

INTRODUCTION AUX RÉSEAUX DE NEURONES
SÉMINAIRE APPRENTISSAGE MACHINE EN MATHÉMATIQUES

Théo Lopès-Quintas

BPCE Payment Services,
Université Paris Dauphine

25 octobre 2023

APPRENTISSAGE SUPERVISÉ

FORMULATION D'UN PROBLÈME DE MACHINE LEARNING

Dans le cadre supervisé, nous avons accès à un dataset \mathcal{D} défini comme :

$$\mathcal{D} = \left\{ (x_i, y_i) \mid \forall i \leq \underbrace{n}_{\text{Nombre d'observations}}, x_i \in \mathbb{R}^{\underbrace{d'}_{\text{Nombre d'informations}}}, y_i \in \mathcal{Y} \right\}$$

Avec $\mathcal{Y} \subseteq \mathbb{R}$ pour un problème de régression et $\mathcal{Y} \subset \mathbb{N}$ dans le cadre d'une classification. Les problèmes de Machine Learning supervisé peuvent souvent s'écrire sous la forme d'une optimisation d'une fonction de perte $\mathcal{L} : \mathbb{R}^d \times \mathcal{M}_{n,d'} \times \mathbb{R}^n \rightarrow \mathbb{R}_+$ comme :

Vecteur des paramètres optimaux

$$\underbrace{\theta^*}_{\text{Vecteur des paramètres optimaux}} = \arg \min_{\theta \in \mathbb{R}^{\underbrace{d}_{\text{Dimension du vecteur de paramètres}}}} \mathcal{L}(\theta, X, y)$$

$\theta \in \mathbb{R}^d$

Dimension du vecteur de paramètres

Dans la suite, pour simplifier les notations, nous omettrons la dépendance de \mathcal{L} en X (matrice des informations) et y (vecteur réponse). Notons qu'en général, nous avons $d \neq d'$ et dans le cas du deep learning, très souvent $d \gg d'$.

PERCEPTRON

PRÉSENTATION

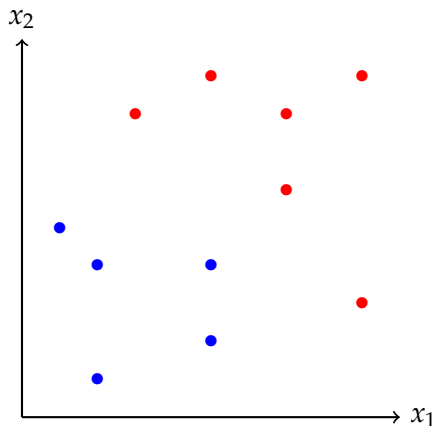


Figure – Visualisation d'un dataset pour un problème de classification avec $d = 2$ et $\mathcal{Y} = \{-1, 1\}$

Ici le dataset est linéairement¹ séparable : on cherche donc à *apprendre* la droite qui permet de séparer les deux classes.

1. Terme de la littérature, *affine* serait plus correct

PERCEPTRON

PRÉSENTATION

L'algorithme du perceptron est décrit dans l'article éponyme [Minsky and Papert, 1969], on reprend ici les notations et les définitions. Le problème que l'on se pose est donc :

$$\begin{aligned} f_w(x) &= \operatorname{sgn}(\langle w, x \rangle) \\ w^* &= \arg \min_{w \in \mathbb{R}^d} \sum_{i=1}^n \max \{0, -y_i \langle w, x_i \rangle\} \end{aligned}$$

Avec f la fonction de décision finale et w le vecteur de poids définissant l'hyperplan séparateur. Le problème d'optimisation est résolu en appliquant la règle de mise à jour observation par observation :

$$w_{t+1} = w_t + yx \mathbb{1}_{y\langle w_t, x \rangle \leq 0}$$

PERCEPTRON

VISUALISATION DE L'APPRENTISSAGE

La règle de mise à jour est :

$$w_{t+1} = w_t + yx\mathbb{1}_{y\langle w_t, x \rangle \leq 0}$$

Elle se visualise dans notre exemple :

