



Programmation concurrente

Les Réseaux de neurones Artificiels

Notions de parallélisme et de concurrence

Université Paris 8
Département Informatique et Technologies du Numérique ITN
Master 1 Informatique
Parcours : Informatique & Big Data

Y. Y. Touati
capaok@gmail.com

Table des matières

Le cerveau

- Partitions du cerveau
- Fonctionnalités
- le neurones dans le cortex cérébral
- Le neurone biologique
- La synapse

Le Perceptron

- Le neurone formel
- Architecture et Formalisme : McCulloch-Pitts
- Architecture et Formalisme : Franck RosenBlatt
- Neurone biologique – Perceptron
- Le MLP

Le deep learning

- Fonctions d'activation
- Contexte multi-classes

Performances d'un réseau de neurones

- La fonction de perte, i.e., Loss

Apprentissage

- Les paradigmes
- Le mécanisme en quelques lignes
- Overfitting vs. Underfitting

Descente du gradient stochastique

- Application sur un Réseau de neurones simple
- Calcul du gradient
- Aspect mathématique
- Convergence – pas d'adaptation – CI
- Hyper-paramètre : Learning rate
- Comparatif des différentes variantes DGS

La Rétropropagation

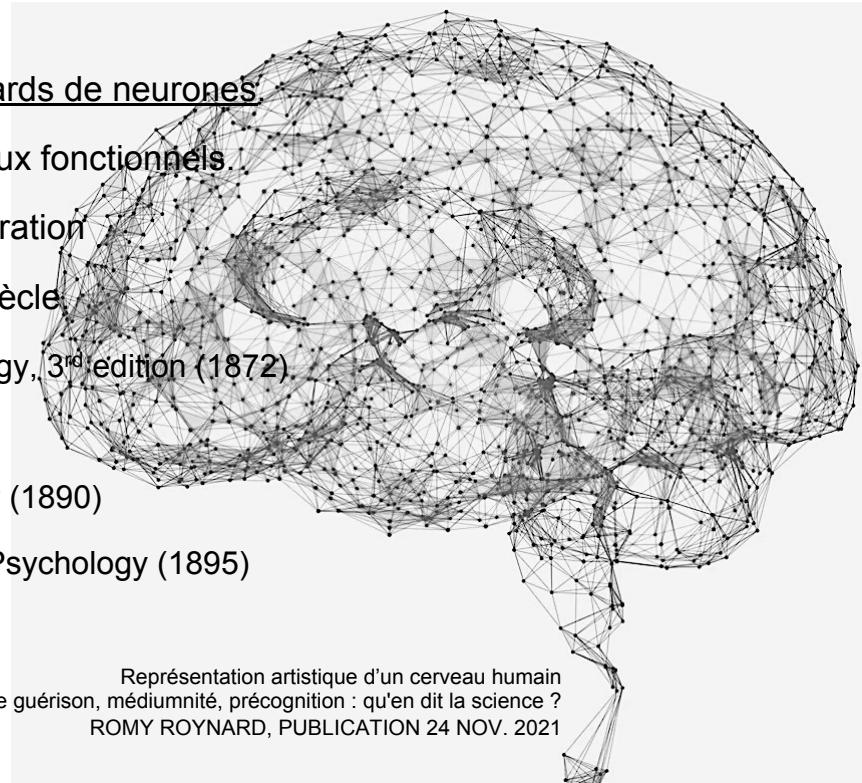
- Etapes de mise en œuvre
- Hyper-paramètre : Learning rate
- Par l'image

Comparatifs DL, ML

Application sur un MLP à 3 couches

Le cerveau

- Organe humain avec plus de 80-100 milliards de neurones
- Organisation : Plusieurs millions de réseaux fonctionnels
- Plasticité cérébrale, capacité de reconfiguration
- Premiers travaux : Vers la fin du 19^{ème} siècle
 - Herbert Spencer, Principles of Psychology, 3rd édition (1872)
 - Theodor Meynert, Psychiatry (1884)
 - William James, Principles of Psychology (1890)
 - Sigmund Freud, Project for a Scientific Psychology (1895)

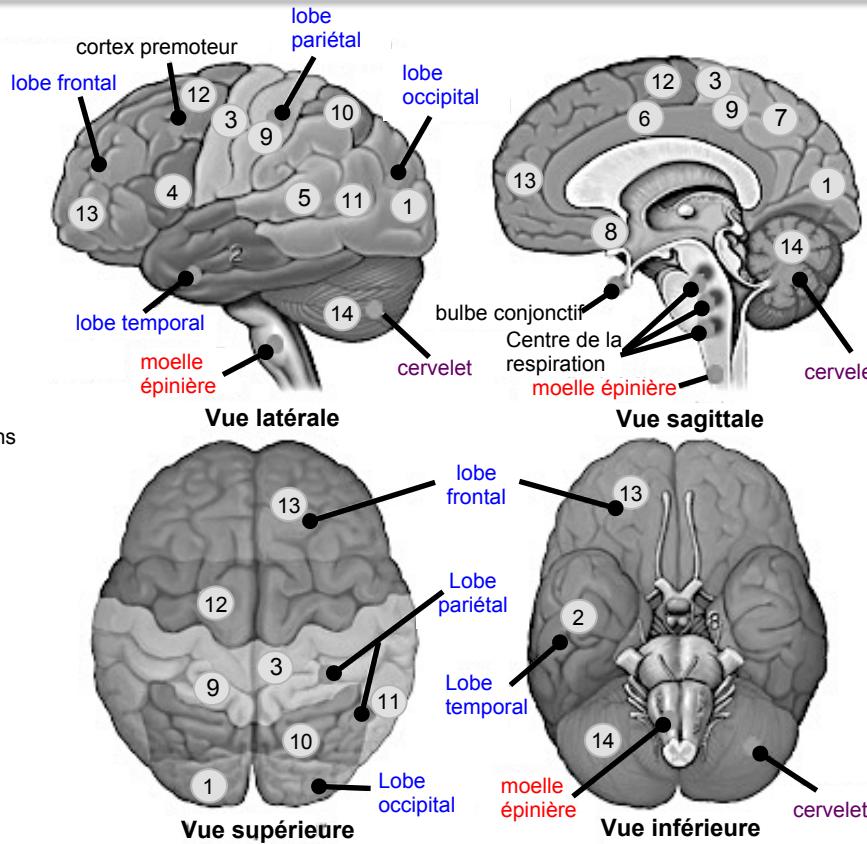


Représentation artistique d'un cerveau humain
Don de guérison, médiumnité, pré cognition : qu'en dit la science ?
ROMY ROYNARD, PUBLICATION 24 NOV. 2021

Le cerveau

Partitions du cerveau

- 1 Aire visuelle
- 2 Aire associative
- 3 Cortex moteur
- 4 Aire du langage (Broca)
- 5 Aire auditive
- 6 Aire émotionnelle
- 7 Aire de l'association des sens
- 8 Aire olfactive
- 9 Cortex somesthésique
- 10 Aire associative
- 11 Aire de compréhension
- 12 Aire pré motrice
- 13 Aire préfrontale
- 14 Cervelet



Fonctionnalités

lobe pariétal Langage
Touché
Odorat
Goût

lobe frontal Réflexion
Discussion
Mémoire
Mouvement

lobe occipital Vision
Couleur
Formes

lobe temporal Ouïe
Apprentissage
Sentiment
Peur

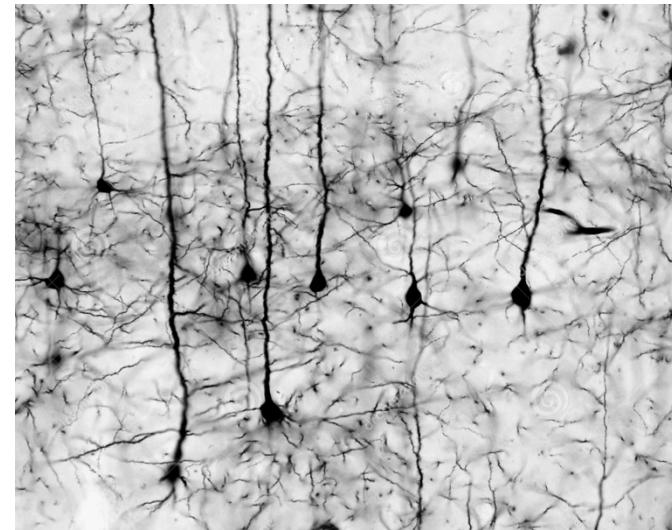
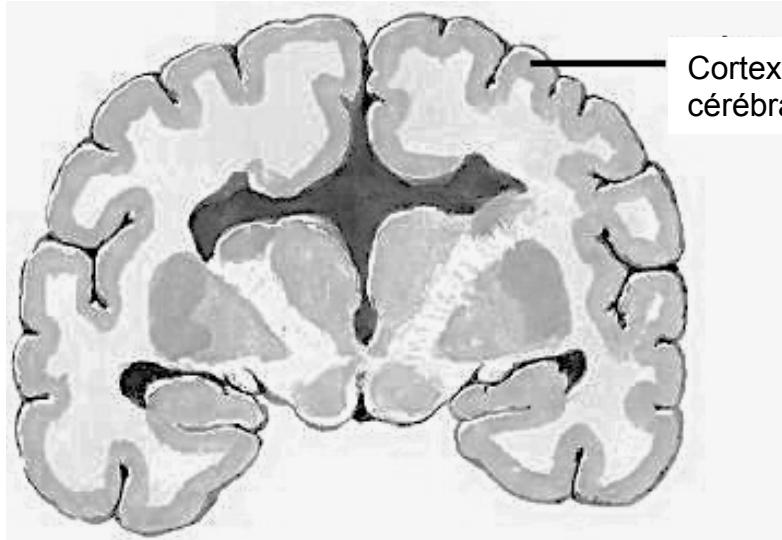
Le cerveau

Fonctionnalités

- Cellule nerveuse excitable formant l'unité fonctionnelle de la base du système nerveux.
- Grande capacité réactive face aux stimulations transformées en impulsions nerveuses, i.e., excitabilité.
- Faculté de transmission d'un signal bioélectrique, i.e., influx nerveux, i.e., conductivité.
- Actuellement : Plusieurs champs d'applications.
(Médecine, Psychologie, Informatique)
- Réseaux de neurones artificiels.

Le cerveau

Neurones dans le cortex cérébral

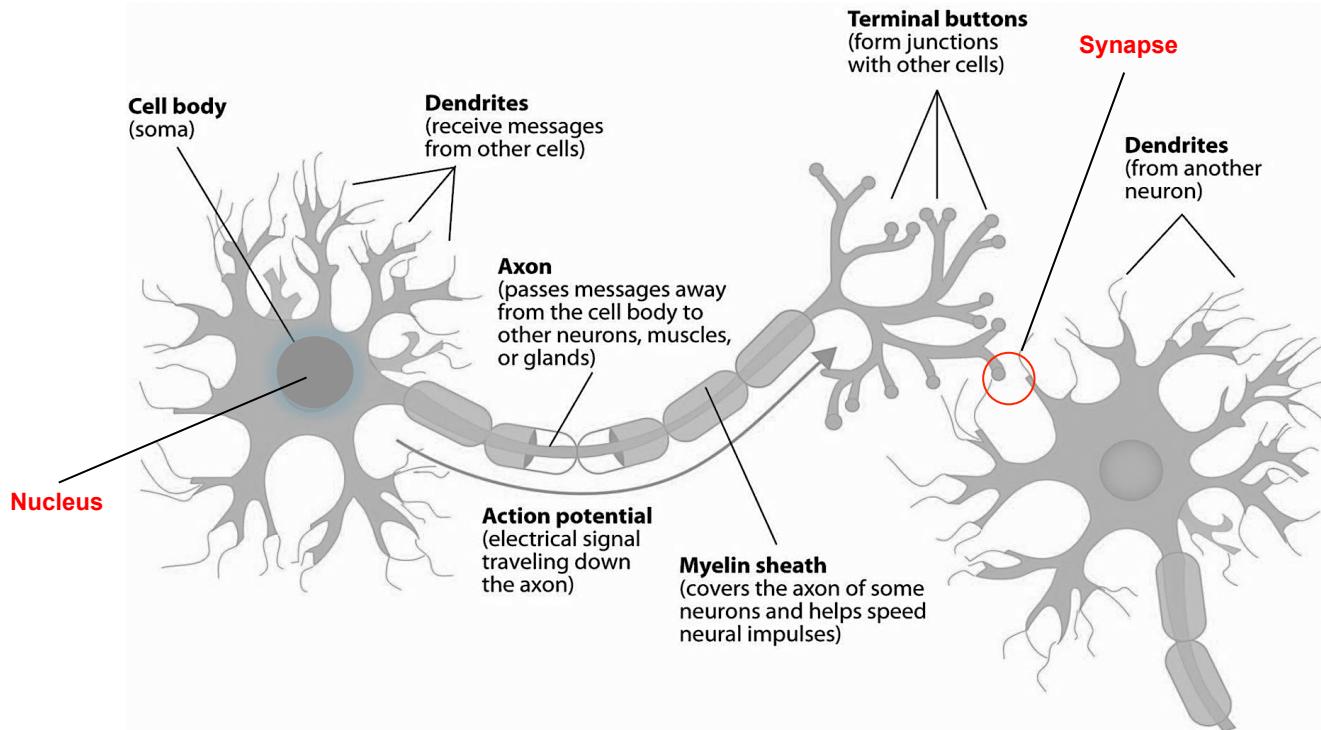


Structuration pyramidale

Le cerveau humain n'exécuterait pas plus de 60 processus en parallèle, Source : Etude publiée par Harris V. Georgiou

