Introduction aux Réseaux de Neurones

Lucas Carpentier

Électif Machine Learning 2024-2025
SeaTech Toulon

December 15, 2024

Plan de la Présentation

Origines des Réseaux de Neurones

- Inspiration biologique : fonctionnement des neurones

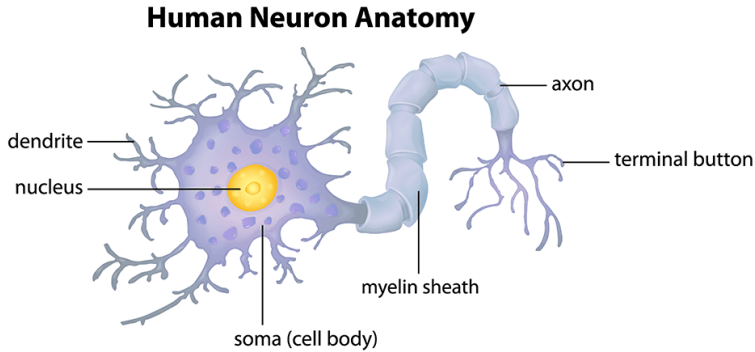


Figure 1: Illustration de l'anatomie d'un neurone humain

Premiers Neurones Artificiels

- Travaux de McCulloch et Pitts (1943)

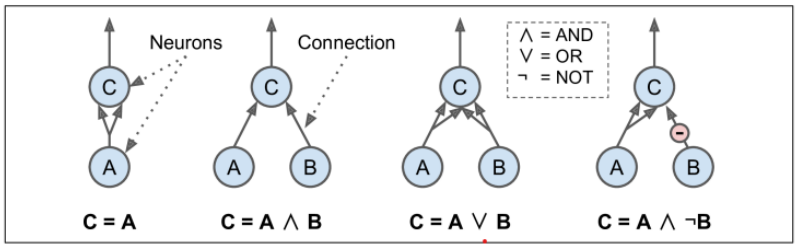


Figure 2: Neurones logiques de McCulloch et Pits

Le Modèle du Perceptron

- Perceptron de Rosenblatt (1957)

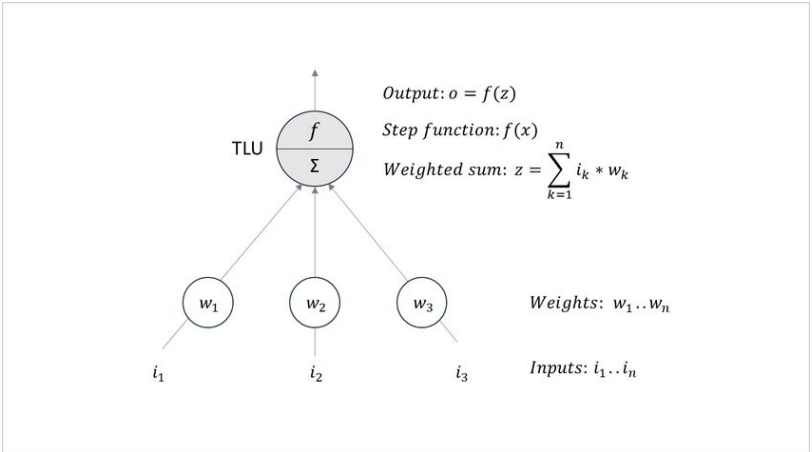


Figure 3: Threshold Logic Unit (TLU) de Rosenblatt

Réseau de Neurone Entièrement Connecté (Dense)

