Mapi3 Machine Learning : Réseaux de Neurones

Contexte $(x, y) \sim \mathbb{P}$. Trouver $f \in F$ tq $y \approx f(x)$.

Réseaux de neurones : Une classe particulière de F's !

$$f_{(\theta_1,\dots,\theta_N)}(x) = f_{\theta_N}^{(N)}\left(\dots\left(f_{\theta_1}^{(1)}(x)\right)\right) \quad (\theta_1,\dots,\theta_N) \in \mathbb{R}^{d_1} \times \dots \times \mathbb{R}^{d_N}.$$

$$F = \left\{ f_{(\theta_1,\dots,\theta_N)} \; ; \; (\theta_1,\dots,\theta_N) \in \mathbb{R}^{d_1} \times \dots \times \mathbb{R}^{d_N} \right\}$$

Vocabulaire :
- $f_\bullet$ = architechture
- $(\theta_1,\dots,\theta_N)$ = poids
- N = profondeur
- $\left(f^{(n)} \circ \dots \circ f^{(1)}\right)$ = layer (ou niveau) n
- $\left(f^{(n)} \circ \dots \circ f^{(1)}\right)(x)$ = activations au layer n.

I. Quelques architechtures classiques

Principe général des réseaux de neurones :

Chaque $f^{(i)}$ est en général très simple, la complexité et l'expressivité provient de la profondeur.

1) Les couches linéaires

Une classe très simple de fonctions est celle des fonctions linéaires.

$f^{(i)}$ ( ) ( )

$\theta_i(x) = W_i x + b_i$; $\theta_i = (W_i, b_i)$.

Vocabulaire :
- W = poids
- b = biais

⚠ Le terme de "poids" a un double usage et le terme de fonction linéaire désigne en fait une fonction affine.

2) Les activations et couches non-linéaires

Simplement composer des applications affines n'est en général pas très intéressant car le résultat est aussi affine.

L'expressivité des réseaux de neurones provient de la composition d'applications linéaires et de rectifications non-linéaires.

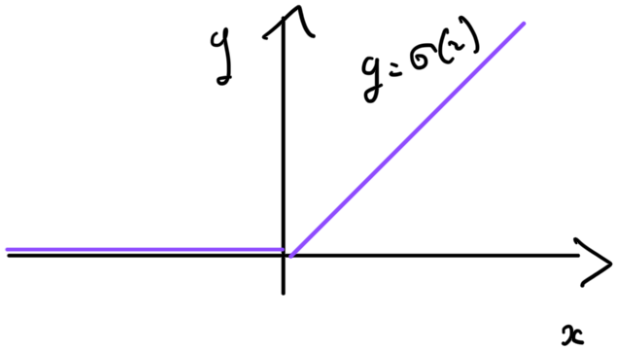
• Les activations

Soient, $P^{(i)}_{\theta_i}(x) = \sigma(W_i x + b_i)$; $\theta_i = (W_i, b_i)$

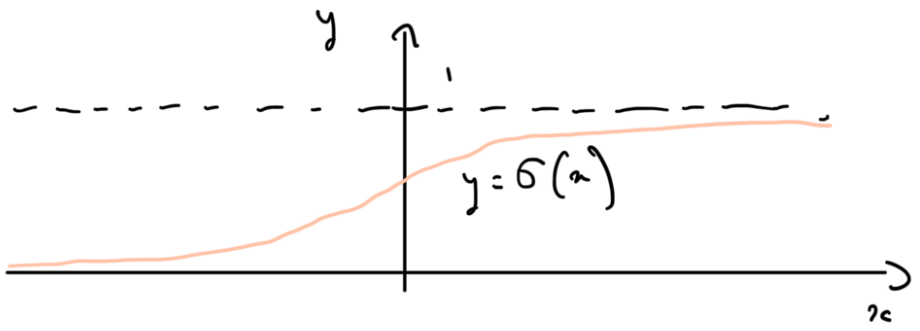
σ : fonction d'activation.

exemples classiques :

