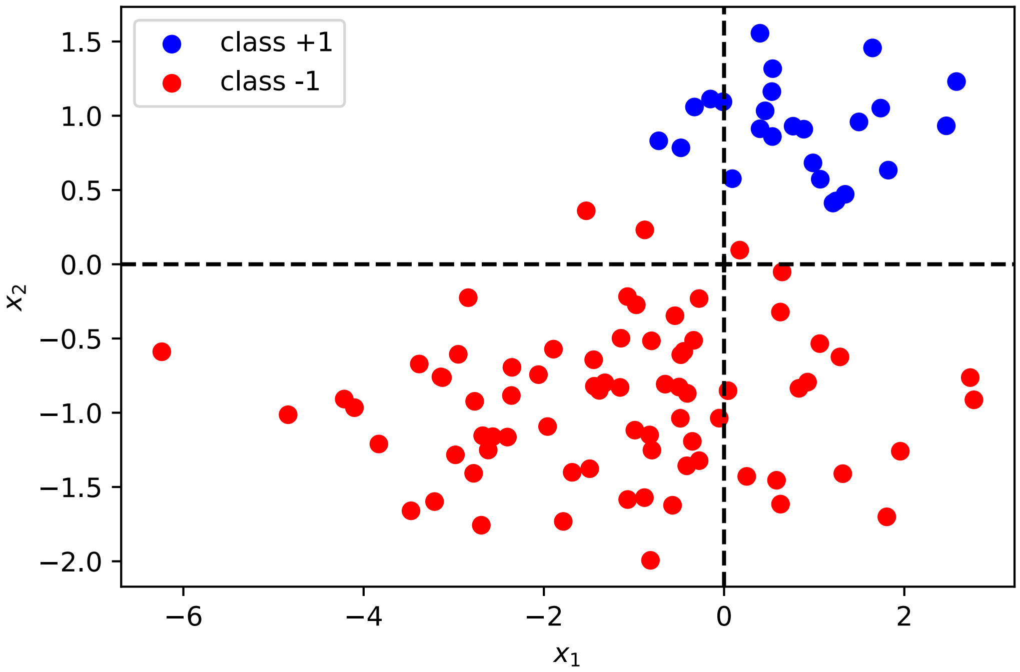
Today – Outline

- **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 x1, x2 : 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: $\sigma(.) \equiv sign(.)$
- Prediction: $\hat{y}_n = sign\Big(f(\vec{x}_n)\Big) \in \{+1,-1\}$
- **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

$$\Theta^* = argmin_\Theta(J(\Theta, X)) = argmin_\Theta\Big(\sum_n^N \mathcal{L}(f_\Theta(\vec{x}_n), t_n)\Big)$$

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)***

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

Here $\sigma(.) \equiv sign(.)$ 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**: sign(.), softmax(.), ReLu(.)
- Classic **Activation functions**: ReLu(.), tanh(.),

Perceptron as a ***Neural Network***
Loss #1 (Naïve attempt #1)

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

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

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

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

Perceptron as a ***Neural Network***
Loss #2 (Naïve attempt #2)

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

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

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

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

Perceptron as a ***Neural Network***
Loss #3 (Better attempt)

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

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

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

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

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

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

what is $ReLu\big(-f_\Theta(\vec{x}_n)t_n\big)$

Perceptron as a ***Neural Network***
Loss #3 (Better attempt)

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

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

.

Perceptron as a ***Neural Network*** *Other Losses* (→ *exercises)*

- What happens without ReLu ?

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

(Hint: there is a problem)

- What if we use a ***smooth*** *activation function* in place of the hard ***readout*** sign(.) ?