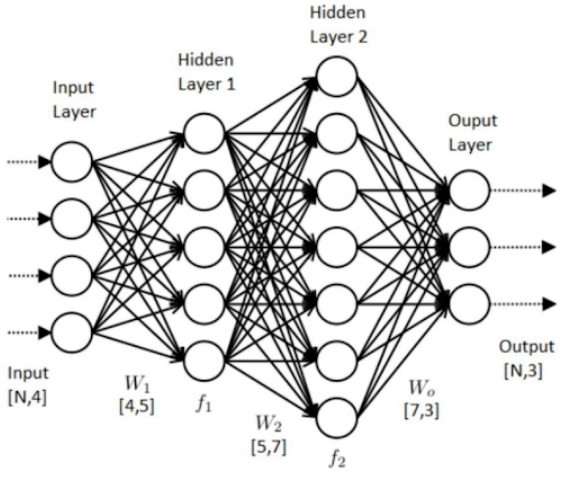


Figure 4: Schéma d'un Réseau de Neurone Artificiel

Architecture Simplifiée d'un Réseau de Neurone

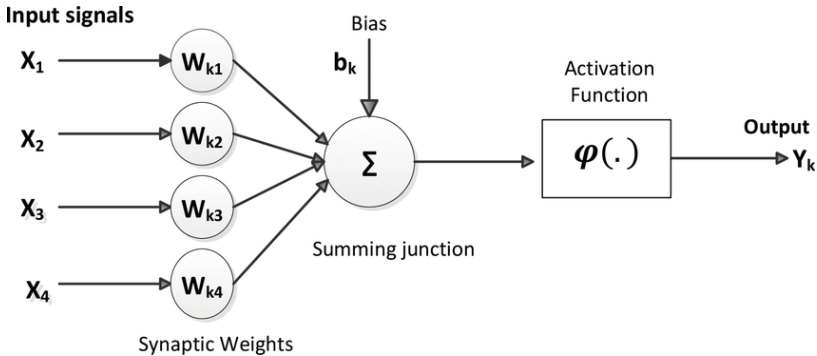


Figure 5: Schéma d'une Couche d'un Réseau de Neurone Artificiel

Fonctions d'Activation

Calcul par Neurone

Chaque neurone utilise la formule :

$$y = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

où ϕ est la fonction d'activation, w_i les poids et b le biais.

- ReLU : $f(x) = \begin{cases} x & \text{si } x > 0 \\ 0 & \text{si } x \leq 0 \end{cases}$ où $f(x) \in \mathbb{R}^+$
- Softmax : $f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$ où $\sum_{i=1}^{n} f(x)_i = 1$
- Tanh : $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ où $f(x) \in [-1, 1]$

Fonctions d'activation

- Quelques exemples:

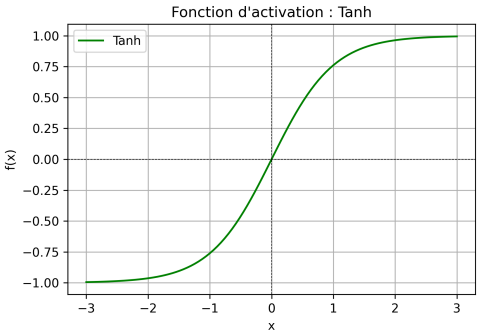


Figure 6: Fonction Tangeante Hyperbolique

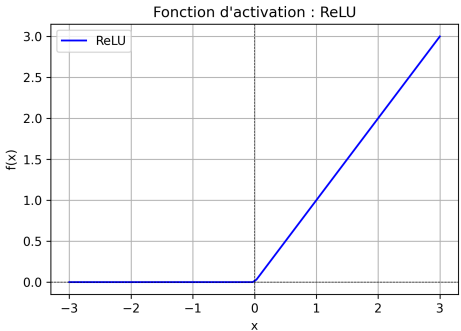


Figure 7: Fonction Rectified Linear Unit

Objectif

- Les fonctions d'activation permettent d'apprendre des relations non linéaires.

Fonction de Coût

Définition

- La fonction de coût $J(\theta)$ mesure l'écart entre les prédictions du réseau $(\hat{y})$ et les valeurs cibles (y).
- Objectif : Trouver les paramètres θ qui minimisent $J(\theta)$.

- Régression : Erreur quadratique moyenne (MSE)

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} (\theta^T x_i - y_i)^2$$

- θ : vecteur des paramètres
- x_i : observation numéro i
- $(\theta^T x_i$: prédiction du modèle pour l'observation i

- Classification : Entropie croisée

$$J(\theta) = -\frac{1}{n} \sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