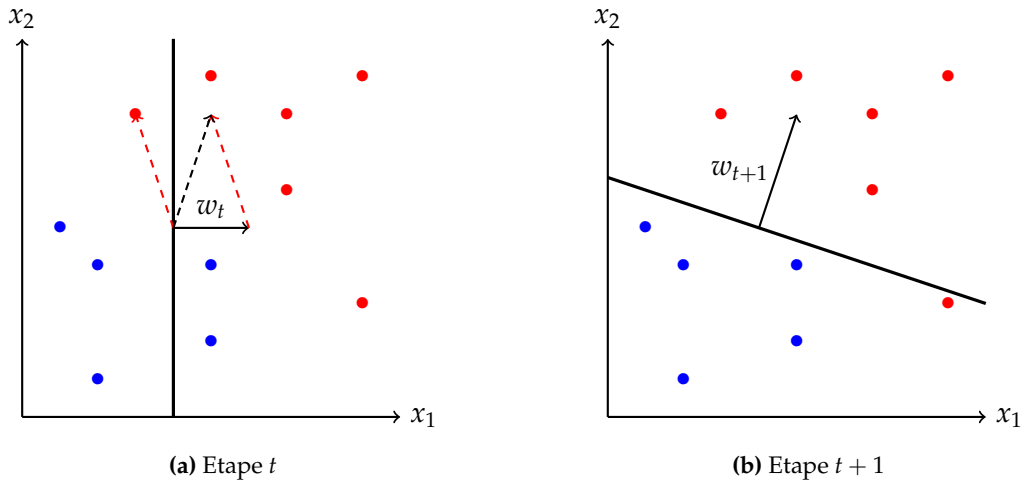


Figure – Intuition géométrique de l'apprentissage du perceptron

PERCEPTRON

LA CONVERGENCE DU PERCEPTRON EST GARANTIE DANS LE CAS LINÉAIREMENT SÉPARABLE

Théorème 1 (Convergence du perceptron)

On suppose que les données issues d'un dataset \mathcal{D} soient linéairement séparables et de plus que :

$$\exists \gamma > 0, \forall i \leq n, \quad y_i \langle w^*, x_i \rangle \leq \gamma$$

On note $R = \max_{1 \leq i \leq n} \|x_i\|$. Alors la k -ième erreur de classification du perceptron aura lieu avant :

$$k \leq \left(\frac{R}{\gamma} \right)^2 \|w^*\|^2$$

γ correspond ici à l'existence d'une *marge* qui sépare les deux classes. Ainsi, plus la marge est grande plus la k -ième erreur arrivera *tôt*.

PERCEPTRON

PREUVE

Puisque le vecteur des paramètres w n'est mis à jour que lors d'une erreur de prédiction, on note w_k le vecteur lorsque la k -ième erreur est commise. Alors :

$$w_k = w_{k-1} + y_i x_i$$

On commence par remarquer que :

$$\begin{aligned}\langle w_k, w^* \rangle &= \langle w_{k-1} + y_i x_i, w^* \rangle \\ &= \langle w_{k-1}, w^* \rangle + y_i \langle x_i, w^* \rangle \\ &\geq \langle w_{k-1}, w^* \rangle + \gamma \quad \text{par définition de } \gamma\end{aligned}$$

Par une récurrence immédiate, on a :

$$\langle w_k, w^* \rangle \geq k\gamma \tag{1}$$

PERCEPTRON

PREUVE

Puisqu'on veut borner le nombre d'erreurs, il faut borner l'apparition de k , et donc majorer le terme $\langle w_k, w^* \rangle$. On peut regarder l'inégalité de Cauchy-Schwarz quand on cherche à majorer de telles quantités. Pour le faire, on doit avoir une idée de la norme de w_k :

$$\begin{aligned}\|w_k\|^2 &= \|w_{k-1}\|^2 + 2y_i \langle w_k, x_i \rangle + y_i^2 \|x_i\|^2 \\ &\leq \|w_{k-1}\|^2 - 2\gamma + R^2 \quad \text{par définition de } \gamma \text{ et } R \\ &\leq kR^2 \quad \text{par récurrence immédiate}\end{aligned}$$

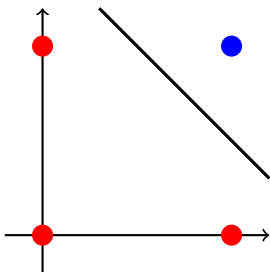
Il ne nous reste plus qu'à appliquer l'inégalité de Cauchy-Schwarz en partant de (1) :

$$\begin{aligned}k^2 \gamma^2 &\leq \langle w_k, w^* \rangle^2 \\ &\leq kR^2 \|w^*\|^2 \quad \text{par Cauchy-Schwarz} \\ k &\leq \left(\frac{R}{\gamma} \right)^2 \|w^*\|^2\end{aligned}$$

