IA et Nous
Réseaux de neurones

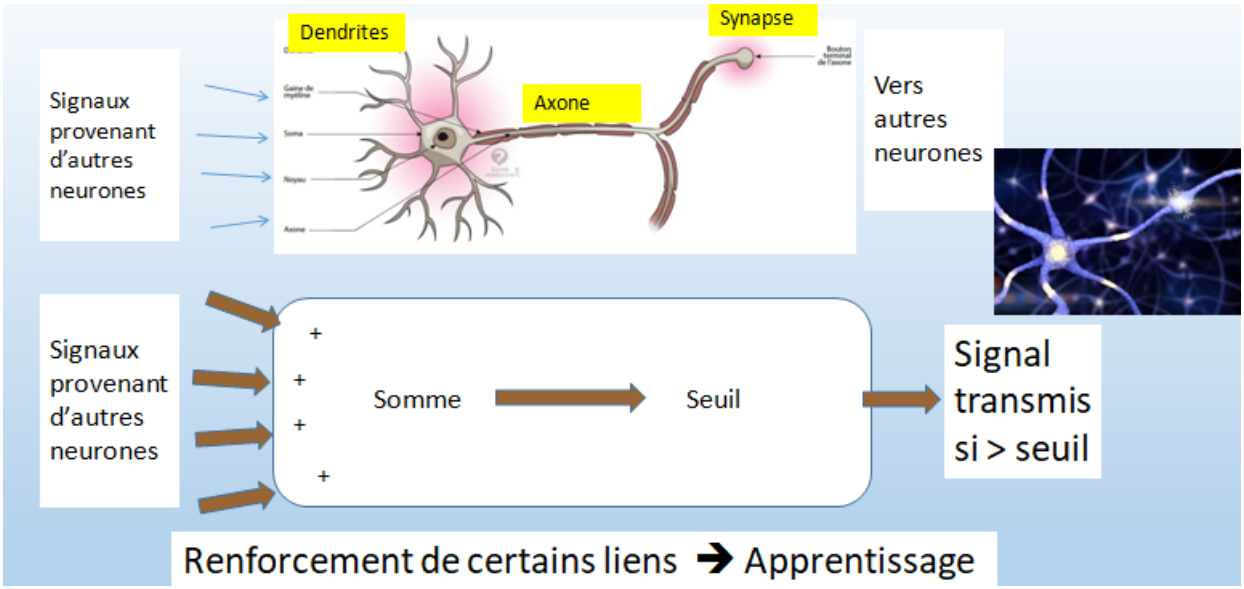
Maj février 2023

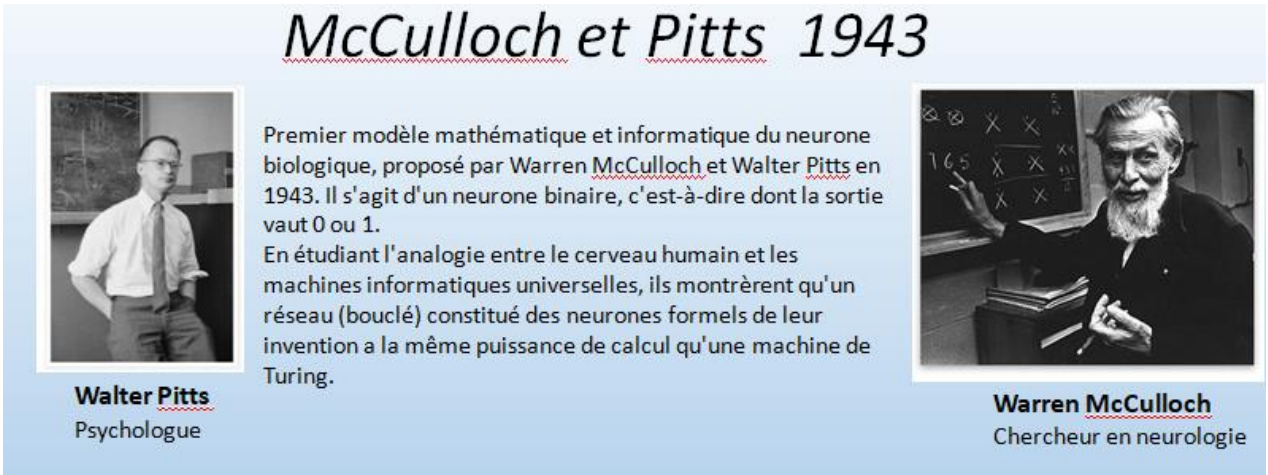
Plan

1. *Le Neurone formel de McCulloch et Pitts*
2. *Le Perceptron de Rosenblatt*
3. *Le Perceptron Multicouche*
4. *Améliorations du perceptron*
5. *Retour au multicouche*
6. *Evolutions Améliorations*
7. *Démonstrations Exemples Exercices*

