Réseau de neurones dense

Introduction au deep learning

Théo Lopes Quintas

BPCE Payment Services,
Université Paris Dauphine

2023-2026

Problème Machine learning

Formulation d'un problème de Machine Learning

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

$$\mathcal{D} = \left\{ (x_i, y_i) \mid \forall i \leqslant n,\ x_i \in \mathbb{R}^{d'}, y_i \in \mathcal{Y} \right\}$$

Nombre d'informations

Nombre d'observations

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 $\mathcal{L} : \mathbb{R}^d \times \mathcal{M}_{n,d'} \times \mathbb{R}^n \to \mathbb{R}_+$ comme :

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^d} \mathcal{L}(\theta, X, y)$$

Vecteur des paramètres optimaux

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 $d >> d'$.

Problème Machine learning

Exemple de fonction de perte : problème de régression

On cherche à minimiser une fonction de perte $\mathcal{L}$ qui s'écrit très souvent en Machine Learning comme la somme d'une fonction sur chaque observation du dataset. On note $\hat{y}_i$ la prédiction d'un algorithme pour l'observation x_i avec $i \leqslant n$ l'index de l'observation i. Pour un problème de régression on a par exemple :

$$\begin{aligned} \mathcal{L}(\theta) &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}\ell_i(\theta) \end{aligned}$$

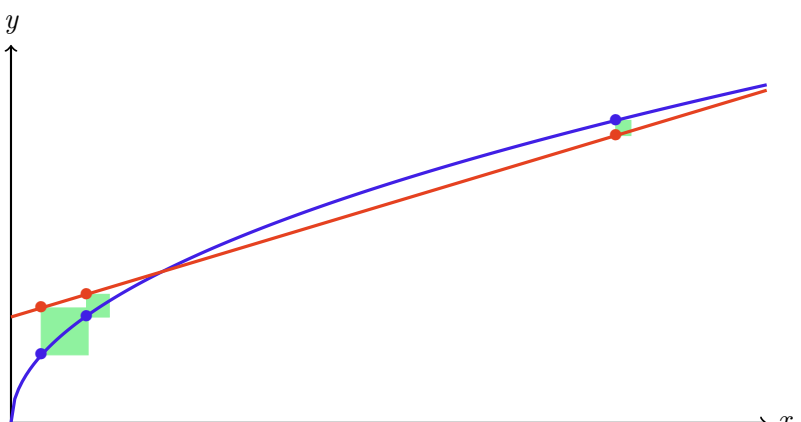


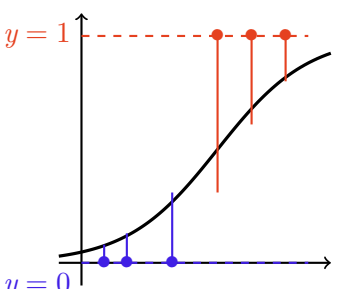
Figure – Visualisation de la MSE entre une régression linéaire et la vraie courbe

Problème Machine learning

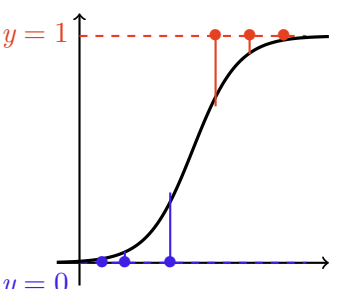
Exemple de fonction de perte : problème de classification

On cherche à minimiser une fonction de perte $\mathcal{L}$ qui s'écrit très souvent en Machine Learning comme la somme d'une fonction sur chaque observation du dataset. On note $\hat{y}_i$ la prédiction d'un algorithme pour l'observation x_i avec $i \leqslant n$ l'index de l'observation i. Pour un problème de classification on a par exemple :

$$\mathcal{L}(\theta) = -\frac{1}{n}\sum_{i=1}^{n} y_i \ln(\hat{y}_i) + (1-y_i)\ln(1-\hat{y}_i) = \frac{1}{n}\sum_{i=1}^{n} \ell_i(\theta)$$



(a) $\mathcal{L}(\theta_1) = 0.3$



(b) $\mathcal{L}(\theta_2) = 0.13$

Figure – Performances pour deux paramètres différent d'une régression logistique

Problème Machine learning

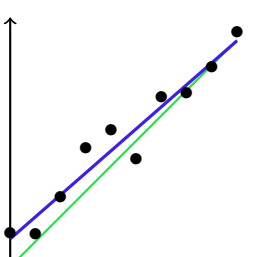
Trade-off biais-variance

Pour minimiser $\mathcal{L}$ on modélise la fonction f inconnue par une forme de fonction $\hat{f}$.