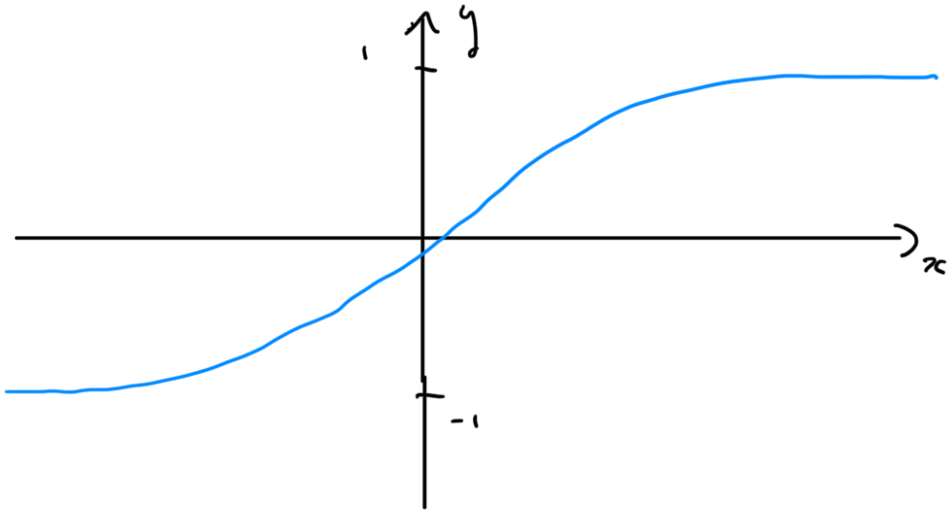
• ReLu : $\sigma(x) = (x_+) = \max(0, x)$.



• Sigmoid : $\sigma(v) = \frac{1}{1 + e^{-v}}$



• Tanh $\qquad \sigma(v) = \dfrac{e^{v} - e^{-v}}{e^{v} + e^{-v}}$



• Le Softmax

En classification, on veut $f(x) \in \Delta_{d-1}$ avec

$$\Delta_{d-1} = \left\{ x \in \mathbb{R}^d \text{ st } x \geq 0 \text{ et } \mathbb{1}^T x = 1 \right\}.$$

L'interprétation est que $f(x)$ encode les probabilités d'être dans telle ou telle classe.

Il faut alors convertir un vecteur de $\mathbb{R}^d$ en un point de Δ_{d-1}. C'est réalisé grâce au Softmax.

$$\text{Softmax}\left(\begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}\right) = \begin{pmatrix} \dfrac{e^{x_1}}{\sum_i e^{x_i}} \\ \vdots \end{pmatrix}$$

$$\begin{pmatrix} \vdots \\ e^{x_d} \\ \hline \sum_i e^{x_d} \end{pmatrix}$$

3) Les fonctions de perte

Un réseau est ensuite entraîné avec une fonction de perte $\ell(\cdot,\cdot)$

• En régression

La fonction de perte est souvent la fonction de perte quadratique

$$\ell(y, f(x)) = \|y - f(x)\|_2^2 .$$

• En classification

En classification, il faut pouvoir comparer deux vecteurs de probabilités de Δ_{d-1}.
C'est effectué avec la cross-entropy loss

$$\ell\left(\begin{pmatrix} y_1 \\ \vdots \\ y_d \end{pmatrix}, \begin{pmatrix} p_1 \\ \vdots \\ p_d \end{pmatrix} = f(x)\right) = -\sum_{i=1}^{d} y_i \log(p_i) .$$

Proposition $\forall p, q \in \Delta_{d-1}$, $\ell(p,q) \geq 0$ et $\ell(p,q) = 0$ si et seulement si $p = q$.

Preuve : $\ell(p,q) - \ell(p,p) = -\sum_i p_i \log(q_i) - \left(-\sum_i p_i \log(p_i)\right)$

$$= -\sum_i p_i \log\left(\frac{q_i}{p_i}\right)$$

$$\geq -\log\left(\sum_i p_i \frac{q_i}{p_i}\right)$$

Jensen

$$= -\log\left(\sum_i q_i\right) = -\log(1) = 0.$$

Il y a égalité dans l'inégalité de Jensen ssi $\forall i,\ p_i = q_i$.

La suite au prochain cours...

II. Optimisation et règle de la chaine

$$f(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(y_i, f_\theta(x_i)\right) + \Omega(\theta).$$

$$\text{d'où : } \nabla_\theta f(\theta) = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta \ell\left(y_i, f_\theta(x_i)\right) + \nabla_\theta \Omega(\theta).$$

Descente de gradient :

- Commencer à θ_0
- Tant que (critère à vérifier pour continuer)

$$\theta_t \leftarrow \theta_{t-1} - \gamma_t \nabla_\theta f(\theta_{t-1})$$

$\uparrow$ "learning rate" à l'itération t.