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

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

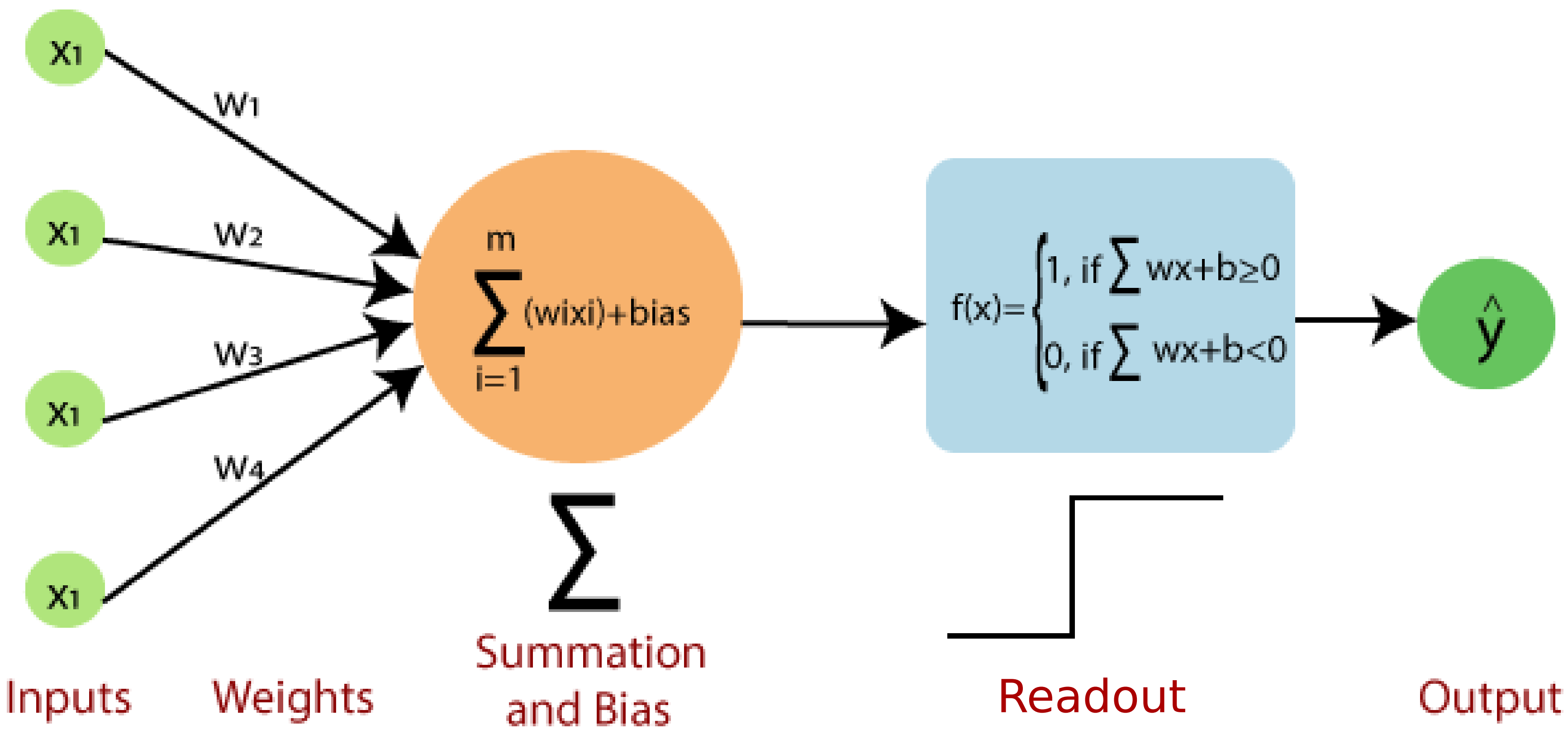
Model: $$y_n = f_\Theta(\vec{x}_n) = \sigma(\vec{w}\vec{x}_n) = \frac{1}{1+e^{-\vec{w}\vec{x}_n}}$$

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

Note: here we included σ into f for convenience.