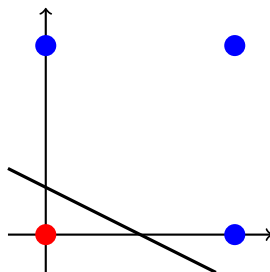
PERCEPTRON

PROBLÈME XOR

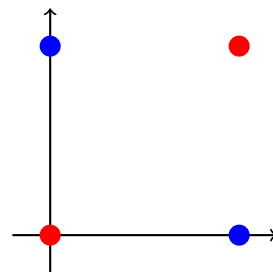
On souhaiterait savoir s'il est possible à l'aide d'un perceptron de reproduire les fonctions logiques NOT, AND, OR et XOR. Pour l'opérateur de négation, $w = -1$ et $b = 0.5$ suffisent pour renvoyer les valeurs attendues.



(a) Problème AND



(b) Problème OR



(c) Problème XOR

Dans le cas de XOR, on ne peut pas trouver un couple (w, b) qui permettent à un perceptron de séparer les classes correctement. Nous savons que $\text{XOR}(A, B) = \text{AND}(\text{NOT}(\text{AND}(A, B)), \text{OR}(A, B))$ donc il faudrait combiner plusieurs perceptrons pour y parvenir !

RÉSEAU DE NEURONES

EXEMPLE D'UN RÉSEAU

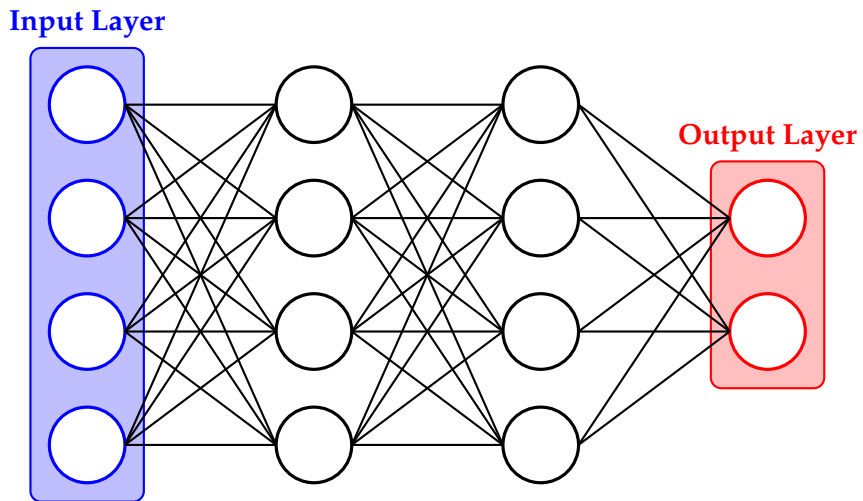


Figure – Exemple d'un réseau de neurones avec 2 couches cachées pour une classification binaire

RÉSEAU DE NEURONES

DESCRIPTION D'UN NEURONE

Un neurone renvoie un nombre après l'application d'une fonction f que l'on appelle fonction **d'activation** : nous reviendrons après sur cette notion.

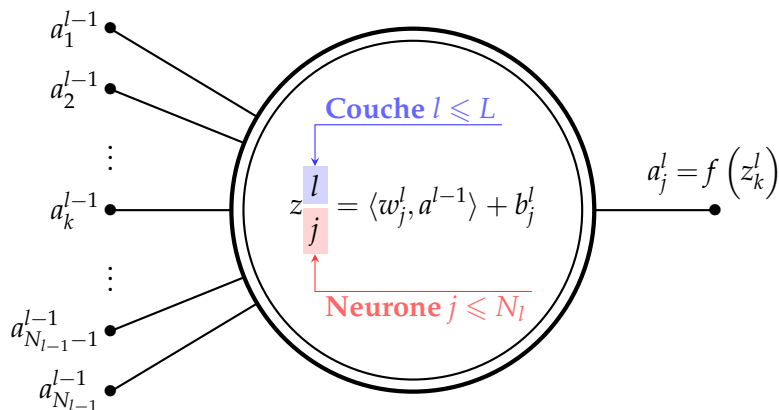


Figure – Neurone j de la couche l paramétré par les poids $w_i^j \in \mathbb{R}^{N_{l-1}}$ et le biais $b_j^l \in \mathbb{R}$

L'objectif est de comprendre comment on peut *apprendre* ces paramètres.