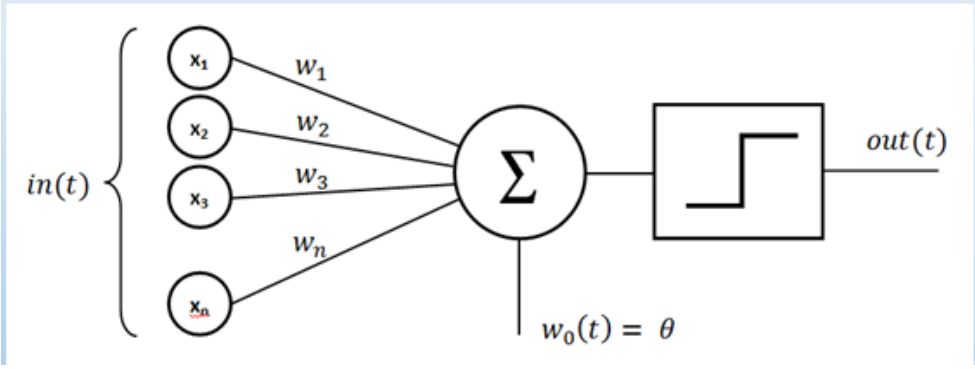
---

1. Le neurone formel de McCulloch et Pitts 1943





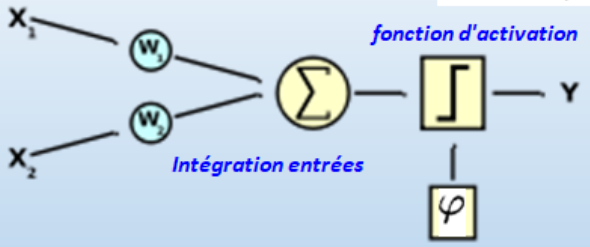
Neurone binaire

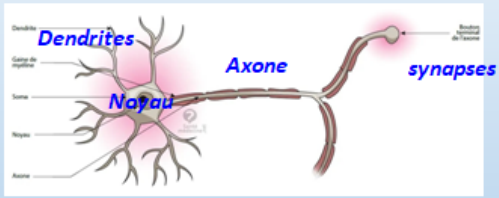


Sortie 0 ou 1

Neurone binaire

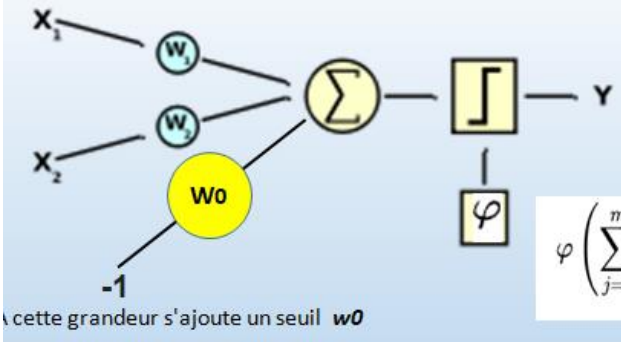
*Dans le modèle de McCulloch et Pitts, à chaque entrée est associé un poids synaptique, c'est-à-dire une valeur numérique notée **w1** pour l'entrée **1** jusqu'à **wm** pour l'entrée **m***





$$w_1 x_1 + \ldots + w_m x_m = \sum_{j=1}^{m} w_j x_j.$$

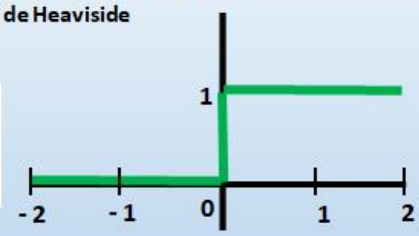
Neurone biologique	Neurone formel
Synapses	Poids des connexions
Axones	Signal de sortie
Dendrites	Signal d'entrée
Noyau ou Somma	Fonction d'activation



A cette grandeur s'ajoute un seuil ***w0***

*Le résultat est transformé par une **fonction d'activation** non linéaire φ (parfois appelée fonction de sortie*
Fonction de Heaviside

$$\varphi\left(\sum_{j=1}^{m} w_j x_j - w_0\right),$$



Si la somme $\sum_{j=1}^{m} w_j x_j$ > ***w0*** Y = 1, sinon Y= 0 :

W0 est donc le seuil d'activation du neurone, d'où le nom de neurone binaire

ThresHold Logic Unit

Au départ le modèle n'est conçu que pour traiter des entrées logiques donc à 0 ou à 1

***Mc Culloch et Pitts** ont montré que l'on pouvait reproduire les **fonctions booléennes** « ET « et « OU »*

H (x1 + x2 + w0) avec H fonction de Heaviside

Selon la valeur de W 0

-1 <= W 0 < 0

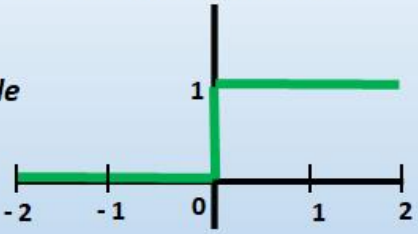
x_1/x_2	0	1
0	0	1
1	1	1

Porte OU

-2 <= W 0 < -1

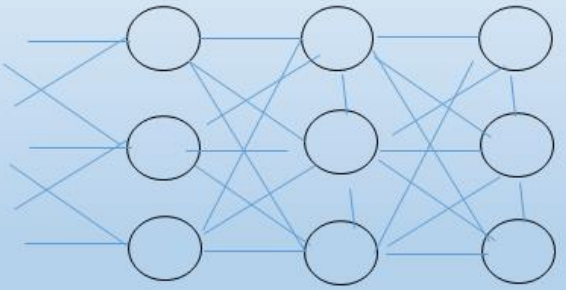
x_1/x_2	0	1
0	0	0
1	0	1

Porte ET



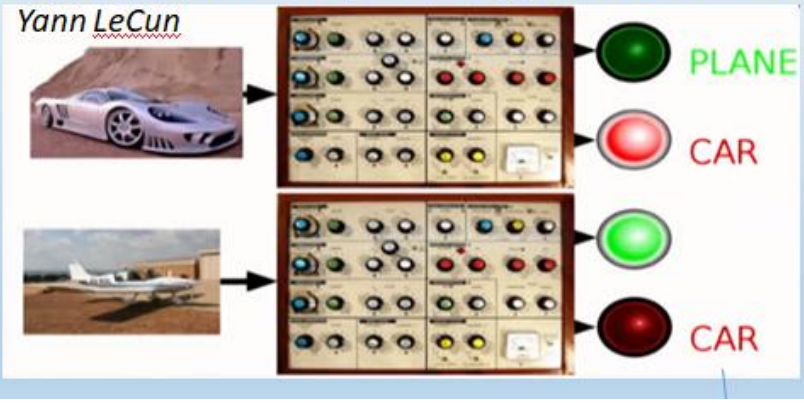
Un neurone à une entrée donne, selon la valeur de Wo la fonction identité (w0> 0) u la fonction « NAND » (w0 <-1)

En combinant des neurones de McCulloch et Pitts,, on peut donc réaliser n'importe quelle fonction logique.
Quand on autorise en outre des connexions formant des boucles dans le réseau, on obtient un système avec la même puissance qu'une machine de Turing



L'engouement fut donc extraordinaire voire démesuré
Certains imaginent que ces machines allaient être capables de remplacer le cerveau humain !!!

Mais même si ce modèle pose les bases de qui est encore aujourd'hui le Deep Learning, il a plusieurs lacunes dont :
- il ne traite que des valeurs logiques
- il ne dispose pas d'algorithme d'apprentissage, ce qui nous oblige à trouver nous-mêmes les valeurs des « poids » Wi pour des applications du monde réel



Maj février 2023

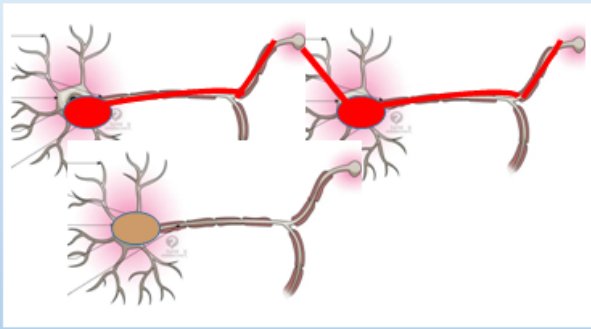
2. Le Perceptron de Rosenblatt 1957

Frank Rosenblatt invente en 1957 le premier réseau de neurones artificiels à une seule couche : le Perceptron

Il implémente sur un IBM 704 le Perceptron et parvient à simuler le processus de mémorisation en appliquant la règle de Hebb.