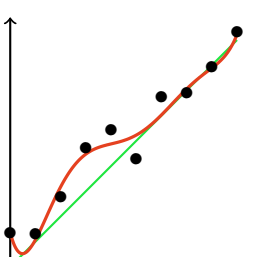
- Bias $\left[\hat{f}(x)\right] = \mathbb{E}\left[\hat{f}(x)\right] - f(x)$: l'écart moyen entre la valeur prédite et la vraie valeur
- $\mathbb{V}\left[\hat{f}(x)\right] = \mathbb{E}\left[\left(\mathbb{E}\left[\hat{f}(x)\right] - \hat{f}(x)\right)\right)^2\right]$: la dispersion moyenne des valeurs prédites autour de la moyenne

La MSE met en relation ces deux quantités : **Erreur incompressible**

$$\text{MSE}(y, \hat{f}(x)) = \left(\text{Bias}\left[\hat{f}(x)\right]\right)^2 + \mathbb{V}\left[\hat{f}(x)\right] + \sigma^2$$



(a) Biais : 0.18, variance : 0.57 et RMSE : 0.21



(b) Biais : 0.16, variance : 0.62 et RMSE : 0.24

Figure – Biais et variance pour un modèle linéaire et un modèle polynomial pour prédire f calculé sur un dataset de test

Problème Machine learning

Descente de gradient

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

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

η_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 minimaux 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.

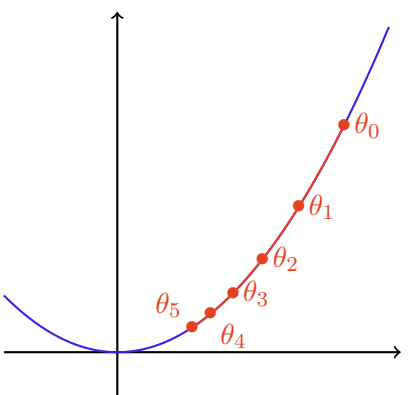


Figure – Exemple d'une descente de gradient pour $f(x) = x^2$

Problème Machine learning

Descente de gradient stochastique

On cherche à minimiser une fonction de perte $\mathcal{L}$ qui s'écrit très souvent en Machine Learning comme la somme d'une fonction sur chaque observations du dataset. On note $\hat{y}_i$ la prédiction d'un algorithme pour l'observation x_i avec $i \leqslant n$ l'index de l'observation i. Pour un problème de régression et de classification, on a par exemple :

$$\begin{aligned} \mathcal{L}(\theta) &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}\ell_i(\theta) \end{aligned}$$

$$\begin{aligned} \mathcal{L}(\theta) &= -\frac{1}{n}\sum_{i=1}^{n} y_i \ln(\hat{y}_i) + (1-y_i)\ln(1-\hat{y}_i) \\ &= -\frac{1}{n}\sum_{i=1}^{n}\ell_i(\theta) \end{aligned}$$

Ainsi, pour une unique mise à jour, on doit appliquer n fonctions ℓ. La descente de gradient stochastique consiste à sélectionner aléatoirement un index i_t et mettre à jour les poids avec uniquement cette observations. La descente de gradient devient alors :

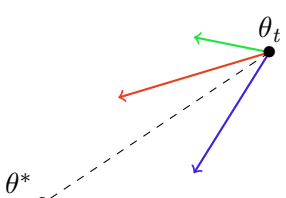
$$\theta_{t+1} = \theta_t - \eta_t \nabla \ell_{i_t}(\theta_t)$$

