

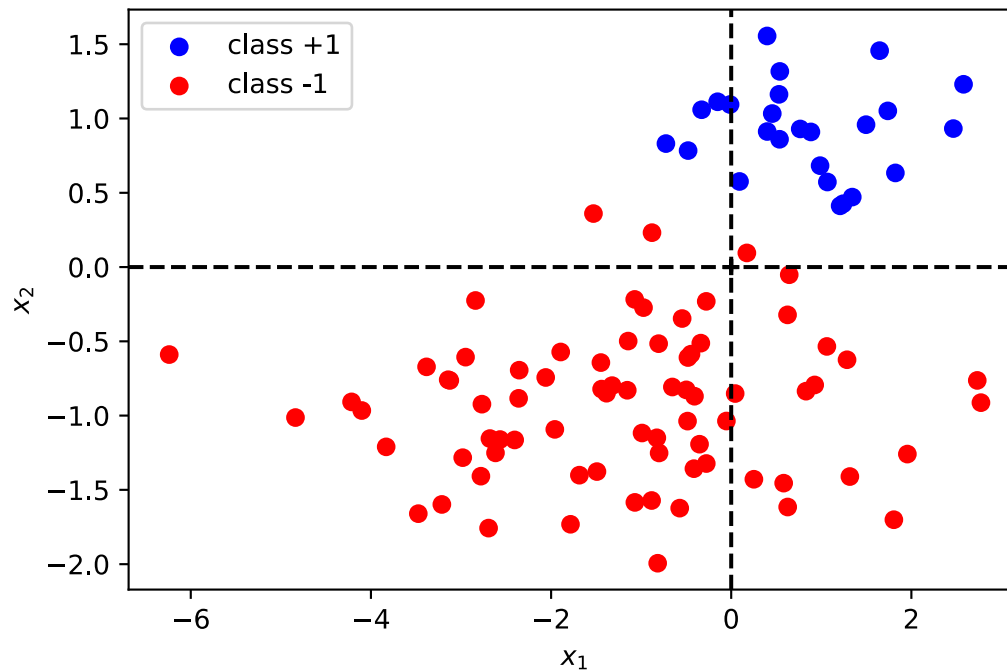
Today – Outline

- Supervised Learning basics:
 - Linear regression
 - Polynomial regression
- Lots of Vocabulary, notations
- Optimization basics: Gradient Descent
- Supervised Learning
 - Classification : Perceptron
 - multiple Loss functions, activations functions
- Optimisation strategies
- Multi-class classification

Intuitive approach (2 variables)

Data: $K=2$ classes – blue and red data points
(e.g. cats and dogs, or brooms and dogs)

- From input features x_1, x_2 : can we separate ?



- Idea: like linear regression (fit the clouds)
then reduce to 2 values, +1 and -1

Binary Classification

Vocabulary

$$X \in \mathbb{R}^{N,d}, \vec{x}^{(n)} = (x_d^{(n)})_{d \in [1, \dots, D]}, X = \{\vec{x}^{(n)}\}_{n \in [1, \dots, N]}$$

- Output (*Ground Truth*) is $t_n \in \{+1, -1\}$

2 classes only → Task is **Binary Classification**

- Model: linear (like LinReg) : $f_{\Theta}(\vec{x}_n) = \vec{w} \cdot \vec{x}_n + b = \vec{w}' \cdot \vec{x}'_n$
- Parameters: $\Theta = (b, \vec{w}) = \vec{w}'$
- **Readout** function: $\sigma(.) \equiv \text{sign}(.)$
- Prediction: $\hat{y}_n = \text{sign}\left(f(\vec{x}_n)\right) \in \{+1, -1\}$
- **Loss** function : **to be found** (next slides)
- Minimization routine: probably Gradient Descent, as usual !

Overall goal: as always in supervised learning, minimize the *discrepancy* between predicted y and Ground Truth t

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}} (J(\Theta, X)) = \underset{\Theta}{\operatorname{argmin}} \left(\sum_n^N \mathcal{L}(f_{\Theta}(\vec{x}_n), t_n) \right) \quad 3$$

Warning

Distinguish the ***model f***:

$$f_{\Theta}(\vec{x}^{(n)}) = \vec{w} \cdot \vec{x}^{(n)} + b \in \mathbb{R}$$

From the ***prediction (decision function)***

$$\hat{y}_n = \text{sign}\left(f(\vec{x}_n)\right) \in \{+1, -1\}$$

Here $\sigma(.) \equiv \text{sign}(.)$ is the ***readout*** (put at the end of the Network), to ***read out*** the result