Perceptron (1960)

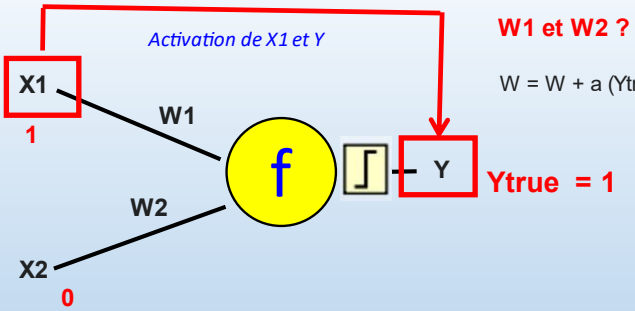
En 1957, un psychologue américain proposa le premier algorithme d'apprentissage ***Franck Rosenblatt*** *avec le* ***Perceptron***



Il utilise la règle de Hebb :
Résumée par la formule : « des neurones qui s'excitent ensemble se lient entre eux. »
(cells that fire together, wire together).
Ils renforcent leurs connexions, leurs liens synaptiques : c'est ce que les neurosciences appellent la plasticité synaptique

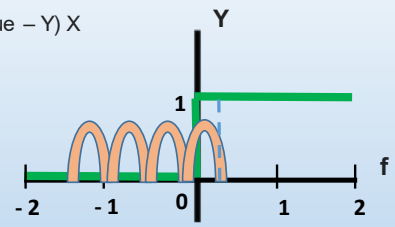
Perceptron

Algorithme d'apprentissage



W1 et W2 ?

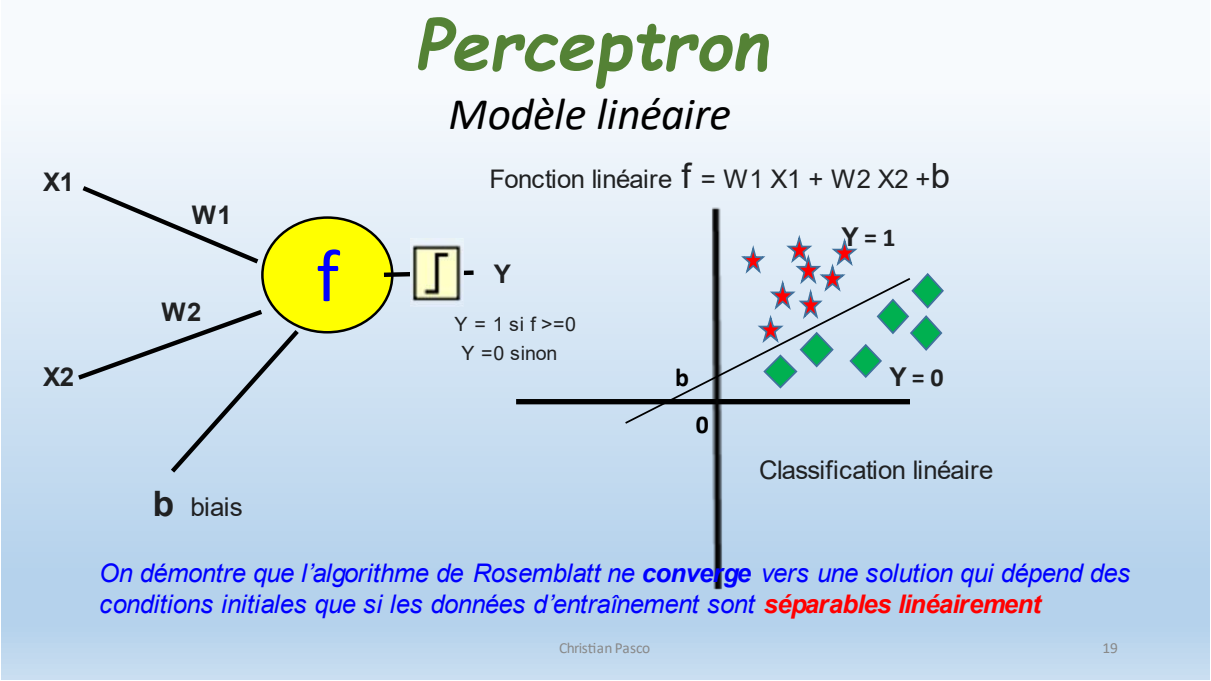
W = W + a (Ytrue – Y) X



Si Y = 0
W1 = W1 + a (1– 0); W1= W1 + a
W2 = W2
Et ainsi de suite tant que Y = 0

W1 = W1+ a

Christian Pasco 18



Là encore , l'engouement fut extraordinaire voire démesuré.
Ce fut ***un boom***

On pensait que, grâce au Perceptron, on pourrait réaliser des machines capables d'écrire, de parler, de marcher, voire même d'avoir une conscience !!!

En 1969*, **Marvin Minsky et Seymour Papert** montrent que les neurones formels ne peuvent enregistrer que des données* ***linéairement séparables****.*

En particulier, un neurone formel ne peut simuler la fonction ***OU exclusif (XOR)***
Or, une grande partie des phénomènes de notre univers ne sont pas linéaires
➔ *le perceptron simple n'a donc pas trop d'intérêt*

Les travaux sur les réseaux de neurones s'arrêtent quasi totalement pendant 10 ans
Premier Hiver de l'IA 1970 -1980 Quasiment plus d'investissements pendant 10 ans

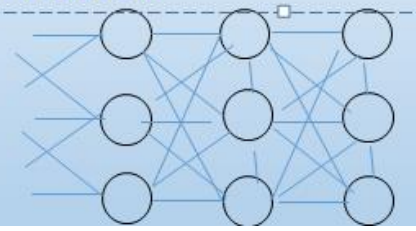
Maj février 2023

3. Le Perceptron Multicouche

Renaissance du connexionnisme

*Au début des **années 80**, redémarrage de travaux sur les **réseaux de neurones***

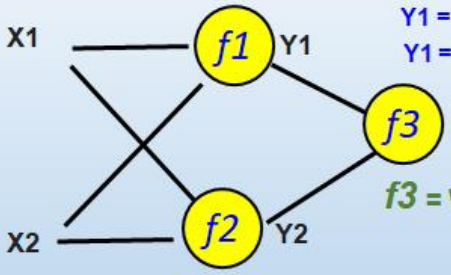
*En **1986**, **Geoffrey Hinton**, un des pères du **Deep Learning**, développe le **Perceptron multicouches***
Idée : en connectant ensemble plusieurs neurones, on devrait pouvoir résoudre des problèmes plus complexes et représenter toute fonction