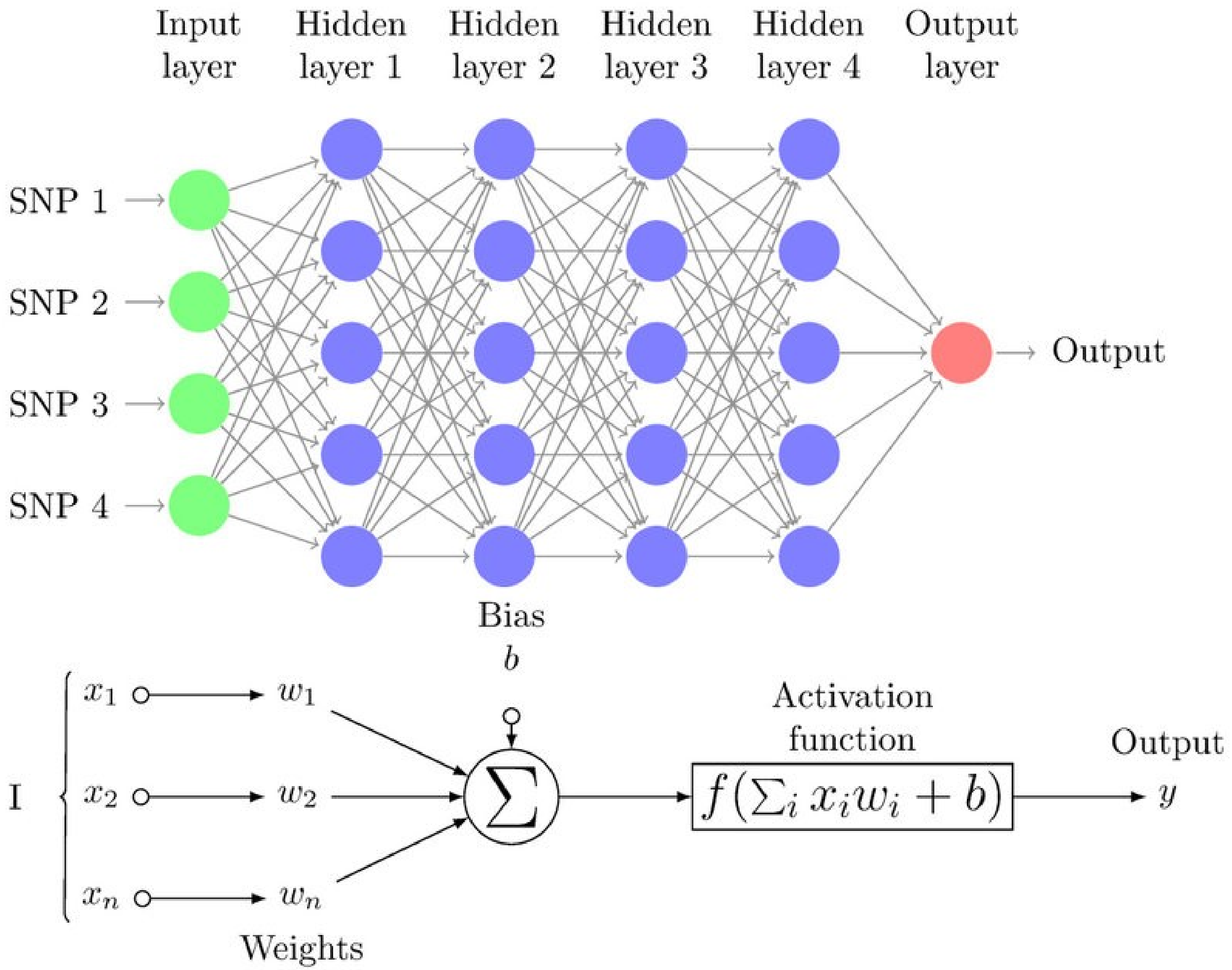
Perspective
Neural Network Diagrams

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



https://static.javatpoint.com/tutorial/tensorflow/images/single-layer-perceptron-in-tensorflow2.png

- 4-layers Multi-Layer Perceptron



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

https://www.researchgate.net/figure/Multi-Layer-Perceptron-MLP-diagram-with-four-hidden-layers-and-a-collection-of-single_fig1_334609713