- The readout is reminiscent of an activation function, but plays a different role.
- The readout does not appear in the Loss function, nor the updates
- Classic **Readout functions**: $\text{sign}(.)$, $\text{softmax}(.)$, $\text{ReLU}(.)$
- Classic **Activation functions**: $\text{ReLU}(.)$, $\tanh(.)$,

Perceptron as a *Neural Network*

Loss #1 (Naïve attempt #1)

Discrepancy=difference → Let's do **like for linear regression**, MSE ?!

$$J_1(\Theta, X) = \frac{1}{2N} \sum^N (\sigma(f_{\Theta}(\vec{x}_n)) - t_n)^2$$

Check if it's a good loss by computing the updates:

$$\vec{\Theta} \rightarrow \vec{\Theta} - \eta \vec{\nabla}_{\Theta} J(\Theta, X)$$

Perceptron as a *Neural Network*

Loss #2 (Naïve attempt #2)

Let's **drop the readout** and use the model (it's more expressive)

$$J_2(\Theta, X) = \frac{1}{2N} \sum^n (f_{\Theta}(\vec{x}_n) - t_n)^2$$

Check if it's a good loss by computing the updates:

$$\vec{\Theta} \rightarrow \vec{\Theta} - \eta \vec{\nabla}_{\Theta} J(\Theta, X)$$

Perceptron as a *Neural Network*

Loss #3 (Better attempt)

Let's use another form of discrepancy: having opposite sign

$$\mathcal{L}(\vec{x}_n, t_n) = \text{ReLU}(-f_{\Theta}(\vec{x}_n)t_n)$$

$$J_{\text{Rosenblatt}}(\Theta, X) = \frac{1}{N} \sum_n \text{ReLU}(-f_{\Theta}(\vec{x}_n)t_n)$$

Check if it's a good loss by computing the updates:

$$\vec{\Theta} \rightarrow \vec{\Theta} - \eta \vec{\nabla}_{\Theta} J(\Theta, X)$$

Remarks: 1. what is $f_{\Theta}(\vec{x}_n)t_n$

2. what is $\text{ReLU}(-f_{\Theta}(\vec{x}_n)t_n)$

3. Remark about encoding: it is now crucial! We must encode with +1/-1, and not just 0 and 1 or 1 and 2...

Perceptron as a *Neural Network*

Loss #3 (Better attempt)

$$J_{Rosenblatt}(\Theta, X) = \frac{1}{N} \sum_n^N ReLu(-f_{\Theta}(\vec{x}_n)t_n)$$

$$\vec{\Theta} \rightarrow \vec{\Theta} - \eta \vec{\nabla}_{\Theta} J(\Theta, X)$$

▪

Perceptron as a *Neural Network*

Other Losses (→ exercises)

- What happens without ReLu ?

$$J_4(\Theta, X) = \frac{1}{N} \sum_n^N (- f_{\Theta}(\vec{x}_n) t_n)$$

(Hint: there is a problem)

- What if we use a ***smooth*** activation function in place of the hard ***readout*** $\text{sign}(\cdot)$?

$$J_5(\Theta, X) = \frac{1}{N} \sum_n^N (\tanh(f_{\Theta}(\vec{x}_n)) - t_n)^2$$

(Hint: this is a decent Loss, *tanh* is an *example* of a *sigmoid*.
See question 4 of the exercise 3, in the 2020's exam)

Perceptron as a *Neural Network*

Other Losses (→ exercises)

- The “**logistic Regression**” is actually a **classification**, with the logistic function as *activation function*, i.e. :

logistic: $\sigma(z) = \frac{e^z}{1 + e^z} = \frac{1}{1 + e^{-z}}$

Model: $y_n = f_{\Theta}(\vec{x}_n) = \sigma(\vec{w}\vec{x}_n) = \frac{1}{1 + e^{-\vec{w}\vec{x}_n}}$

Loss:

$$J_{logistic}(\Theta, X) = -\frac{1}{N} \sum_n^N \left(t_n \log(y_n) + (1 - t_n) \log(1 - y_n) \right)$$