Problème Machine learning

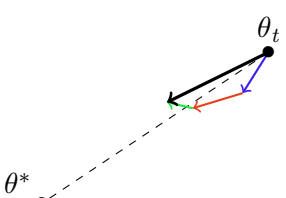
Descente de gradient stochastique par batch

Afin d'obtenir des convergences plus proche du minimum, nous pouvons considérer n_B observations puis mettre à jour. Si $n_B = 1$ on obtient la descente de gradient stochastique et si $n_B = n$ on retrouve la descente de gradient classique. La descente de gradient par mini-batch est donc un compromis entre les deux descentes présentées.
On note $\mathcal{B}_t$ l'ensemble des index choisit aléatoirement tel que $|\mathcal{B}_t| = n_B$, on a :

$$\theta_{t+1} = \theta_t - \eta_t \left(\frac{1}{n_B} \sum_{i \in \mathcal{B}_t} \nabla \ell_i(\theta_t) \right)$$



(a) Calcul des $-\nabla \ell_i(\theta_t)$ pour un batch $\mathcal{B}_t$



(b) Mise à jour des poids

En résumé

Perceptron

Au commencement il y avait le perceptron

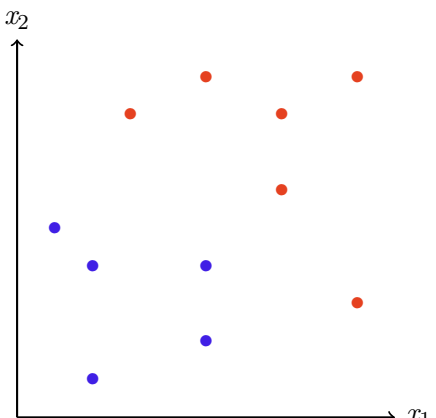


Figure – Visualisation d'un dataset pour un problème de classification avec $d = 2$ et $\mathcal{Y} = \{-1, 1\}$

Ici le dataset est linéairement [1] 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

Au commencement il y avait le perceptron

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 :

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

Dans le cas où les données sont linéairement séparable, la convergence est garantie [2].

2. Voir annexe.

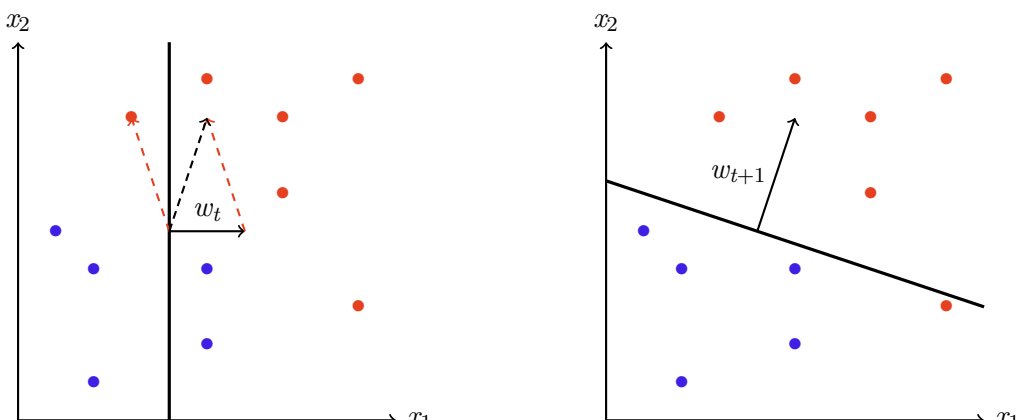
Perceptron

Au commencement il y avait le perceptron

La règle de mise à jour est :

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

Elle se visualise dans notre exemple :



