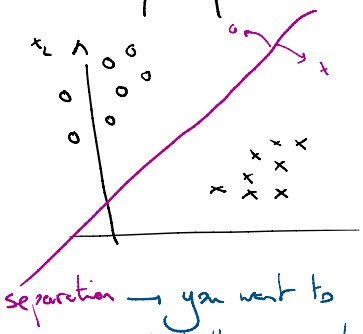
Supervised learning

martes, 16 de febrero de 2021 10:36

III Supervised Learning

A] The perceptron



I want to find a function such that

$$\begin{cases} f((x_1, x_2) \in \times) \to \times \\ f((x_1, x_2) \in o) \to o \end{cases}$$

separation → you want to find this separation automatically

in D dimensions, how do you define a line?

→ $w_0 + \sum_{i=1}^{D} w_i x_i = 0$ eq of a line

(x_i: coordinates)

parameters of the line: $\{w_0, \vec{w}\}$ (w_0: the bias)

(weights)

Let's define the model: $y = \text{sgn}\left(w_0 + \sum_{i=1}^{D} w_i x_i\right)$

(y: output; x_i: input)

We have to discretize the labels of the dataset

$\times$: +1

o: −1

I want to find $\{w_0, \vec{w}\}$ s.t. $\forall i$ in my dataset

$$\tilde{y}_i = f(w_0 + \sum w_i x_i)$$

↳ label of data i

Perceptron algo: for each i, change the weights :. if i is not correctly classified

$$\vec{w} \leftarrow \vec{w} + \tilde{y}_i \vec{x}_i$$

$$w_0 \leftarrow w_0 + \tilde{y}_i$$

You can show the this algo reaches a solution if: the dataset is linearly seperable

B] Multiclass Perceptron

Notation: $\vec{x} = \begin{pmatrix} 1 \\ \vec{x} \end{pmatrix} \in \mathbb{R}^{D+1}$, $\vec{w} = \begin{pmatrix} w_0 \\ \vec{w} \end{pmatrix}$

pb with binary perceptron:
- sign fct cannot be derived
- no information on the proba of a data to belong to one cluster.

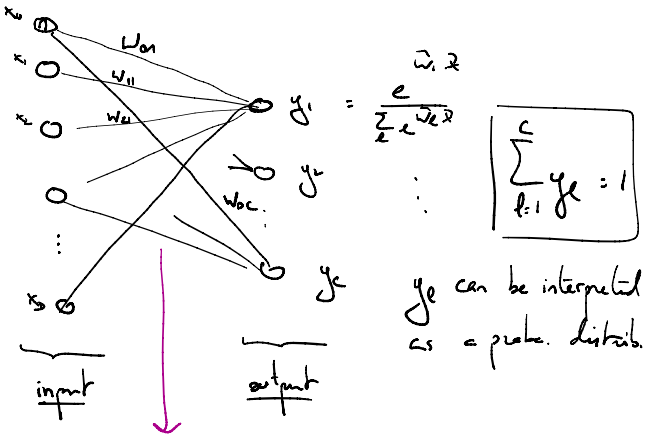
We will have C classes.

i) Let's define the model: Instead of 1 output I will have C of them

output n°h: $y_h = \frac{\exp\left(\sum_i w_{hi} x_i\right)}{\sum_{l=1}^{C} \exp\left(\sum_i w_{li} x_i\right)} = \frac{\exp(\vec{w}_h \cdot \vec{x})}{\sum_{l=1}^{C} \exp(\vec{w}_l \cdot \vec{x})}$

for each output, I have an associated set of weights $\vec{w}_h$
$\vec{w}_h \in \mathbb{R}^{D+1}$

- graphical representation



y_l can be interpreted as a proba. distrib.

the edges represent the weights

- How classes are separated ?

$$\frac{y_h}{y_l} = \frac{e^{\vec{w}_h \cdot \vec{x}}}{e^{\vec{w}_l \cdot \vec{x}}} = \exp\left((\vec{w}_h - \vec{w}_l) \cdot \vec{x}\right)$$

it defines an hyperplan of dim $D-1$ between class h & l.