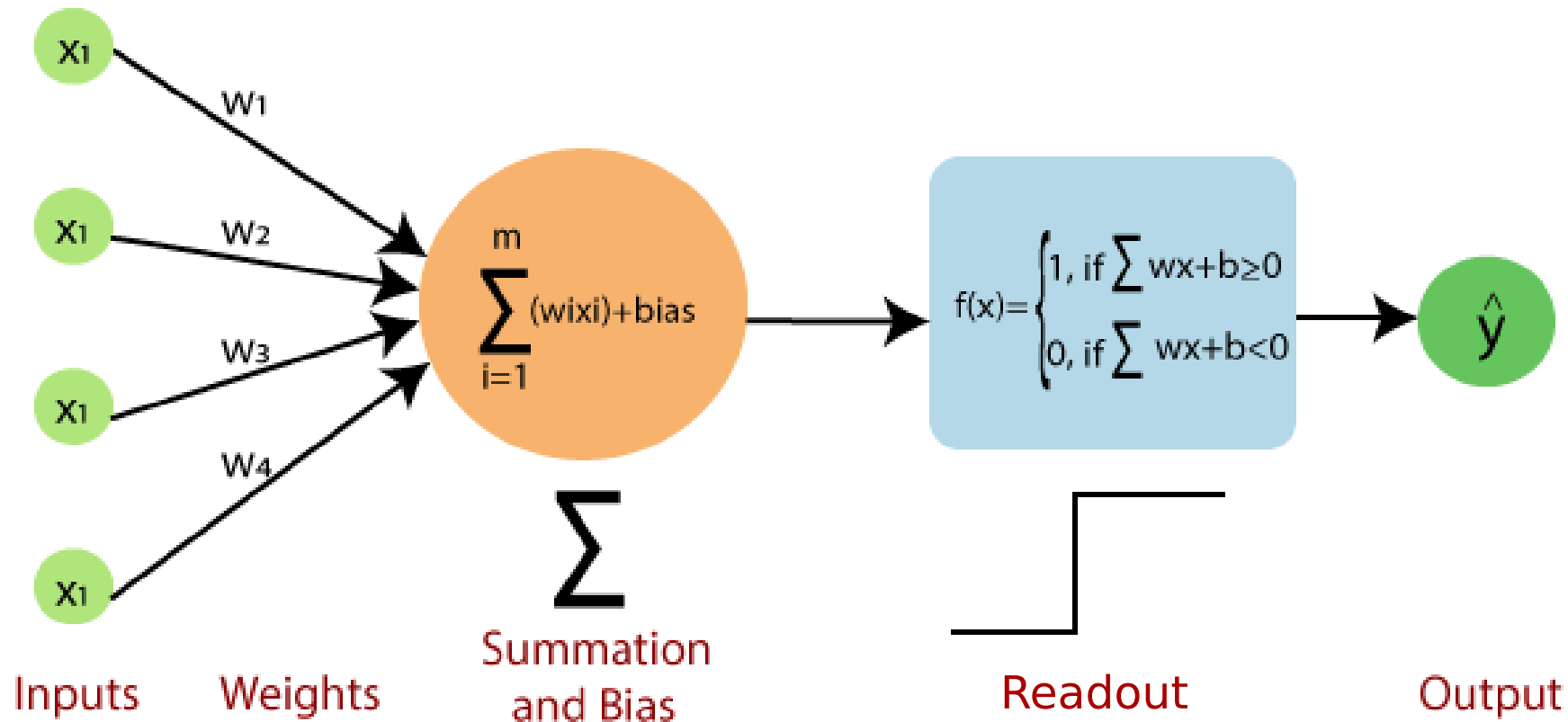
Note: here we included σ into f for convenience.

- Encoding: here, $t=0$ or 1 (not $-1, +1$)