Exemple de Fonction de Coût

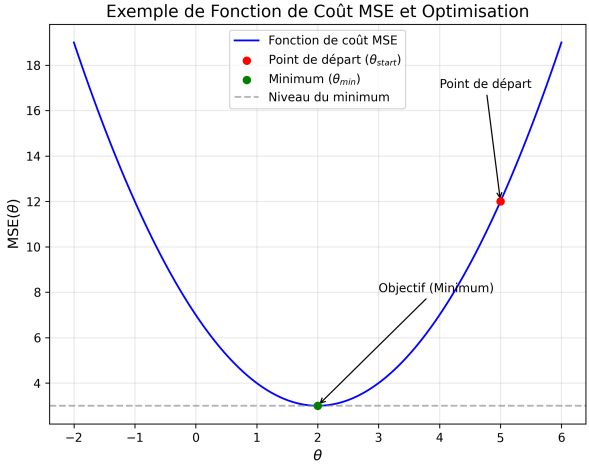


Figure 8: Fonction Coût Pour des Variables Linéaires

Critère d'Optimisation : Maximum de Vraisemblance

Définition

- Le but est de maximiser la vraisemblance des données par rapport au modèle :

$$\theta^* = \arg\max_{\theta} \prod_{i=1}^{m} P_{\text{model}}(x^{(i)}; \theta).$$

- En passant au logarithme pour simplifier le calcul :

$$\theta^* = \arg\max_{\theta} \sum_{i=1}^{m} \log P_{\text{model}}(x^{(i)}; \theta).$$

- On passe au logarithme car :
  - fonction monotone donc même objectif
  - produit devient somme
  - évite la multiplication de termes très faibles (risque d'underflow)

Critère d'Optimisation : Divergence KL

Définition

- La divergence de Kullback-Leibler mesure la non-similarité entre deux distributions de probabilité :

$$D_{\mathrm{KL}}(P||Q) = \sum_{x \in X} P(x) \log \left(\frac{P(x)}{Q(x)} \right).$$

- En apprentissage supervisé l'objectif est de minimiser la non-similarité entre le training set et le modèle :

$$\theta^* = \arg\min_{\theta} D_{\mathrm{KL}}(P_{\mathrm{data}}||P_{\mathrm{model}}).$$

Lien avec la Vraisemblance

- Minimiser $D_{\mathrm{KL}}(P_{\mathrm{data}}||P_{\mathrm{model}})$ revient à maximiser le maximum de vraisemblance.
- Objectif : aligner $P_{\mathrm{model}}(x;\theta)$ avec $P_{\mathrm{data}}(x)$.

Descente de Gradient

Objectif

- Minimiser la fonction de coût $J(\theta)$ en ajustant les paramètres θ.
- Utilisation des gradients pour connaître la direction de la descente.

- Mise à jour des paramètres

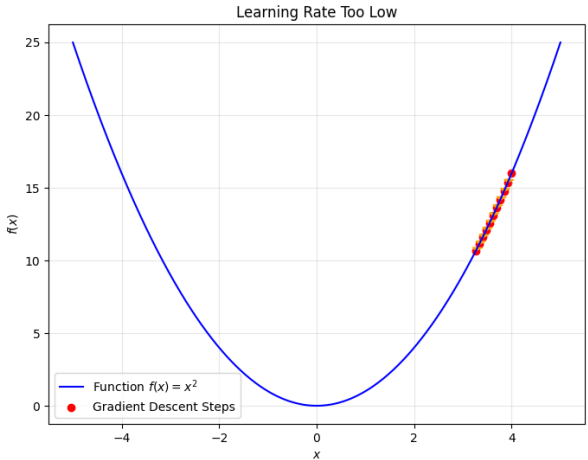
$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta)$$

- η : Taux d'apprentissage, contrôle la vitesse de mise à jour.
- $\nabla_\theta J(\theta)$: Gradient de $J(\theta)$ par rapport à θ.

Défi

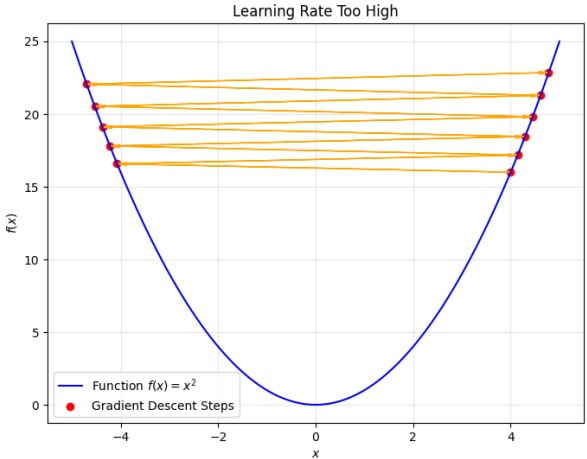
- Calculer $\nabla_\theta J(\theta)$ efficacement dans des réseaux de neurones complexes.