Le cerveau

Neurone biologique

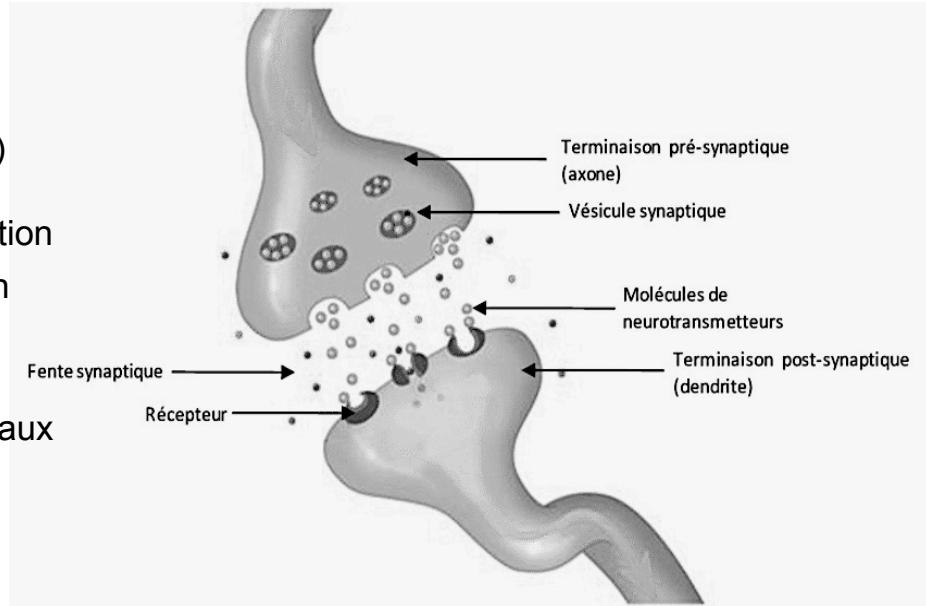


Screenshot from : R. Rojas: Neural Networks,
Springer-Verlag, Berlin, 1996

Le cerveau

La synapse

- Jonction fonctionnelle neurone-neurone ou neurone-cellule (musculaire, sensoriels, etc.)
- **Mécanisme** : Conversion d'un potentiel d'action déclenché dans le neurone présynaptique en un signal dans la cellule postsynaptique.
- Utilisation de neurotransmetteurs ou de signaux électriques pour transmettre l'information.



Méthodes analytiques pour la détection de phénomènes biologiques de sécrétion à l'échelle de la cellule unique.
Anne Meunier - McGill University
Septembre 2011

Le PERCEPTRON

Le neurone formel

- Unité élémentaire des RNA connecté à d'autres neurones pour calculer des fonctions simples / complexes.
- **Historique** : Invention du neurone formel par **McCulloch-Pitts** dans les années 50. Les coefficients synaptiques sont constant (Pas d'apprentissage).
- Règle de Hebb (1949) : ***Cells that fire together, wire together.***
 - Description des variations neuronales pendant un processus d'apprentissage.
 - **Plasticité synaptique** : L'efficacité synaptique augmente lors d'une stimulation présynaptique répétée et persistante de la cellule postsynaptique.

McCulloch, W. S., Pitts, W. :
A Logical Calculus of the Ideas Immanent in Nervous Activity, Bulletin
of Mathematical Biophysics, vol. 5, pp. 115-133, 1943.

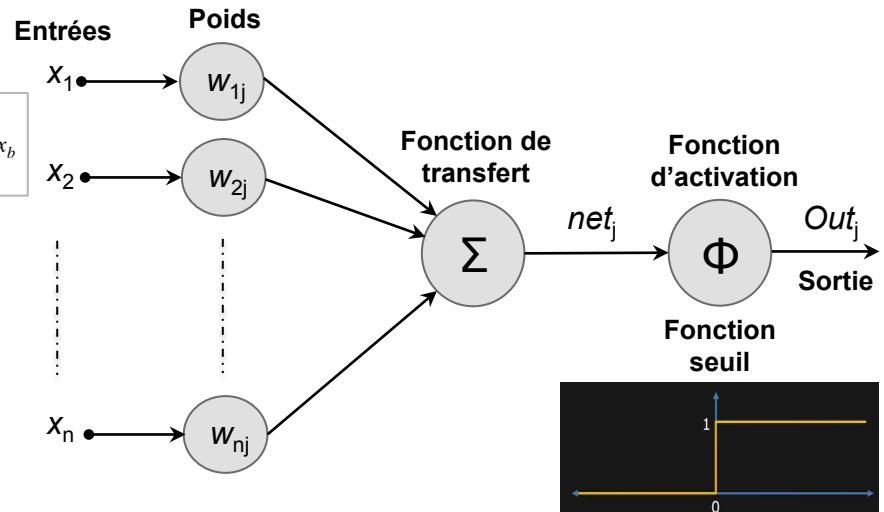
Hebb, D.O. :
The organization of behavior. New York: Wiley & Sons, 1949.

Le PERCEPTRON

Architecture et Formalisme : McCulloch-Pitts

- Représentation d'un neurone j de McCulloch-Pitts à n entrées x_1, x_2, \dots, x_n avec un neurone de polarisation (biais) x_b déclenchant la valeur 1.
- Sortie du neurone j :

$$net_j = \sum_i (w_{j,x_i} \cdot x_i) + w_{j,x_b} \cdot 1 = \sum_i (w_{j,x_i} \cdot x_i) + w_{j,x_b}$$



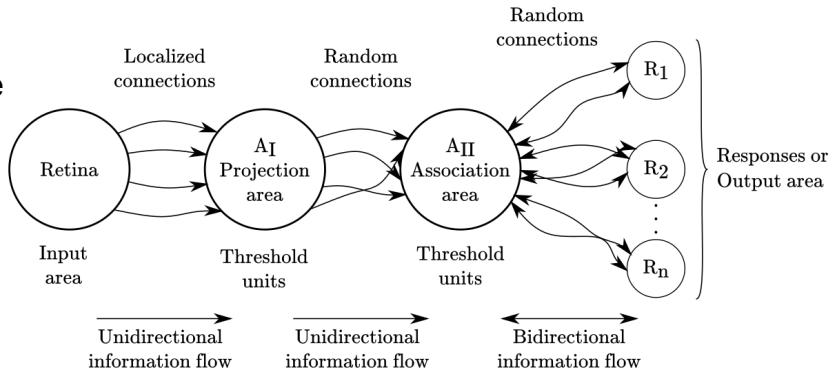
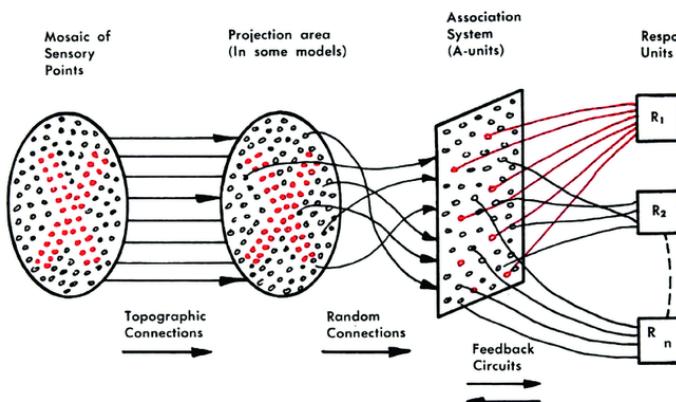
- La fonction d'activation, fonction seuil, appliquée pour le neurone j :

$$\Phi(net_j) = \begin{cases} 0 & net_j < 0 \\ 1 & net_j \geq 0 \end{cases}$$

Le PERCEPTRON

Architecture et Formalisme : Franck RosenBlatt

- Amélioration (1958) : Première machine apprenante
- Prise en compte de l'erreur en sortie.
- 1^{er} modèle informatique des RNA dans la BI.



Ancêtre des réseaux de neurones, le Perceptron associait des cellules photoélectriques et des câbles pour reconnaître des lettres de l'alphabet.

Rosenblatt, F. :

- The Perceptron: A probabilistic model for information storage and organization in the brain, Psychological Review, 65(6), 1958.
- The Perceptron, A perceiving and recognizing automaton, Cornell Aeronautical Laboratory, 1957.

Le PERCEPTRON

Neurone biologique – Perceptron : Mise en correspondance

- Un perceptron peut *apprendre* avec les règles de Hebb/RosenBlatt.
 - Les **synapses** : Poids de pondération des connexions.
 - Le **soma** ou **corps cellulaire** : Fonction de transfert ou **fonction d'activation**.
 - L'**axone** : Sortie du neurone.

Single-layer perceptron can learn only linearly separable patterns

Limites du perceptron monocouche Marvin Minsky & Seymour Papert (1969) :

- Traitement des données complexes et massives
- Modèle d'apprentissage : basique et limité dans ses applications (Classes linéaire).

Minsky, M., Papert, S. :

Perceptrons: An Introduction to Computational Geometry, The MIT Press, Cambridge MA, 1969.

Dans les années 70, mise en veille des investissements dans l'IA et les RNA : **Hiver de l'IA**.

Le PERCEPTRON

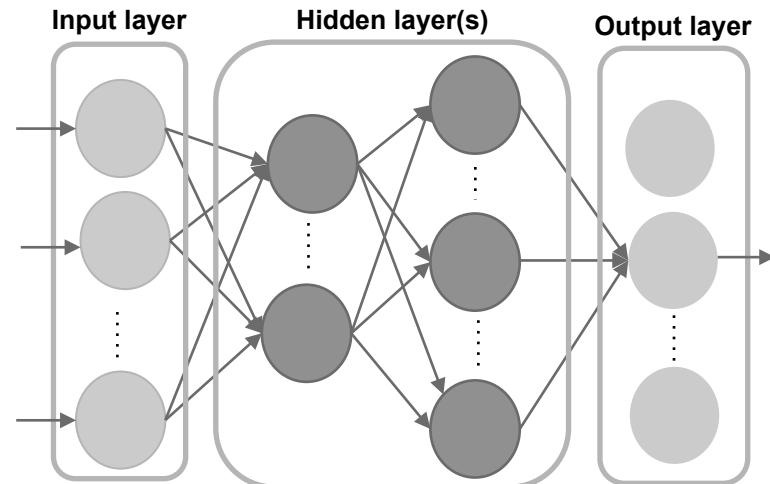
Le MLP

- Un réseau de perceptrons interconnectés entre-eux.
- Classe de RNA prédictifs avec au moins :
 - Une couche d'entrée (Input layer) avec plus d'un neurone en entrée.
 - Une couche de sortie (Output layer) qui peut être constituée de plus d'un neurone.
 - Une ou des couche(s) cachée(s) (Hidden layers) difficilement interprétable.

Forte interconnectivité et éloignement des valeurs d'entrée ou de sortie connues.
- Technique d'apprentissage supervisé.
- Rétropropagation des erreurs (années 80).

Rumelhart, D.E., Hinton, G.E., Williams, R.J. :
Learning internal representations by error propagation, 1986, pp. 318–362 in
Parallel Distributed Processing: Explorations in the Microstructures of
Cognition, eds., MIT Press, Cambridge, MA

- Fonctions d'activation : Non linéaires sur tout les nœuds (sauf l'entrée).
- Traitement des data non séparables linéairement.

