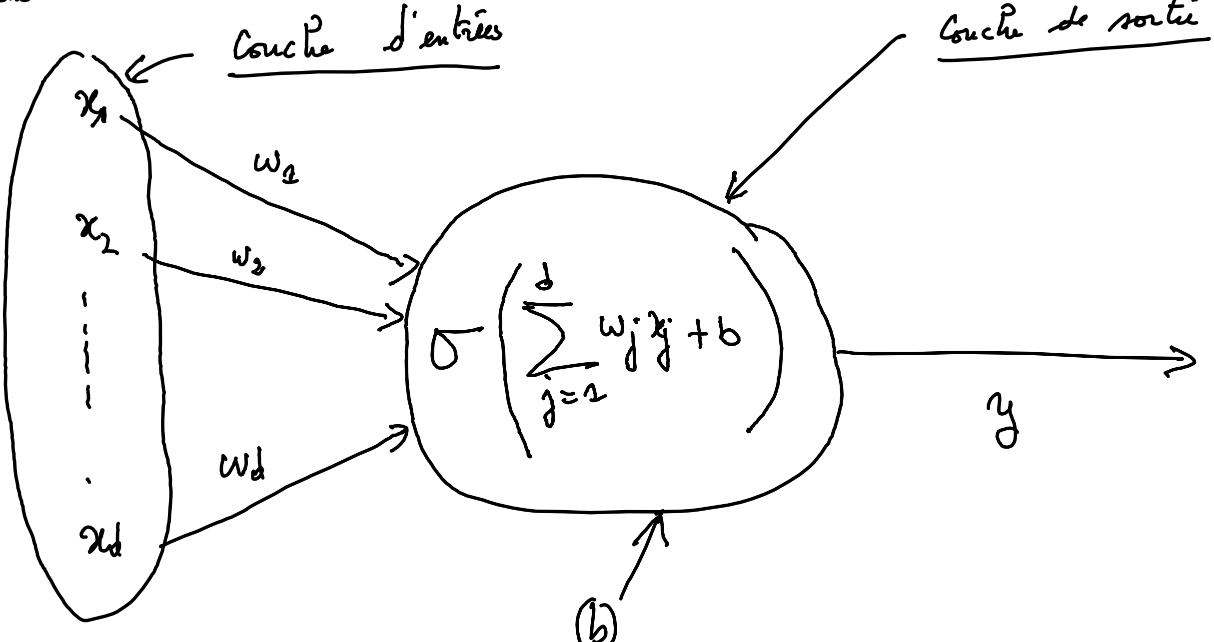
I) Architecture d'un réseau de neurones :

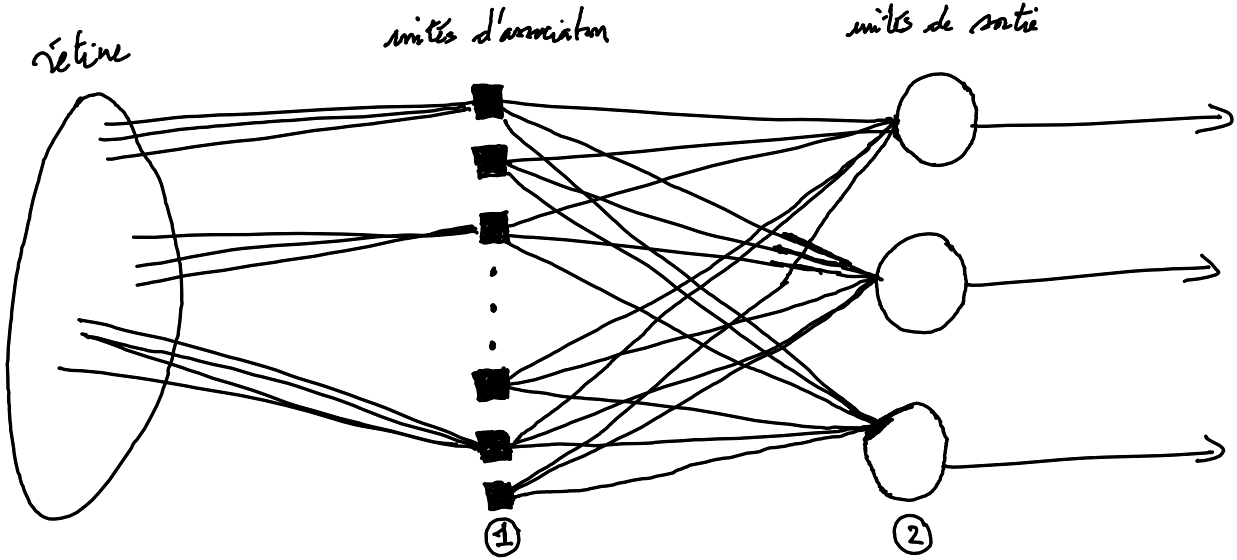
- Le neurophysiologiste Warren McCulloch et le mathématicien Walter Pitts ont réalisé un portrait à l'aide d'un circuit électrique.

- Un neurone de McCulloch-Pitts prend des entrées binaires, calcule une somme pondérée et renvoie 0 si le résultat est inférieur au <u>seuil</u> et 1 dans le cas contraire.



– À la fin des années 50, Frank Rosenblatt, psychologue à Cornell, a travaillé sur les systèmes de décision présents dans l'oeil d'une mouche, qui déterminent sa réaction de fuite.

– En 1958, il a proposé l'idée d'un perceptron qu'il a appelé "Mark I Perceptron". Il s'agissait d'un système avec une relation entrée-sortie simple, modélé par un neurone de McCulloch-Pitts.