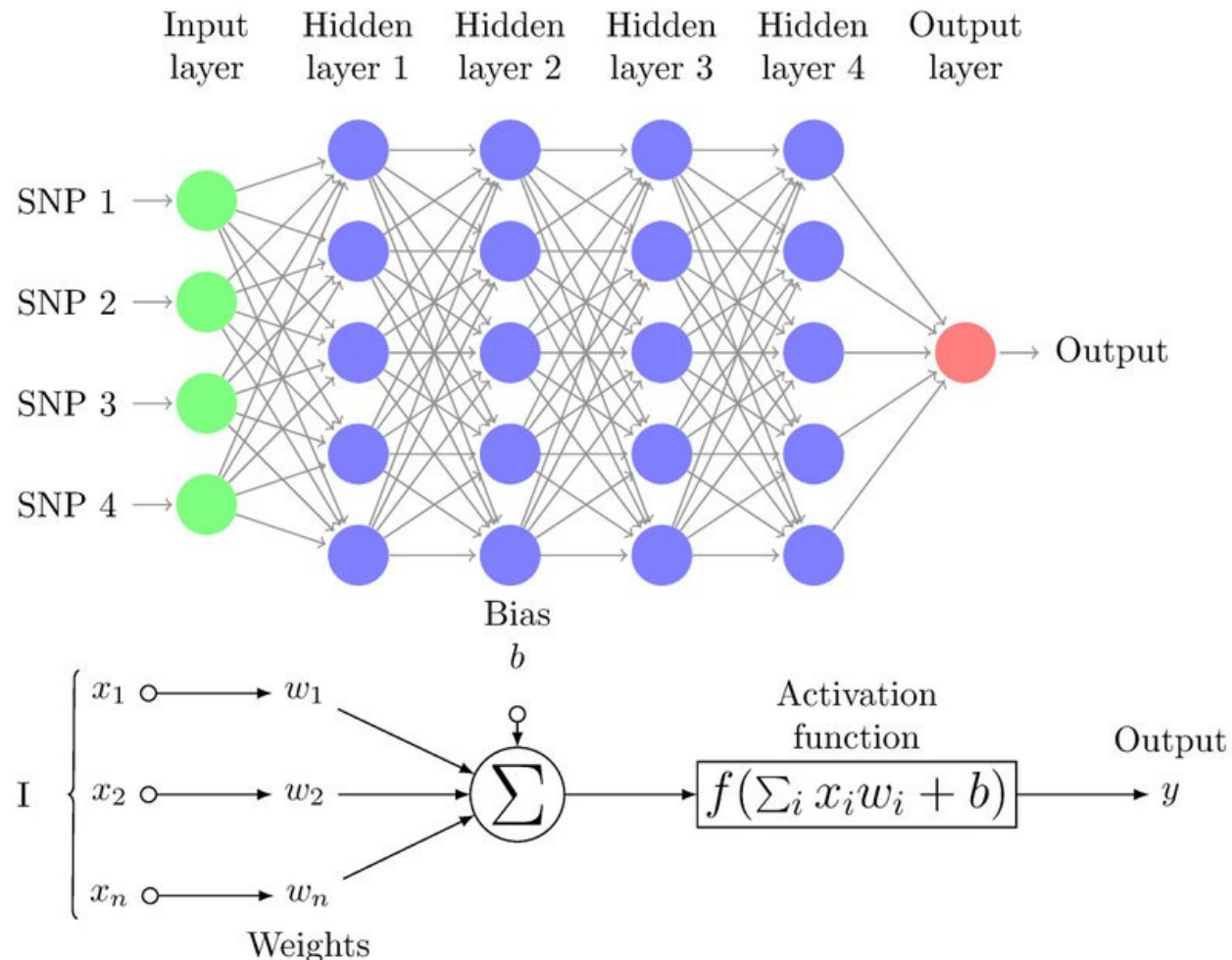
Perspective

Neural Network Diagrams

Our single-layer perceptron is a first instance of a “Neural Network” (with no hidden layer, only an input layer):



