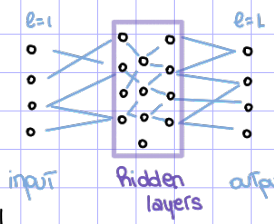
8 HOW TO COMPUTE THE WEIGHT ?

8.1 FFNN recall



All the layer are fully connected with σ_ℓ the activation function of the layer ℓ

$$y_i^\ell = \sum_{j=1}^{N_{\ell-1}} \sigma_\ell \left(w_{ij}^\ell \, x_j^{\ell-1} \right) \quad \forall i \in (1,..,N_\ell) \quad \forall \ell \in (2,..,L)$$

(ℓ = layer; $N_{\ell-1}$ = input before layer; w_{ij}^ℓ = weight; $x_j^{\ell-1}$ = non linear; y_i^ℓ = neuron i; y := output)

$\Rightarrow$ The non-linearity is contained in σ_ℓ

↓ Rewrite in compact form $y^\ell = f_\ell (w^\ell, y^{\ell-1})$ ($y^{\ell-1} \equiv x^{\ell-1}$ in which we consider the input of layer j is the output of layer $j-1$)

Imagine to know the weight w (you already solve the optimization problem), the algorithm to the evaluation process is

ALGORITHM to compute y

```
y' = x        → input
for ℓ = 2,..,L
    z^ℓ = w^ℓ y^{ℓ-1}
    y^ℓ = σ^ℓ(z^ℓ)
end
y = y^L       → output
```

↳ The optimization problem is the offline phase that is NOT fast

This is similar to the online phase that is very fast

How to compute the weight? TRAINING PHASE: we have different strategies

1. **Supervised Learning**: we have known snapshot $\longrightarrow \{x^n, y^n\}_{n=1,..,N}$ that minimize the loss (x: input, n: layer)
   ↳ the network is trained using example of input & output given by a teacher $\Rightarrow$ **goal**: learn the rule to map $x \mapsto y$
2. **Unsupervised learning**: we do NOT have snapshot $\longrightarrow$ PINN: Physics Informed Neural Network → See next lessons
   ↳ the network is trained using only inputs: no outputs are given $\Rightarrow$ **goal**: discover hidden patterns in the data
3. **Reinforcement Learning**: You give information and reword and the net learn
   ↳ the network is trained in a dynamic environment to perform a fixed goal: while training, the environment gives feedback (positive & negative reward) and the network try to maximizes

8.2 Supervised learning: we want to approximate the target function

Consider the function $\underline{f}: K \subset \mathbb{R}^n \longrightarrow \mathbb{R}^m$ (compact) that is the function in the hidden layer that produce the output

($\underline{f}$ unknown) $x \longmapsto y$

We know N "samples": $(x^n, y^n = \underline{f}(x^n))$

Problem - Given FFNN fixed $(L, N_\ell, \ell = 2,..,L)$

Find w^ℓ s.t.

$y = f(w^\ell, x)$ → where f is an approximation of $\underline{f}$

IDEA: we want to find w so the f approximate $\underline{f}$ (the hidden function) that we do NOT have, but we have the model (FFNN) so we know (*)

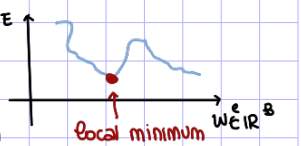
We suppose to know some snapshots $(x^n, \bar{y}^n)_{n=1,..,N}$ and we select the **loss function** n-th loss error

$$E_n = \frac{1}{2} \| y^n - \bar{y}^n \|_2^2 = \frac{1}{2} \| f(w^\ell, x^n) - \bar{y}^n \|_2^2$$

(y^n predicted value; $\bar{y}^n$ real value)

↓ The **total loss error** is $E = \frac{1}{N} \sum_{n=1}^{N} E_n(w)$ and we want to find a **local minimum**

we try to get a minimization problem

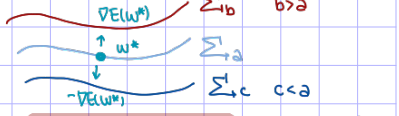


We need a **minimization method** $\longrightarrow$ to solve the minimization problem

could be NOT unique

a. **GRADIENT DESCENT METHOD**: based on the gradient of the loss function

Find $\min_{w \in \mathbb{R}^B} E(w)$ where $\Sigma^1_a := \{ w \in \mathbb{R}^B : E(w) = a \}$ $\forall a \in \mathbb{R}$ are the level curve