RÉSEAU DE NEURONES

DESCENTE CLASSIQUE

La méthode la plus utilisée pour résoudre ce genre de problème est la descente de gradient :

$$\theta_{t+1} = \theta_t - \eta_t \nabla \mathcal{L}(\theta_t)$$

↑
Learning rate

La convergence est garantie dans le cas où \mathcal{L} est convexe. Ce n'est pas le cas dans un réseau de neurones, mais il est montré [Choromanska et al., 2015] que la majorité des minima locaux d'un réseau de neurones sont *proches* en terme de valeur optimale. Ainsi, obtenir un minima local est satisfaisant.

Quand on travaille avec des grands datasets, le coût de calcul/temps est grand si l'on calcule $\nabla \mathcal{L}(\theta_t)$ pour la totalité de la base. Ainsi, on préfère la descente de gradient **stochastique** par batch : on applique la descente de gradient à l'ensemble de la base de données en découpant en petit lots (batch) aléatoires et en mettant à jour θ à chaque fois.

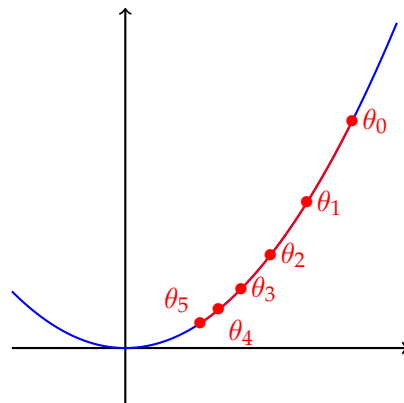


Figure – Exemple d'une descente de gradient pour $f(x) = x^2$

ALGORITHME DE BACK-PROPAGATION

COMMENT CALCULER LES DÉRIVÉES PARTIELLES ?

On souhaite mettre en place la descente de gradient pour pouvoir *apprendre* l'ensemble des poids.
Nous avons besoin de calculer pour le neurone $j \leq N_l$ à la couche $l \leq L$ les gradients $\frac{\partial \mathcal{L}}{\partial w_{j,k}^l}$ et $\frac{\partial \mathcal{L}}{\partial b_j^l}$ en conservant les notations d'un neurone $j \leq N_l$ à la couche $l \leq L$:

$$\begin{aligned} z_j^l &= \langle w_j^l, a^{l-1} \rangle + b_j^l \\ a^{l-1} &= f(z^{l-1}) \end{aligned}$$

La dernière équation étant vectorisée. Par le théorème de dérivation des fonctions composées, on obtient :

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial b_j^l} &= \frac{\partial \mathcal{L}}{\partial b_j^l} \frac{\partial b_j^l}{\partial z_j^l} \quad \text{mais} \quad \frac{\partial b_j^l}{\partial z_j^l} = 1 \\ &= \frac{\partial \mathcal{L}}{\partial z_j^l} \end{aligned} \qquad \begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{j,k}^l} &= \frac{\partial \mathcal{L}}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{j,k}^l} \quad \text{mais} \quad \frac{\partial z_j^l}{\partial w_{j,k}^l} = a_k^{l-1} \\ &= a_k^{l-1} \frac{\partial \mathcal{L}}{\partial z_j^l} \end{aligned}$$

ALGORITHME DE BACK-PROPAGATION

COMMENT CALCULER LES DÉRIVÉES PARTIELLES DE LA DERNIÈRE COUCHE ?

Pour le neurone $j \leq N_l$ à la couche $l \leq L$ on a obtenu que $\frac{\partial \mathcal{L}}{\partial b_j^l} = \frac{\partial \mathcal{L}}{\partial z_j^l}$ et $\frac{\partial \mathcal{L}}{\partial w_{j,k}^l} = a_k^{l-1} \frac{\partial \mathcal{L}}{\partial z_j^l}$.