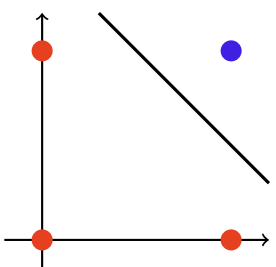
(a) Etape t **(b)** Etape $t+1$

Figure – Intuition géométrique de l'apprentissage du perceptron

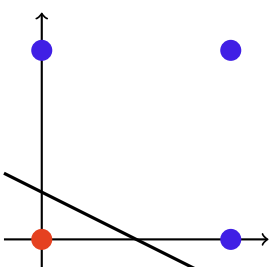
Perceptron

Le problème XOR : premier hiver

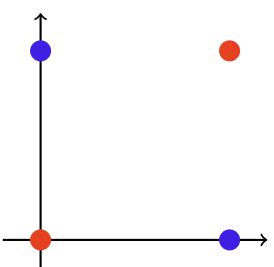
On souhaiterait savoir s'il est possible à l'aide d'un perceptron de reproduire les fonctions logiques NOT, AND, OR et XOR. Pour l'opérateur de négation, $w = -1$ et $b = 0.5$ suffisent pour renvoyer les valeurs attendues.



(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 $\mathrm{XOR}(A, B) = \mathrm{AND}(\mathrm{NOT}(\mathrm{AND}(A, B)), \mathrm{OR}(A, B))$ donc il faudrait combiner plusieurs perceptrons pour y parvenir !

Réseau de neurones

Organisation d'un réseau dense

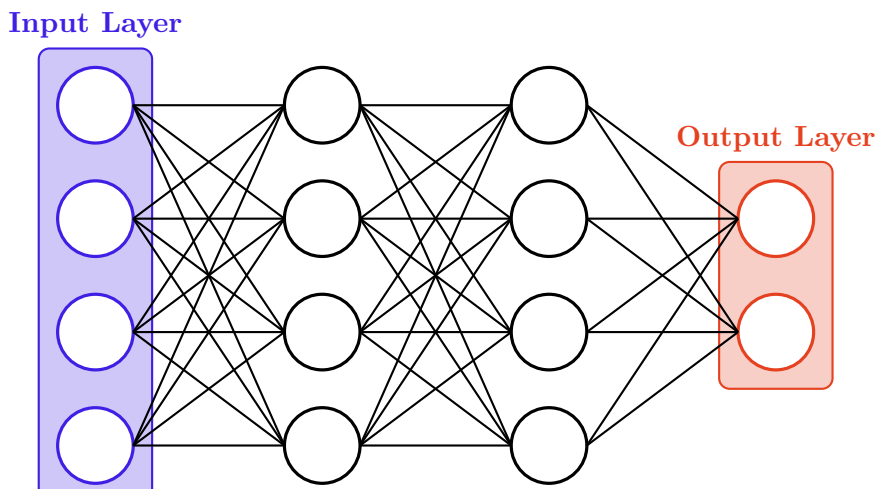


Figure – Exemple d'un réseau de neurones avec 2 couches cachées pour une classification binaire

Réseau de neurones

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.

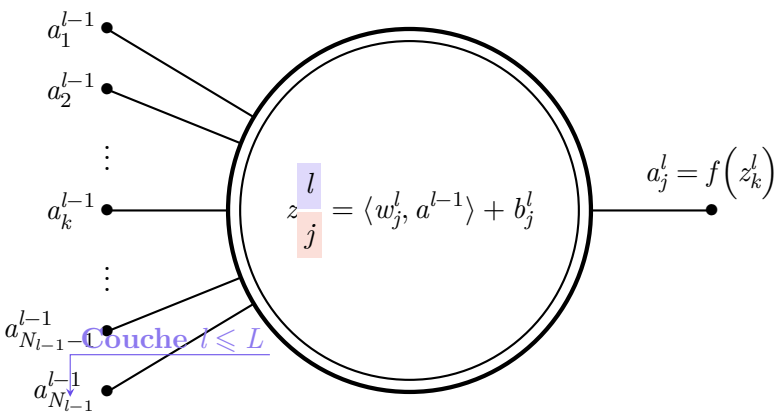


Figure – Neurone j de la couche l paramétré par les poids $w_i^j \in \mathbb{R}^{N_{l-1}}$ et le biais $b_j^l \in \mathbb{R}$

Une question centrale est de comprendre comment on peut *apprendre* ces paramètres.