To go in a curve with lower value we use the gradient $\Rightarrow$ **iterative step**: $w_i = w^{i-1} - \alpha \nabla E(w^{i-1})$ $\alpha > 0 \in \mathbb{R}^+$ (learning rate)

BUT this works well iif the dimension is NOT so high, because we have $\nabla E(w) = \sum_{n=1}^{N} \nabla E_n(w)$ so it depends on N := number of snapshots that has to be high to learn properly

↓

b. Solution? **STOCHASTIC GRADIENT DESCENT ALGORITHM**: compute the gradient for only some snapshots (Garantees only to reach a local minimum, not a global one)

$$w_i = w_{i-1} - \alpha \frac{1}{M} \sum_{n \in S_M} \nabla E_m (w_{i-1}) \qquad M \ll N$$

↓ take $M \neq$ snapshots, chosen in a random way, S_M is called **mini-batch**

Example - •) Real gradient $\nabla E = \sum_{n=1}^{N} \nabla E_n$

•) Stocastic gradient set $S_M = \{n_1, \dots, n_M\}$ $n_i \in \{1, \dots, N\}$ $\forall i \in \{1, \dots, M\}$ $\tilde{\nabla E} = \frac{1}{M} \sum_{n \in S_M} \nabla E_n$

mini-batch: selected randomly at each epoc, is a subset of all the snapshots

Each iteration of the stochastic gradient descent is called **epoch**

ADAM (Adaptive Moment Estimation) ALGORITHM: improvement of the STOCHASTIC in which exploit the previous computation all the previous gradient are called momentum of E

This is the standard used

L-BGFS: Second order algorithm → exploit the Hessian (estimate), so it is more precise → following quasi-Newton method

(Limited Memory Broyden-Fletcher-Goldfarb-Shanno)

c. How to compute $\nabla E(w)$? **BACKPROPAGATION** → Yann le Cun 1988

Consider $E(w) = \frac{1}{M} \sum_{m=1}^{M} \frac{1}{2} \| y^m(w) - \bar{y}^m \|_2^2$, where i.e. $w = \begin{pmatrix} w^1_{11} \\ w^1_{12} \\ \vdots \\ w^1_{24} \end{pmatrix}$ if $\in \mathbb{R}^{3,2}$ $W^1 \longrightarrow W^2 \in \mathbb{R}^{2,4}$ perceptron

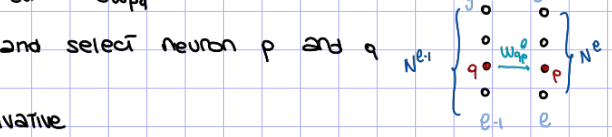
From definition, we have $\nabla E_n(w) = \frac{1}{2} \cdot 2 (y^n(w) - \bar{y}^n) \frac{\partial y^n(w)}{\partial w}$, where $y^n(w) = f(w, x^n)$

($y^n(w)$: we know; $\bar{y}^n$: we do not know; $\frac{\partial y^n(w)}{\partial w}$: how to compute that?)

we know everything, but not the partial derivative $f: \mathbb{R}^S \to \mathbb{R}^m$ $\frac{\partial f}{\partial w} = \left\{ \frac{\partial f_j}{\partial w^e_{pq}} \right\}$ → we need to compute the Jacobian

Imagine to fix the layers $e-1, e$ that are fully connected, and select neuron p and q

$y^e = \sigma_e(w^e \cdot y^{e-1})$



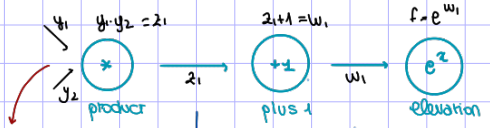
So we can use the chain rule to compute the partial derivative