Association de 3 neurones

f1 = W11 X1 + W12 X2 + b1

Y1 = 1 si *f1* >=0
Y1 = 0 sinon

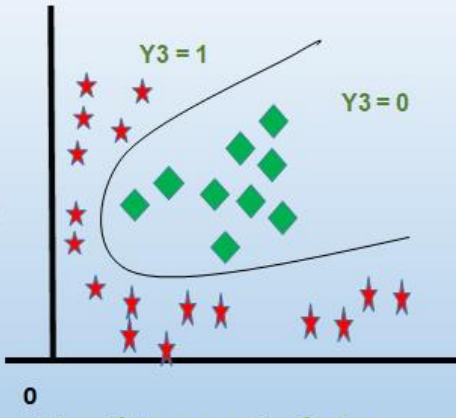


f3 = W31 Y1 + W32 Y2 + b3

Y3 = 1 si *f3* >=0
Y3 = 0 sinon

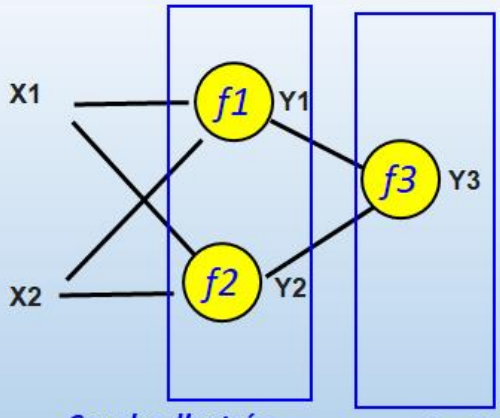
f2 = W21 X1 + W22 X2 + b2

Y2 = 1 si *f2* >=0
Y2 = 0 sinon



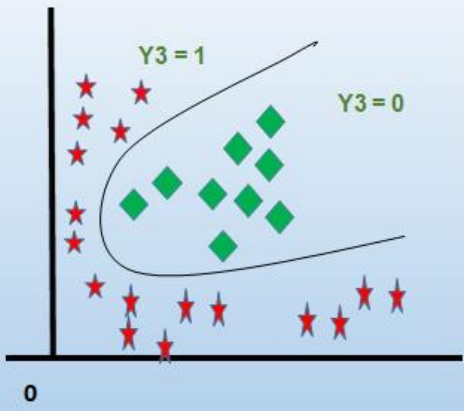
Modèle non linéaire

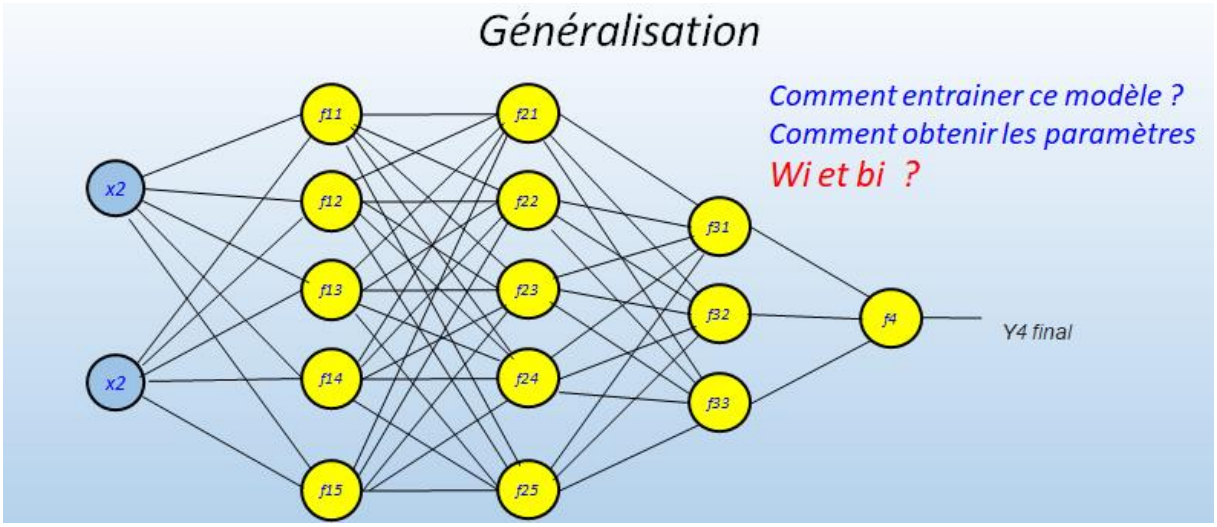
Association de 3 neurones

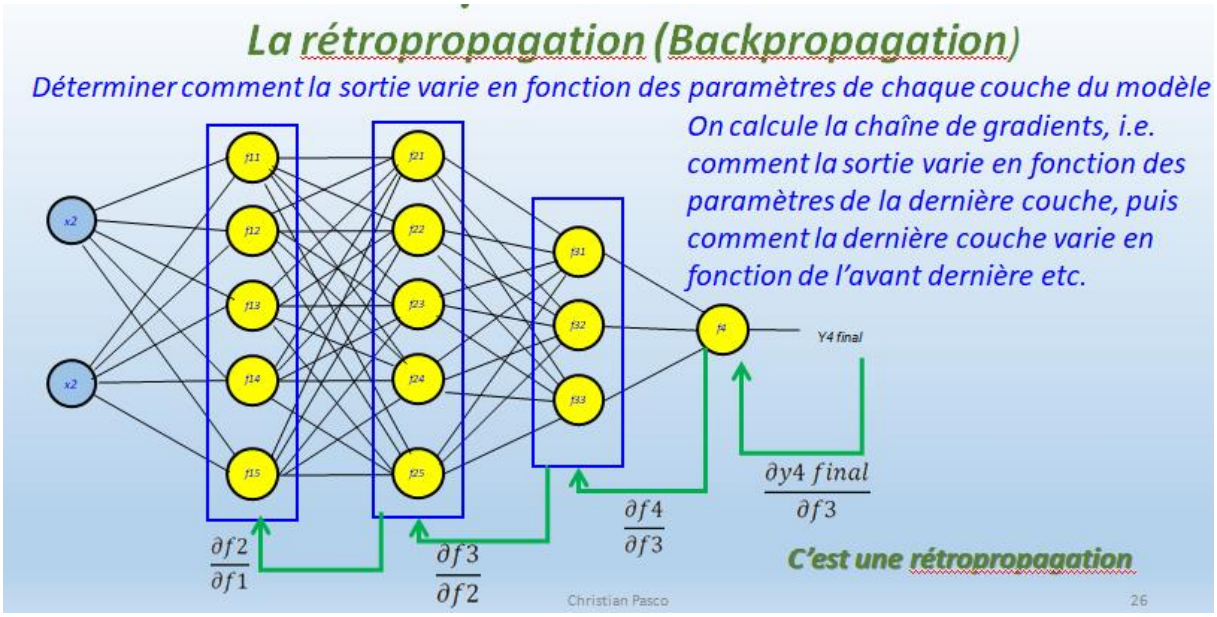


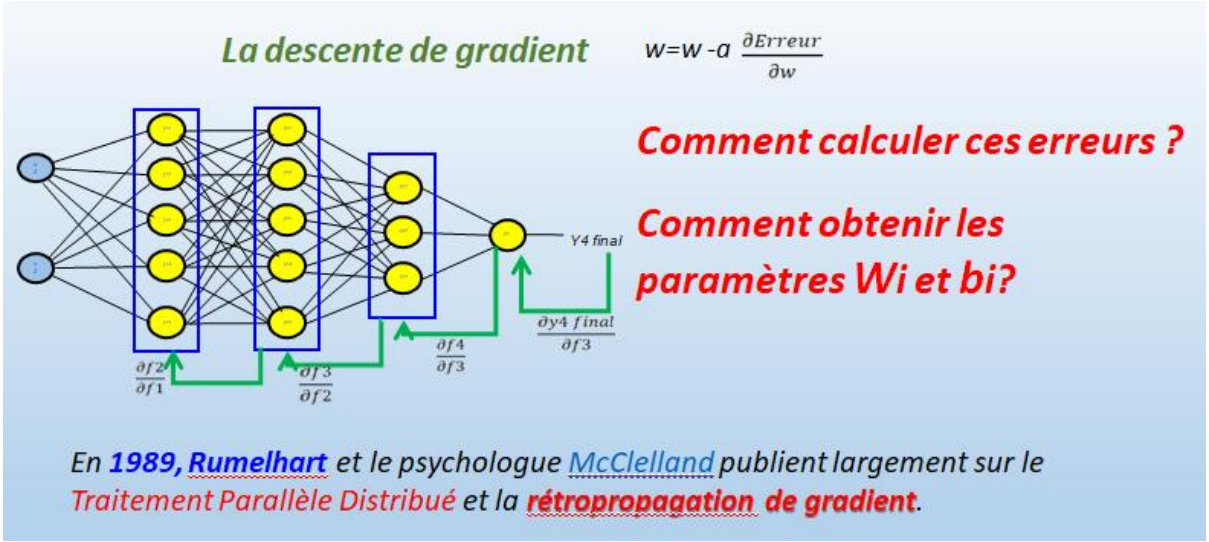
Couche d'entrée à 2 neurones

Couche de sortie à 1 neurone



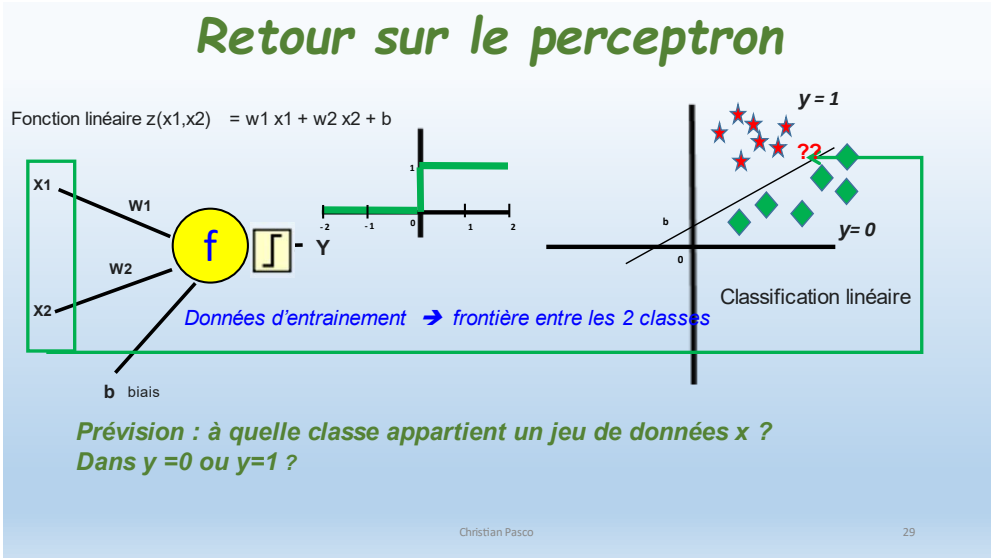

Généralisation
Comment entrainer ce modèle ?
Comment obtenir les paramètres
Wi et bi ?
x2
x2
f11
f12
f13
f14
f15
f21
f22
f23
f24
f25
f31
f32
f33
f4
Y4 final


La rétropropagation (Backpropagation)
Déterminer comment la sortie varie en fonction des paramètres de chaque couche du modèle
On calcule la chaîne de gradients, i.e. comment la sortie varie en fonction des paramètres de la dernière couche, puis comment la dernière couche varie en fonction de l'avant dernière etc.
Y4 final
$\frac{\partial f2}{\partial f1}$
$\frac{\partial f3}{\partial f2}$
$\frac{\partial f4}{\partial f3}$
$\frac{\partial y4\ final}{\partial f3}$
C'est une rétropropagation
Christian Pasco
26


La descente de gradient
$w=w-a\ \frac{\partial Erreur}{\partial w}$
Comment calculer ces erreurs ?
Comment obtenir les paramètres Wi et bi?
Y4 final
$\frac{\partial f2}{\partial f1}$
$\frac{\partial f3}{\partial f2}$
$\frac{\partial f4}{\partial f3}$
$\frac{\partial y4\ final}{\partial f3}$
En 1989, Rumelhart et le psychologue McClelland publient largement sur le Traitement Parallèle Distribué et la rétropropagation de gradient.

