learning"

2. How the Perceptron algorithm works?

Perceptron update rule (through origin):

if $y^{(t)} \neq h(x^{(t)}; \theta^{(k)})$ then

$$\theta^{(k+1)} = \theta^{(k)} + y^{(t)} x^{(t)}$$

Perceptron update rule (general case):

if $y^{(t)} \neq h(x^{(t)}; \theta^{k}, \theta_0^{k})$ then

$$\theta^{(k+1)} = \theta^{(k)} + y^{(t)} x^{(t)}$$

$$\theta_0^{(k+1)} = \theta_0^{(k)} + y^{(t)}$$

3. What is the guarantee of the Perceptron algorithm when the dataset is linearly separable?

Converges after a finite number of mistakes! ← Perceptron convergence theorem

Acknowledgements

- A lot of slides and content of this lecture are adopted from:
  - McGill COMP-652 Machine Learning
  - SUTD 50.007 Machine Learning, Spring 2023 (Asst Prof. Malika Meghjani)
  - MIT 6.036 Introduction to Machine Learning