Taux d'Apprentissage Trop Faible



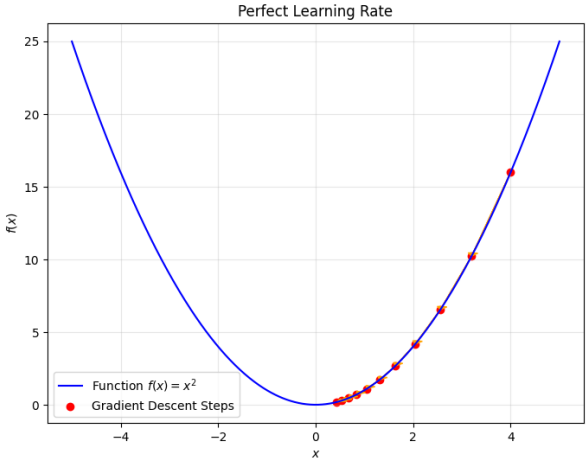
- **Observation :** La descente est stable mais extrêmement lente.

Taux d'Apprentissage Trop Élevé



- **Observation :** Convergence jamais atteinte.

Taux d'Apprentissage Optimal



- **Observation :** Convergence rapide et efficace.

Impact des Taux d'Apprentissage Adaptatifs

Optimisateurs Adaptatifs

Les optimisateurs modernes ajustent dynamiquement le taux d'apprentissage pour améliorer la convergence :

- **Adagrad** : Convient aux paramètres nécessitant des taux plus faibles, mais tend à réduire trop le taux à long terme.
- **RMSprop** : Stabilise la convergence en régulant les oscillations, idéal pour les problèmes non convexes.
- **Adam** : Combine les avantages d'Adagrad et RMSprop, en intégrant une estimation des moments d'ordre 1 et 2.

Avantages des Taux Adaptatifs

- Réduction des oscillations et stabilisation de la descente.
- Moins sensible au choix du taux d'apprentissage initial.
- Performances améliorées sur des modèles complexes (CNN, RNN, etc.).

Descente de Gradient

Objectif

- Minimiser la fonction de coût $J(\theta)$ en ajustant les paramètres θ.
- Utilisation des gradients pour connaître la direction de la descente.

- Mise à jour des paramètres

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta)$$

- η : Taux d'apprentissage, contrôle la vitesse de mise à jour.
- $\nabla_\theta J(\theta)$: Gradient de $J(\theta)$ par rapport à θ.

Défi

- Calculer $\nabla_\theta J(\theta)$ efficacement dans des réseaux de neurones complexes.

Origines de la Rétropropagation

- Seppo Linnainmaa, **algorithme de différentiation automatique en mode inverse**.
- Idée de calculer les dérivées dans des chaînes de calcul (règle de la chaîne) de manière efficace.
- Base de l'algorithme de rétropropagation que nous utilisons toujours aujourd'hui.
- En 1985, Rumelhart, Hinton et Williams ont appliqué cette technique au contexte des réseaux de neurones.
- La rétropropagation permet d'ajuster les poids en fonction de l'erreur en minimisant la fonction de coût $J(\theta)$ via la descente de gradient.

Backpropagation

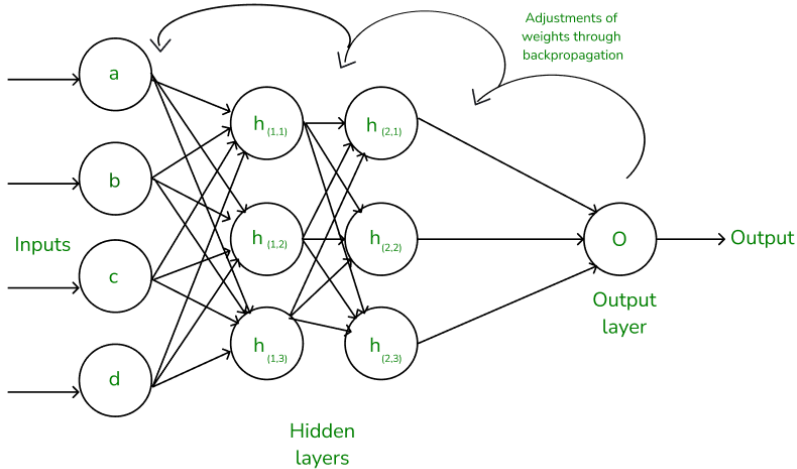


Figure 9: Principe de la Rétropropagation dans un Réseau de Neurone

La règle de la chaîne

Principe général

Si une variable y dépend d'une seconde variable u, qui dépend elle-même d'une variable x, alors :