⚠ Lorsque n est grand, calculer $\nabla_\theta f\left(= \frac{1}{n}\sum_{i=1}^{n} \dots\right)$ peut être trop cher, il est alors possible de remplacer $\nabla_\theta f$ par un <u>estimateur</u> calculé sur un <u>sous-ensemble du jeu de données</u> uniquement

Soit $B \subseteq [\![1, n]\!]$

$$\nabla_\theta^{(B)}(\theta) = \frac{1}{|B|}\sum_{b \in B} \nabla_\theta \ell\left(y_b, f_\theta(x_b)\right) + \nabla_\theta \Omega(\theta)$$

Descente de gradient stochastique :

- Commencer à θ_0
- Tant que (critère à vérifier pour continuer)
  - Sélectionner $B \subseteq [\![1, n]\!]$ selon la règle choisie (aléa ou déterministe)
  - $\theta_t \leftarrow \theta_{t-1} - \gamma_t \nabla_\theta^{(B)} f(\theta_{t-1})$

B s'appelle un "batch".

1) Algorithme de backpropagation

$$x \xrightarrow{A_1 \cdot + b_1} a_1 \xrightarrow{\sigma(\cdot)} b_1 \xrightarrow{A_2 \cdot + b_2} a_2 \cdots a_N \xrightarrow{\sigma(\cdot)} b_N \xrightarrow{A_{out} \cdot + b_{out}} a_{out} \xrightarrow{P(y, \cdot)}$$

Comment calculer le gradient en les paramètres de $P(y, h(x))$?

h : fonction encodée par le réseau.

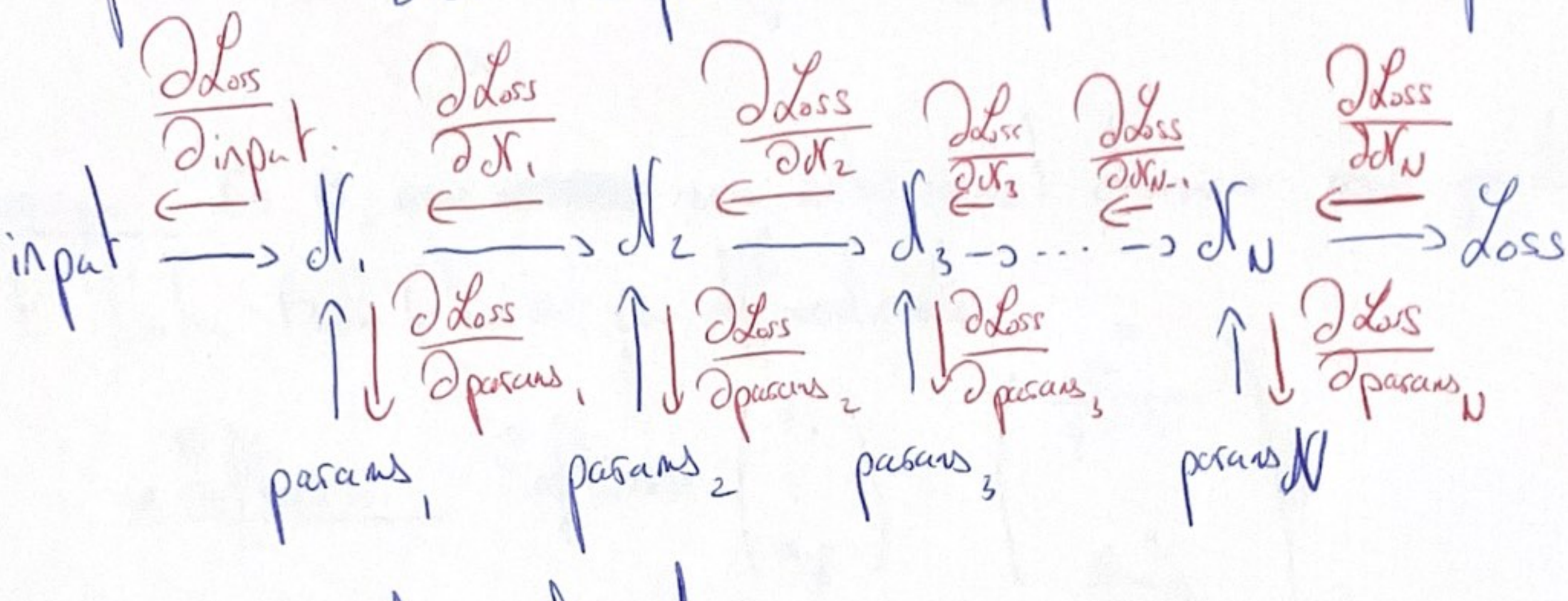
Solution : Règle de la chaîne !