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 $j \leqslant N_l$ à la couche $l \leqslant L$ les gradients $\dfrac{\partial \mathcal{L}}{\partial w_{j,k}^l}$ et $\dfrac{\partial \mathcal{L}}{\partial b_j^l}$ en conservant les notations d'un neurone $j \leqslant N_l$ à la couche $l \leqslant L$:

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

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

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial b_j^l} &= \frac{\partial \mathcal{L}}{\partial b_j^l} \frac{\partial b_j^l}{\partial z_j^l} \quad \text{mais} \quad \frac{\partial b_j^l}{\partial z_j^l} = 1 \\ &= \frac{\partial \mathcal{L}}{\partial z_j^l} \end{aligned}$$

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

Algorithme de back-propagation

Comment calculer les dérivées partielles de la dernière couche ?

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

On définit $\delta_j^l = \dfrac{\partial \mathcal{L}}{\partial z_j^l}$ 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 $j \leqslant N_L$ 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 $\dfrac{\partial \mathcal{L}}{\partial a_j^L}$ 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 $j \leqslant N_l$ à la couche $l \leqslant L$ on a obtenu que $\dfrac{\partial \mathcal{L}}{\partial b_j^l} = \dfrac{\partial \mathcal{L}}{\partial z_j^l}$ et $\dfrac{\partial \mathcal{L}}{\partial w_{j,k}^l} = a_k^{l-1} \dfrac{\partial \mathcal{L}}{\partial z_j^l}$.

De plus, $\delta_j^L = f'(z_j^L) \dfrac{\partial \mathcal{L}}{\partial a_j^L}$ 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 : probablement de proche en proche. On considère le neurone $j \leqslant N_l$ à la couche $l < L$:

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

Algorithme de back-propagation

Explosion et disparition des gradients

Pour le neurone $j \leqslant N_l$ à la couche $l \leqslant L$ 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 :

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

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 inverse[3] s'appelle *exploding gradient*.
Il faut donc choisir avec plus de précaution la fonction d'activation f.

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

En résumé

Comment intégrer une non-linéarité ?

Fonction d'activation classique

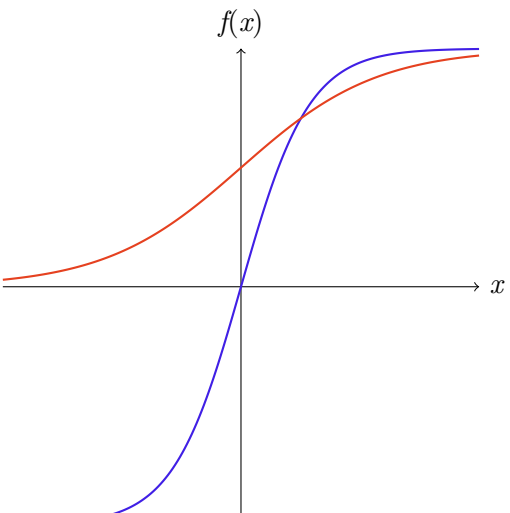


Figure – Fonctions sigmoid et tangente hyperbolique

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.

Comment intégrer une non-linéarité ?

Fonction d'activation ReLU et variantes

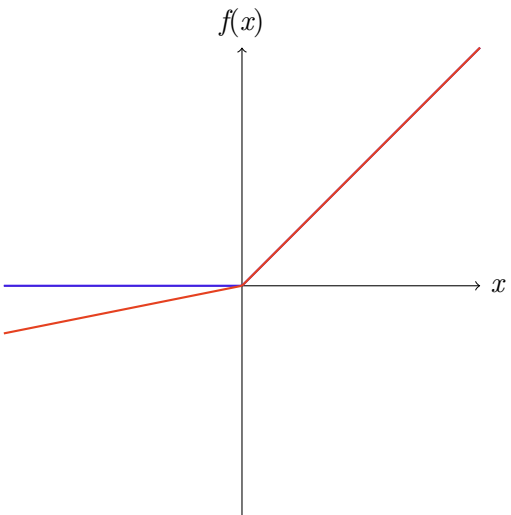


Figure – Fonctions ReLU et Leaky ReLU

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 $\alpha \geqslant 0$ un hyper-paramètre.