On définit $\delta_j^l = \frac{\partial \mathcal{L}}{\partial z_j^l}$ la quantité commune à ces deux gradients. δ_j^l s'interprète comme l'erreur *brute* d'un neurone. Si l'on considère les neurones les plus proches de la sortie, donc à la couche L , on a pour un neurone $j \leq N_L$ avec le théorème de dérivation des fonctions composées :

$$\begin{aligned} \delta_j^L &= \frac{\partial \mathcal{L}}{\partial a_j^L} \frac{a_j^L}{\partial z_j^L} \quad \text{mais} \quad a_j^L = f(z_j^L) \\ &= f'(z_j^L) \frac{\partial \mathcal{L}}{\partial a_j^L} \end{aligned}$$

Calculer la valeur de $\frac{\partial \mathcal{L}}{\partial a_j^L}$ est aisé puisqu'il suffit de comparer la valeur prédite par le réseau à la valeur attendue. En combinant ce résultat à la question précédente, nous savons calculer la mise à jour des poids et des biais de la dernière couche.

ALGORITHME DE BACK-PROPAGATION

COMMENT CALCULER LES DÉRIVÉES PARTIELLES DU RESTE DU RÉSEAU ?

Pour le neurone $j \leq N_l$ à la couche $l \leq L$ on a obtenu que $\frac{\partial \mathcal{L}}{\partial b_j^l} = \frac{\partial \mathcal{L}}{\partial z_j^l}$ et $\frac{\partial \mathcal{L}}{\partial w_{j,k}^l} = a_k^{l-1} \frac{\partial \mathcal{L}}{\partial z_j^l}$.

De plus, $\delta_j^l = f'(z_j^l) \frac{\partial \mathcal{L}}{\partial a_j^l}$ ce qui implique que l'on sait calculer les gradients pour les poids des neurones de la dernière couche. Pour calculer le reste des gradients, il faut probablement réussir à exprimer δ_j^l en fonction de δ_j^{l+1} : probablement de proche en proche. On considère le neurone $j \leq N_l$ à la couche $l < L$:

$$\begin{aligned} \delta_j^l &= \frac{\partial \mathcal{L}}{\partial z_j^l} \\ &= \sum_{k=1}^{N_{l+1}} \frac{\partial \mathcal{L}}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} \quad \text{et} \quad \frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{k,j}^{l+1} f'(z_j^l) \\ &= \sum_{k=1}^{N_{l+1}} w_{k,j}^{l+1} f'(z_j^l) \frac{\partial \mathcal{L}}{\partial z_k^{l+1}} \\ &= \sum_{k=1}^{N_{l+1}} w_{k,j}^{l+1} \delta_k^{l+1} f'(z_j^l) \quad \text{par définition des } \delta_j^l \end{aligned}$$

ALGORITHME DE BACK-PROPAGATION

EXPLOSION ET DISPARITION DES GRADIENTS

Pour le neurone $j \leq N_l$ à la couche $l \leq L$ on a obtenu que $\frac{\partial \mathcal{L}}{\partial b_j^l} = \delta_j^l$ et $\frac{\partial \mathcal{L}}{\partial w_{j,k}^l} = a_k^{l-1} \delta_j^l$ avec

$$\delta_j^l = \sum_{k=1}^{N_l} w_{k,j}^{l+1} \delta_k^{l+1} f'(z_j^l).$$

Si l'on note $\gamma = \max_{x \in \mathbb{R}} f'(x)$ alors :

$$|\delta_j^l| \leq \gamma^{L-l+1} \left| \sum_{k=1}^{n_l} w_{k,j}^{l+1} \times \left(\sum_{k=1}^{n_{l+1}} w_{k,j}^{l+2} \times \left(\dots \left(\sum_{k=1}^{n_{L-1}} w_{k,j}^L \delta_j^L \right) \dots \right) \right) \right|$$

Dans les premiers réseaux de neurones la fonction sigmoid était utilisée et dans ce cas $\gamma = \frac{1}{4}$ ce qui pouvait conduire à une disparition de l'information des gradients dans les réseaux profonds. Ce phénomène est appelé *vanishing gradient*. Le phénomène inverse² s'appelle *exploding gradient*. Il faut donc choisir avec plus de précaution la fonction d'activation f .