Les connexions entre 1 et 2 ne peuvent pas être optimisées.

Paramètres d'un perceptron :

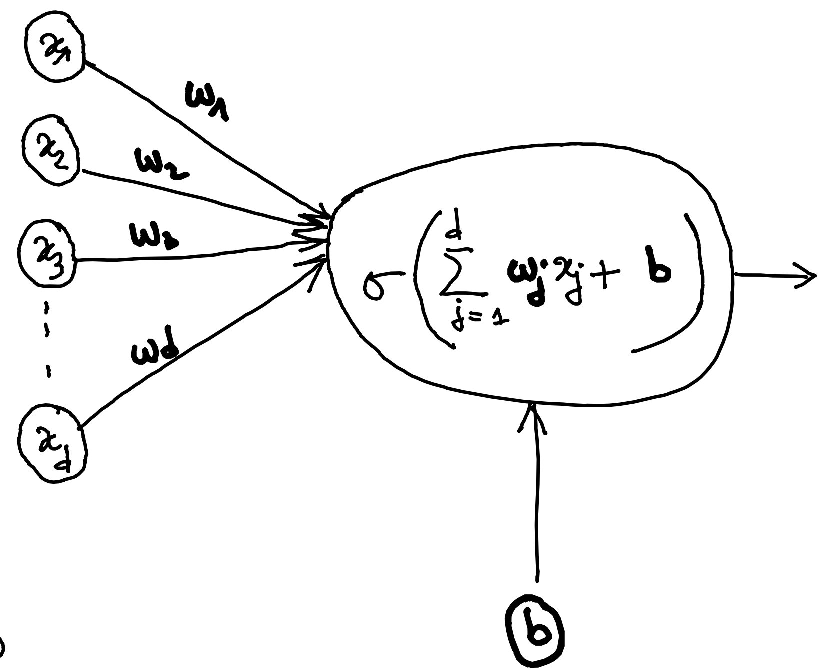
- Fonction d'activation

$$\sigma(z) = \mathbb{1}_{]-\infty, 0[}(z)$$

"Learnable" :

- Poids $w = (w_1, ---, w_d) \in \mathbb{R}^d$

- biais $\underline{\underline{b}} \in \mathbb{R}$



Comment estimer (w, b) ?

Algorithme du Perceptron :

Nous avons les données $D_n = \{(x_i, y_i), i = 1, \dots, d\}$ avec $y_i \in \{-1, +1\}$.

On note $\tilde{w} = (w_1, \dots, w_d, b)$ et $\tilde{x}_i = (x_i, 1)$.

Algorithme du perceptron : (premier algorithme d'apprentissage)

* $\tilde{w} = 0$
* Pour chaque donnée $(\tilde{x}_i, y_i)$
  - Si $y_i \langle \tilde{w}, \tilde{x}_i \rangle \leq 0$ modifier $\tilde{w} = \tilde{w} + y_i \tilde{x}_i$
  - Sinon ne pas modifier $\tilde{w}$.

Questions :

- Pourquoi ça marche ? (intuition)
- Est-ce lié à une descente du gradient ?

Descente du gradient :

1. départ avec $\tilde{w} = 0$ (vecteur de zéros)

2. mise à jour $\tilde{w} \longleftarrow \tilde{w} - \eta \nabla \ell(\tilde{w})$

   ℓ est la fonction de perte à minimiser.

3. arrêt si $\tilde{w}$ ne bouge plus.

Une donnée est mal-classée si $y_i \langle \tilde{w}, \tilde{x}_i \rangle \leq 0$. On veut minimiser la perte

$$\ell(\tilde{w}) = - \sum_{i \in \mathcal{M}_{\tilde{w}}} y_i \langle \tilde{w}, \tilde{x}_i \rangle$$

où $\mathcal{M}_{\tilde{w}}$ est l'ensemble des indices des données mal-classées par $\tilde{w}$.

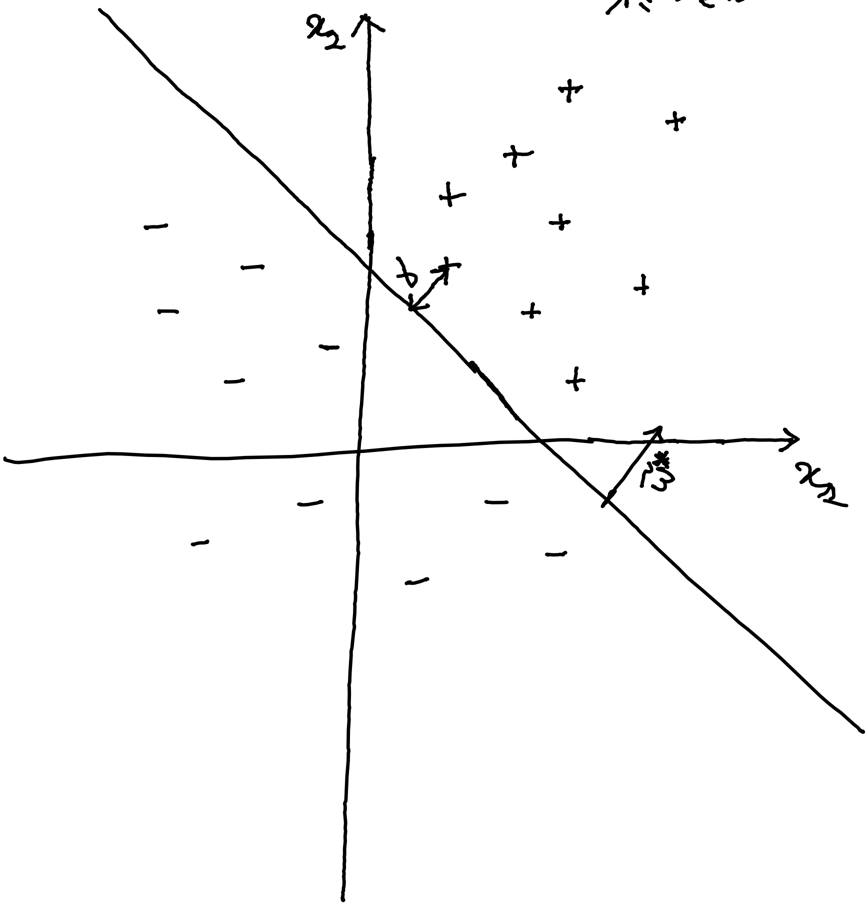
Descente du gradient : (stochastique)