En résumé

Comment initialiser les poids du réseau de neurone ?

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 :

Nombre de couche du réseau

Vecteur d'entrée de la couche l

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

Nombre de neurones de la couche l

Matrice des poids du neurone k à la couch

Nous faisons les hypothèses supplémentaires :

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

$$\forall l \leqslant L,\ \mathbb{V}\left[z^l\right] = \mathbb{V}\left[z^1\right] \tag{1}$$

Comment initialiser les poids du réseau de neurone ?

Cas d'une fonction d'activation impaire

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

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

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,

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

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

Comment initialiser les poids du réseau de neurone ?

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

Comment initialiser les poids du réseau de neurone ?

Cas de la fonction ReLU

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

$$\begin{aligned}
\mathbb{E}\left[x^2\right] &= \mathbb{E}\left[f(z)^2\right] \\
&= \int_{-\infty}^{+\infty} f(t)^2 p(t) \mathrm{d}t \text{ avec } p \text{ la densité de probabilité associé à } z \\
&= \int_{0}^{+\infty} t^2 p(t) \mathrm{d}t \\
&= \frac{1}{2}\mathbb{V}\left[z\right]
\end{aligned}$$

Nous avons donc cette fois :

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

A nouveau, une condition suffisante pour que la relation (1) soit vérifée pour la passe forward est :

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

Comment initialiser les poids du réseau de neurone ?

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

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

En résumé

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.

Minsky, M. and Papert, S. (1969).
Perceptrons.
MIT press.

Bibliographie II


Shazeer, N. (2020).
Glu variants improve transformer.
arXiv preprint arXiv :2002.05202.

Annexe : 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 :

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

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

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

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*.

Annexe : 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 :

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

Par une récurrence immédiate, on a :

$$\langle w_k, w^* \rangle \geqslant k\gamma \tag{2}$$

Annexe : 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 :

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

Il ne nous reste plus qu'à appliquer l'inégalité de Cauchy-Schwarz en partant de (2) :

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

Annexe : Réseau de neurones

La fonction de perte n'est pas en général convexe

Considérons le réseau de neurones suivant avec comme fonction d'activation ReLU :

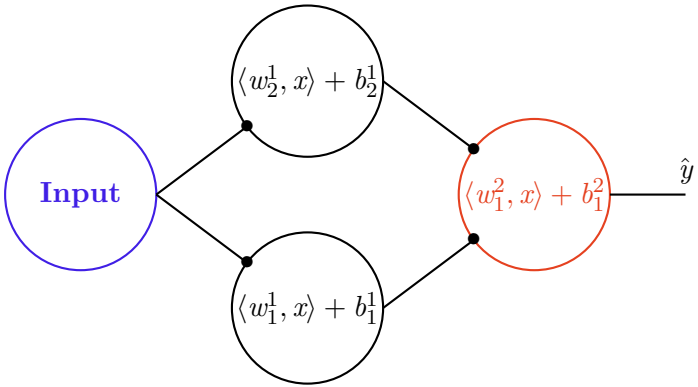


Figure – Réseau de neurone avec une couche caché de deux neurones

Pour le dataset $\mathcal{D} = \{(-1,-1),(1,1)\}$ et la fonction de perte $\mathcal{L}(\theta) = \frac{1}{2}\sum_{i=1}^{2}(\hat{y}_i - y_i)^2$, on a que :

- θ_1 : $w_1^1 = 1, w_2^1 = -1, w_1^2 = (1,-1)$ et $b_1^1 = b_2^1 = b_1^2 = 0$ donne $\mathcal{L}(\theta_1) = 0$
- θ_2 : $w_1^1 = -1, w_2^1 = 1, w_1^2 = (-1,1)$ et $b_1^1 = b_2^1 = b_1^2 = 0$ donne $\mathcal{L}(\theta_2) = 0$

Si la fonction de perte $\mathcal{L}$ est convexe, alors en prenant une combinaison linéaire entre les deux θ obtenu, on doit avoir encore un minimum. Mais ce n'est pas le cas !