Si ρ et γ sont différentiables, $d(g \circ \rho) = (dg \circ \rho) \circ d\rho$

Ce qui donne, en notation informelle :

Si $N = N(input, params)$, et $Loss = Loss(N)$,

(*) $\frac{\partial Loss}{\partial params} = \frac{\partial Loss}{\partial N} \frac{\partial N}{\partial params}$; $\frac{\partial Loss}{\partial input} = \frac{\partial Loss}{\partial N} \frac{\partial N}{\partial input}$ (**)



Algorithme Forward Backward

- Forward : Calculer $N(x, \theta)$ en gardant les activations dans chaque noeud

- Backward : En remontant à partir de la Loss,

  pour chaque noeud N

  - Calculer $\frac{\partial Loss}{\partial params(N)}$ (avec (*))
  - Calculer $\frac{\partial Loss}{\partial input}(N)$ (avec (**))

2) Calculs des gradients à la main

• **Relu** $\quad \text{Relu}\begin{pmatrix} x_1 \\ \vdots \\ x_6 \end{pmatrix} = \begin{pmatrix} (x_1)_+ \\ \vdots \\ (x_6)_+ \end{pmatrix}$

donc $\dfrac{\partial \text{Relu}(x)_j}{\partial x_i} = \delta_{ij} \mathbb{1}_{x_i > 0}$

remarque : En 0, nous avons arbitrairement choisi un sous-gradient car la fonction Relu(·) n'est pas différentiable.

• **Softmax** $\quad \text{Softmax}\begin{pmatrix} x_1 \\ \vdots \\ x_6 \end{pmatrix} = \begin{pmatrix} \dfrac{e^{x_1}}{\sum_5 e^{x_5}} \\ \vdots \\ \dfrac{e^{x_6}}{\sum_5 e^{x_5}} \end{pmatrix}$

$$\frac{\partial \text{Softmax}(x)_j}{\partial x_i} = \text{Softmax}(x)_j \left(\delta_{ij} - \text{Softmax}(x)_i\right)$$

• **Affine :** $\quad \text{aff}(x) = Ax + b$

$$\frac{\partial \text{aff}(x)_j}{\partial x_i} = A_{ji} \qquad \frac{\partial \text{aff}(x)_i}{\partial b_i} = 1$$

$$\frac{\partial \text{aff}(\cdot)_b}{\partial A_{ij}} = \delta_{ib}\, x_j$$

III Approximation Universelle

Nous allons prouver que les réseaux de neurones ont la capacité d'approximer les fonctions continues aussi proche que l'on veut.

Réseaux à une couche cachée

$$f(x) = \sum_{j=1}^{m} \eta_j \left(\omega_j^T x + b_j \right)_+$$

Étape 1 : Les fonctions continues sont approchables par des fonctions continues affines par morceaux en $\|\cdot\|_\infty$.

Soit $g \in C^0([a,b], \mathbb{R})$.

Alors d'après le théorème de Heine, g est absolument continue.

En effet, sinon, $\exists \varepsilon > 0, \forall \eta, \exists a_\eta, b_\eta \in [a,b]$ tq $|a_\eta - b_\eta| < \eta$, $|g(a_\eta) - g(b_\eta)| \geq \varepsilon$.

$\eta_n = \frac{1}{n}$ donne deux suites $(a_n)_{n \geq 1}$ et $(b_n)_{n \geq 1}$

Comme $[a,b]$ est compact, on extrait $a_{\varphi(n)} \to a_\infty \in [a,b]$

De même, on extrait $b_{\varphi \circ \psi(n)} \to b_\infty \in [a,b]$

Alors $a_{\varphi \circ \psi(n)} \to a_\infty$; $b_{\varphi \circ \psi(n)} \to b_\infty$

Comme $\forall n, |a_{\varphi \circ \psi(n)} - b_{\varphi \circ \psi(n)}| \leq \frac{1}{\varphi \circ \psi(n)} \leq \frac{1}{n}$, $a_\infty = b_\infty$.

Par continuité, $|g(a_{\varphi \circ \psi(n)}) - g(b_{\varphi \circ \psi(n)})| \xrightarrow[n \to +\infty]{} 0$ ce qui est absurde.