1. Sélectionner au hasard un indice $i \in \mathcal{M}_{\tilde{w}}$

2. Mettre à jour $\tilde{w} \leftarrow \tilde{w} - \eta \nabla \ell_i(\tilde{w}) = \tilde{w} + \eta y_i \tilde{x}_i$.

Algorithme du perceptron : $\eta = 1$.

Exercice : On note $R = \max_{1 \le i \le n} \|x_i\|$. Soit $\tilde{w}^*$ l'hyperplan optimal de marge $\gamma = \min_{1 \le i \le n} y_i \langle \tilde{w}^*, \tilde{x}_i \rangle$ où $\|\tilde{w}^*\| = 1$.



Théorème : (Block (1962) et Novikoff (1963))

Supposons que le jeu de données $D_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$ est linéairement séparable ($\gamma > 0$). On initialise $\tilde{w}_0 = 0$. Le nombre de mises à jour k de l'algorithme du perceptron est borné par

$$k + 1 \le \frac{1 + R^2}{\gamma^2}.$$

Inégalité de Cauchy-Schwarz : $\langle \tilde{w}^*, \tilde{w}_{k+1} \rangle \leq \|\tilde{w}_{k+1}\|_2$

car $\|\tilde{w}^*\|_2 = 1$ avec égalité si et seulement si $\tilde{w}_{k+1} \propto \tilde{w}^*$. Par construction,

$$\begin{aligned}
\langle \tilde{w}^*, \tilde{w}_{k+1} \rangle &= \langle \tilde{w}^*, \tilde{w}_k \rangle + y_i \langle \tilde{w}^*, \tilde{x}_i \rangle \\
&\geqslant \langle \tilde{w}^*, \tilde{w}_k \rangle + \gamma \\
&\geqslant \langle \tilde{w}^*, \tilde{w}_0 \rangle + (k+1)\gamma \\
&\geqslant (k+1)\gamma \quad \text{car } \tilde{w}_0 = 0.
\end{aligned}$$

Nous avons aussi :

$$\begin{aligned}
\|\tilde{w}_{k+1}\|^2 &= \|\tilde{w}_k\|^2 + \|\tilde{x}_i\|^2 + 2\langle \tilde{w}_k, y_i \tilde{x}_i \rangle \\
&\leqslant \|\tilde{w}_k\|^2 + R^2 + 1 \\
&\leqslant (k+1)(1+R^2)
\end{aligned}$$

$\|\tilde{x}_i\|^2 = R^2 + 1$ et $\langle \tilde{w}_k, y_i \tilde{x}_i \rangle \leq 0$ car le point $(\tilde{x}_i, y_i)$ est mal-classé.

$$(k+1)\gamma \leq \sqrt{(k+1)(1+R^2)} \iff k+1 \leqslant \frac{1+R^2}{\gamma^2}.$$

Perceptron : bilan et critiques

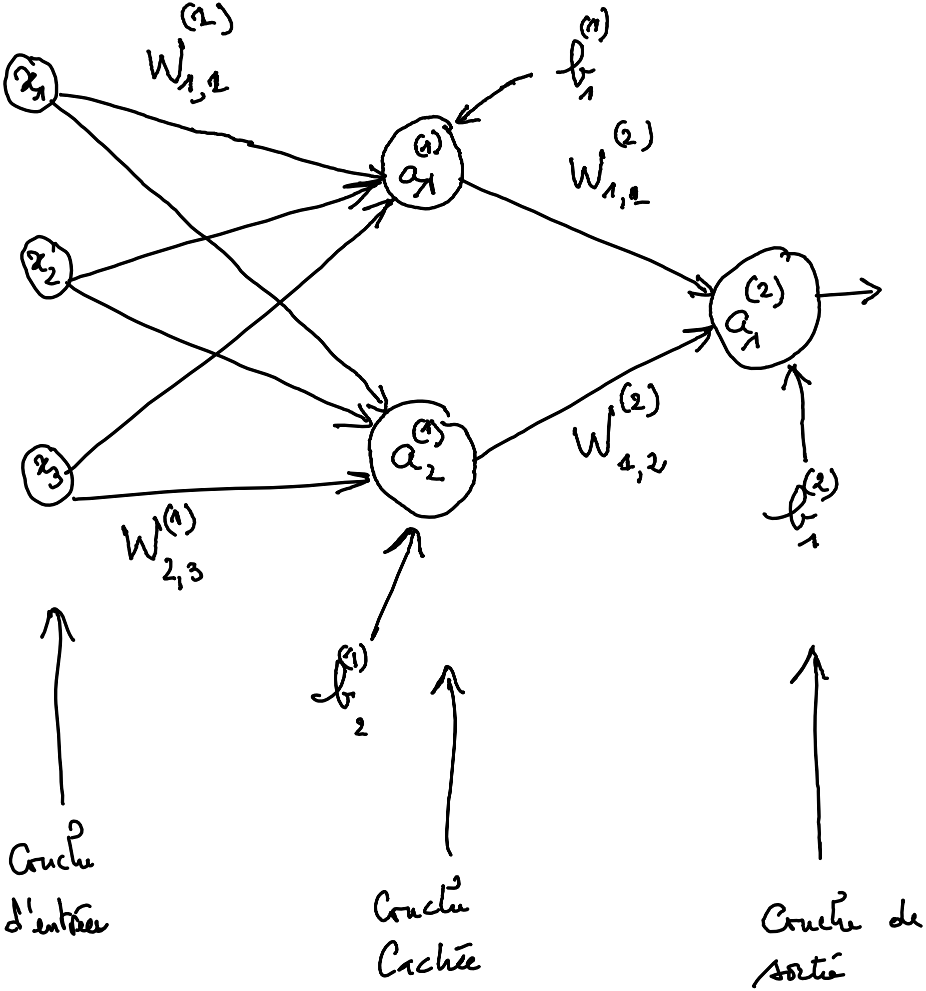
Bilan :