2. Avoir des gradients qui sont de plus en plus grand quand on remonte le réseau

FONCTIONS D'ACTIVATION

FONCTION D'ACTIVATION CLASSIQUE

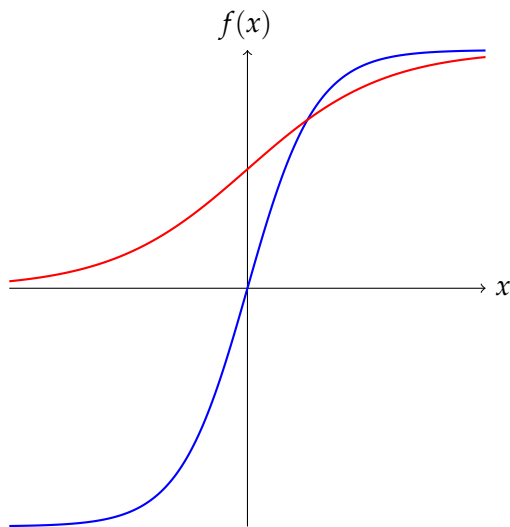


Figure – Fonctions **sigmoid** et **tangente hyperbolique**

Initialement les fonctions **sigmoid** et **tangente hyperbolique** ont été choisies pour être les fonctions d'activation dans un réseau de neurones :

$$\begin{aligned}\sigma(x) &= \frac{1}{1 + e^{-x}} \\ \tanh(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}}\end{aligned}$$

Ces deux fonctions peuvent amener à des gradients qui disparaissent ou explosent quand la back-propagation a lieu.

FONCTIONS D'ACTIVATION

FONCTION D'ACTIVATION RELU ET VARIANTES

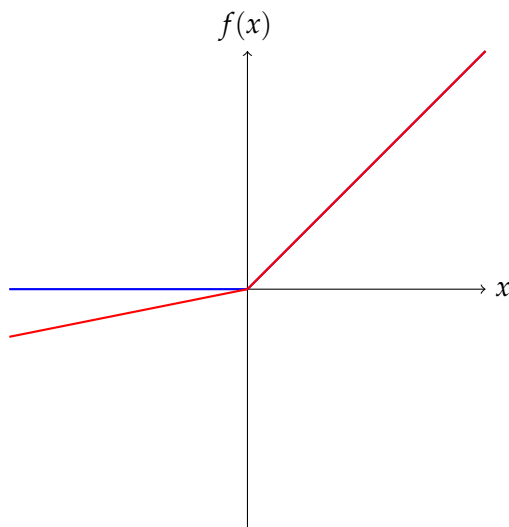


Figure – Fonctions ReLU et Leaky ReLU

Une fonction plus simple a été proposée :

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

Elle présente beaucoup moins souvent les deux phénomènes néfastes, mais en amène un autre : la mort de neurones. Puisque le gradient est fréquemment zéro, certains neurones *n'apprennent* plus. Ainsi, de nombreuses propositions ont vu le jour, dont :

$$\text{Leaky ReLU}(x) = \max\{\alpha x, x\}$$

Avec $\alpha \geq 0$ un hyper-paramètre.

FONCTIONS D'ACTIVATION

FONCTIONS D'ACTIVATION MODERNE

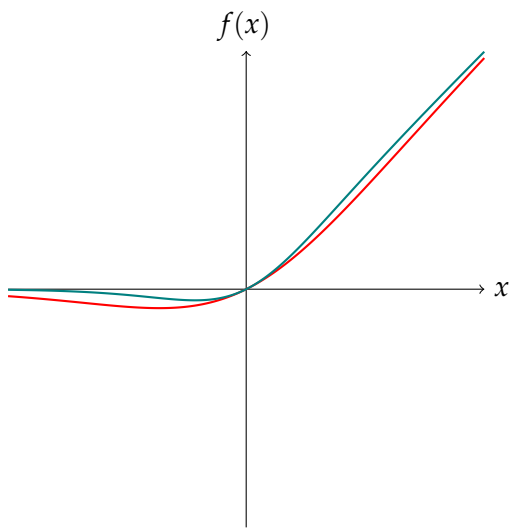


Figure – SiLU et GELU

Des variantes ont été proposées et on montre des performances équivalentes voire meilleures que [ReLU](#) sur une variété de problèmes. Nous citerons spécifiquement GELU et SiLU [Hendrycks and Gimpel, 2016].

$$\text{SiLU}(x) = x\sigma(x) \quad \text{avec} \quad \sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\text{GELU}(x) = x\phi(x) \quad \text{avec} \quad \phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-\frac{t^2}{2}} dt$$

Dans les architectures les plus poussées on observe même de nouveaux blocs dédiés à l'activation comme présenté dans [Dauphin et al., 2016] et [Shazeer, 2020].