- convexity property: let's have two pts A & B in class k according to my model

$$\Rightarrow \quad \vec{w}_k \cdot \vec{x}_A > \vec{w}_j \cdot \vec{x}_A \,, \quad \forall j \neq k$$
$$\vec{w}_k \vec{x}_B > \vec{w}_j \vec{x}_B \,, \quad \text{——}$$

hence $\forall$ pt defined as $\vec{x} = \lambda \vec{x}_A + (1-\lambda)\vec{x}_B$, $\lambda \in [0,1]$

we have: $\vec{w}_k \vec{x} = \lambda \vec{w}_k \cdot \vec{x}_A + (1-\lambda)\vec{w}_k \cdot \vec{x}_B$

$$> \lambda \vec{w}_j \vec{x}_A + (1-\lambda)\vec{w}_j \vec{x}_B \quad \forall j \neq k$$

II) Learning of the model

we have to define a { loss / objective } function and { minimize / maximize } it

Since we have an interpretation in terms of proba, we can define a Likelihood.

we said: proba($\vec{x}$ is in class k) = $y_k(\vec{x})$

→ we want to maximize the proba that $\vec{x}$ is in the correct class

Dataset: $(\vec{x}_d, \tilde{y}_d)$, $d = 1, \dots, n$

↳ label of data d

we will transform it into a one-hot vector.

instead of having $\tilde{y}_d = 2 \rightarrow \vec{\tilde{y}}_d = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$ = { 1 on coord 2 / 0 elsewhere }

For data $(\vec{x}, \vec{\tilde{y}})$ I define the likelihood: $p_n(\vec{x} \in \text{class } \tilde{y})$

$$= \prod_{k=1}^{c} \left(y_k(\vec{x}) \right)^{\tilde{y}_k}$$

→ log-likelihood: $\boxed{\mathcal{L}(\vec{x}) = \sum_k \tilde{y}_k \log\left(y_k(\vec{x}) \right)}$ (cross-entropy)

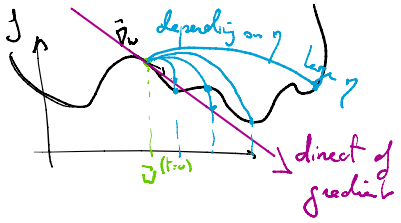
we will make a gradient ascent of $\mathcal{L}$

we will make a gradient ascent of $\mathcal{L}$

Compute : $\boxed{\frac{\partial \mathcal{L}}{\partial w_i} = x_i\left(\tilde{y}_e - y_e(\vec{x})\right)}$ strongly suggest to derive it

iteration rule : $w_i \leftarrow w_i - \eta \cdot x_i\left(\tilde{y}_e - y_e(\vec{x})\right)$

In practice : • η ? the learning rate



• Mini-batch : the dataset is partitioned into small groups of mb data

. $\frac{n}{mb}$ groups / mini-batches

. each group/mini-batch contains mb data points

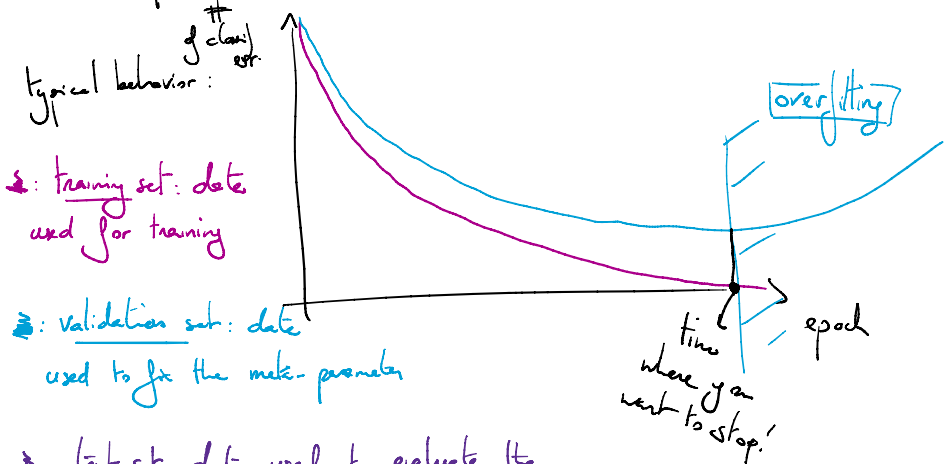
. a data-pt belong to only one group/mini-batch

typically $mb \in [8, 200]$

the update becomes : $w_i \leftarrow w_i - \eta \frac{1}{mb} \sum_{d \in \text{batch}(b)} x_i\left(\tilde{y}_e - y_e(\vec{x})\right)$

• An epoch : it is when all the data of the dataset have been used one time (only one) to update the parameters