4 . Retour sur le perceptron Améliorations

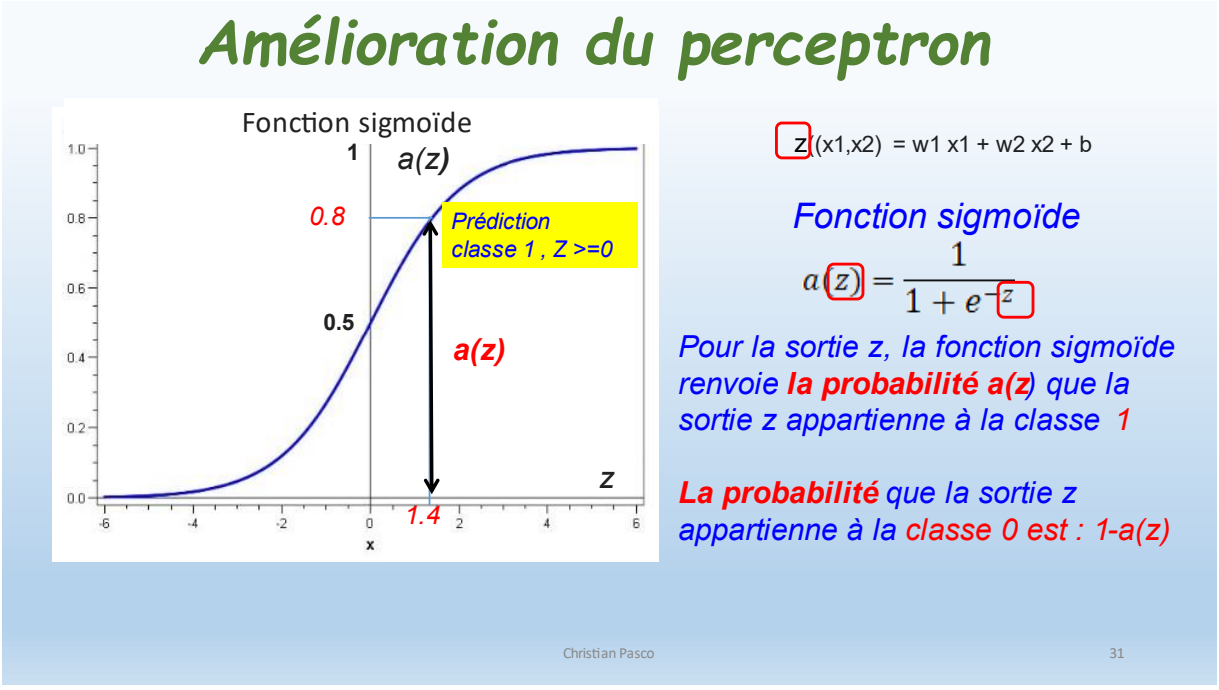
Rappel Perceptron

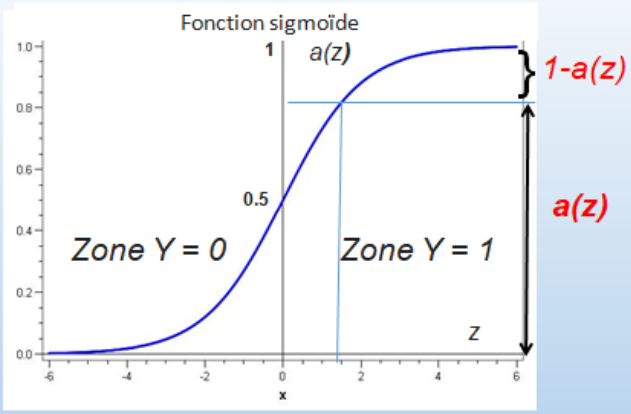


Pourrait on améliorer le modèle pour qu'il nous fournisse une probabilité ?

En effet, si une donnée se situe à la frontière de séparation, il y a une incertitude sur le résultat binaire 0 ou 1

Plus une donnée est éloignée de la frontière de décision, plus il est probable qu'elle appartient à sa classe



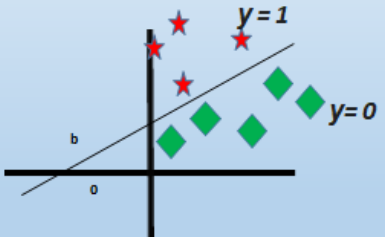


Probabilité
Loi de Bernouilli

$$P(Y = y) = a(z)^y \, X \, (1 - a(z))^{1-y}$$

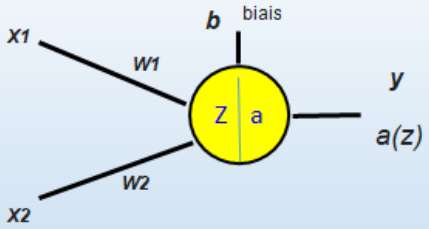
$P(Y = 1) = a(z)$

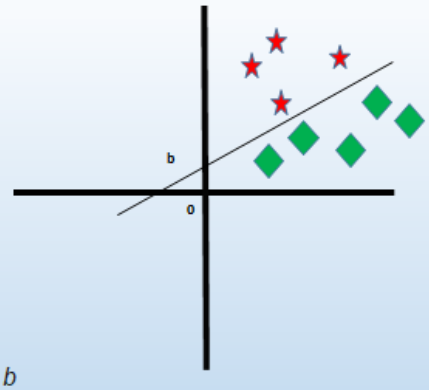
$P(Y = 0) = 1 \, X \, (1 - a(z))$



Amélioration du perceptron

Résumé





Perceptron monocouche

Fonction linéaire z $z(x1,x2) = w1\ x1 + w2\ x2 + b$

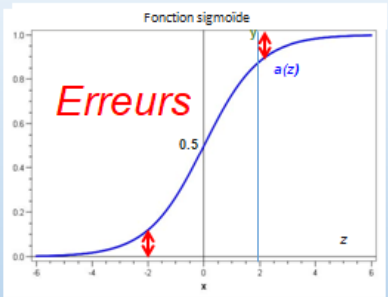
Fonction d'activation a $a(z) = \frac{1}{1 + e^{-z}}$ *Sigmoïde*

Retourne une probabilité P $P(Y = y) = a(z)^y \, X \, (1 - a(z))^{1-y}$ *Loi de Bernouilli*

Calcul de l'erreur du modèle

Fonction de coût

Il s 'agit maintenant de calculer l'erreur entre le résultat produit par la fonction d'activation a(z) et la valeur réelle y

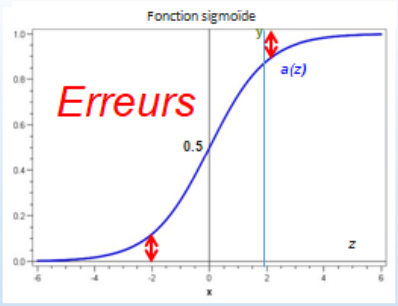


En Machine Learning, c'est le rôle de la fonction de coût

On a
- *un jeu de données étiquetées y =0*
- *un jeu de données étiquetées y=1*
- *les sorties a (z)*
- *les erreurs*

Calcul de l'erreur du modèle

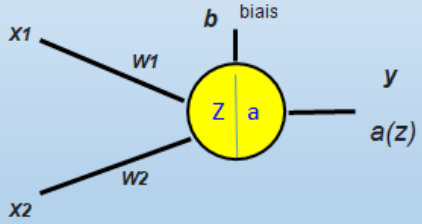
Fonction de coût



En machine Learning on utilise la fonction ***Log Loss***

Log Loss $L = -\frac{1}{m}\sum_{i=1}^{m} y_i \log(a_i) + (1-y_i)\log(1-a_i)$

- *m = nombre de données*
- *yi = donnée d'entrée n° i*
- *ai = sortie produite n° i*



Pourquoi ? D'où cela vient ?