* Nous avons un jeu de données $D_n = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
* Nous utilisons l'algorithme du perceptron pour estimer les poids $\boldsymbol{w}$ et le terme dit de biais $\boldsymbol{b}$.
* On va prédire en utilisant la fonction $f_{(w,b)}(x) = \mathbb{1}_{\langle w, x\rangle + b > 0}$

Limites :

* La frontière de décision est linéaire ! (trop simple).
* L'algorithme du perceptron ne converge pas si les données ne sont pas linéairement séparables : dans ce cas, l'algorithme ne doit pas être utilisé.
* Donc, en pratique : **à ne pas utiliser**.

Réseau de neurones avec une couche cachée :



Notations :

1. $W^{(l)}_{i,j}$: poids entre le neurone $\underline{j}$ de la couche $l-1$ et le neurone $\underline{\underline{i}}$ de la couche l.

2. $b^{(l)}_j$: biais du neurone j de la couche l.

3. $a^{(l)}_j$: sortie du neurone j de la couche l.

4. $z^{(l)}_j$: entrée du neurone j de la couche l telle que

$$a_j^{(l)} = \sigma\left(z_j^{(l)}\right).$$

Estimation de poids et biais : (descente du gradient)

- La prédiction est donnée par $f_\theta(x)$

- Minimisation du risque empirique

$$\underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, f_\theta(X_i)) = \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell_i(\theta)$$

Descente du gradient stochastique :

Tant que $\|\theta_t - \theta_{t-1}\| \geq \varepsilon$:

* On tire $I_t \subset \{1, \dots, n\}$
* $\theta_{t+1} = \theta_t - \eta_t \left(\frac{1}{|I_t|} \sum_{i \in I_t} \nabla_\theta \ell_i(\theta_t) \right)$

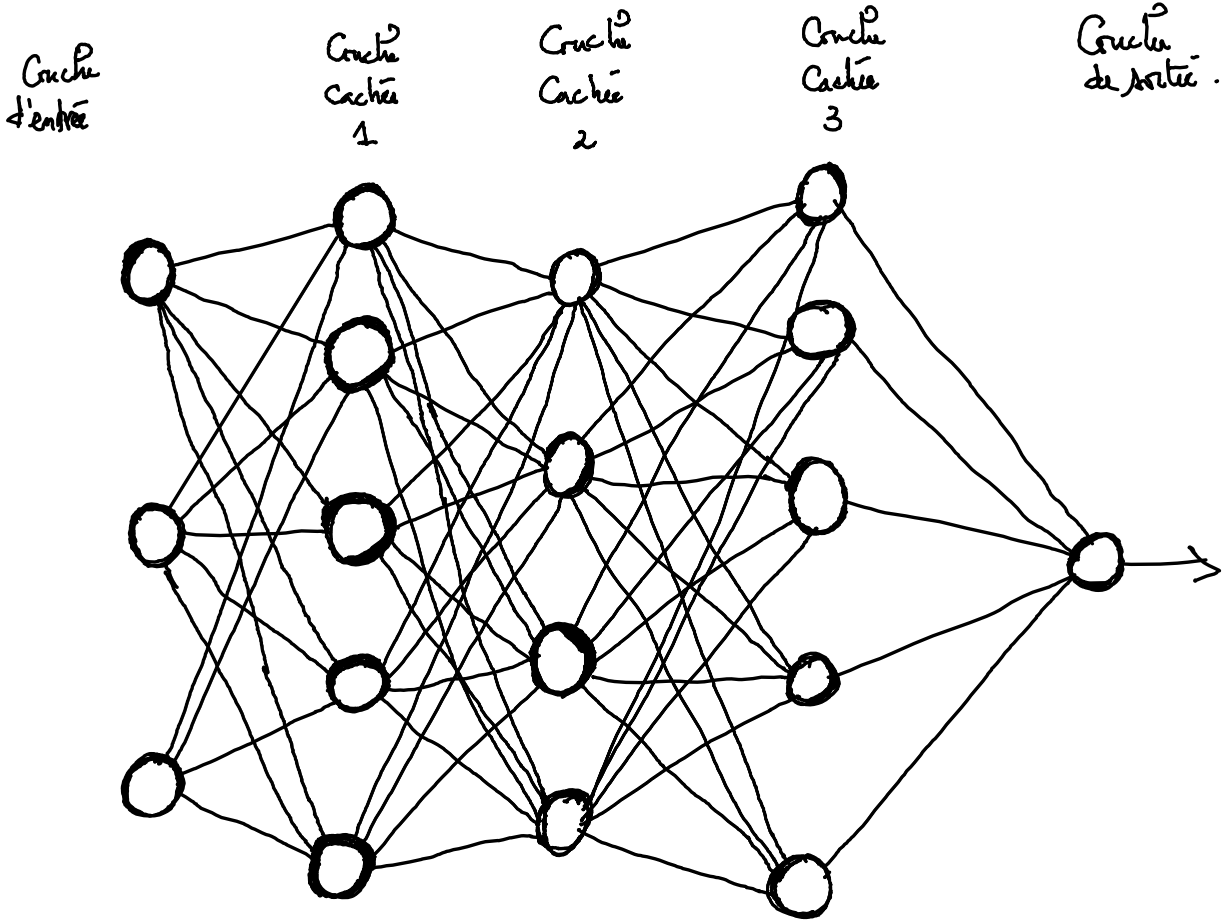
Peut-on calculer $\nabla_\theta \ell_i$ de manière efficace ?