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

***Online* Perceptron Algorithm:** take examples 1 by 1

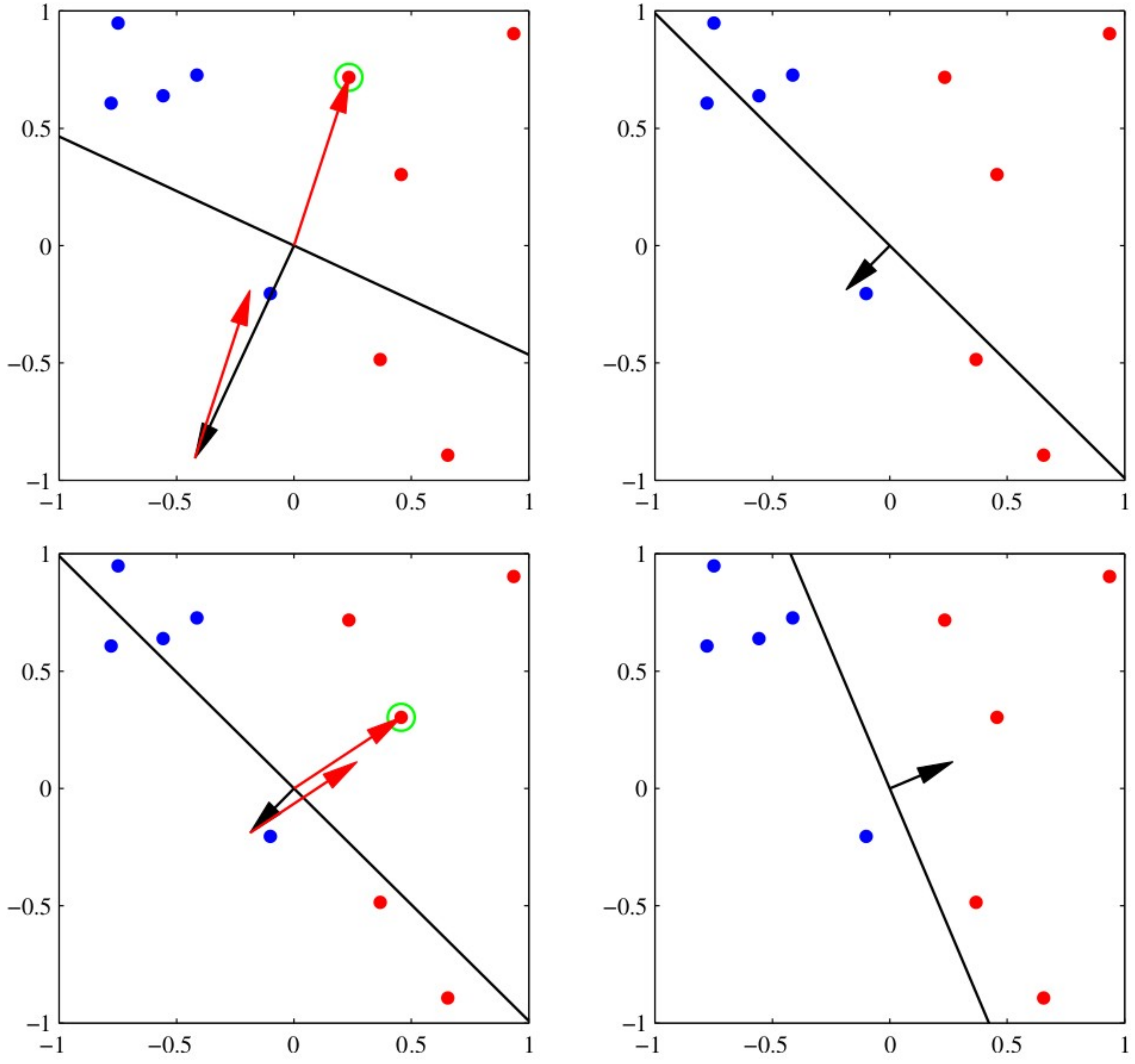
- Initialize $\Theta = \Theta_0$
- For each $\vec{x}^{(n)}$,
  - if $\hat{y}_n \neq t_n$ (misclassified), then push towards correct side:

$$\vec{\Theta} \rightarrow \vec{\Theta} - \eta \vec{\nabla}_\Theta \mathcal{L}(\vec{x}_n, t_n) \qquad \mathcal{L}(\vec{x}_n, t_n) = ReLu\Big(- f_\Theta(\vec{x}_n) t_n \Big)$$