- 4-layers Multi-Layer Perceptron



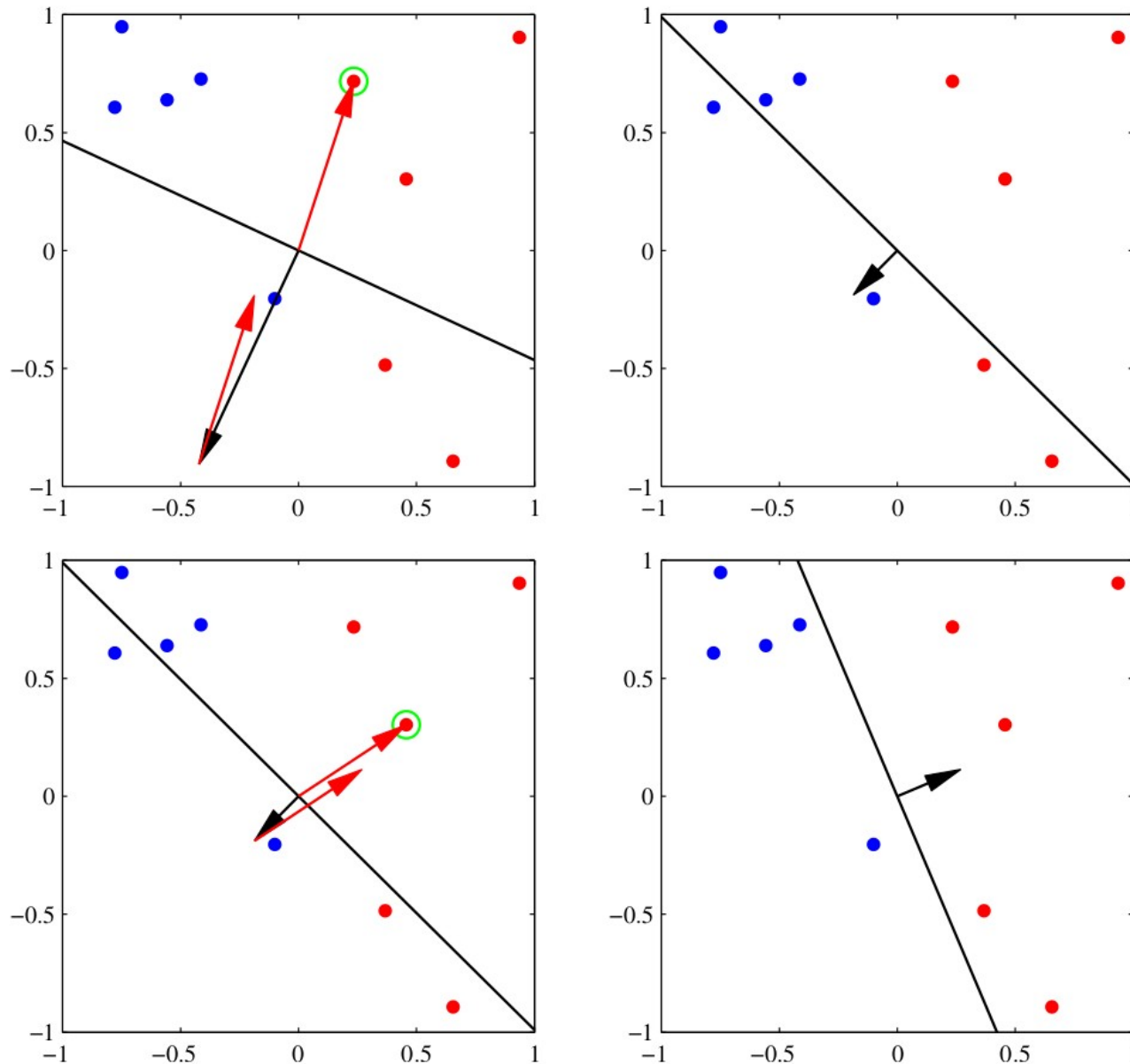
Multi-Layer Perceptron: see the Deep Learning course, OPT4 (Caio Corro)

Historical Perceptron: Rosenblatt's, **Online** Updates

Online Perceptron Algorithm: take examples 1 by 1

- Initialize $\Theta = \Theta_0$
- For each $\vec{x}^{(n)}$,
 - if $\hat{y}_n \neq t_n$ (misclassified), then push towards correct side:
$$\vec{\Theta} \rightarrow \vec{\Theta} - \eta \vec{\nabla}_{\Theta} \mathcal{L}(\vec{x}_n, t_n) \quad \mathcal{L}(\vec{x}_n, t_n) = \text{ReLU}(-f_{\Theta}(\vec{x}_n)t_n)$$
 - else, nothing (exple is already correctly classified)
- Note: here **online** does not mean connected to the internet, but means that *we take examples as they come*, as if they were a **flux** and not a static heap of data