Rétropropagation du gradient :

* Connu avec les travaux de Rumelhart, McClelland, Hinton en 1986.
* Remonte à Werbos en 1974.
* C'est une formule de dérivation en chaîne.

- C'est le secret de fonctionnement des réseaux de neurones.
- Au cœur du fonctionnement des réseaux de neurones profonds.

Idée de rétropropagation du gradient :



Equations de rétropropagation :

Un reseau de neurones avec L couches, avec une sortie vectorielle et une fonction de perte quadratique :

$$C = \frac{1}{2} \| y - a^{(L)} \|^2$$

Par définition :

$$\delta_j^{(\ell)} = \frac{\partial C}{\partial z_j^{(\ell)}}$$

Les 4 équations de la rétropropagation du gradient sont données par :

$$\delta^{(L)} = \nabla_a C \odot \sigma'(z^{(L)}) \quad \text{---- (1)}$$

$$\delta^{(\ell)} = \left((w^{(\ell+1)})^T \delta^{(\ell+1)} \right) \odot \sigma'(z^{(\ell)}) \quad \text{---- (2)}$$

$$\frac{\partial C}{\partial b_j^{(\ell)}} = \delta_j^{(\ell)} \quad \text{---- (3)}$$

$$\frac{\partial C}{\partial w_{j,k}^{(\ell)}} = a_k^{(\ell-1)} \delta_j^{(\ell)} \quad \text{---- (4)}$$

<u>Preuve:</u> $\delta^{(L)} = \nabla_a C \odot \sigma'(z^{(L)})$, en appliquant la dérivation par chaine :

$$\delta_j^{(L)} = \sum_k \frac{\partial C}{\partial a_k^{(L)}} \frac{\partial a_k^{(L)}}{\partial z_j^{(L)}}.$$

où $z_j^{(L)}$ l'entrée du jème neurone de la couche L. Comme l'activation $a_k^{(L)}$ ne dépend seulement de l'entrée $z_j^{(L)}$, nous avons:

$$\delta_j^{(L)} = \frac{\partial C}{\partial a_j^{(L)}} \times \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}$$

Comme $a_j^{(L)} = \sigma(z_j^{(L)})$, nous avons :

$$\underbrace{\delta_j^{(L)} = \frac{\partial C}{\partial a_j^{(L)}} \sigma'(z_j^{(L)})}_{},$$

c'est la jème composante de l'équation (1).

Maintenant, on veut vérifier que

$$\delta^{(l)} = \left[(w^{(l+1)})^T \delta^{(l+1)}\right] \odot \sigma'(z^{(l)}) \quad \text{-------} (2)$$

<u>Dérivation par chaîne:</u>

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}}$$

$$= \sum_k \frac{\partial C}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}.$$

$$= \sum_k \delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}$$

Rappelons que : $z_k^{(l+1)} = \sum_j w_{kj}^{(l+1)} \sigma(z_j^{(l)}) + b_k^{l+1}$, nous avons donc

$$\delta_j^{(l)} = \sum_k \delta_k^{(l+1)} w_{kj}^{(l+1)} \sigma'(z_j^{(l)}).$$

On peut maintenant vérifier que:

$$\frac{\partial C}{\partial b_j^{(\ell)}} = \delta_j^{(\ell)} \quad \text{----} \quad (3)$$

Dérivation par chaine:

$$\frac{\partial C}{\partial b_j^{\ell}} = \sum_k \frac{\partial C}{\partial z_k^{(\ell)}} \frac{\partial z_k^{(\ell)}}{\partial b_j^{\ell}}$$

$z_j^{(\ell)}$ dépend seulement de b_j^{ℓ} et $z_j^{(\ell)} = \sum_k W_{jk}^{(\ell)} \sigma\left(z_k^{(\ell-1)}\right) + b_j^{\ell}$.

donc

$$\frac{\partial C}{\partial b_j^{(\ell)}} = \delta_j^{(\ell)}.$$

Maintenant, on s'intéresse :

$$\frac{\partial C}{\partial w_{j,k}^{(\ell)}} = a_k^{(\ell-1)} \, \delta_j^{(\ell)} \quad \text{-----} \quad (4)$$

<u>Dérivation par chaîne :</u>

$$\frac{\partial C}{\partial w_{j,k}^{(\ell)}} = \sum_m \frac{\partial C}{\partial z_m^{(\ell)}} \frac{\partial z_m^{(\ell)}}{\partial w_{j,k}^{(\ell)}}$$

Comme $z_m^{(\ell)} = \sum_k w_{mk}^{(\ell)} \sigma\left(z_k^{\ell-1}\right) + b_m^{\ell}$, nous avons

$$\frac{\partial z_m^{(\ell)}}{\partial w_{j,k}^{(\ell)}} = \sigma\left(z_k^{(\ell-1)}\right) \mathbb{1}\{m=j\} .$$

<u>Donc :</u>

$$\frac{\partial C}{\partial w_{j,k}^{(\ell)}} = \delta_j^{(\ell)} \sigma\left(z_k^{(\ell-1)}\right) = a_k^{(\ell-1)} \delta_j^{(\ell)} .$$

Algorithme de rétropropagation :

Soit $\delta_j^{(\ell)} = \dfrac{\partial C}{\partial z_j^{(\ell)}}$ où $z_j^{(\ell)}$ est l'entrée du neurone j de la couche ℓ.