En général dans un réseau de neurone, nous ne sommes pas dans un cadre convexe. Nous souhaitons accélérer la descente de gradient ainsi qu'obtenir des garanties pratique que la méthode *classique* n'offre pas, pour s'assurer d'une *meilleure convergence*.

Annexe : Descente de gradient stochastique

Descente en espérance

Notons que la descente de gradient stochastique n'est pas vraiment une *descente* : nous ne pourrons garantir une descente qu'en espérance. On suppose que $\mathcal{L}$ est différentiable et β-smooth, qu'il existe un minimum pour cette fonction et que chaque ℓ_i soit différentiable continue.
Alors, puisque $\mathcal{L}$ est β-smooth :

$$\mathcal{L}(\theta_{t+1}) \leqslant \mathcal{L}(\theta_t) + \langle \nabla \mathcal{L}(\theta_t), \theta_{t+1} - \theta_t \rangle + \frac{\beta}{2} \| \underbrace{\theta_{t+1} - \theta_t}_{\eta_t \nabla \ell_{i_t}(\theta_t)} \|^2$$

Ainsi, en espérance sur le choix aléatoire de i_t, on a :

$$\mathbb{E}[\mathcal{L}(\theta_{t+1})] \leqslant \mathcal{L}(\theta_t) + \eta_t \langle \nabla \mathcal{L}(\theta_t), \mathbb{E}[\nabla \ell_{i_t}(\theta_t)] \rangle + \frac{\beta \eta_t^2}{2} \mathbb{E}\left[\|\nabla \ell_{i_t}(\theta_t)\|^2 \right]$$

Cela ne nous assure pas pour autant une garantie de descente en espérance. Nous devons faire des hypothèses supplémentaires.

Annexe : Descente de gradient stochastique

Hypothèse 1

Un index i_t d'une mise à jour t est tiré selon :

- i_t *ne dépend pas des index* $i_0, \dots, i_{t-1}$
- $\nabla \ell_{i_t}(\theta_t)$ *est un estimateur non biaisé de* $\nabla \mathcal{L}(\theta_t)$
- $\mathbb{E}\left[\|\nabla \ell_{i_t}(\theta_t)\|^2\right] \leqslant \sigma^2 + \|\mathcal{L}(\theta)\|^2$

Si les indices sont tirés uniformément alors les deux premières hypothèses sont vérifiées. Pour le troisième point, s'il existe $M > 0$ tel que pour tout itérations t on a $\|\nabla \ell_{i_t}(\theta_t)\| \leqslant M$, alors il est vérifié. Dans ce cas, on a une garantie de descente en espérance :

$$\mathbb{E}\left[\mathcal{L}\left(\theta_{t+1}\right)\right] \leqslant \mathcal{L}\left(\theta_t\right) - \left(\eta_t - \frac{\beta \eta_t^2}{2}\right) \|\nabla \mathcal{L}(\theta_t)\|^2 + \frac{\beta \eta_t^2}{2} \sigma^2 \tag{3}$$

En supposant que $\eta_t \leqslant \dfrac{1}{\beta}$.

Annexe : Descente de gradient stochastique

Garantie de convergence

On se place dans le cadre d'une optimisation d'une fonction de perte $\mathcal{L}$ non convexe, mais β-smooth. On conserve les hypothèses (1) précédentes. On note $\theta^* = \arg\min_{\theta \in \mathbb{R}^d} \mathcal{L}(\theta)$

Théorème 2 (Learning rate fixe)

On considère une descente de gradient stochastique avec $\eta_t = \eta$ tel que $\eta \in \left]0, \frac{1}{\beta}\right]$. Alors, pour tout $T \geqslant 1$:

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=0}^{T-1} \|\nabla \mathcal{L}(\theta_t)\|^2\right] \leqslant \eta\beta\sigma^2 + \frac{2(\mathcal{L}(\theta_0) - \mathcal{L}(\theta^*))}{\eta T}$$

On a donc que $\lim_{T\to+\infty} \mathbb{E}\left[\min_{0\leqslant t\leqslant T-1} \|\nabla \mathcal{L}(\theta_t)\|^2\right] \in [0, \eta\beta\sigma^2]$: le bruit nous empêche de converger vers un point de gradient nul. Ce n'est pas non plus une garantie de convergence vers le minimum global.