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}.$$

Principe mathématique

- Soient $f(x)$ et $g(x)$ deux fonctions telles que $h(x) = f(g(x))$. - La dérivée de $h(x)$ est donnée par :

$$h'(x) = f'(g(x)) \cdot g'(x)$$

Utilité

La règle de la chaîne est essentielle pour analyser les dépendances successives entre les variables, en décomposant le calcul des dérivées en étapes simples.

Backpropagation et règle de la chaîne

- La **backpropagation** permet de calculer efficacement les gradients de la fonction de coût $\mathcal{L}$ par rapport aux paramètres w et b.
- Elle utilise la **règle de la chaîne**, qui permet de propager les dérivées à travers les couches d'un réseau de neurones.

Règle de la chaîne appliquée :

$$\frac{\partial \mathcal{L}}{\partial w^{[l]}} = \frac{\partial \mathcal{L}}{\partial a^{[l]}} \cdot \frac{\partial a^{[l]}}{\partial z^{[l]}} \cdot \frac{\partial z^{[l]}}{\partial w^{[l]}}$$

Étapes générales :

1. Calculer l'erreur à la sortie ($\delta^{[L]}$).
2. Propager cette erreur à travers les couches ($\delta^{[l]}$).
3. Mettre à jour les poids et biais avec :

$$w^{[l]} \leftarrow w^{[l]} - \alpha \cdot \frac{\partial \mathcal{L}}{\partial w^{[l]}}$$

Exemple : Calcul des gradients avec entropie croisée

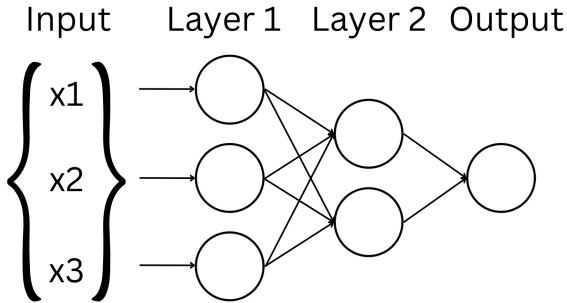


Figure 10: Schéma du réseau utilisé

$$z_i = w_i^T \cdot x_{i-1} + b_i$$
$$a_i = f(z_i)$$
$$pred = f(w_3^T \cdot x_2 + b_3)$$

Exemple : Calcul des gradients avec entropie croisée

Fonction de coût :

$$\mathcal{L} = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$$

Gradient de la fonction de coût pour $w^{[3]}$:

$$\frac{\partial \mathcal{L}}{\partial w^{[3]}} = -\left[y^{(i)}\frac{\partial}{\partial w^{[3]}}\left(\log\left(\sigma(w^{[3]}a^{[2]}+b^{[3]})\right)\right) + (1-y^{(i)})\frac{\partial}{\partial w^{[3]}}\left(\log\left(1-\sigma(w^{[3]}a^{[2]}+b^{[3]})\right)\right)\right.$$

Simplification : En appliquant les dérivées, on obtient :

$$\frac{\partial \mathcal{L}}{\partial w^{[3]}} = -\frac{1}{m}\sum_{i=1}^{m}\left[(y^{(i)}-\hat{y}^{(i)})\cdot a^{[2]\top}\right]$$

où :

$$\hat{y}^{(i)} = \sigma(w^{[3]}a^{[2]}+b^{[3]})$$

Gradient pour $w^{[2]}$

Expression générale avec la règle de la chaîne :

$$\frac{\partial \mathcal{L}}{\partial w^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}} \cdot \frac{\partial z^{[3]}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial w^{[2]}}$$

Développement des termes :

- $\frac{\partial \mathcal{L}}{\partial a^{[3]}} = (\hat{y} - y)$,
- $\frac{\partial a^{[3]}}{\partial z^{[3]}} = a^{[3]}(1 - a^{[3]})$,
- $\frac{\partial z^{[3]}}{\partial a^{[2]}} = w^{[3]\top}$,
- $\frac{\partial a^{[2]}}{\partial z^{[2]}} = a^{[2]}(1 - a^{[2]})$,
- $\frac{\partial z^{[2]}}{\partial w^{[2]}} = a^{[1]\top}$.