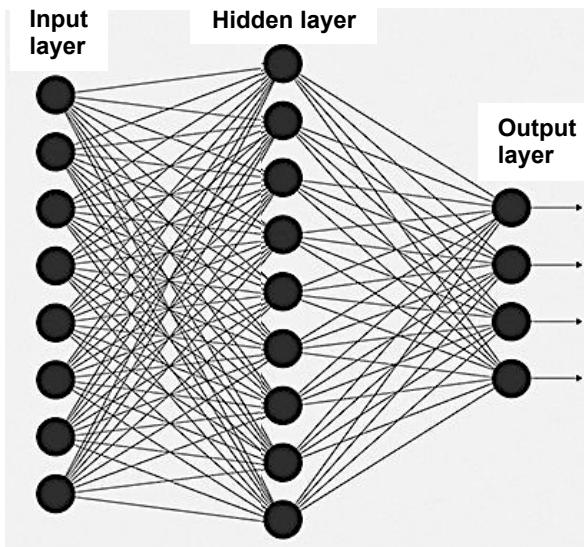


On parle de réseaux profonds (Deep NN), s'il existe au moins 2 couches cachées.

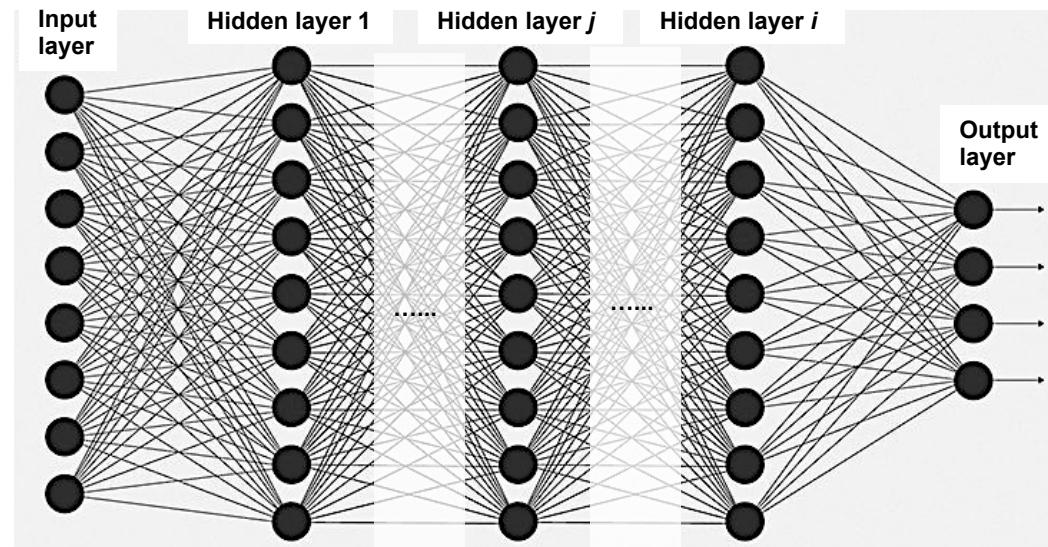
Le PERCEPTRON

MLP : Non-deep or Deep NN

Non-deep feedforward Neural Network



Deep Neural Network

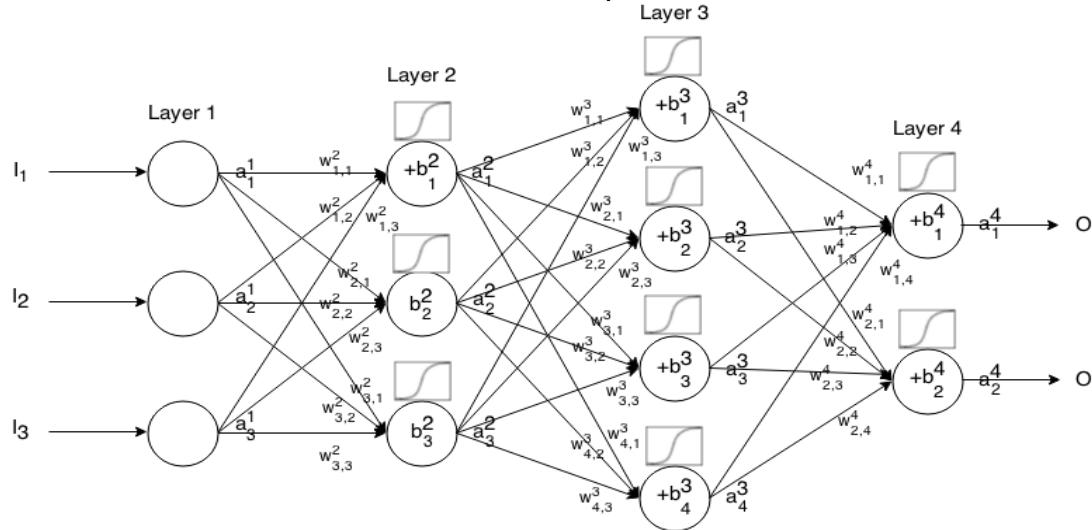


Le Deep learning

- Banche du Machine Learning (1950)
 - Implémentation d'algorithmes d'apprentissage capables d'imiter les actions du cerveau humain.
 - Le RN est une machine capable d'apprendre à partir de ses traitements internes des données non structurées (son, texte, image, etc.).
(Dans le ML, les data sont quantitatives et structurées)
- Dès les années 90
 - Utiliser des techniques statistiques à grande échelle pour déceler au sein d'un corpus de data, de l'information cachée sans utiliser de règles explicitement définies par un opérateur humain (Systèmes experts).
- Très populaire en 2010
 - Reconnaissance d'images et compréhension, traitement du langage parlé, finance et marketing, santé, réseaux sociaux, Youtube, etc.

Le Deep learning

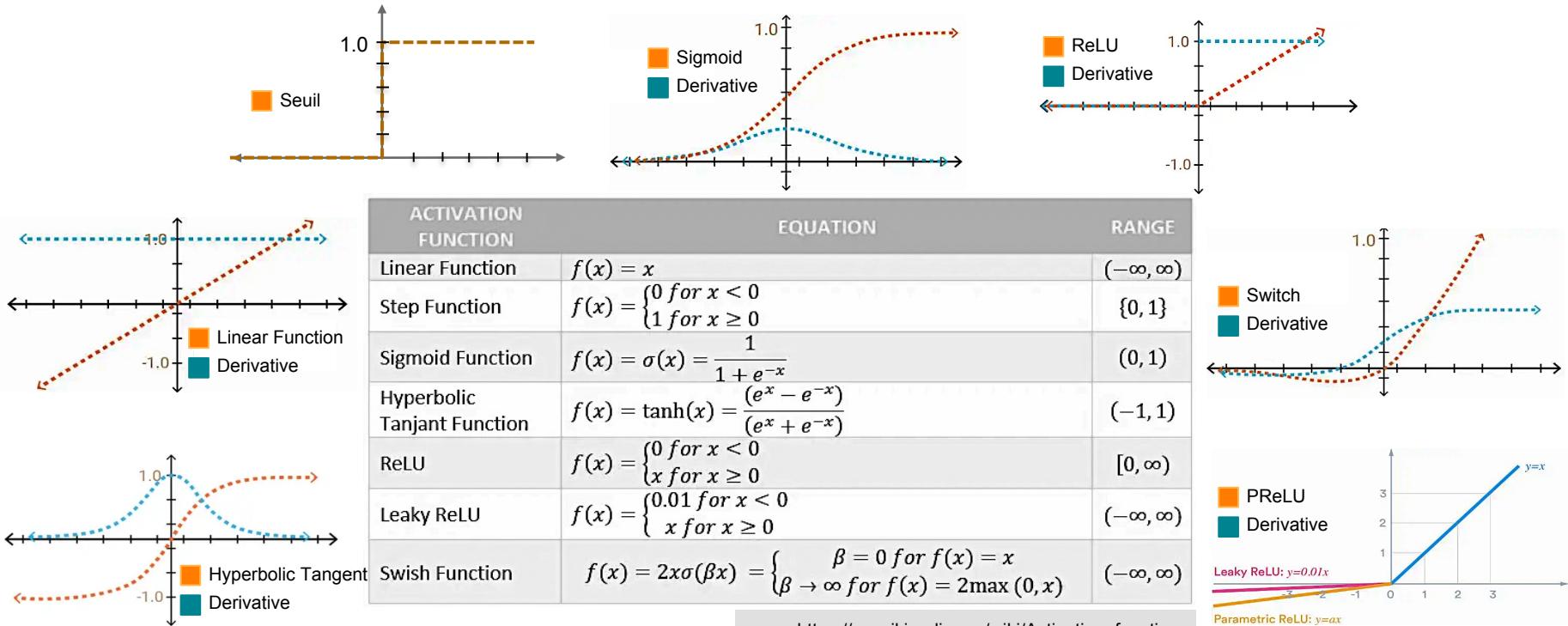
- Le RN dispose de **plusieurs couches** : entrée, sortie et cachées avec chacune des unités de traitement des données d'entrée utiles pour les couches suivantes.



*Exemple d'un réseau très simple avec paramètres étiquetés.
Mais beaucoup de calculs! Comment résoudre ce problème ?*

Le Deep learning

Fonctions d'activation : Quelques unes !



Le Deep learning

Fonctions d'activation : Quelques explications !

Fonction Seuil

Très peu utilisé en pratique (que pour expliquer des concepts théoriques). Destinée pour les RN simples, elle produit 0 ou 1. Utile pour la classification mais ne considère pas les petites variations.

Fonction linéaire

Seule fonction linéaire, elle est utilisée que dans la couche de sortie pour des tâches simples comme l'interprétabilité, et résoudre les problèmes de régression.

Fonction Sigmoïde

Très utile dans le domaine de la classification, et est plus sensible aux petits changements que la fonction Seuil. Elle peut causer des blocages au niveau de l'apprentissage.

Fonction Tangente Hyperbolique

Très similaire à la fonction sigmoïde mais définie dans $[-1, +1]$. Elle a l'avantage que sa dérivée est plus raide, impliquant plus de valeurs donc une plage plus large pour un apprentissage et une notation plus rapides. Mais encore une fois, le problème des gradients aux extrémités de la fonction persiste.

Fonction ReLU

Fonction d'activation la plus utilisée dans presque tous les RN Convolutifs ou l'apprentissage profond. Les valeurs négatives diminuent la capacité du modèle à s'adapter ou à s'entraîner correctement. Très performante, particulièrement dans le Vanishing gradient. La diminution très rapide des valeurs des gradients lors de la Rétropropagation entraîne l'annulation du gradient et l'arrêt de l'apprentissage.

Fonction Switch

Proposée par Google Brain, elle a tendance à fonctionner mieux que ReLU sur des modèles plus profonds à travers un certain nombre dataset complexes. Elle est différentiable en tout point contrairement à la fonction ReLU.

Fonction Leaky ReLU

Fonction d'activation Relu modifiée avec les mêmes applications que ReLU. Très performante, particulièrement dans le Vanishing gradient.

Le processus d'apprentissage est plus rapide que dans ReLU.

https://en.wikipedia.org/wiki/Activation_function
https://fr.wikipedia.org/wiki/Fonction_d%27activation

Le Deep learning

Contexte Multi-Classe

Remarque :

- Toutes les fonctions d'activation citées ci-dessus, n'ont de sens que pour une seule sortie, que ce soit une étiquette (label) continue ou une prédiction d'une classification binaire (soit 0 ou 1).

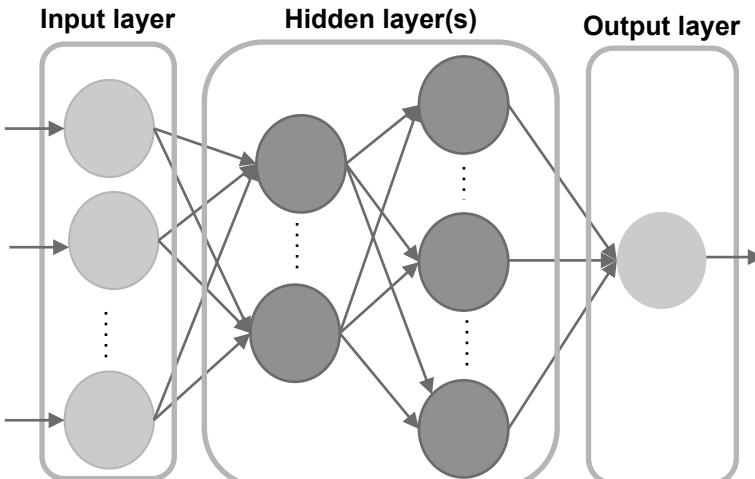
Types de familles classes multiples :

- Classes non-exclusives : un point de données peut se voir attribuer plusieurs classes/catégories. Les images peuvent avoir plusieurs tags (plage, famille, vacances, etc...).
- Classes mutuellement exclusives : une seule classe par point de données. Les images peuvent être classées par niveaux de gris (noir et blanc) ou en couleur. Elle peut pas être les deux à la fois.