$\frac{\partial f_j}{\partial w^e_{pq}} = \frac{\partial f_j}{\partial y^e_p} \cdot \frac{\partial y^e_p}{\partial w^e_{pq}}$

$\frac{\partial y^e_p}{\partial w^e_{pq}}$: because I derive wrt w^e_{pq} so all the other element becomes ϕ

$\frac{\partial f_j}{\partial y^e_p}$: we do not know i.e. if $f_j = e^{(y_1 y_2 + 1)} \Rightarrow \frac{\partial f_j}{\partial y_1} = e^{y_1 y_2 + 1} \cdot y_2$ $\frac{\partial f_j}{\partial y_2} = e^{y_2 y_1 + 1} \cdot y_1$

BUT how to compute that in NN logic?



$f = e^{y_1 y_2 + 1} \longrightarrow \frac{\partial f}{\partial f} = 1$ where $\frac{\partial f}{\partial w_1} = e^{w_1}$, $\frac{\partial w_1}{\partial z_1} = 1$, $\frac{\partial z_1}{\partial y_1} = y_2$

chain rule $\frac{\partial f}{\partial y_1} = \frac{\partial f}{\partial z_1} \frac{\partial z_1}{\partial y_1}$

chain rule $\frac{\partial f}{\partial z_1} = \frac{\partial f}{\partial w_1} \frac{\partial w_1}{\partial z_1}$

chain rule $\frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial F} \frac{\partial F}{\partial w_1}$

Hence, go back we have all the element and we can compute everything

$\frac{\partial f}{\partial y_1} = \frac{\partial z_1}{\partial y_1} \cdot \frac{\partial w_1}{\partial z_1} \cdot \frac{\partial f}{\partial w_1} \cdot \frac{\partial f}{\partial F} = y_2 e^{y_1 y_2 + 1}$

Starting to the end, we know everything so we can compute the derivative!

Now, we can apply this IDEA to compute $\frac{\partial f_j}{\partial y^e_p} = \frac{\partial f_j}{\partial y^{e+1}_h} \cdot \frac{\partial y^{e+1}_h}{\partial y^e_p}$, where by definition $y^{e+1}_h = \sigma_{e+1}\left(\sum_{q=1}^{N_e} w^{e+1}_{hq} y^e_q \right)$

$= f(w, x^n) = y^n$

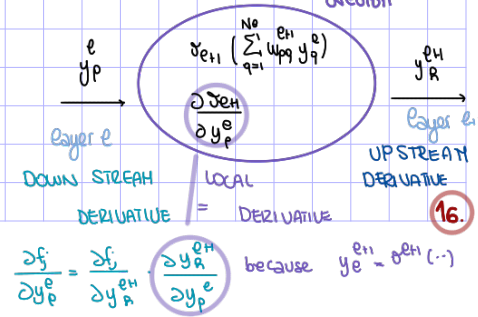
$\frac{\partial y^{e+1}_h}{\partial y^e_p} = \sigma'_{e+1}(w^{e+1} y^e) w^{e+1}_{hp}$

BUT! Know we do NOT know this element

We can put in the case $e+1 = L \Rightarrow \frac{\partial f}{\partial y^L} = \frac{\partial y^L}{\partial y^L} = I$ identity

So we can do all the steps: $\frac{\partial f}{\partial y^{L-1}} \to \frac{\partial f}{\partial y^{L-2}} \to \dots \to \frac{\partial f}{\partial y^1} \to \frac{\partial y^1}{\partial x}$ ⇒ zoom

And $\frac{\partial y^1}{\partial x} = \frac{\partial}{\partial x_i} \left(\sigma_1(w^1 \cdot x) \right) = \sigma'_1(w^1 \cdot x) w^1_{ij}$



$\frac{\partial f_j}{\partial y^e_p} = \frac{\partial f_j}{\partial y^{e+1}_R} \cdot \frac{\partial y^{e+1}_R}{\partial y^e_p}$ because $y^{e+1}_e = \sigma^{e+1}(\dots)$

To see more detail on UNIVERSAL APPROXIMATION THEORMS see lecture notes Lesson 8