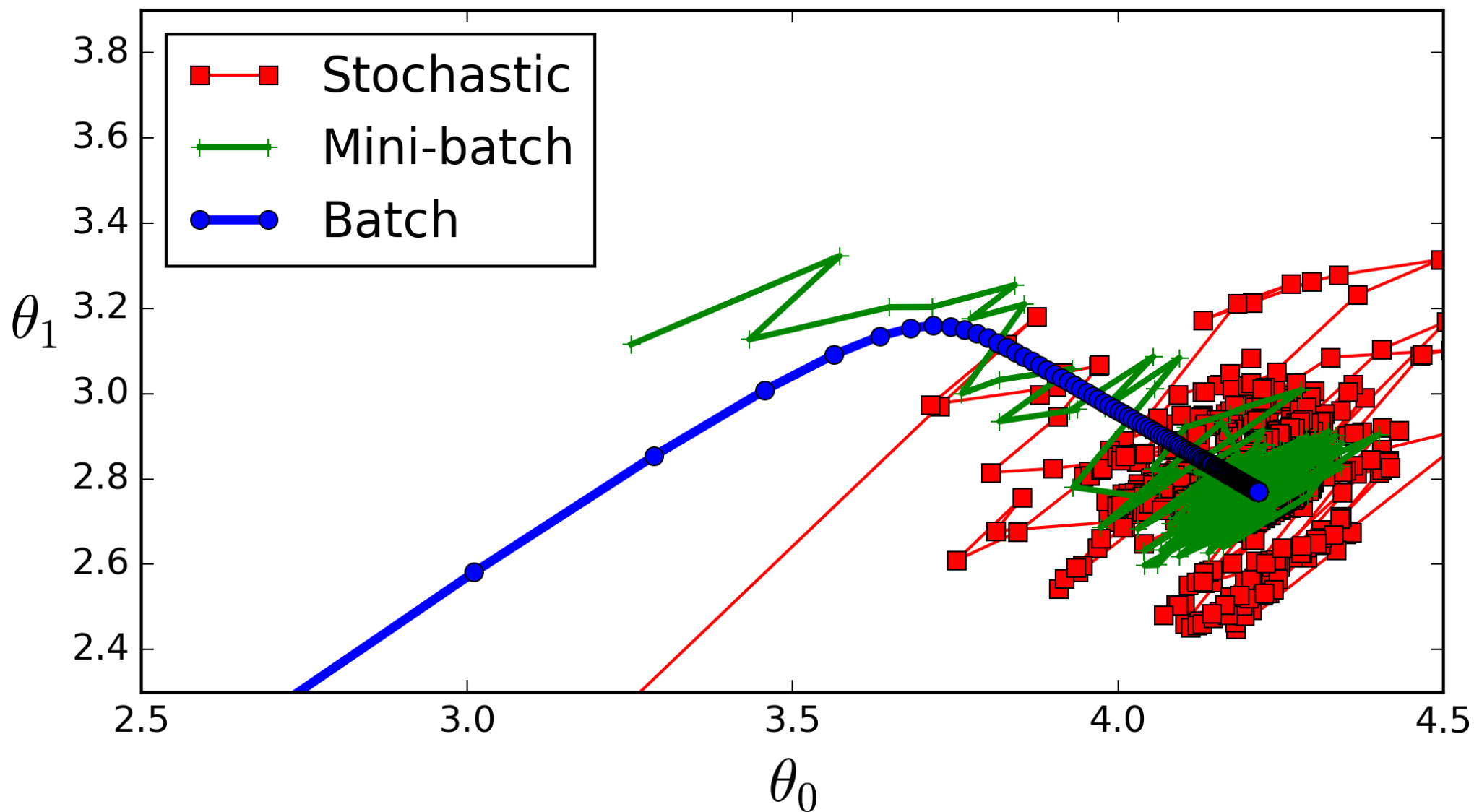
Online Learning ("hand-crafted" algorithm)