Le Deep learning

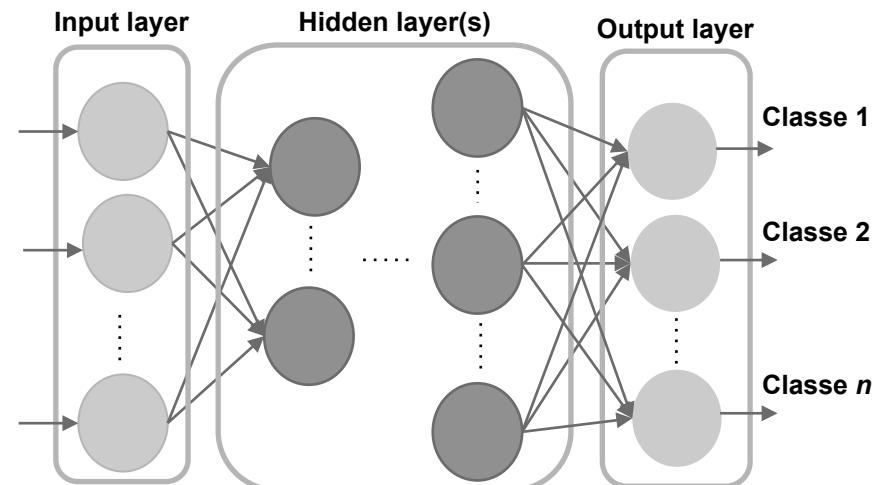
Contexte Multi-Classe : Organisation

Cas d'un RN mono-classe



Un nœud unique qui produit une seule valeur de régression continue ou une classification binaire (0 ou 1).

Cas d'un RN multi-classe

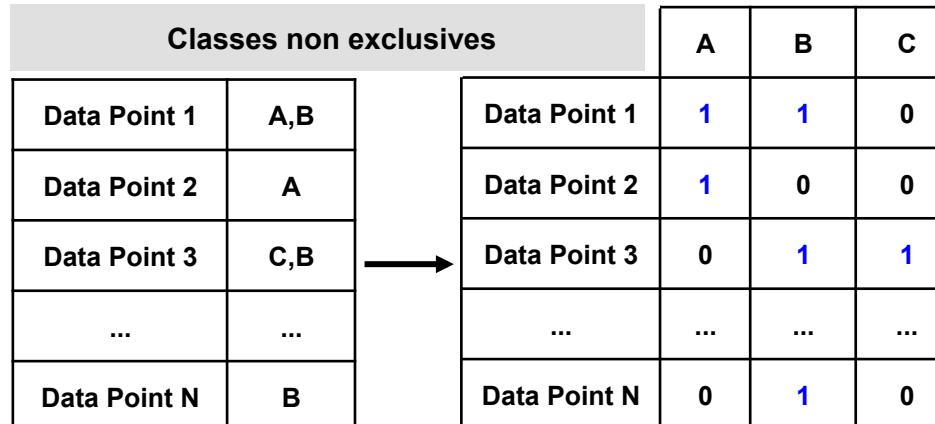


Organisation neuronale en plusieurs classes.
Un nœud de sortie par classe.
(One hot Encoding)

Le Deep learning

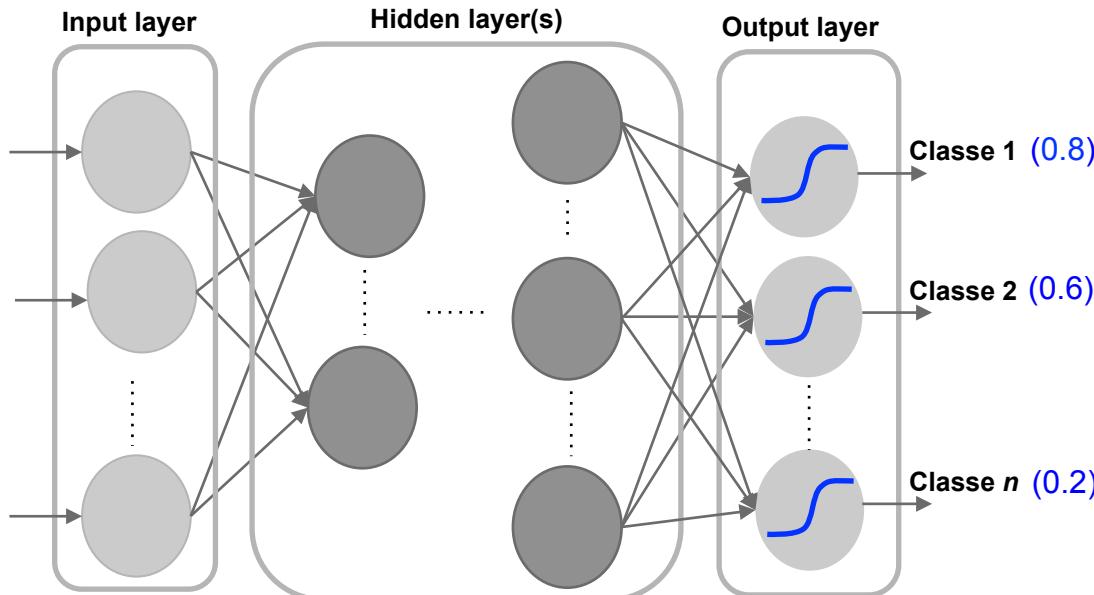
Contexte Multi-Classe : Cas de classes non exclusives

- Fonction d'activation de classification pour la couche de sortie : Sigmoïde
- Chaque neurone calcule une valeur en sortie (0 ou 1) correspondant à la probabilité de se voir attribuer une classe.



Le Deep learning

Contexte Multi-Classe : Cas de classes non exclusives



Chaque neurone peut produire une sortie indépendamment des autres classes, ce qui permet à un point de données dans la fonction de se voir attribuer plusieurs classes, avec une probabilité d'appartenance plus importante pour la classe 1.

Le Deep learning

Contexte Multi-Classe : Cas de classes mutuellement exclusives

- Fonction d'activation de classification pour la couche de sortie : **Softmax**
- Elle peut être utilisée en machine learning pour convertir un score en probabilité.
- La fonction **Softmax** calcule la distribution des probabilités de l'événement sur K événements différents. Elle détermine les probabilités de chaque classe cible parmi toutes les autres classes cibles.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad \text{for } i = 1, \dots, K$$

- L'intervalle sera de **0** à **1**, et **la somme de toutes les probabilités sera égale à un**.
- Le modèle retourne les probabilités de chaque classe et la classe cible choisie aura la probabilité la plus élevée.

Le Deep learning

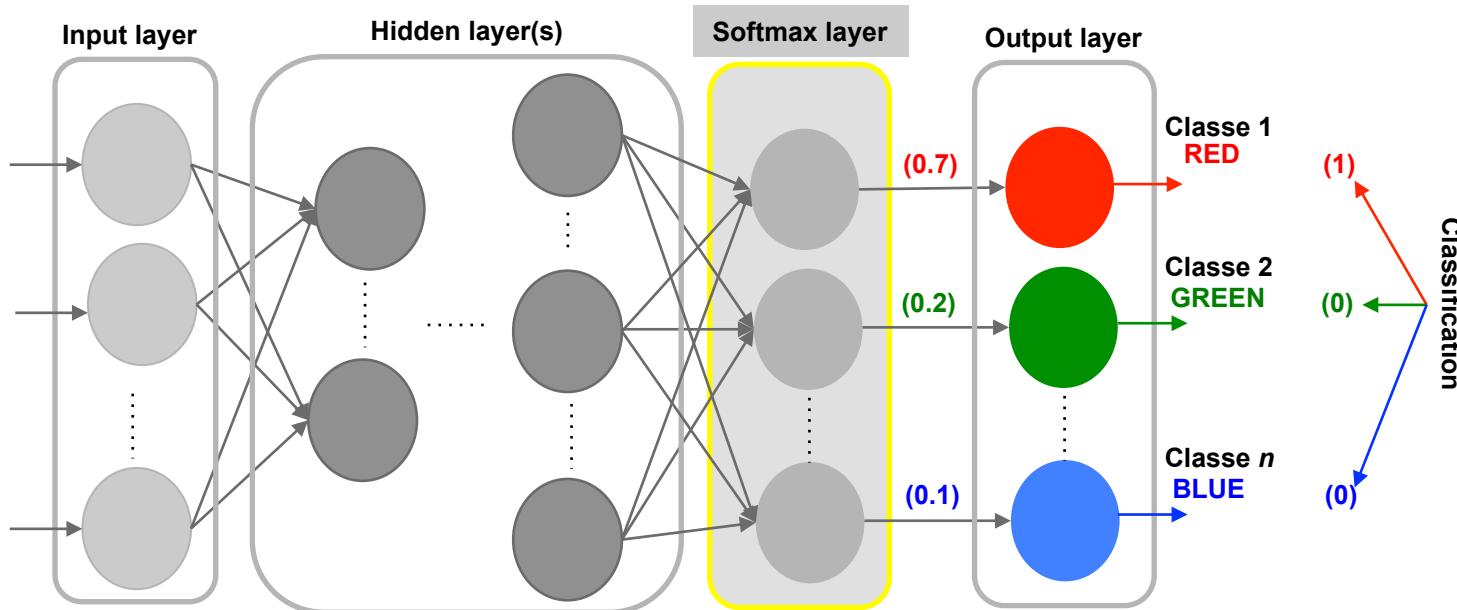
Contexte Multi-Classe : Cas de classes mutuellement exclusives

Classes mutuellement exclusives		RED	GREEN	BLUE
Data Point 1	RED	1	0	0
Data Point 2	GREEN	0	1	0
Data Point 3	BLUE	0	0	1
...
Data Point N	RED	1	0	0

Chaque point de données ne peut se voir attribuer qu'une seule classe.

Le Deep learning

Contexte Multi-Classe : Cas de classes mutuellement exclusives



Softmax est implémenté via une couche de réseau de neurones juste avant la couche de sortie. La couche Softmax doit comporter le même nombre de nœuds que la couche de sortie.

Performances d'un Réseau de neurones

La fonction de perte, i.e., Loss

- Comment évaluer les performances du réseaux de neurones ?
 - Prédiction, précision de classification par rapport aux labels/dataset ...
 - Robustesse par rapport aux changements de ses paramètres intrinsèques.
 - Coûts engendrés en termes de consommation de ressources.
 - Amélioration des performances possibles.
 - Mécanismes d'optimisation paramétrique ...
- Analyser l'évolution de la **fonction de coût** ou fonction de perte, i.e., **loss**, pendant l'entraînement, i.e., **apprentissage**, du réseau.

Performances d'un Réseau de neurones

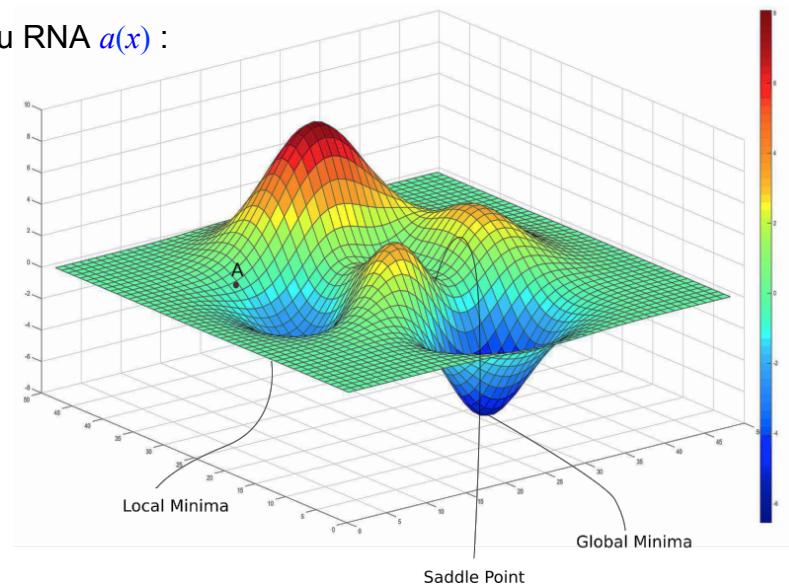
La fonction de perte, i.e., Loss

– Fonction coût quadratique

- Ecart entre les valeurs réelles $y(x)$ et les prédictions du RNA $a(x)$:

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

- Quelles sont les différents valeurs des paramètres du RNA qui minimisent la fonction coût C ?



Source : <https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/>

Performances d'un Réseau de neurones

La fonction de perte, i.e., Loss

Classification

- Entropie croisée binaire - *Binary Cross-entropy*

Cas particulier de la cross-entropy, elle renvoie une mesure de la dissimilarité entre deux probabilités quantifiant leur différence. Au maximum, deux classes sont considérées.