• Meta-parameters & evaluation



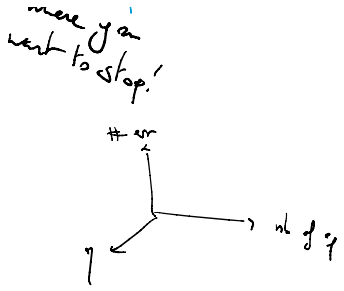
: training set : data used for training

: validation set : data used to fix the meta-parameters

: test set : data used to evaluate the

used to fix the meta-parameter

- test set: data used to evaluate the performance of your network



Other meta-parameter: . nb of epoch
. learning rate η
. L_2 regularization λ_2
. L_1 ——— λ_1

L_2-reg: you add : $-\frac{\lambda_2}{2}\sum_{i,j} w_{ij}^2$ to the log-likelihood

~ gaussian prior on the weights

L_1-reg: : $-\lambda_1 \sum_{i,j} |w_{ij}|$ to the log-l

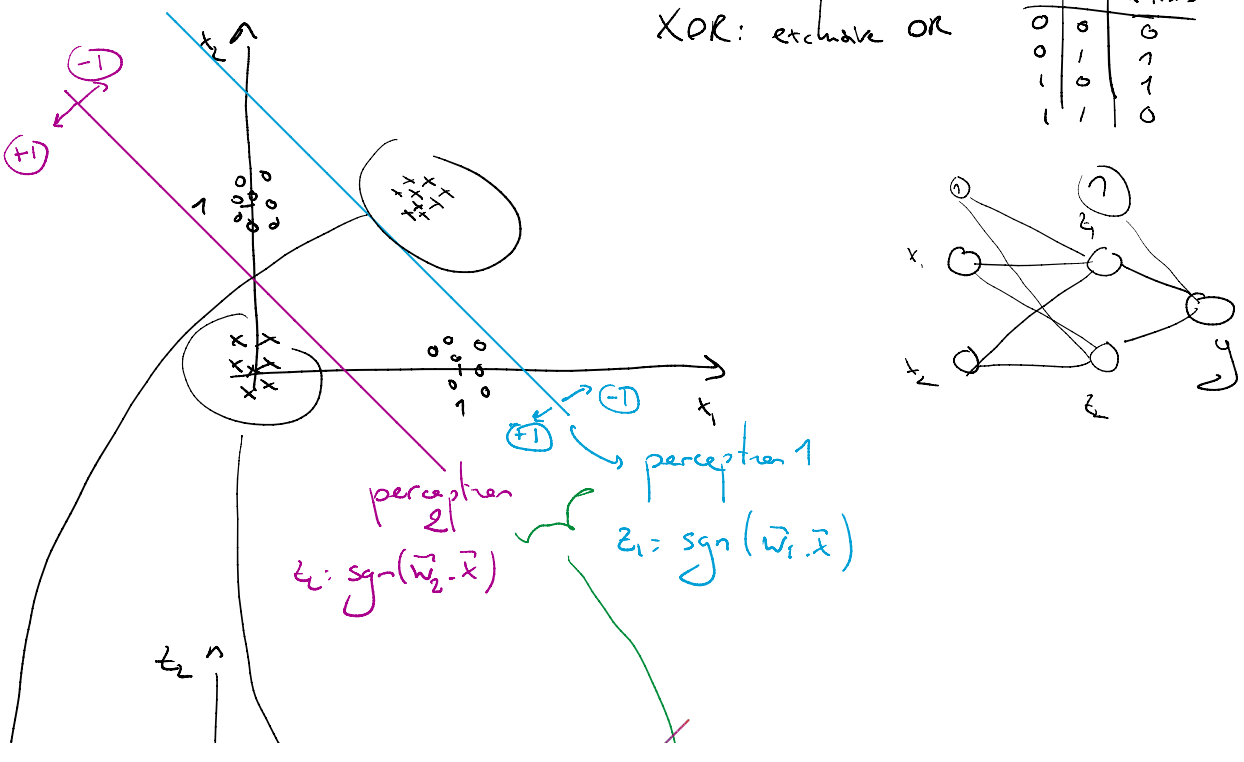
~ Laplace prior on the weights

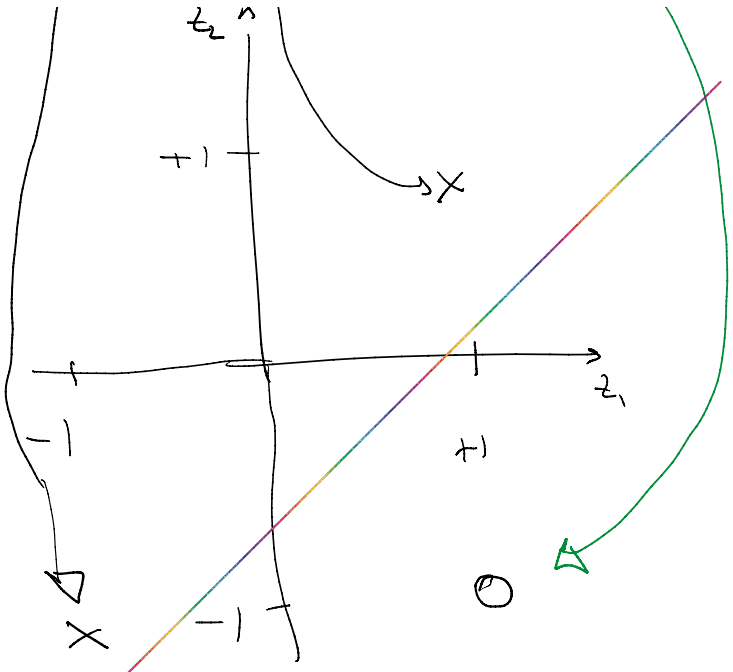
C] Multilayer Perceptron

1) Why multilayer is good ?

XOR: exclusive OR

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

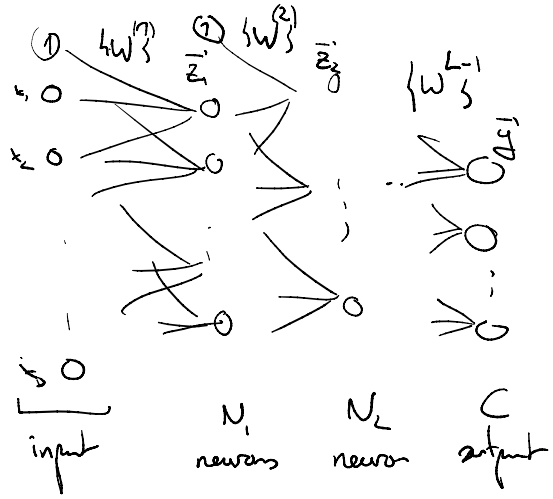




↳ perceptron $y = \text{sgn}\left(t_1 w_1' + t_2 w_2' + w_0' \right)$

Multilayer perceptron is a good model for complex dataset.

II) Definitions :



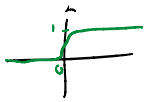
$$\vec{z}_i^{(1)} = f_1\left(\vec{w}_i^{(1)} \cdot \vec{x} \right), \quad f_1 \text{: an activation function}, \quad i = 1, \dots, N_1$$