Fonction de coût

Vraisemblance

La vraisemblance indique la plausibilité du modèle vis-à-vis de vraies données

Si pour une donnée étiquetée de classe 1, le modèle sort une probabilité de 80%, alors le modèle est vraisemblable

Pour qualifier la plausibilité d'un modèle, on multiplie les probabilités :

Likelihood
Vraisemblance

$$L = \prod_{i=1}^{m} P(Y = y_i) \qquad L = \prod_{i=1}^{m} a_i^{y_i} \times (1-a_i)^{1-y_i}$$

Loi de Bernouilli

Fonction de coût

Vraisemblance et Log Loss

On va chercher à maximiser la vraisemblance

Pour les calculs il va être plus facile d'utiliser le log : additions au lieu de multiplications:

$$LL = \prod_{i=1}^{m} a_i^{y_i} \times (1-a_i)^{1-y_i}$$

Log Likelihood
Log de la Vraisemblance

$$LL = \log \prod_{i=1}^{m} a_i^{y_i} \times (1-a_i)^{1-y_i}$$

Log Likelihood $LL = \sum_{i=1}^{m} y_i \log a_i + (1-y_i)\log(1-a_i)$ *À maximiser*

Log Loss $L = -\frac{1}{m}\sum_{i=1}^{m} y_i \log(a_i) + (1-y_i)\log(1-a_i)$ *À minimiser*

Algorithme de descente de gradient

Minimiser les erreurs du modèle

Algorithme de descente de gradient

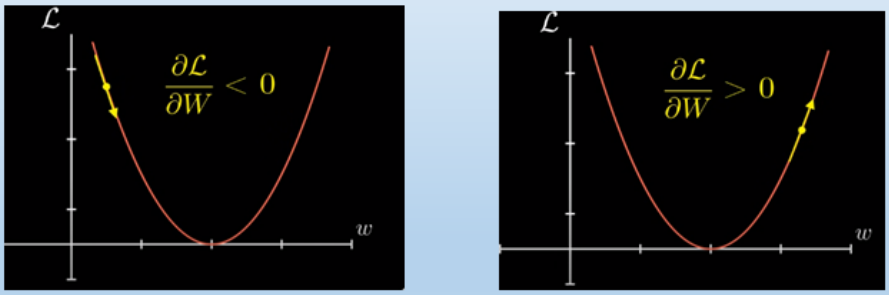
Minimiser les erreurs du modèle

On va donc chercher à minimiser les erreurs du modèle en minimisant la fonction de coût Log Loss

Il faut donc déterminer comment elle varie en fonction des différents paramètres

w et b

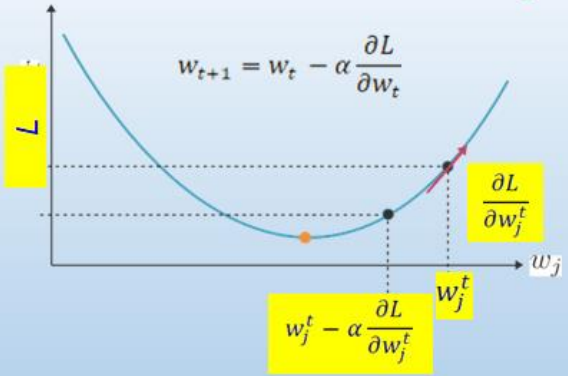
C'est pourquoi on calcule un gradient ou la dérivée de la fonction de coût Log Loss:

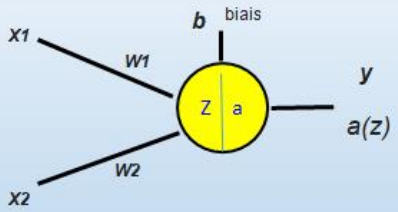


Algorithme de descente de gradient

Minimiser les erreurs du modèle

Minimisation de la fonction de coût Log Loss



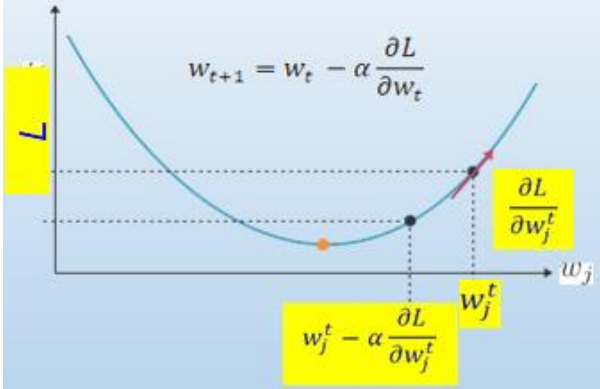


Résumé

On va donc chercher à minimiser les erreurs du modèle en minimisant la fonction de coût Log Loss

Nombre de jeux d'entrée	*m vecteurs (x1,x2)*
Intégration des entrées	*z(x1,x2) = w1 x1 + w2 x2 + b*
Fonction d'activation	$a(z) = \frac{1}{1+e^{-z}}$ Sa dérivée est *a(1-a)*
Fonction de coût	$L = -\frac{1}{m}\sum_{i=1}^{m} y_i \log(a_i) + (1-y_i)\log(1-a_i)$
Descente de gradient	$w_{t+1} = w_t - \alpha \frac{\partial L}{\partial w_t}$

Calcul des gradients



$$\frac{\partial L}{\partial w_1} = \frac{1}{m}\sum_{i=1}^{m}(a_i - y_i)x_1$$

$$\frac{\partial L}{\partial w_2} = \frac{1}{m}\sum_{i=1}^{m}(a_i - y_i)x_2$$

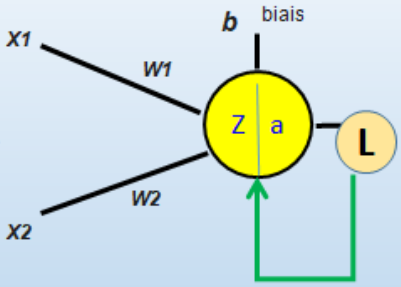
$$\frac{\partial L}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(a_i - y_i)$$

Rappel de l'algorithme initial du Perceptron 1957
W = W + a (Ytrue – Y) X

En résumé

Ajustement des paramètres w1, w2 et b

Jeu de m données x_i^1, x_i^2, b_i



1. *Forward Propagation*
2. *Cost Function*
3. *Backward Propagation*
4. *Gradient Descent*