- Entropie croisée Catégorielle - *Categorical Cross-entropy*

Elle se base sur la Binary Cross-Entropy, mais traite des problématiques multi-classes.

- Perte SVM multi-classes – *Hinge Loss*

Alternative à la fonction de perte d'entropie croisée qui a été principalement développée pour l'évaluation du modèle de machine à vecteurs de support (SVM).

Régression

- Mean Square Error / Quadratic Loss / L2 Loss

- Mean Absolute Error / L1 Loss

- Huber Loss / Smooth Mean Absolute Error

- Log-Cosh Loss

- Quantile Loss

Apprentissage du réseau

- Concept :
 - Adaptation des paramètres du réseau, i.e., (*poids* , *biais*), pour améliorer la précision du résultat.
 - Minimisation des **erreurs d'observation** avec un taux d'erreur proche de 0.
 - **Fonction de coût** évaluée périodiquement au cours de l'apprentissage. Tant que sa production continue de baisser, l'apprentissage continue.
 - L'**apprentissage** tente de réduire le total des différences entre les observations.

Apprentissage du réseau

Les paradigmes

- Apprentissage supervisé
 - Utilisation de Dataset préalables désirés pour guider les sorties d'un modèle.
 - Adaptation des paramètres par optimisation d'un coût, i.e., EQM*.
 - **Classification** (Reconnaissance de formes).
 - **Régression** (Approximation des fonctions quelconques).
- Apprentissage non supervisé
 - Utilisation de données non étiquetées pour déduire des regroupements.
 - Regroupement, estimation, compression et filtrage.
- Apprentissage par renforcement
 - Concept d'agent autonome.
 - Politique optimale d'association à l'état courant une action à exécuter.
 - Apprentissage des actions à partir d'un retour d'expériences, en optimisant une récompense (+ ou -) dans le temps, procurée par l'environnement.
 - Routage des véhicules, gestion des ressources naturelles, santé.

*EQM Erreur quadratique moyenne

Apprentissage du réseau

Le mécanisme en quelques lignes

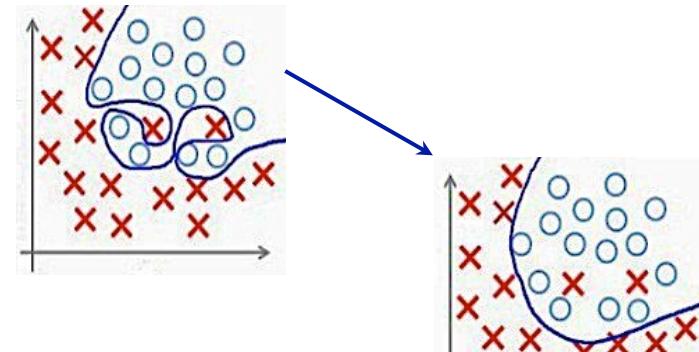
- Les différentes étapes :
 - Présentation des données, i.e., Dataset au processus d'apprentissage et extraction de caractéristiques.
 - Phase d'entraînement du modèle.
 - Phase de test et de déploiement du modèle.
 - Prédiction : phase de mise en œuvre opérationnel en ligne du modèle par optimisation d'une fonction coût, i.e., *perte*.

Apprentissage du réseau

Overfitting vs. Underfitting

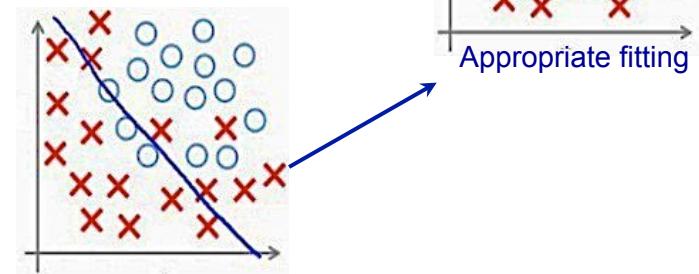
Overfitting Sur-apprentissage

- Modélisation parfaite des données d'entraînement.
- Le modèle d'apprentissage intègre aussi des bruits (Impact sur la généralisation).
- Modèles non paramétriques et non linéaires (Arbres de décision).
- Solution : Technique de ré-échantillonnage pour estimer la précision du modèle : **k-fold cross validation**.



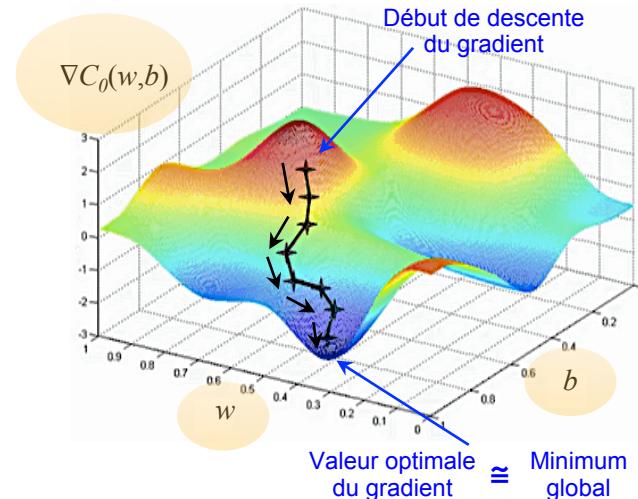
Underfitting Sous-apprentissage

- Le modèle ne peut ni modéliser les données d'apprentissage ni généraliser sur de nouvelles données.
- Un modèle sous-entraîné a de mauvaises performances sur les données entraînées.



Descente de gradient stochastique

- Algorithme adaptatif destiné pour entraîner la plupart des modèles type Machine Learning et des RNA.
- L'idée :
 - Réduction des erreurs dans $C()$ en utilisant les données d'apprentissage.
 - Optimisation du modèle en augmentant sa précision via la mise à jour de ses paramètres.
 - Localisation du minimum global.
- Avantages :
 - Efficacité des calculs et rapidité.
 - Parallélisme des calculs (simultanéité).

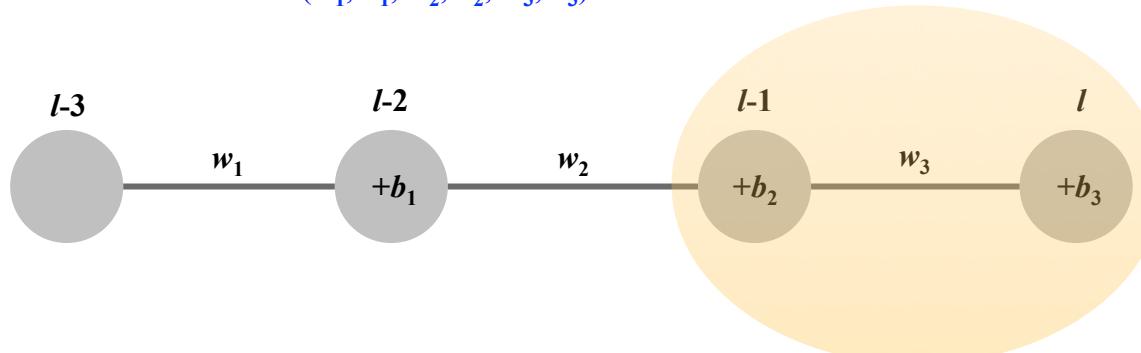


Descente de gradient stochastique

Application sur un Réseau de neurones simple

- Développement par l'exemple

- Soit un RNA dans lequel chaque couche, i.e., Layer, $l-i$ ($i=1..3$) possède un seul neurone.
- Chaque entrée i disposera d'un couple de valeurs poids et d'un biais (w_i, b_i) .
- La fonction coût C est : $C(w_1, b_1, w_2, b_2, w_3, b_3)$



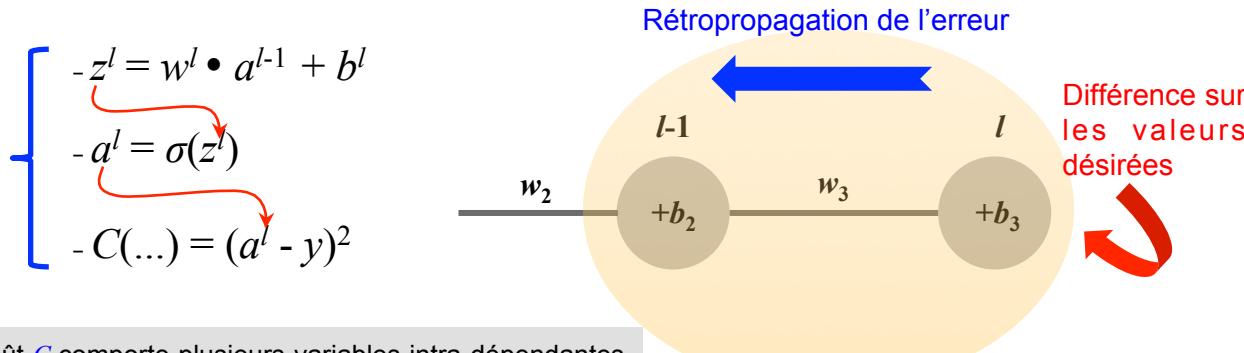
- Soit $z = w \cdot x + b$.
- Appliquons une fonction d'activation : $a = \sigma(z)$.
- Considérons les deux dernières couches l et $l-1$.

$$\left. \begin{array}{l} - z^l = w^l \cdot a^{l-1} + b^l \\ - a^l = \sigma(z^l) \\ - C(\dots) = (a^l - y)^2 \end{array} \right\} \rightarrow$$

Descente de gradient stochastique

Calcul du Gradient

- Evolution de la fonction coût $C()$ / variation de ses paramètres (w_i, b_i)
 - Dérivation en chaîne :



La fonction coût C comporte plusieurs variables intra-dépendantes ce qui explique le passage à la dérivation partielle.

- Utilisation du GRADIENT, soit : $\nabla C(w, b)$
pour ajuster les paramètres, i.e., *poids et biais*, et minimiser la sortie du vecteur d'erreur sur la dernière couche de sortie.

Descente de gradient stochastique

Aspect mathématique

- Formulation mathématique :

$$z^l = w^l \bullet a^{l-1} + b^l \quad \longleftarrow \text{Traitements d'un neurone couche } l.$$

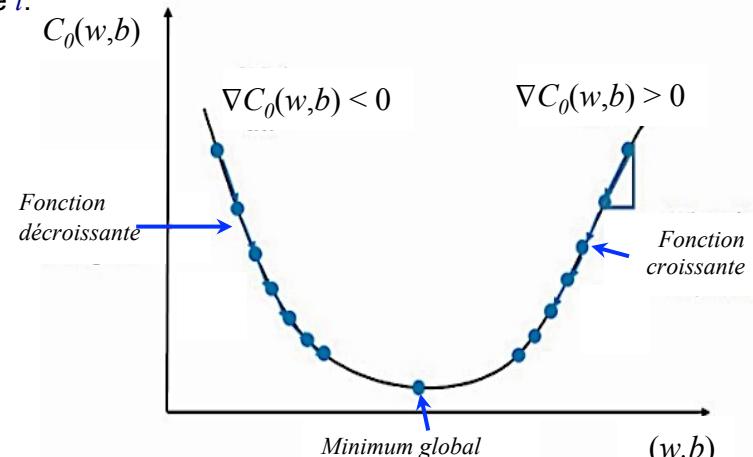
$$a^l = \sigma(z^l) \quad \longleftarrow \text{Fonction d'activation alimentant la couche suivante.}$$

$$C_0(w, b) = (a^l - y)^2$$

- Calcul du gradient en chaîne :

$$\nabla C_0(w, b) = \left(\frac{\partial C_0}{\partial w}, \frac{\partial C_0}{\partial b} \right) \quad \left\{ \begin{array}{l} \frac{\partial C_0}{\partial w^L} = \frac{\partial C_0}{\partial a^L} \frac{\partial a^L}{\partial z^L} \frac{\partial z^L}{\partial w^L} \\ \frac{\partial C_0}{\partial b^L} = \frac{\partial C_0}{\partial a^L} \frac{\partial a^L}{\partial z^L} \frac{\partial z^L}{\partial b^L} \end{array} \right.$$

Le calcul du gradient se fait à chaque itération, i.e., pas d'apprentissage, taux d'apprentissage ou learning rate η .