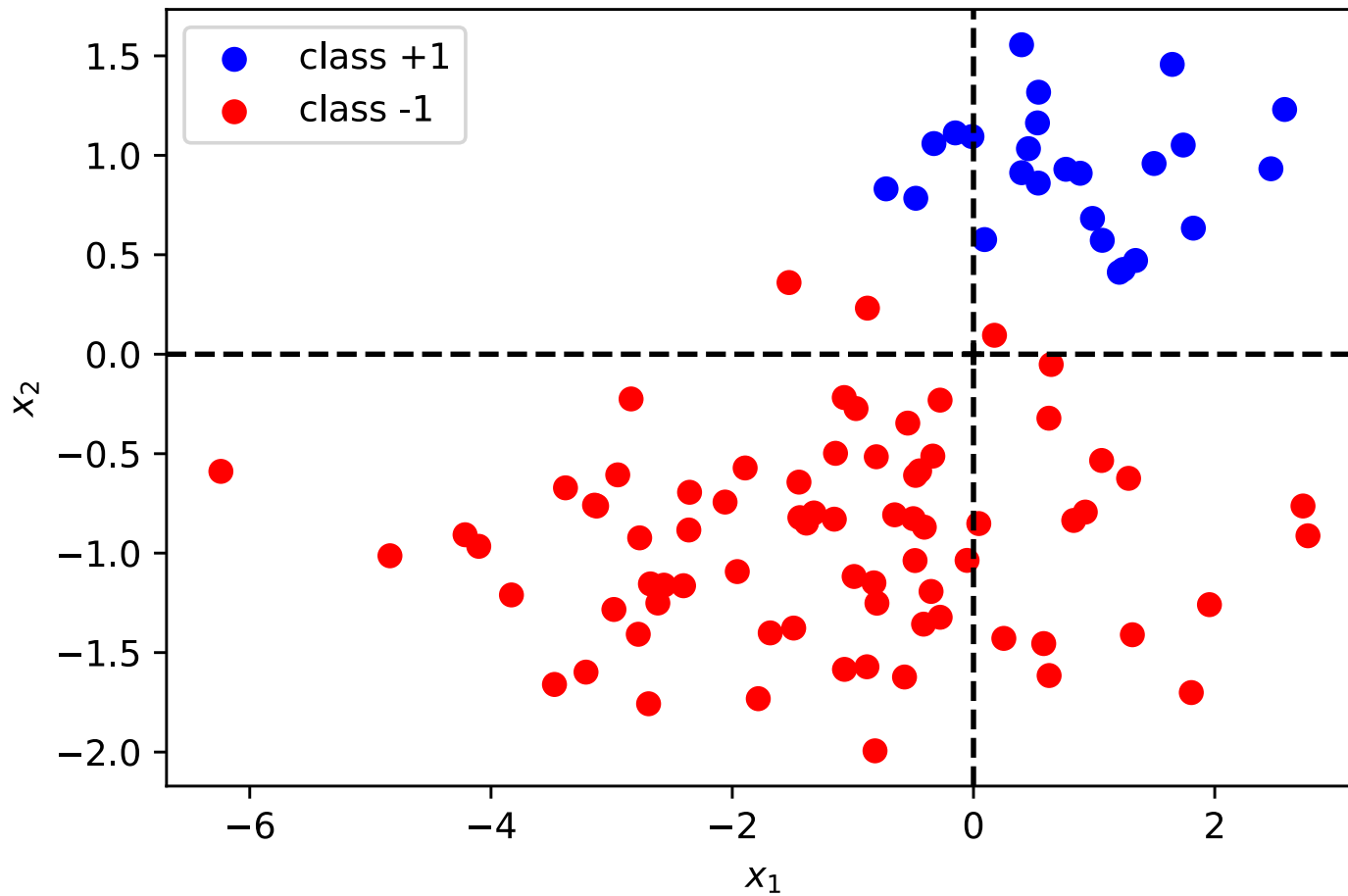
First nuance: various **optimization strategies**

- **Examples** seen one by one:
“Online” learning
- **Stochastic Gradient Descent (SGD):** $(m=1)$
~looks like *Online* (but more random)
- Examples seen all at once $(m=N)$
global update (optimization viewpoint)
- Intermediate solution: **batch size** m , $(m>1)$
mini-batch Gradient Descent