Annexe : SGD par mini-batch

Réduction de variance

On conserve les hypothèses (1) que l'on a faite sur la fonction de perte et l'aléatoire de tirage des index dont les deux dernières conditions sont :

- $\nabla \ell_{i_t}(\theta_t)$ est un estimateur non biaisé de $\nabla \mathcal{L}(\theta_t)$
- $\mathbb{E}\left[\|\nabla \ell_{i_t}(\theta_t)\|^2\right] \leqslant \sigma^2 + \|\mathcal{L}(\theta)\|^2$

On obtient le résultat suivant :

Proposition 1 (Variance)

La variance d'une estimation par descente de gradient stochastique par mini-batch utilisant n_B échantillons avec remise vérifie :

$$\mathbb{E}\left[\left\|\frac{1}{n_B}\sum_{i\in\mathcal{B}_t}\nabla \ell_i(\theta_t)\right\|^2\right] \leqslant \frac{\sigma^2}{n_B} + \|\nabla \mathcal{L}(\theta_t)\|^2$$

Annexe : SGD par mini-batch

Garantie de convergence

A l'aide de la proposition 1 nous pouvons déduire un résultat similaire au théorème 2 qui le généralise.

Théorème 3 (Learning rate fixe pour SGD par mini-batch)

On considère une descente de gradient stochastique par mini-batch avec remise de taille n_B *avec* $\eta_t = \eta$ *tel que* $\eta \in \left]0, \frac{1}{\beta}\right]$. *Alors, pour tout* $T \geqslant 1$ *:*

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=0}^{T-1} \|\nabla \mathcal{L}(\theta_t)\|^2\right] \leqslant \frac{\eta\beta\sigma^2}{n_B} + \frac{2(\mathcal{L}(\theta_0) - \mathcal{L}(\theta^*))}{\eta T}$$

On obtient alors que $\lim_{T\to+\infty} \mathbb{E}\left[\min_{0\leqslant t\leqslant T-1} \|\nabla\mathcal{L}(\theta_t)\|^2\right] \in \left[0, \frac{\eta\beta\sigma^2}{n_B}\right]$.

Il est important de noter que l'on ne traite que du cas où le mini-batch est construit avec remise et que l'on a supposé des indépendances, distributions identiques et des estimations non biaisées. Il est difficile de s'en assurer en pratique.

Annexe : Fonctions d'activation

Fonctions d'activation moderne

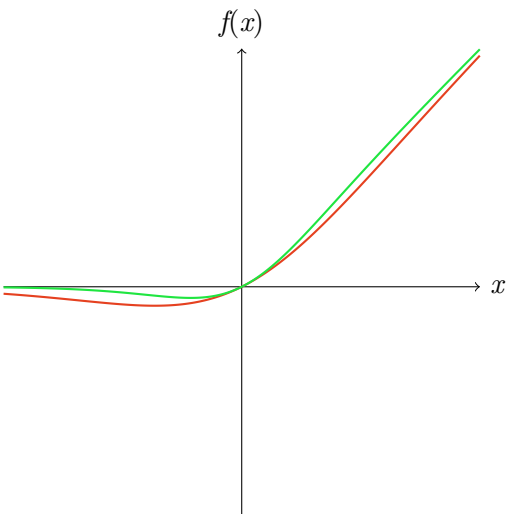


Figure – SiLU et GELU

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

$$\begin{aligned} \text{SiLU}(x) &= x\sigma(x) \quad \text{avec} \quad \sigma(x) = \frac{1}{1+e^{-x}} \\ \text{GELU}(x) &= x\phi(x) \quad \text{avec} \quad \phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} \mathrm{d}t \end{aligned}$$

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].