  - 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**

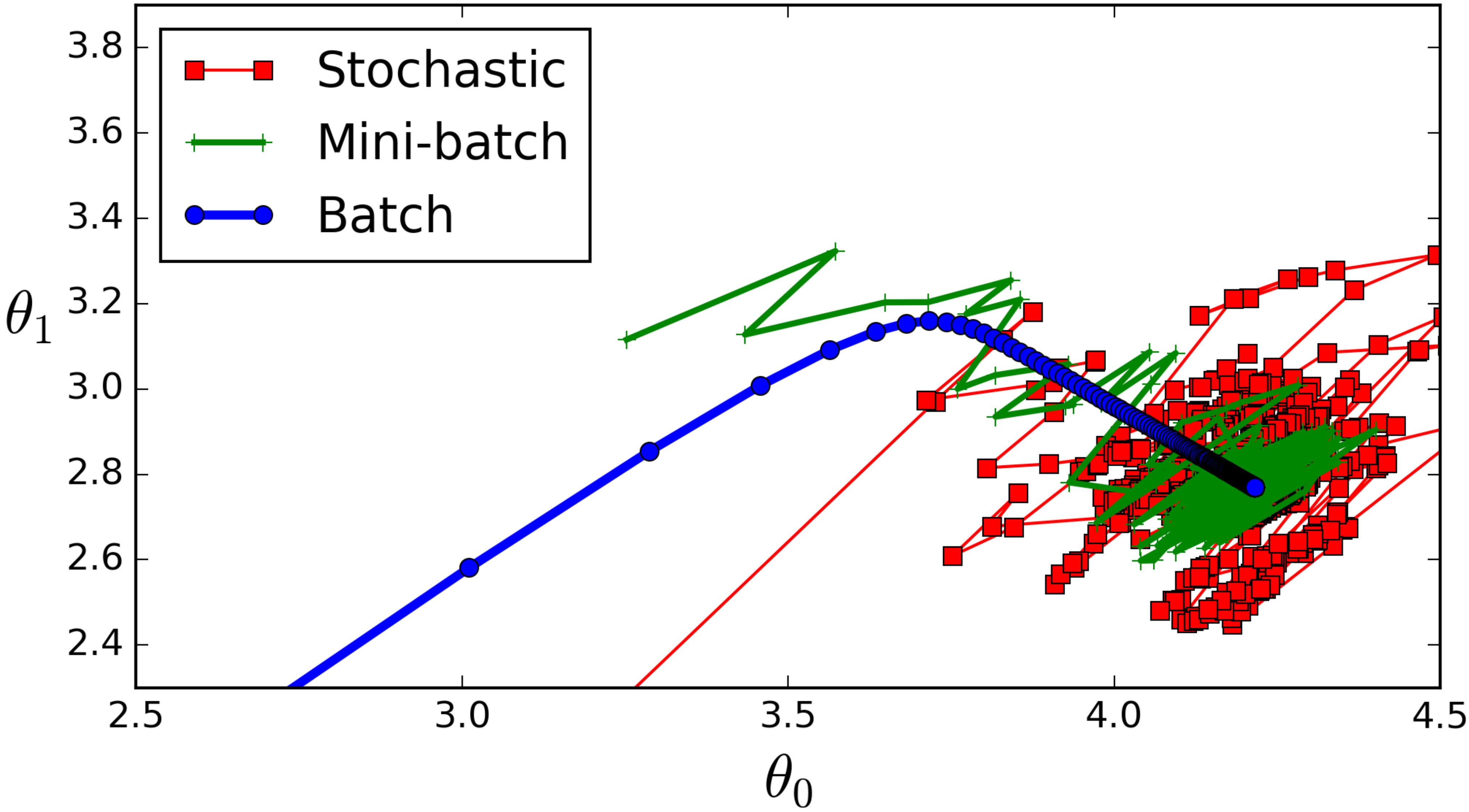
- ***Examples*** seen one by one:
  "Online" learning

- ***Stochastic*** *Gradient Descent* (**SGD**): $(m{=}1)$
  ~looks like *Online* (but more random)