SGD vs *mini-batch* vs *full batch*

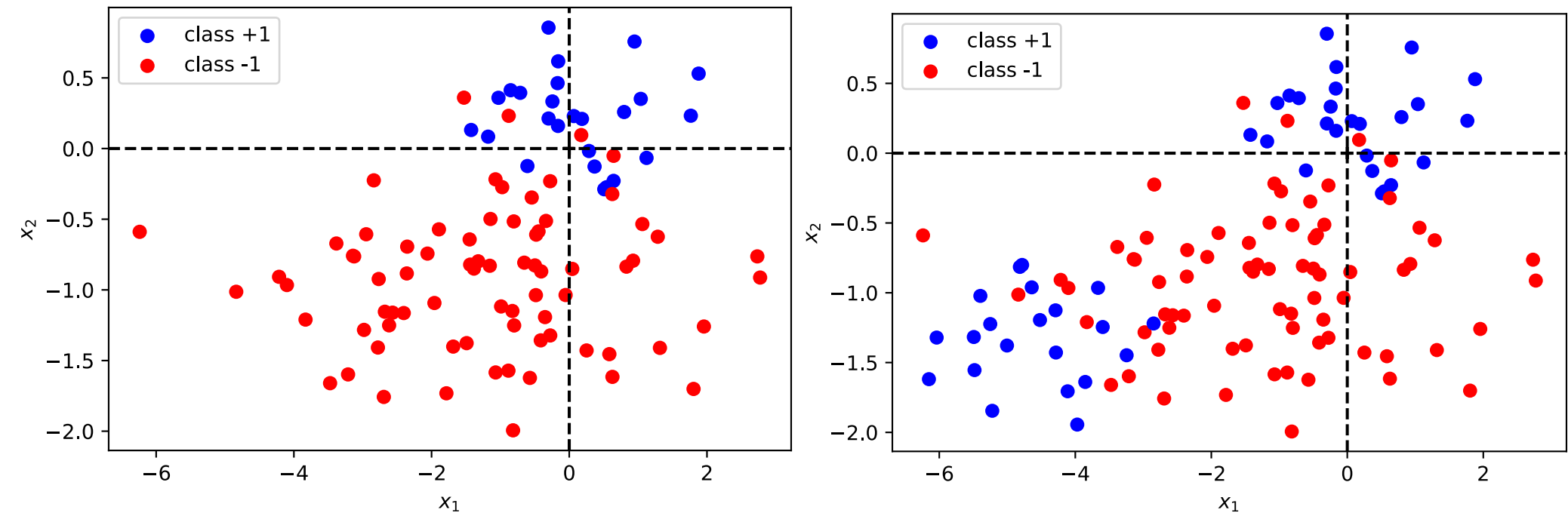


Geometrical Interpretation



More realistically...

Data may be **non linearly separable**



Can we hope for **convergence** ?

→ For the ***online*** choice, it's not so great.

→ For others, it's ok, it can converge to *something*

Multi-class classification

- If there are $K > 2$ **classes**. Various strategies:

One-versus-rest (OVR) strategy:

- Return to Binary Classif., a point is either class k or “not class k ”.

→ You now have K classifiers $W_{K,d} = \{\vec{w}_1, \dots, \vec{w}_K\}$

- You have K times more parameters !
- Which one to choose ?
The one that is the most on the correct side of the hyperplane:

$$\hat{y}_n = \operatorname{argmax}_k (f_{\Theta}(\vec{x}_n)) = \operatorname{argmax}_k (\vec{w}_k \vec{x}_n)$$

- What is a good Loss ?

Multi-class classification

Building a good Model+Loss for a **Multiclass** Perceptron:

- **Encode** classes into **one-hot vectors**

Ground truth of type: $t_n = \vec{e}_k = (0, \dots, 0, 1, 0, \dots, 0)$

Network output : $\vec{y}^{(n)} = (y_1^{(n)}, \dots, y_K^{(n)})$

- Use **softmax**(z) : $z \in \mathbb{R}^K, \text{softmax}(\vec{z})_j = \frac{\exp(z_j)}{\sum_k \exp(z_k)}$
- Model: assume $W_{K,d} = \{\vec{w}_1, \dots, \vec{w}_K\}$

$$(y_n)_j = \text{softmax}(W_{K,d}\vec{x}_n)_j = \frac{\exp(\vec{w}_j \cdot \vec{x}_n)}{\sum_k \exp(\vec{w}_k \cdot \vec{x}_n)}$$

Trick: insert $z_k = \vec{w}_k \vec{x}_n$ or $z_j = \vec{w}_j \vec{x}_n$

- Readout: $\hat{y}_n = \text{argmax}_k((y_n)_k) = \text{argmax}_k(\vec{w}_k \vec{x}_n)$
- Loss ? : see exercise “Multi class classification” in TD or “TP2.2-MultiClass-Classification.ipynb”

Multi-class classification

Two classic strategies :

- OVR: **one-versus-rest** (K)
How to choose the winner ? Take the max.
(argmax).
- OVO: **one-versus-one** ($K(K-1)/2$)
How to choose the winner ? Take the one with
most votes, typically.

References:

- **Linear classifiers in general:**
 - Bishop book, page 179-196, section 4.1
- Loss Function J2 (MSE for classif)
 - Bishop book, page **184-186**, section 4.1.3
(**Least squares for classification**)
- **Perceptron:**
 - Bishop book, page **192-196**, section 4.1.7
(The **perceptron algorithm**)
- **Multi-Layer Perceptron:** see the Deep Learning course, OPT4 (Caio Corro)

Key concepts

- **Classification**
- **Readout** (vs activation function)
- **Model** vs Prediction (without readout, with it)
- Non trivial **losses**
- **Activations** : ReLu, softmax, sigmoids, logistic
- **Strategies**: Online, SGD, **mini-batch**, **full batch**
- Hyperplanes, Linearly Separable / non linearly separable data
- Encoding, **one-hot** vectors
- **Multi-class** Classification, **OVR** and OVO strategies