Entraînement d'un réseau de neurones :

(a) Initialiser aléatoirement les poids et biais du réseau

(b) Pour toutes les données $(x_i)_{i \in B}$ dans le lot $\underline{B}$.

à répéter jusqu'à convergence.

1. Feed forward : faire passer toutes les données du lot dans le réseau et stocker les valeurs la fonction d'activation et sa dérivée de chaque neurone.
2. Perte : Calculer la perte moyenne sur tout le lot du réseau.
3. Rétropropagation : Calculer récursivement les vecteurs $\delta^{(\ell)}$ en partant de $\ell = L$ à $\ell = 1$ à l'aide des équations (1) et (2). Calculer le gradient avec les équations (3) et (4).
4. Optimisation : Mise à jour des poids et biais à l'aide d'un pas de gradient

Vocabulaire des réseaux de neurones :

* (Mini) batch size : nombres de données dans un feedforward / backward. Plus la taille est grande, plus vous avez besoin de mémoire.

* Nombre d'itérations : nombre de mises à jour des paramètres pendant toute la durée de l'apprentissage.

* Époque (epoch) : nombre d'itérations nécessaires pour que le réseau ait "vu" autant d'observations qu'il y en a dans le jeu de données d'apprentissage.

Par exemple : pour un jeu de données de taille 12800, si on utilise un batch size de 128, nous avons besoin de 100 itérations pour "compléter 1 epoch".

Déclaration et entraînement d'un réseau de neurones :

Structure du réseau :

* Nombre de couches / neurones par couche
* Fonctions d'activation
* Unité de sortie
* Couches particulières (dropout et normalisation)

Optimisation :

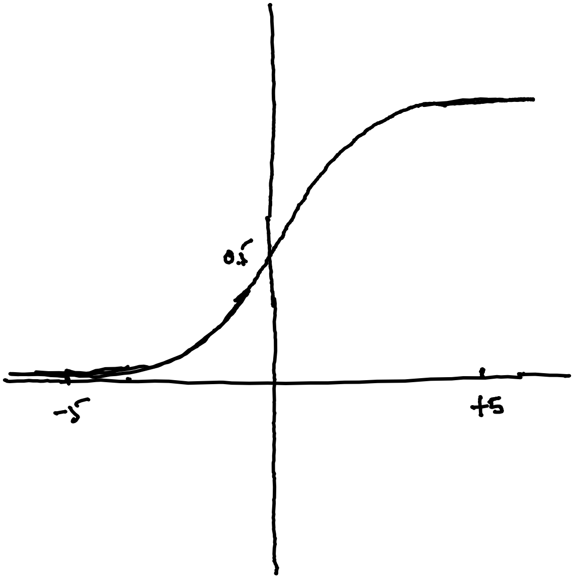
* Fonction de perte
* Algorithme d'optimisation
* Initialisation des poids / biais.

Nombre de neurones et couches cachées :

* Aucune règle particulière
* Lire des articles de recherche autour de la question et tester les propositions
* tester des variations et regarder les performances.
* Attention : trop de pifométrie (sans base théorique).

* Quelques travaux :
    * Network pruning (Blalock et al. 2020).
    * AgEBO-Tabular (Egele et al. 2020).

Fonctions d'activation : Sigmoïd



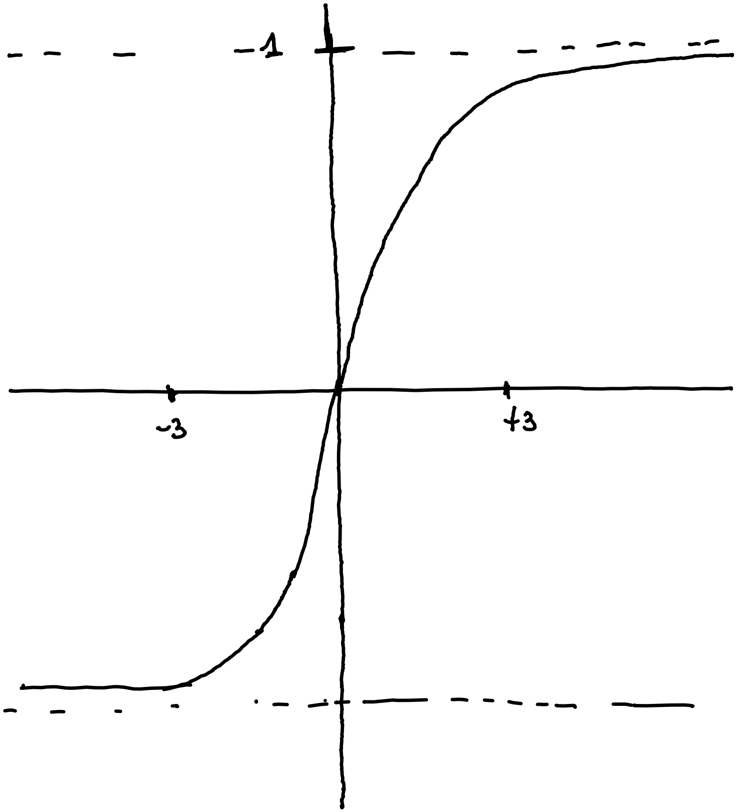
Commentaires :

* Fonctions saturées en raison de asymptotes horizontales :
  1. Gradient proche de zéro dans ces zones.
  2. Normalisation des données de chaque couche pour éviter.

$$\sigma : x \longmapsto \frac{\exp(x)}{1 + \exp(x)}$$

* exp(x) peut coûter un peu de temps.