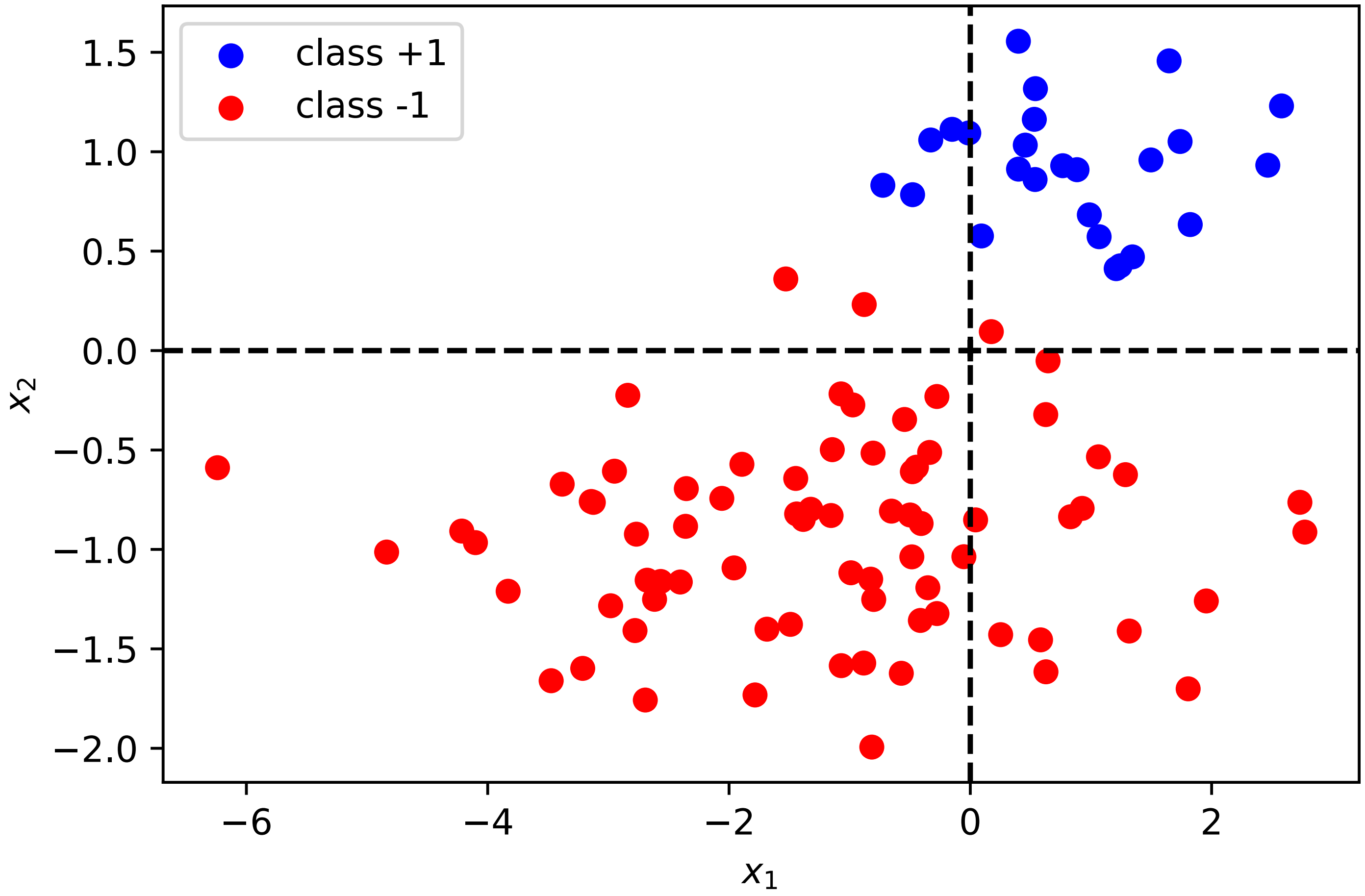
- Examples seen all at once $(m{=}N)$
  global update **(optimization viewpoint)**

- Intermediate solution: ***batch size*** m, $(m{>}1)$
  mini-batch Gradient Descent