INITIALISATION DES POIDS

COMMENT LIMITER LES EFFETS D'EXPLOSION OU DE DISPARITION DES GRADIENTS ?

On considère un réseau de neurones à $L \in \mathbb{N}^*$ couches. On reprend la majorité des notations précédentes, mais on note z_k^l le résultat du neurone k à la couche l défini par :

$$\forall l \leq L, \forall k \in N_l, z_k^l = \langle w_k^l, x^l \rangle$$

Diagram illustrating the notation for the output of a neuron k at layer l :

- L : Nombre de couche du réseau (Number of layers of the network)
- N_l : Nombre de neurones de la couche l (Number of neurons in layer l)
- x^l : Vecteur d'entrée de la couche l (Input vector of layer l)
- w_k^l : Matrice des poids du neurone k à la couche l (Weight matrix of neuron k at layer l)

Nous faisons les hypothèses supplémentaires :

- ▶ Indépendance et distribution : $(w^l)_{l \leq L}$, $(x^l)_{l \leq L}$ et $(z^l)_{l \leq L}$ sont indépendants et identiquement distribués (respectivement)
- ▶ Indépendance : $\forall l \leq L, w^l \perp\!\!\!\perp x^l$
- ▶ Poids centrés : $\mathbb{E}[w^l] = 0$ et la distribution est symétrique autour de 0
- ▶ Résultats centrés : $\mathbb{E}[z^l] = 0$ et la distribution est symétrique autour de 0

On souhaite obtenir une bonne propagation des signaux dans les deux sens afin d'éviter une explosion ou une disparition des gradients. Formellement :

$$\forall l \leq L, \mathbb{V}[z^l] = \mathbb{V}[z^1] \quad (2)$$

INITIALISATION DES POIDS

CAS D'UNE FONCTION D'ACTIVATION IMPAIRE

Calculons pour le neurone $j \leq N_l$ à la couche $l \leq L$:

$$\begin{aligned}\mathbb{V} \left[z_j^l \right] &= \mathbb{V} \left[\langle w_j^l, x^l \rangle \right] \\ &= n_{\text{input}} \mathbb{V} \left[w_{j,k}^l x_k^l \right] \quad \text{par indépendance et car identiquement distribués, pour } k \leq n_{\text{input}} \\ &= n_{\text{input}} \left(\mathbb{V} \left[w_{j,k}^l \right] \mathbb{V} \left[x_k^l \right] + \mathbb{E} \left[w_{j,k}^l \right]^2 \mathbb{V} \left[x_k^l \right] + \mathbb{E} \left[x_k^l \right]^2 \mathbb{V} \left[w_{j,k}^l \right] \right) \\ &= n_{\text{input}} \mathbb{V} \left[w_{j,k}^l \right] \mathbb{E} \left[\left(x_k^l \right)^2 \right] \quad \text{car } \mathbb{E} \left[w_{j,k}^l \right] = 0\end{aligned}$$

Rappelons que $x^l = f \left(z^{l-1} \right)$ avec f la fonction d'activation entre la couche $l-1$ et la couche l . Si f est impaire, alors $\mathbb{E} \left[x^l \right] = \mathbb{E} \left[f \left(z^{l-1} \right) \right] = 0$ donc $\mathbb{E} \left[\left(x_j^l \right)^2 \right] = \mathbb{V} \left[z_j^{l-1} \right]$. Ainsi,

$$\forall l \leq L, \forall j \leq N_l, \quad \mathbb{V} \left[z_j^l \right] = n_{\text{input}} \mathbb{V} \left[w_{j,k}^l \right] \mathbb{V} \left[z_j^{l-1} \right] = \prod_{i=1}^{l-1} \left(n_{\text{input}} \mathbb{V} \left[w_{j,k}^i \right] \right) \mathbb{V} \left[z_j^1 \right]$$

Une condition suffisante pour que l'on respecte l'équation (2) est que $\mathbb{V} \left[w_{j,k}^l \right] = \frac{1}{n_{\text{input}}}$

INITIALISATION DES POIDS

CAS D'UNE FONCTION D'ACTIVATION IMPAIRE

Nous venons de reproduire le raisonnement de l'article [Glorot and Bengio, 2010]. Pour résoudre le problème final, choisir la condition qui conviendra pour les deux passes (*forward* et *backward*), les auteurs proposent de prendre la moyenne entre les deux conditions. Plus tard, nous ferons références au nombre de neurones de la couche précédente (respectivement suivante) par *fan in* (respectivement *fan out*).