Fonction d'activation : Tangente hyperbolique :



Commentaires :

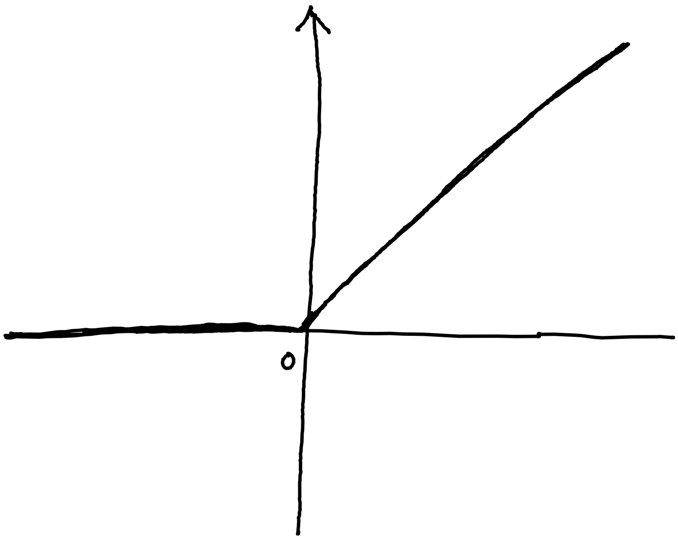
* Fonction centrée.
* Saturée
* Calcul de exp(x)

Notons que :

$$\tanh(x) = 2\sigma(2x) - 1.$$

$$\tanh : x \longmapsto \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

Fonction d'activation : Rectified Linear Unit (ReLU)

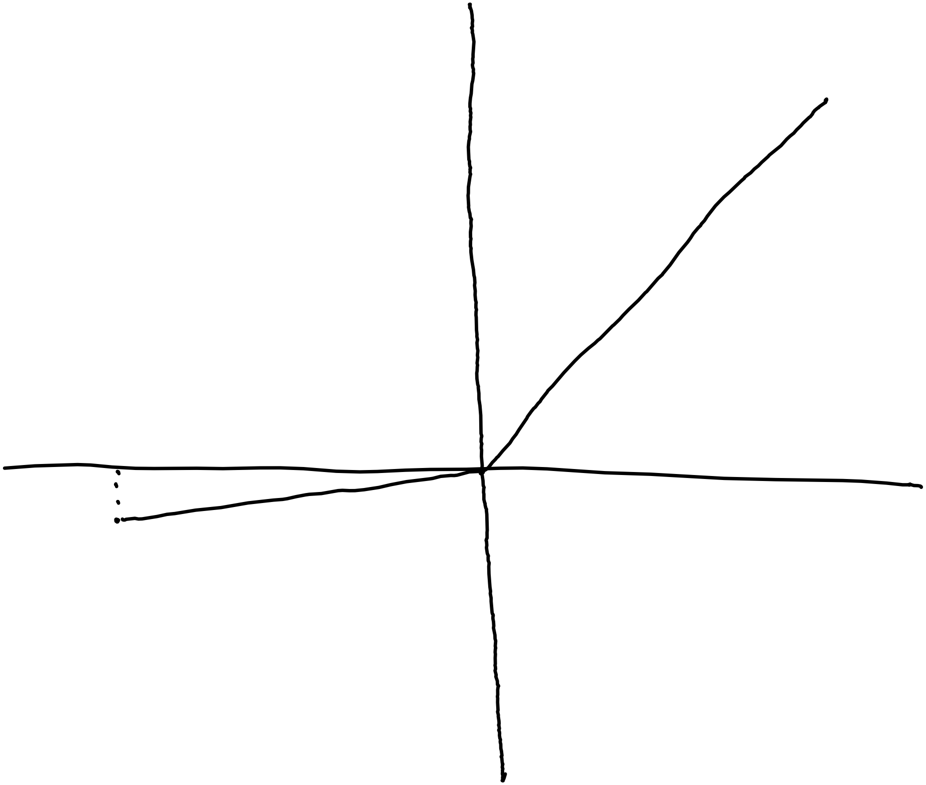


Commentaires :

* Non saturée à $+\infty$
* Saturée en $x \leq 0$.
* Calcul peu coûteux.

$$\text{ReLU} : x \longmapsto \max(0, x)$$

Fonction d'activation ReLU avec paramètre :



Parametric ReLU : $x \longrightarrow \max(\alpha x, x)$

* Leaky ReLU : $\alpha = 0{,}1$

* Absolute value rectification :