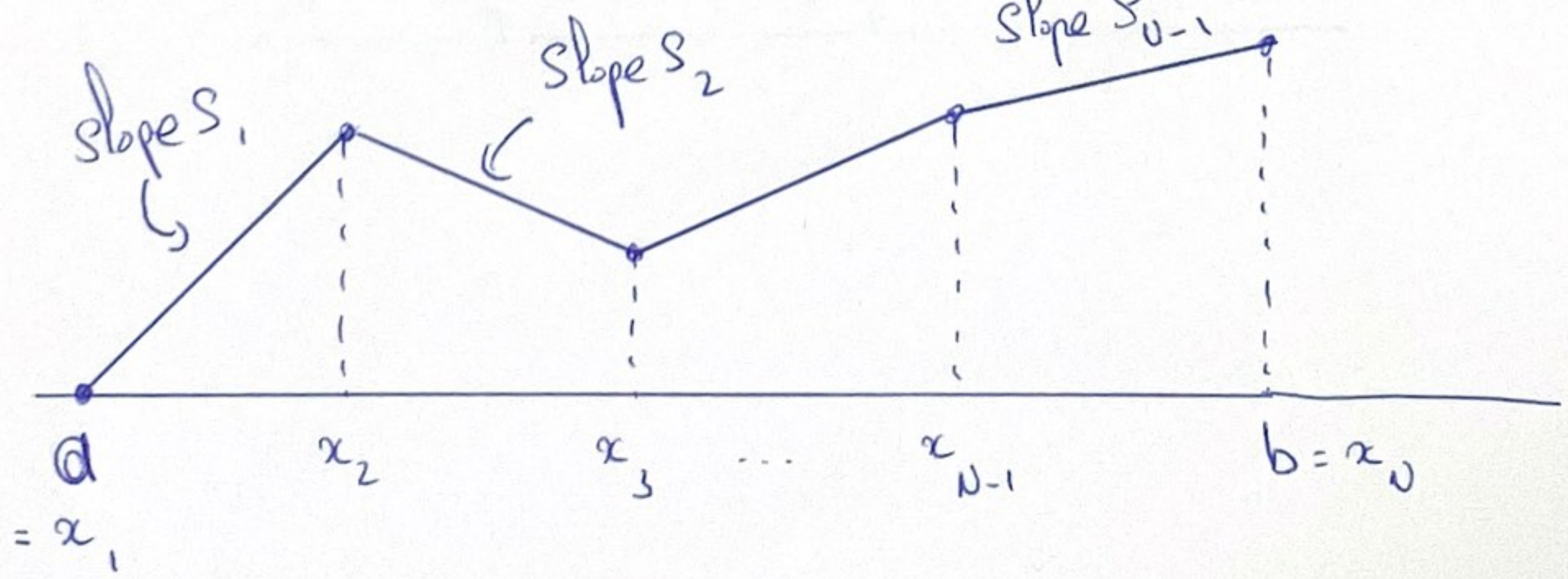
Comme g est absolument continue, Soit $\varepsilon > 0$, il existe $\eta > 0$ tq $|x, y| < \eta \Rightarrow |g(x) - g(y)| \leq \varepsilon$.

Considérons la fonction ($a = 0$ pour simplifier)

$$g_\eta(x) = \begin{cases} g(k\eta) + (x - k\eta)\dfrac{g((k+1)\eta) - g(k\eta)}{\eta} & \text{si } \begin{cases} x \in [k\eta, (k+1)\eta) \\ \text{et } [k\eta, (k+1)\eta) \subset [a,b] \end{cases} \\ g(k\eta) + (x - k\eta)\dfrac{g(b) - g(k\eta)}{b - k\eta} & \text{si } \begin{cases} k\eta \in (a,b) \\ (k+1)\eta > b \\ x \in [k\eta, b) \end{cases} \\ g(b) \text{ si } x = b \end{cases}$$

Alors g_η est continue et satisfait $\forall x \in [a,b] \; |g(x) - g_\eta(x)| \leq 2\varepsilon$

Étape 2: Toutes les fonctions continues et affines par morceaux sur $[a,b]$ peuvent s'écrire comme un réseau de neurones à une couche cachée



- Cas n°1 : $f(a) = 0$

$$f(x) = s_1 (x - x_1)_+ \quad \text{sur} \quad [x_1, x_2]$$

$$f(x) = s_1 (x - x_1)_+ + (s_2 - s_1)(x - x_2)_+ \quad \text{sur} \quad [x_1, x_3].$$

...

$$f(x) = \sum_{i \geq 1} (s_i - s_{i-1})(x - x_i)_+ \quad \text{sur} \quad [x_1, x_N]$$

avec la convention $s_0 = 0$.

- Cas général :

On voit f sur $[a, b]$ comme la restriction d'une fonction f' affine par morceaux et continue sur $[a-1, b]$.

$$f(x) = \sum_{i \geq 0} (s_i - s_{i-1})(x - x_i)_+$$

où $x_0 = a - 1$, $s_0 = f(a)$ et $s_{-1} = 0$