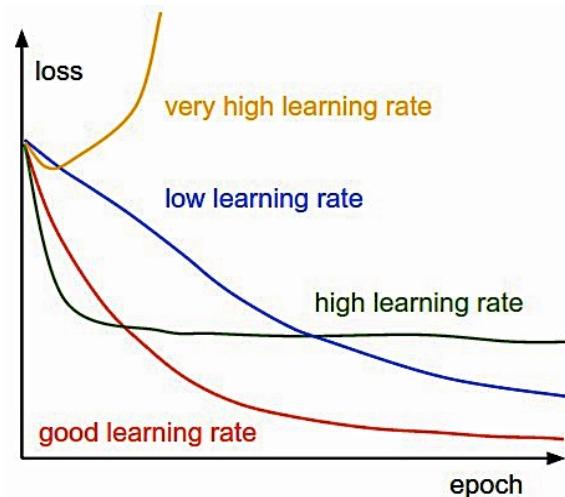
<https://cs231n.github.io/neural-networks-3/>

Descente de gradient stochastique

Convergence – pas d'adaptation – Conditions Initiales CI

- Dépendance : Convergence – pas d'adaptation – CI.
 - Avec de petits pas d'adaptation, lenteur de la convergence vers le minimum global.
 - Des pas d'adaptation plus grands, rapidité de convergence mais risque de divergences.
 - Les CI peuvent influer aussi sur le temps de convergence.
- Comment trouver la bonne valeur pour votre hyperparamètre : Learning rate ?

Malheureusement pas de formule magique !!
Du tâtonnement ...



<https://cs231n.github.io/neural-networks-3/>

Descente de gradient stochastique

Hyper-paramètre : Learning rate

- Méthode **descente adaptative du gradient** : Commencer par des pas d'apprentissage très grand puis diminuer à proximité du minimum.
- Autres méthodes implémentées dans [**Keras**](#) par exemple, comme :
 - Méthode [**Adagrad**](#) Adaptive Gradient Algorithm .
 - Méthode [**Adadelta**](#) Version delta de [**Adagrad**](#).
 - Méthode d'[**Adam**](#) ADaptive Moment estimation.
 - Méthode [**RMSProp**](#) Root Mean Square Propagation .
 - ...

On se limitera à en citer quelques unes

Descente de gradient stochastique

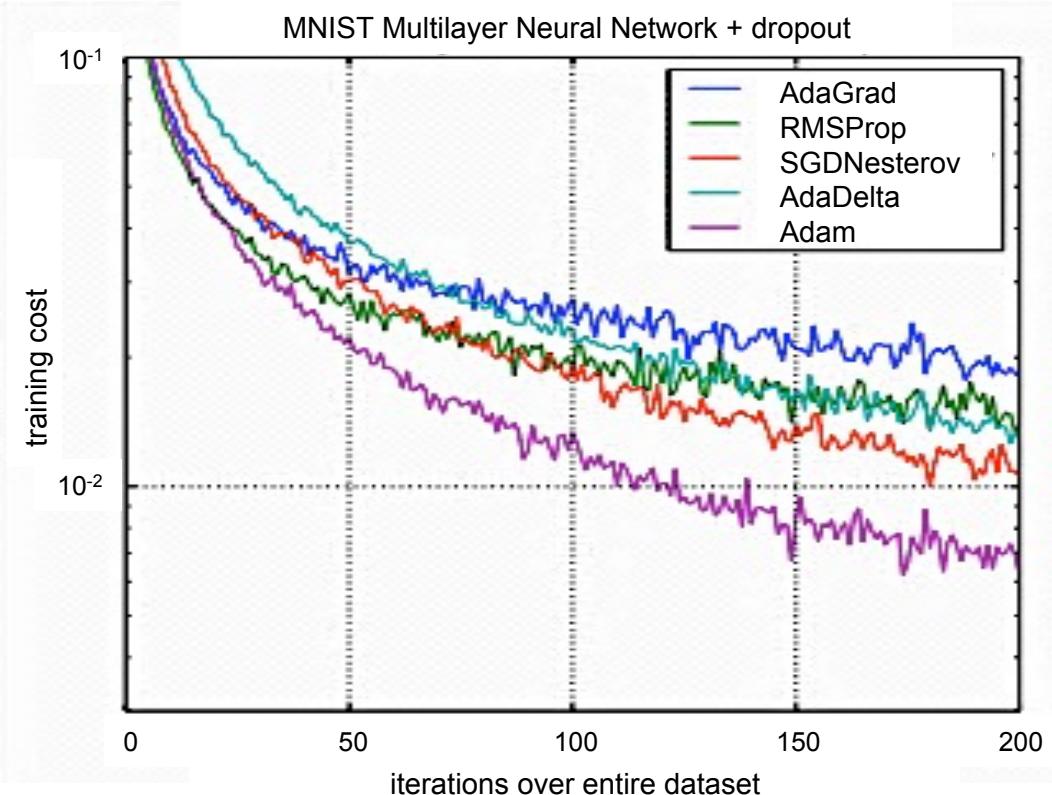
Comparatif des différentes variantes DGS

Adam vs. RMSprop, AdaGrad, Adadelta et SGDNesterov Momentum

RMSprop, Adadelta et Adam sont très similaires. Cependant, En pratique, Adam est le plus recommandé par défaut. Il est au-dessus, à la fin de la phase d'optimisation quand les gradients sont éparses. Il est aussi intéressant d'essayer SGDNesterov Momentum comme alternative.

Adam est simple à mettre en œuvre.
Calcul efficace avec peu de mémoire.
Adapté pour des corpus volumineux et bruités.
Peu de réglages pour les hyper-paramètres.

<https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>



La Rétropropagation

- Algorithme introduit dès les années 1970 mis en avant en 1986.
([D. Rumelhart, G. Hinton et R. Williams](#)).
- Idée :
 - Amélioration de l'efficacité de l'algorithme comparativement aux approches d'apprentissage antérieures.
- Aujourd'hui :
 - Cheval de bataille de l'apprentissage dans les RNA.
 - Calcul des gradients (dérivées partielles) de la fonction de coût.
 - Plusieurs méthodes d'optimisation.
- Les plus :
 - Rapidité de calcul de la modification des coûts / variation des paramètres des RNA.
 - Informations détaillées et interprétation naturelle et intuitive des données du réseau (rapidité et détails).

La Rétropropagation

Etapes de mise en œuvre

- Notations :

w_{jk}^l : Poids de la liaison neurone k de la couche ($l-1$) au neurone j de la couche (l).

b_j^l : Biais du neurone j de la couche (l).

a_j^l : FA du neurone j de la couche (l) en relation avec la FA de la couche ($l-1$).

- Etape 1 : Calculer la sortie intermédiaire.

$$a_j^l = \sigma \left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \right)$$

(Sous forme matricielle)

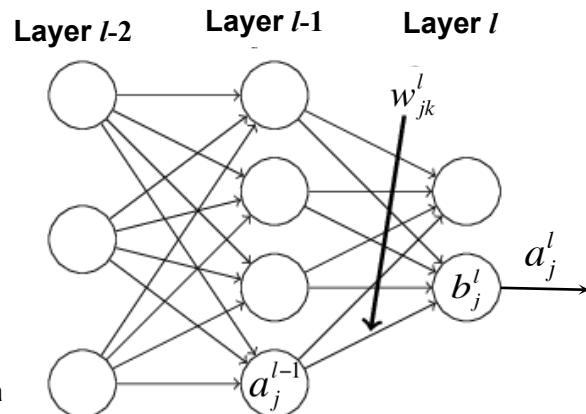
FA ($l-1$) vers la couche suivante (l).

Somme sur tous les neurone k dans la couche ($l-1$).

$$a^l = \sigma(w^l a^{l-1} + b^l)$$

$$a^l = \sigma(z^l) \rightarrow z^l = w^l \cdot a^{l-1} + b^l$$

Entrée pondérée de la FA pour le neurone j dans la couche l .



La Rétropropagation

Etapes de mise en œuvre

- Etape 2 : Calculer la fonction coût.

$$C(w, b) = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

Déterminer les paramètres (w, b) afin que la sortie du réseau puisse approximer la sortie désirée du modèle $y(x)$ par rapport à l'ensemble des entrées d'apprentissage x .

n étant le nombre total d'exemples d'apprentissage.
 L désigne le nombre de couches dans le réseau.

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

Une autre réécriture de la fonction coût quadratique pour un seul exemple d'apprentissage x .

- Etape 3 : Dérivées partielles en chaîne dans une couche l .

$$\nabla C(w_{jk}^l, b_j^l) = \left(\frac{\partial C}{\partial w_{jk}^l}, \frac{\partial C}{\partial b_j^l} \right) \rightarrow \left\{ \begin{array}{l} \frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{jk}^l} \\ \frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial b_j^l} \end{array} \right. \quad (1)$$

La Rétropropagation

Etapes de mise en œuvre

- *Introduction d'une valeur intermédiaire, i.e., erreur en sortie δ^L .*

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l} = \frac{\partial C}{\partial a_j^l} \cdot \sigma'(z_j^l) \longrightarrow \text{Erreurs dans le } j^{\text{ème}} \text{ neurone de la couche } l.$$

Écriture plus conventionnelle :

$$\delta^L \equiv \frac{\partial C}{\partial z^L} = \frac{\partial C}{\partial a^L} \cdot \sigma'(z^L) \quad (2)$$

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

$$\text{avec : } \nabla_a C(w, b) = (a^L - y)$$

Produit de Hadamard, i.e., Schur, élément/élément sur 2 vecteurs de même dimension. Il existe des bibliothèques qui fournissent ce calcul matriciel.

Expression vectorielle très facile à calculer avec une bibliothèque comme **Numpy** par exemple.

- **Etape 4** : Calculer l'erreur généralisée δ^l sur toute la couche l (Rétropropagation de l'erreur).

$$\delta^l = (w^{l+1})^T \delta^{l+1} \odot \sigma'(z^l) \quad (3)$$

Connaissant l'erreur δ^{l+1} , et en appliquant la matrice de pondération transposée, $(w^{l+1})^T$, de la couche $(l+1)$, intuitivement cela revient à rétropropager l'erreur à travers une FA en mesurant l'erreur à la sortie de la $j^{\text{ème}}$ couche.

(2) et (3) calculent l'erreur δ^l pour n'importe quelle couche du réseau. On applique (2) pour calculer δ^L puis (3) pour calculer δ^{L-1} puis à nouveau (2) pour calculer δ^{L-2} , et ainsi de suite, tout au long du réseau.

La Rétropropagation

Etapes de mise en œuvre

- Etape 5 : Calculer le taux de variation de la fonction coût / paramètres de pondération, poids et biais.

A partir de (1), (2) et (3), nous avons :

$$\left\{ \begin{array}{l} \frac{\partial C}{\partial w_{jk}^l} = a_j^{l-1} \delta_j^l \rightarrow \text{Plus la FA du neurone } j \text{ de la couche } (l-1) \text{ est petit, plus la variation du gradient / poids de pondération est petit. On parle d'apprentissage lent, i.e., les poids émis par les neurones à faible activation apprennent lentement.} \\ \frac{\partial C}{\partial b_j^l} = \delta_j^l \rightarrow \text{Le taux de variation du coût par rapport à tout biais dans le réseau correspondrait à l'erreur } \delta_j^l. \end{array} \right.$$

- Mise à jours :

$$(w_{jk}^l, b_j^l)^{(i+1)} = (w_{jk}^l, b_j^l)^{(i)} - \eta \cdot \nabla C(w_{jk}^l, b_j^l)^{(i)}$$

Itération.

η étant le pas d'adaptation.

Gradient.