$$\vec{z}_i^{(2)} = f_2\left(\vec{w}_i^{(2)} \cdot \vec{z}^{(1)} \right), \quad i = 1, \dots, N_2$$

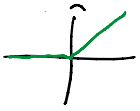
$$\vdots$$

$$\dots \quad \exp\left(\vec{w}_k^{(L-1)} \cdot \vec{z}^{(L-2)} \right)$$

$$y_h = \frac{\exp\left(\vec{w}_h^{(L-1)} \cdot \vec{z}^{(L-1)}\right)}{\sum_\ell \exp\left(\vec{w}_\ell^{(L-1)} \cdot \vec{z}^{(L-1)}\right)}$$

Typical activation fct: - sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$ 

- ReLu: rectified linear unit

$\text{relu}(x) = \max(0, x)$ 

possible output: . for classif: softmax $\frac{e^{\vec{w}_k \vec{x}}}{\sum_\ell \exp(\vec{w}_\ell \vec{x})}$

. regression in $\mathbb{R}$: Linear act (identity)

. regression in $[0,1]$: sigmoid

iii) Learning: maximizing the likelihood

$p(\vec{x} \in \text{class } y)$

$\mathcal{L}(x) = \sum_h \tilde{y}_h \log\left(y_h(\vec{x})\right)$

$\frac{\partial \mathcal{L}}{\partial w^{(1)}}, \frac{\partial \mathcal{L}}{\partial w^{(2)}}, \dots, \frac{\partial \mathcal{L}}{\partial w^{(L-1)}}$

$\vec{z}_i^{(L-2)} \left(\tilde{y}_\ell - y_\ell(\vec{x})\right)$

chain rule