Expression finale :

$$\frac{\partial \mathcal{L}}{\partial w^{[2]}} = (\hat{y} - y) \cdot a^{[3]}(1 - a^{[3]}) \cdot w^{[3]\top} \cdot a^{[2]}(1 - a^{[2]}) \cdot a^{[1]\top}$$

Gradient pour $w^{[1]}$

Expression générale avec la règle de la chaîne :

$$\frac{\partial \mathcal{L}}{\partial w^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}} \cdot \frac{\partial z^{[3]}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}} \cdot \frac{\partial z^{[1]}}{\partial w^{[1]}}$$

Expression finale :

$$\frac{\partial \mathcal{L}}{\partial w^{[1]}} = (\hat{y} - y) \cdot a^{[3]}(1 - a^{[3]}) \cdot w^{[3]\top} \cdot a^{[2]}(1 - a^{[2]}) \cdot w^{[2]\top} \cdot a^{[1]}(1 - a^{[1]}) \cdot a^{[0]\top}$$

Différentiation en mode inverse : pourquoi l'utiliser ?

1. Avantages de la différentiation en mode inverse

- Méthode efficace pour calculer les dérivées dans des modèles complexes comme les réseaux de neurones.
- Particulièrement adaptée pour les fonctions avec de nombreuses entrées (poids et biais) et peu de sorties (valeurs prédictives).

2. Importance du caching des variables pendant la passe avant

- Les variables intermédiaires ($z^{[l]}, a^{[l]}$) sont **mises en cache** pendant la passe avant.
- Cela évite des recalculs inutiles pendant la rétropropagation, accélérant considérablement le processus.
- Le caching est crucial pour que la différentiation en mode inverse reste optimale.

Comparaison de Différents Types de Descente de Gradient

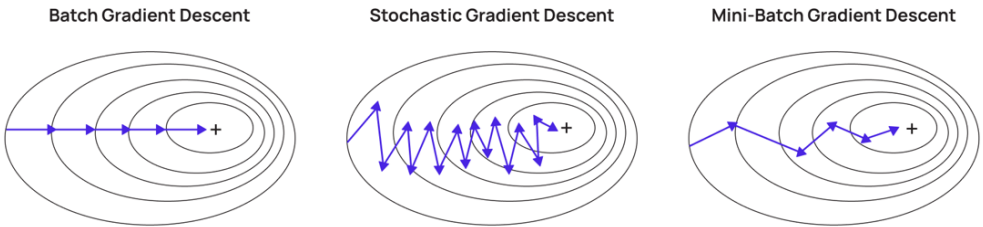


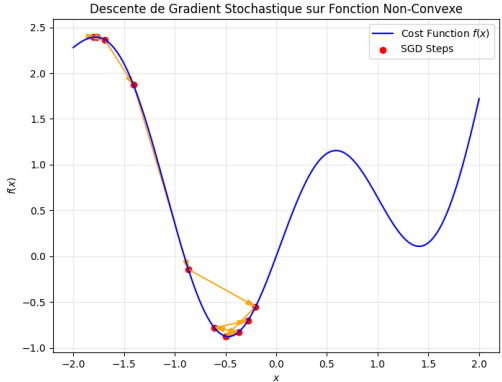
Figure 11: Comparaison de Méthodes de Descente de Gradient

- **Remarque :** Ces optimisateurs sont tous utilisables dans Scikit-Learn, Tensorflow ou encore PyTorch.

Avantages de la Descente de Gradient Stochastique

Problème des minimums locaux

- Les fonctions de coût complexes présentent souvent des minimums locaux.
- La descente de gradient classique peut se bloquer dans ces minimums.



Problèmes : Gradient Vanishing et Gradient Explosion

Gradient Vanishing

- Dans les réseaux profonds, les gradients deviennent extrêmement petits au fur et à mesure qu'ils sont propagés en arrière.
- Résultat :
  - Les couches proches de l'entrée ne sont pas correctement mises à jour.
  - L'entraînement devient inefficace, voire impossible.
- Lié aux fonctions d'activation comme **sigmoïde** ou **tanh**.

Gradient Explosion

- Les gradients deviennent exponentiellement grands pendant la rétropropagation.
- Résultat :
  - Les paramètres du réseau divergent (valeurs infinies ou NaN).
  - Instabilité du modèle.
- Souvent causé par des poids initiaux mal choisis ou l'absence de régularisation.