Notons que nous n'avons supposé aucune loi pour l'initialisation des poids à part des conditions d'indépendances, distributions et que la loi doit être symétrique autour de 0 et d'espérance 0. Les cas les plus classiques sont :

- ▶ Normale : $\mathcal{N}(0, \sigma^2) : \sigma = \sqrt{\frac{2}{\text{fan in} + \text{fan out}}}$
- ▶ Uniforme : $\mathcal{U}([-a, a]) : a = \sqrt{\frac{6}{\text{fan in} + \text{fan out}}}$

INITIALISATION DES POIDS

CAS DE LA FONCTION ReLU

Puisque cette fois f n'est pas impaire mais égale à $f(x) = \max\{0, x\}$, nous avons que :

$$\begin{aligned}\mathbb{E}[x^2] &= \mathbb{E}[f(z)^2] \\ &= \int_{-\infty}^{+\infty} f(t)^2 p(t) dt \text{ avec } p \text{ la densité de probabilité associé à } z \\ &= \int_0^{+\infty} t^2 p(t) dt \\ &= \frac{1}{2} \mathbb{V}[z]\end{aligned}$$

Nous avons donc cette fois :

$$\forall l \leq L, \forall j \leq N_l, \quad \mathbb{V}[z_j^l] = \frac{n_{\text{input}}}{2} \mathbb{V}[w_{j,k}^l] \mathbb{V}[z_j^{l-1}] = \prod_{i=1}^{l-1} \left(\frac{n_{\text{input}}}{2} \mathbb{V}[w_{j,k}^i] \right) \mathbb{V}[z_j^1]$$

A nouveau, une condition suffisante pour que la relation (2) soit vérifiée pour la passe forward est :

$$\mathbb{V}[w_{j,k}^l] = \frac{2}{n_{\text{input}}}$$

INITIALISATION DES POIDS

CAS DE LA FONCTION ReLU






C'est très proche de la condition quand l'on considère une fonction d'activation impaire, mais cela résulte en un écart significatif lors de l'entraînement. Le raisonnement suivi a été décrit dans l'article [He et al., 2015].

Contrairement à la résolution proposée par Xavier Glorot et Yoshua Bengio, la valeur de la variance pour les poids est obtenue en sélectionnant le *fan in* ou le *fan out* : on le note *fan mode*.

- ▶ Normale : $\mathcal{N}(0, \sigma^2)$: $\sigma = \sqrt{\frac{2}{\text{fan mode}}}$
- ▶ Uniforme : $\mathcal{U}([-a, a])$: $a = \sqrt{\frac{6}{\text{fan mode}}}$

On peut obtenir des distributions précises également pour le Leaky ReLU.

BIBLIOGRAPHIE I

-  Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. (2015).
The loss surfaces of multilayer networks.
In *Artificial intelligence and statistics*. PMLR.
-  Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. (2016).
Language modeling with gated convolutional networks.
In *International conference on machine learning*, pages 933–941. PMLR.
-  Glorot, X. and Bengio, Y. (2010).
Understanding the difficulty of training deep feedforward neural networks.
In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.
JMLR Workshop and Conference Proceedings.
-  He, K., Zhang, X., Ren, S., and Sun, J. (2015).
Delving deep into rectifiers : Surpassing human-level performance on imagenet classification.
In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.
-  Hendrycks, D. and Gimpel, K. (2016).
Gaussian error linear units (gelus).
arXiv preprint arXiv :1606.08415.

BIBLIOGRAPHIE II



Minsky, M. and Papert, S. (1969).

Perceptrons.

MIT press.



Shazeer, N. (2020).

Glu variants improve transformer.

arXiv preprint arXiv :2002.05202.