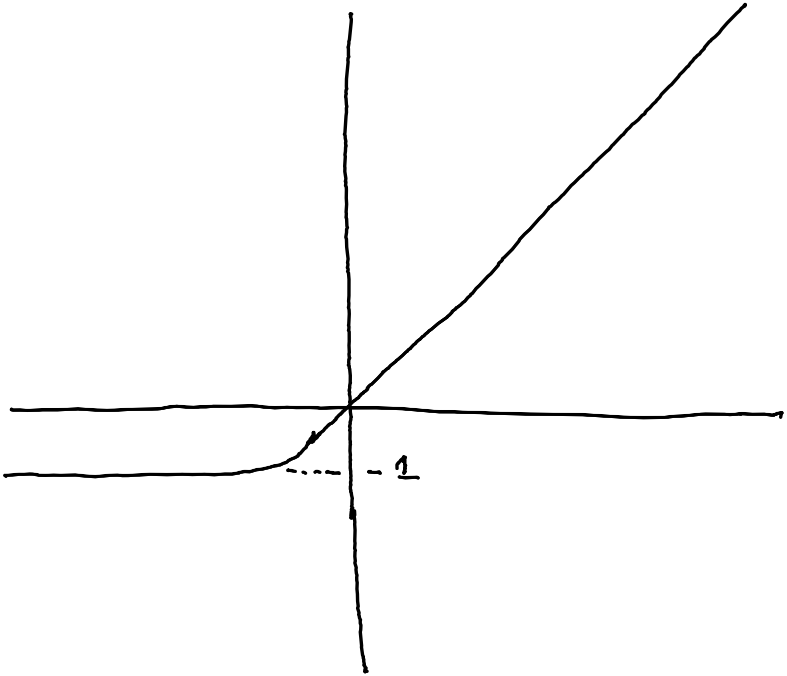
$\alpha = -1$

* Parametric ReLU :

$\underline{\alpha}$ est optimisé durant l'apprentissage.

" Empirical evaluation of rectified activations in convolutional network "
Xu et al. (2015)

Linear Unit Exponentielle (ELU) :



Commentaires :

* Proche de ReLU mais dérivable partout

* $\alpha = 1$ souvent

* Robuste au bruit dit-on.

$$\text{ELU} : x \longmapsto \begin{cases} x & \text{si } x \geq 0 \\ \alpha\left(\exp(x) - 1\right) & \text{sinon} \end{cases}$$

Swish:

$$\text{Swish}: x \longmapsto x \, \frac{\exp(\beta x)}{1 + \exp(\beta x)}$$

Maxout:

$$x \longmapsto \max(w_1 x + b_1, w_2 x + b_2, w_3 x + b_3) \quad \text{- -- } k = 3$$

* fonction linéaire par morceaux.
* linéaire avec différents régimes. (k régimes)
* Nombres de paramètres $\times k$.

Conclusions pour les fonctions d'activation:

* Utiliser ReLU (ou Swish).
* Tester Leaky ReLU, maxout et ELU.
* Tester Tanh mais sans beaucoup d'espoir.
* Ne pas utiliser sigmoid

Les unités de sortie ou neurones de sortie :

- Unité de sortie linéaire : $\hat{y} = W^T h + b$

- Unité de sortie sigmoïd : utiliser pour prédire une réponse binaire : $\mathbb{P}(Y=1 \mid h) = \sigma(W^T h + b)$ où $\sigma(t) = \dfrac{e^t}{1+e^t}$.

- Unité de sortie softmax : pour prédire une classe $\in \{1, \dots, K\}$

$$\mathrm{softmax}(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

où chaque z_i est une activation d'un neurone de la couche précédente donnée par $z_i = W_i^T h + b_i$.

Régression logistique multinomiale :

Pour $1 \leq k \leq K$ $\quad \log\left(\frac{\mathbb{P}[Y_i = k]}{Z}\right) = \beta_k X_i$

Alors, pour tout $1 \leq k \leq K$,

$$\mathbb{P}[Y_i = k] = Z e^{\beta_k X_i}$$

où : $$Z = \frac{1}{\sum_{k=1}^{K} e^{\beta_k X_i}}$$

donc : $$\mathbb{P}[Y_i = k] = \frac{e^{\beta_k X_i}}{\sum_{\ell=1}^{K} e^{\beta_\ell X_i}}.$$

Fonctions de perte :

- Erreur quadratique moyenne :

$$\frac{1}{n}\sum_{i=1}^{n} \ell(Y_i, f_\theta(X_i)) = \frac{1}{n}\sum_{i=1}^{n} (Y_i - f_\theta(X_i))^2$$

- Erreur absolue moyenne :

$$\frac{1}{n}\sum_{i=1}^{n} \ell(Y_i, f_\theta(X_i)) = \frac{1}{n}\sum_{i=1}^{n} |Y_i - f_\theta(X_i)|$$

- Erreur binaire :

$$\frac{1}{n}\sum_{i=1}^{n} \ell(Y_i, f_\theta(X_i)) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{Y_i \neq f_\theta(X_i)\}.$$

Fonctions de perte :

- Cross-entropy (ou la vraisemblance négative) :

$$\ell(y_i, f_\theta(x_i)) = -\log\left(\left[f_\theta(x_i)\right]_{y_i}\right).$$

Cross-entropy :

- très utilisée.

$$-\log\left(\mathbb{P}\left[Y=Y_i \mid X_i = x_i\right]\right) = -\log\left(\sigma\left((2y-1)(W^T h + b)\right)\right)$$

où $\sigma(t) = \dfrac{e^t}{1+e^t}$.

Initialisation des poids : une loi $\mathcal{N}(0, \sigma^2)$

* Si $\sigma^2 \ll 1$ problème d'absence d'activation.

* Si $\sigma^2 \ggg 1$ phénomène de saturation.

Idée : w_j un poids entre une couche j et $j+1$

* biais à zéro et $w_j \sim \mathcal{N}\left(0, \frac{\sqrt{2}}{n_j}\right)$

* $w_j \sim \mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$

n_j : taille de la couche j