Exemple de Problème de Vanishing Gradient

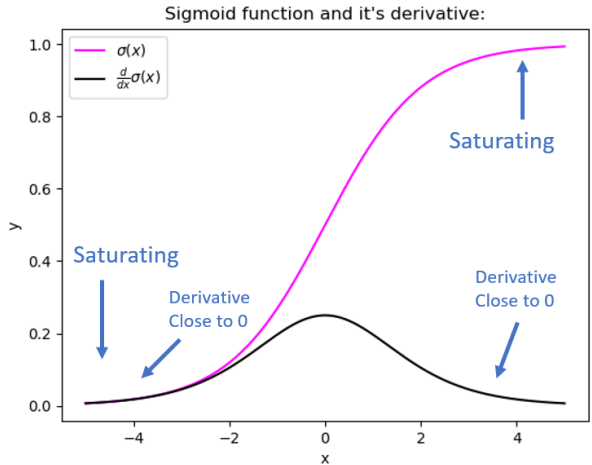


Figure 12: Explication des Causes de la Disparition du Gradient pour la Fonction Sigmoïde

Régularisation : Pourquoi ?

Objectifs principaux

- **Réduire le surapprentissage (overfitting)** : Empêche le modèle de mémoriser les données bruitées en limitant sa complexité.
- **Encourager des modèles simples** : Pénalise les modèles aux paramètres trop grands ou inutiles.
- **Stabiliser l'entraînement** : Évite les comportements instables et assure une convergence fiable.

Techniques principales

- **L1/L2 Regularization** : Limite la magnitude des poids.
- **Dropout** : Désactive aléatoirement certains neurones pour améliorer la robustesse.

Régularisation : Ridge, Lasso et Elastic Net

Ridge (L2 Regularization)

- Pénalité basée sur la somme des carrés des poids ($||w||_2^2$) :

$$J(\theta) = \text{Loss} + \lambda \sum_{i=1}^{n} w_i^2$$

- Réduit la magnitude des poids pour éviter les surajustements.
- Préserve tous les paramètres mais les rend plus petits.

Lasso (L1 Regularization)

- Pénalité basée sur la somme des valeurs absolues des poids ($||w||_1$) :

$$J(\theta) = \text{Loss} + \lambda \sum_{i=1}^{n} |w_i|$$

Régularisation : Ridge, Lasso et Elastic Net

Elastic Net

- Combine les pénalités de Ridge et Lasso :

$$J(\theta) = \text{Loss} + \alpha\lambda \sum_{i=1}^{n} |w_i| + (1-\alpha)\lambda \sum_{i=1}^{n} w_i^2$$

- Bénéficie des avantages des deux méthodes :
  - Sélectionne les caractéristiques (Lasso).
  - Stabilise les poids (Ridge).
- Paramètre α contrôle le ratio entre L1 et L2.

Conclusion et Types de Réseaux Formés

- **Perceptrons Multicouches (MLP)** :
  - Réseaux de base utilisés pour des tâches de classification ou régression simples.
- **Réseaux Convolutionnels (CNN)** :
  - Conçus pour traiter des données spatiales (images, vidéos).
- **Réseaux Récurrents (RNN, LSTM, GRU)** :
  - Idéaux pour des données séquentielles (texte, séries temporelles).
- **Transformers** :
  - Dominants dans les applications modernes de NLP.

Références

- Neptune AI. *Vanishing and Exploding Gradients: Debugging, Monitoring, and Fixing*. Disponible sur : `https://neptune.ai/blog/vanishing-and-exploding-gradients-debugging-monitoring-fixing`
- Pierre Giguère. *Deep Learning Optimization*, Université Laval à Québec. Disponible sur : `http://www2.ift.ulaval.ca/~pgiguere/cours/DeepLearning/03-Deep_Learning_Optimization.pdf`
- Andrew NG. *Gradient Descent Techniques and Optimization Strategies CS229 Stanford*. Vidéo : `https://www.youtube.com/watch?v=zUazLXZZA2U&t=2641s`
- Valentin Gies. *Embedded AI - Master 2 Course*. Disponible sur : `https://www.vgies.com/downloads/Cours/Cours_Embedded_AI_M2_V_GIES.pdf`