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

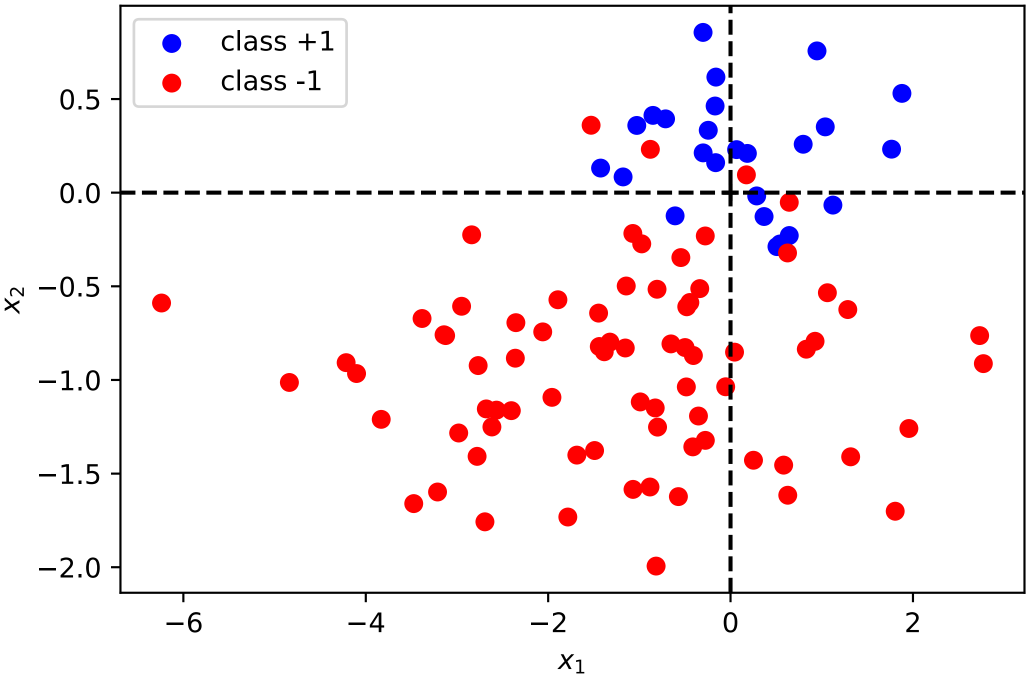


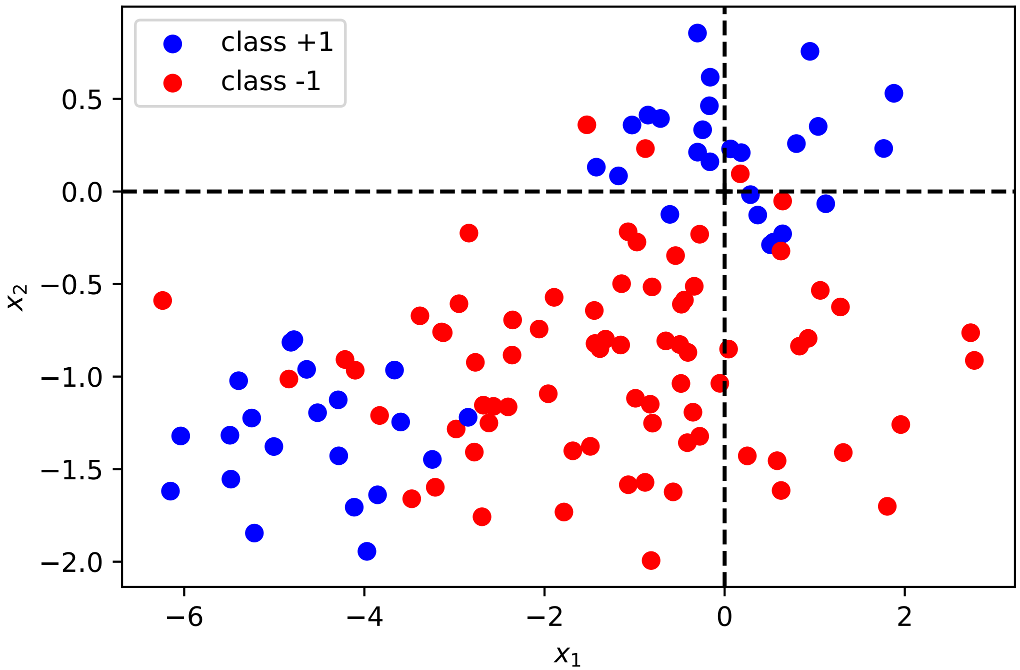
Geometrical Interpretation



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 $W_{K,d} = \{\vec{w}_1, \ldots, \vec{w}_K\}$
- You have K times more parameters !
- Which one to choose ?
  The one that is the most on the correct side of the hyperplane:

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

- 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: $t_n = \vec{e}_k = (0, \ldots, 0, 1, 0, \ldots, 0)$

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

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

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

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

- Readout: $\hat{y}_n = \mathrm{argmax}_k((y_n)_k) = \mathrm{argmax}_k(\vec{w}_k \vec{x}_n)$
- Loss ? : see exercise "*Cross Entropy*" in TD sheet.

Multi-class classification

Other (expensive) strategies :

- ***one-versus-rest*** (K) (without *softmax* , also)
- ***one-versus-one*** $(K(K\text{-}1)/2)$

References:

- **Linear classifiers in general**:
  → Bishop book, page 179-196, section 4.1
- Loss Function J2 (MSE for classif)
  → Bishop book, page **184-18**6, 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
- Multi-class Classification, OVR and OVO strategies