La Rétropropagation

Hyper-paramètre : Learning rate

- Comment trouver la bonne valeur pour votre hyper-paramètre : Learning rate ?
 - Méthode du momentum pour accélérer la convergence. La mise à jours s'effectue :

$$(w_{jk}^l, b_j^l)^{(i+1)} = (w_{jk}^l, b_j^l)^{(i)} - \eta \cdot \nabla C(w_{jk}^l, b_j^l)^{(i)} + \alpha \cdot \eta \cdot \nabla C(w_{jk}^l, b_j^l)^{(i-1)}$$

η learning rate α momentum ≈ 0.9

- Méthode Time-based learning qui met à jours le pas d'apprentissage η / itérations i :

$$\eta^{(i+1)} = \frac{\eta^{(i)}}{1 + d \cdot i}$$

d coefficient de régularisation < 0.01

- Méthode exponentielle :

$$\eta^{(i+1)} = \eta^{(0)} \cdot e^{-d \cdot i}$$

$\eta^{(0)}$ pas d'apprentissage initial

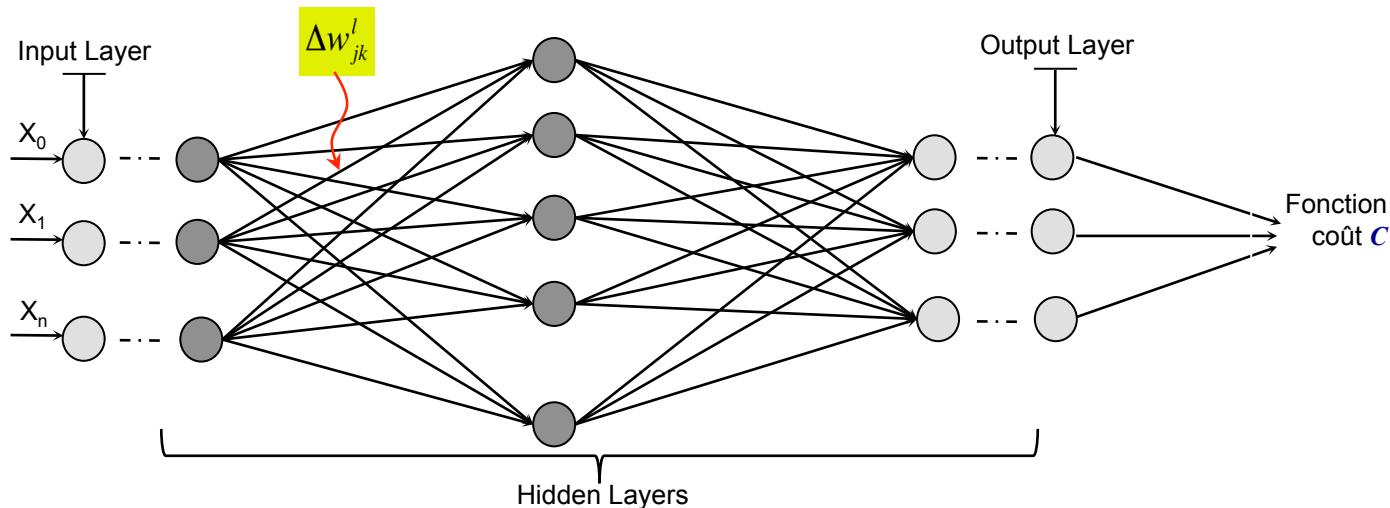
On se limitera à en citer quelques unes

La Rétropropagation

Par l'image

M. Nielsen, *Neural Networks and Deep Learning: Introduction to the core principles*, 2019.

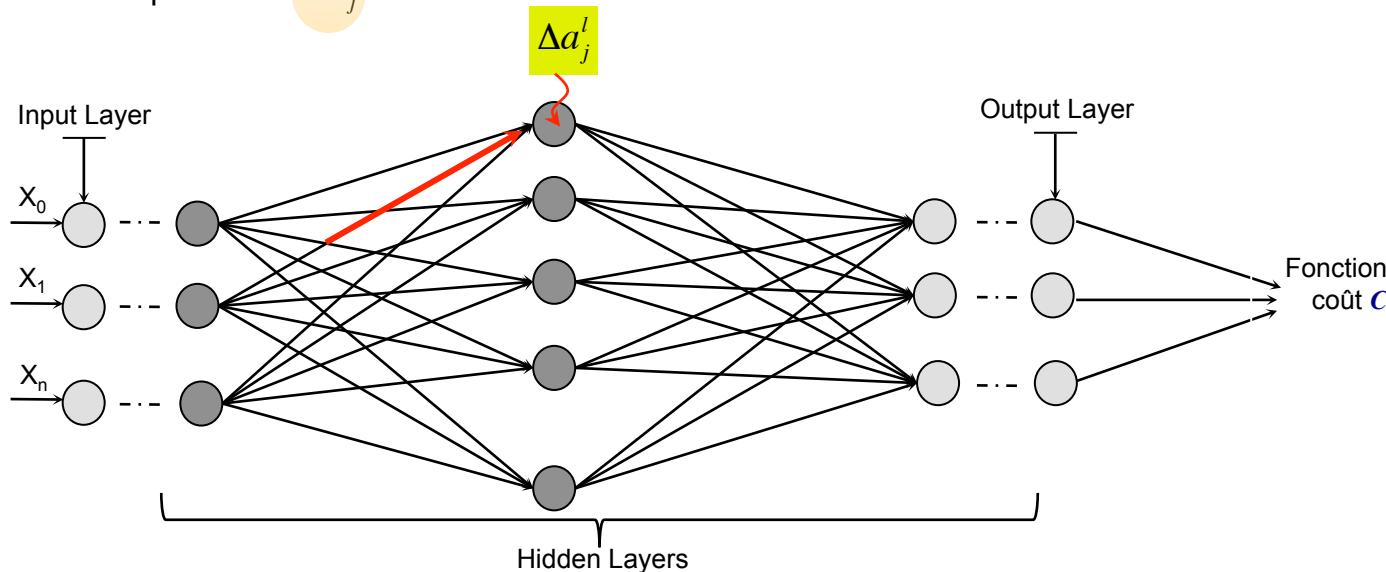
- Considérons une petite modification du poids w_{jk}^l : Δw_{jk}^l



La Rétropropagation

Par l'image

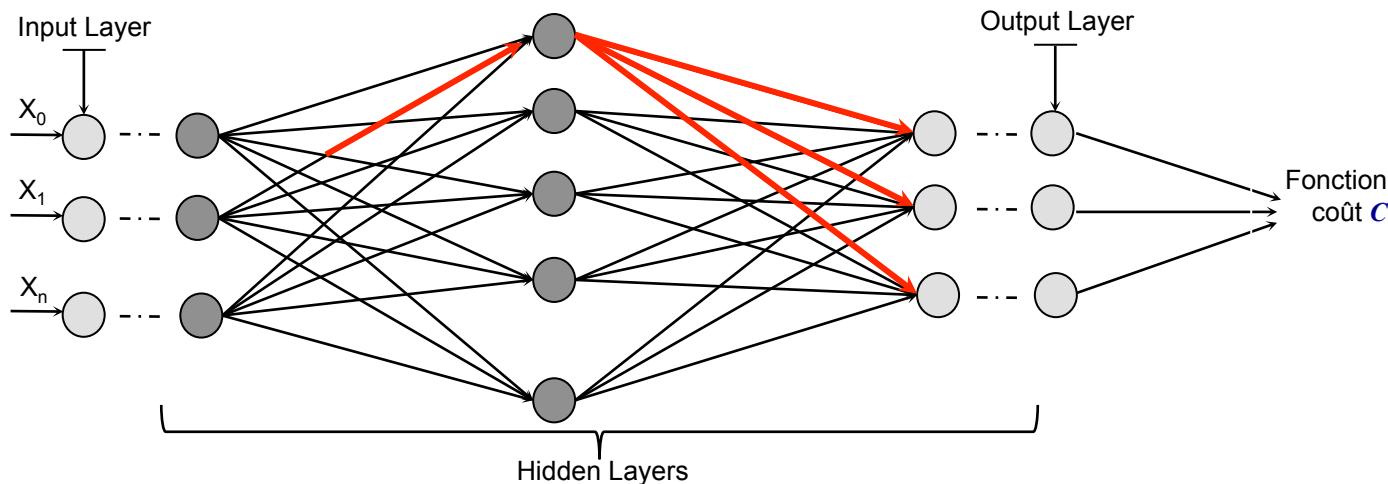
- Une variation dans l'activation de la sortie du neurone correspondant : Δa_j^l



La Rétropropagation

Par l'image

- Des changements qui entraînent d'autres modifications des activations de la couche suivante.

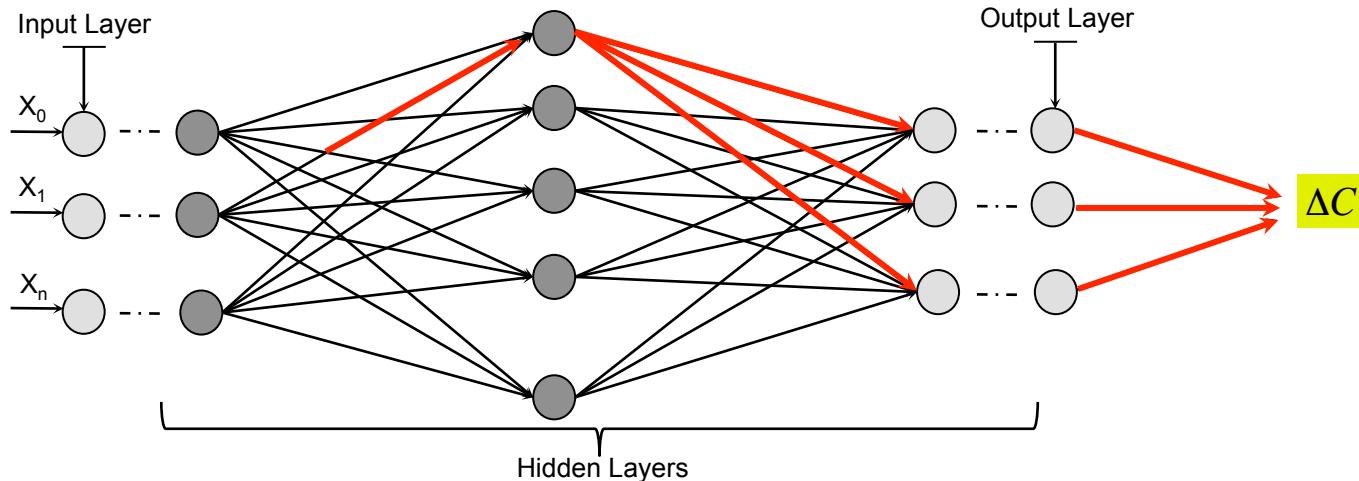


La Rétropropagation

Par l'image

- Effet domino qui entraîne des modifications tour-à-tour dans les couches suivantes, et ainsi de suite jusqu'à atteindre la couche finale, impliquant des changements dans la fonction coût C .

$$\Delta C \approx \frac{\partial C}{\partial w_{jk}^l} \Delta w_{jk}^l \quad (4)$$

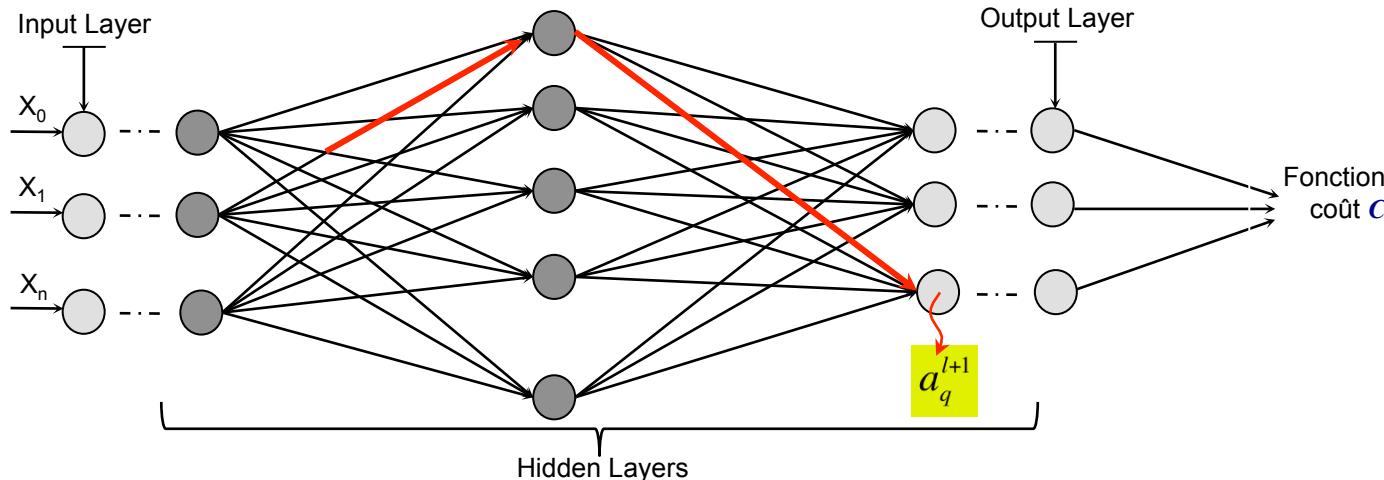


La Rétropropagation

Par l'image

- Le changement Δw_{jk}^l provoque un petit changement Δa_j^l dans l'activation du neurone j dans la l ème couche.

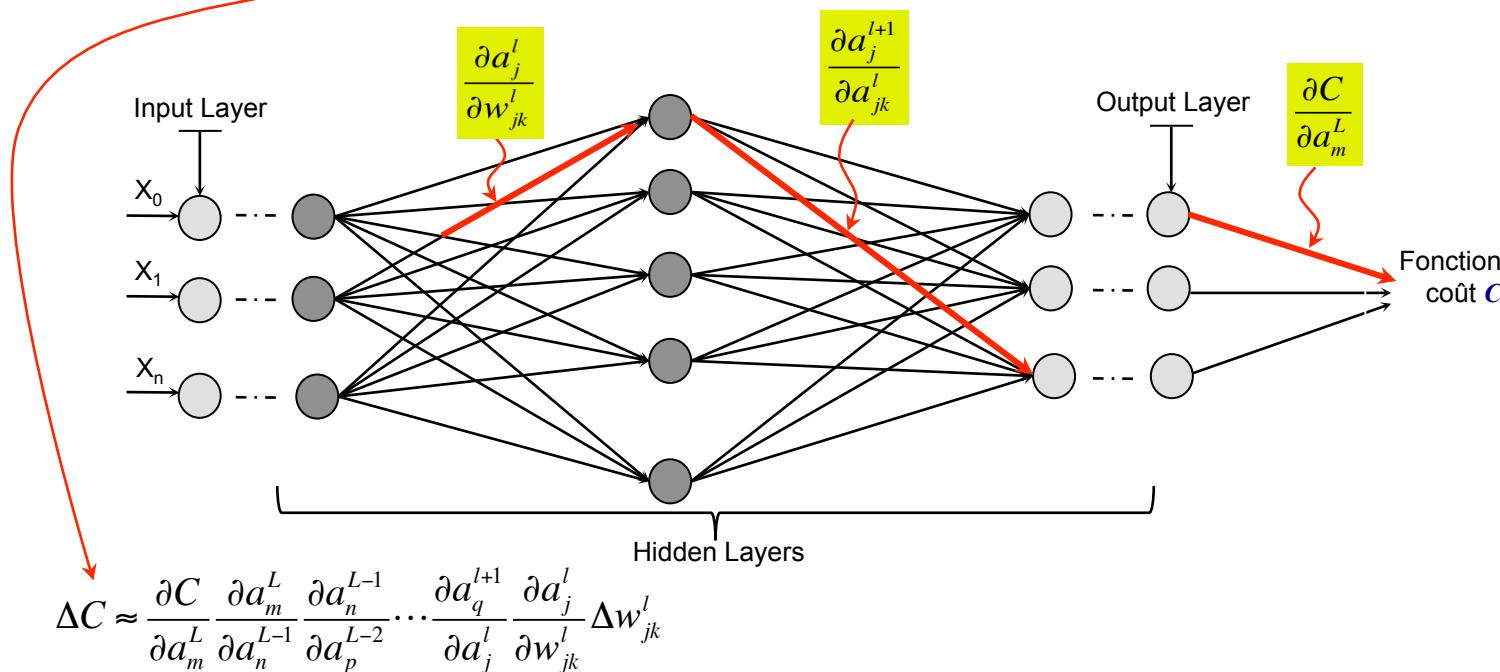
$$\Delta a_q^{l+1} \approx \frac{\partial a_q^{l+1}}{\partial a_j^l} \Delta a_j^l$$
$$\Delta a_q^{l+1} \approx \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial a_{jk}^l} \Delta w_{jk}^l$$



La Rétropropagation

Par l'image

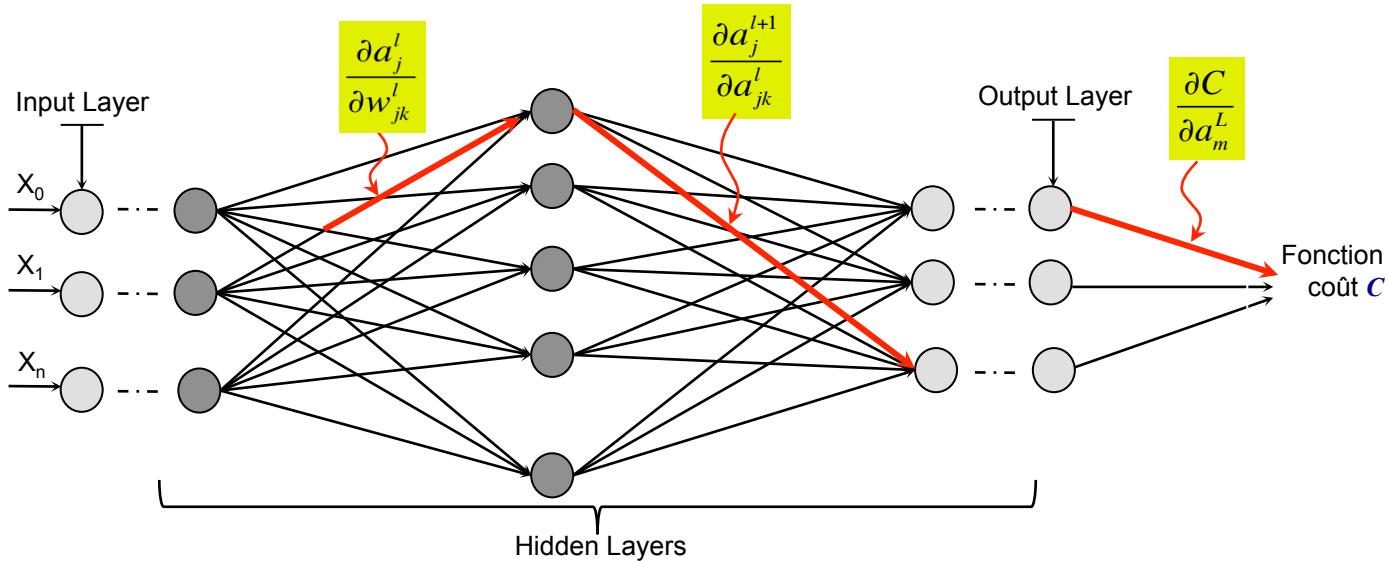
- Tout changement d'activation entraîne un changement de l'activation suivante, et, enfin, $(a_j^l, a_q^{l+1}, \dots, a_n^{L-1}, a_m^L)$ un changement de **coût** en sortie.



La Rétropropagation

Par l'image

- La variation du coût total : $\Delta C \approx \sum_{mnp\cdots q} \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \cdots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$



La Rétropropagation

Par l'image

- En comparant avec (4) :
$$\frac{\partial C}{\partial w_{jk}^l} \approx \sum_{mnp\cdots q} \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \cdots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l}$$
 [\(Interprétation intuitive\)](#)
- Les abords entre neurones impliquent des facteurs de vitesse liés aux dérivées partielles des activations de neurones par rapport aux activations des autres neurones.
- Le taux de changement total :

$$\frac{\partial C}{\partial w_{jk}^l}$$

est simplement la somme des facteurs de taux de tous les chemins depuis le poids initial jusqu'au coût final.

La Rétropropagation

En résumé

(1) **Entrée x** : définir l'activation correspondante a^1 pour la couche d'entrée.

(2) **Anticipation** : Pour chaque couche ($l = 2, 3, \dots, L$)

$$a^l = \sigma(w^l a^{l-1} + b^l) \quad \text{avec : } a^l = \sigma(z^l) \quad \text{et} \quad z^l = w^l \bullet a^{l-1} + b^l$$

(3) **Calcul de l'erreur en sortie** : $\delta^L = \nabla_a C \odot \sigma'(z^L)$ avec : $\nabla_a C(w, b) = (a^L - y)$

(4) **Rétropropagation de l'erreur** : Calculer l'erreur généralisée δ^l sur toute la couche l .

$$\delta^l = (w^{l+1})^T \delta^{l+1} \odot \sigma'(z^l) \quad (\text{Pour chaque } l = L-1, L-2, \dots, 2)$$

(5) **Taux de variation de la fonction coût** : $\frac{\partial C}{\partial w_{jk}^l} = a_j^{l-1} \delta_j^l \quad \frac{\partial C}{\partial b_j^l} = \delta_j^l$

(6) **Mise à jours** : $(w_{jk}^l, b_j^l)^{(i+1)} = (w_{jk}^l, b_j^l)^{(i)} - \eta \cdot \nabla C(w_{jk}^l, b_j^l)^{(i)}$

Comparatifs DL, ML

Caractéristiques	Deep Learning	Machine Learning
Données	Non structurée avec une grande masse de données (>1MO).	Structurée et maîtrisée pour la prédiction.
Dépendances matérielles	GPU pour l'optimisation des calculs et d'opérations de multiplication matricielle.	Ne nécessite pas de grande ressources.
Processus de caractérisation	Utilisation des données pour l'apprentissage de fonctionnalités tout en créant de nouvelles.	Nécessite la création et l'identification des fonctionnalités avec précision par les utilisateurs.
Apprentissage	Parcourt le processus d'apprentissage en résolvant les problèmes de bout-en-bout. Autonome.	Décomposition du processus d'apprentissage en plusieurs phases. Nécessite l'intervention humaine.
Temps d'exécution	Beaucoup de temps pour l'entraînement aux nombreuses couches dans le réseau.	Peu de temps pour l'entraînement (allant de quelques secondes à quelques heures).
Sortie	Plusieurs formats : un texte, une partition ou un son.	Généralement une valeur numérique, i.e., un score ou une classification.

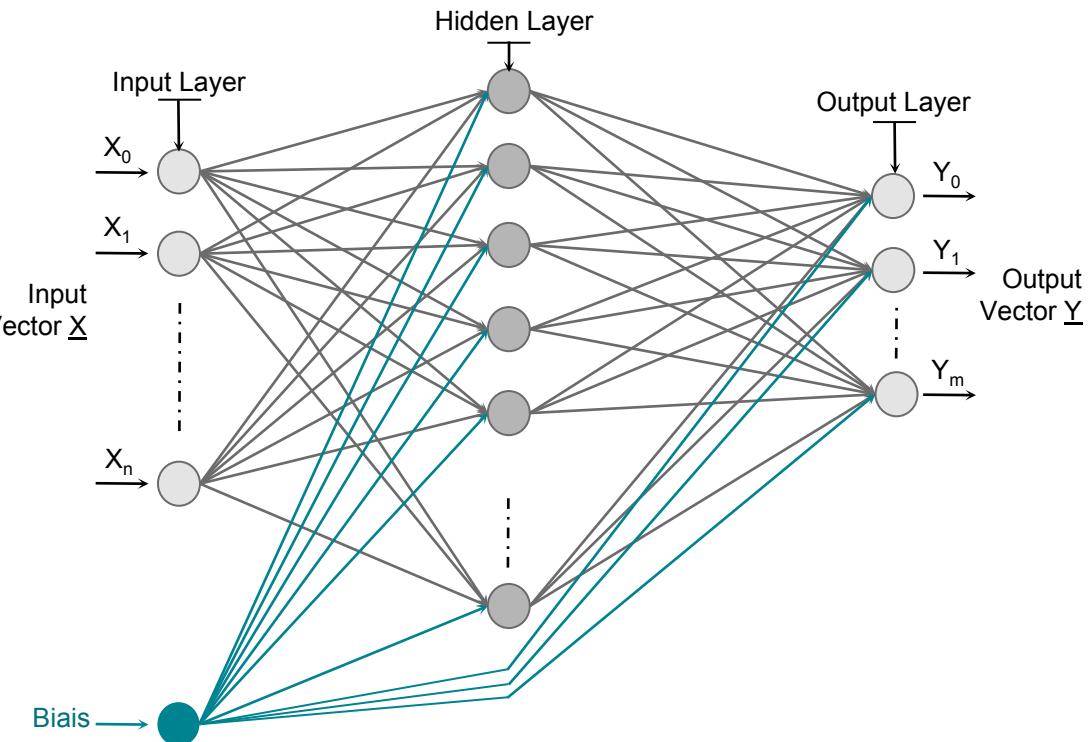
Application à l'identification de fonctionnalités logiques

Projet
#monôme

MLP à 3 couches

Study case

- Identification des fonctionnalités des portes logiques types **AND**, **OR**, **XOR**....
- Implémentation des Threads avec procédure bloquantes pour la mise en œuvre de plusieurs réseau en ligne.
- Mise en œuvre en langage C.
- Structure neuronale RNA-MLP.
- 2 entrées - 1 sortie.
- Fonctions d'activation : Tangente hyperbolique.
- ...



Some logic gates



AND

Inputs		Output
0	0	0
1	0	0
0	1	0
1	1	1



OR

Inputs		Output
0	0	0
1	0	1
0	1	1
1	1	1



EXCLUSIVE OR

Inputs		Output
0	0	0
1	0	1
0	1	1
1	1	0



NAND

Inputs		Output
0	0	1
1	0	1
0	1	1
1	1	0



NOR

Inputs		Output
0	0	1
1	0	0
0	1	0
1	1	0



EXCLUSIVE NOR

Inputs		Output
0	0	1
1	0	0
0	1	0
1	1	1

Any Digital Logic Circuit can be implemented using Perceptron.