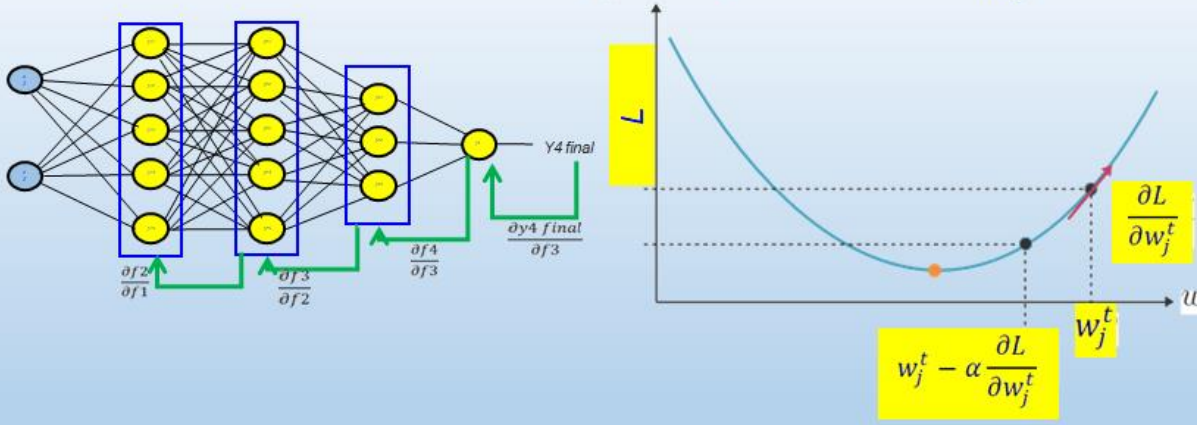
Maj février 2023

5 . Retour au multicouche

Extension au Perceptron Multicouche

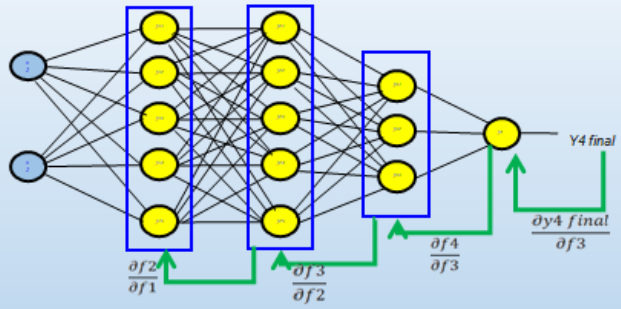
La descente de gradient

$w=w-\alpha \frac{\partial Erreur}{\partial w}$ ⇒ $w_{t+1} = w_t - \alpha \frac{\partial L}{\partial w_t}$



En résumé

Jeu de m données



Ajustement des paramètres w et b

1. *Forward Propagation*
2. *Cost Function*
3. *Backward Propagation*
4. *Gradient Descent*

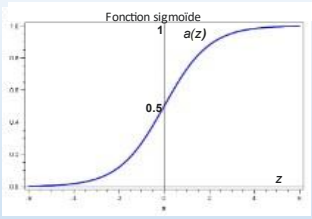
Les stratégies d'apprentissage

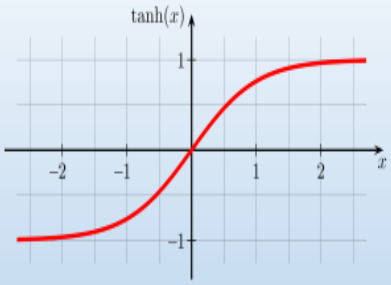
Apprentissage hors ligne	Apprentissage en ligne
• requiert souvent de charger en mémoire l'ensemble des poids et des données d'apprentissage (entrées et éventuellement sorties désirées correspondantes) ; • ne peut satisfaire une contrainte de temps réel ; • autorise de réinitialiser sans risque l'apprentissage.	• prend en compte les observations itérativement, au fur et à mesure et demande de ce fait moins de mémoire, moins de calculs ; • est compatible au temps réel ; • "subit" l'ordre dans lequel les observations sont accessibles.

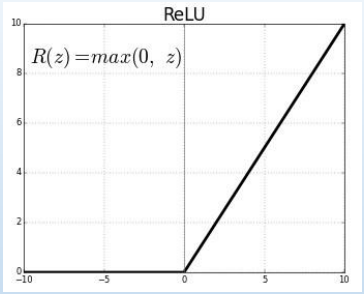
6. Evolutions- Améliorations

Fonctions d'activation

Utilisation d'autres fonctions d'activation







Fonction sigmoïde

$$a(z) = \frac{1}{1+e^{-z}}$$

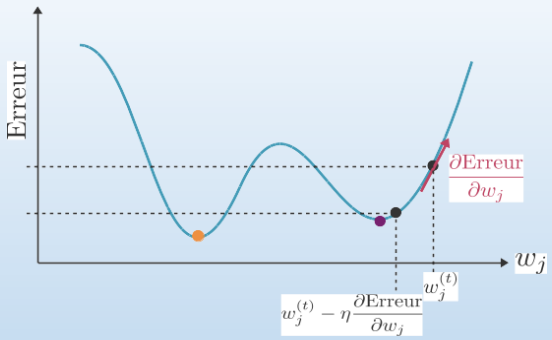
Fonction tangente hyperbolique

$$\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$$

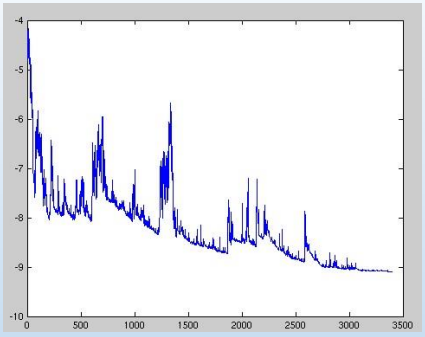
Fonction ReLu
Rectified Linear Unit

Descente de gradient stochastique

Stochastique : aléatoire, lié au hasard



Loss



Deux difficultés :

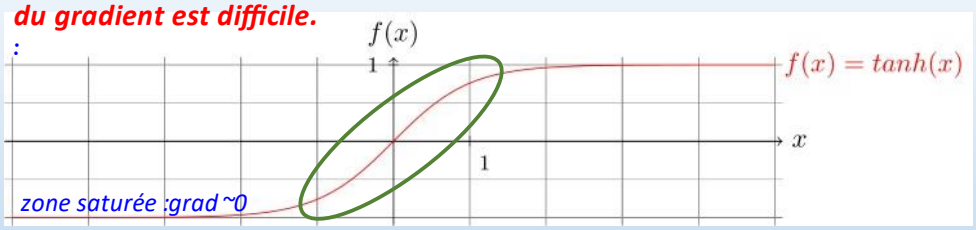
- *Minimum local ➔ tirage aléatoire et itérations*
- *Quantité de données telle que le calcul) va devenir vitre très coûteux en machine*

➔ ***On travaille à chaque epoch sur des échantillons (lots) „ choisis de façon aléatoire***

…

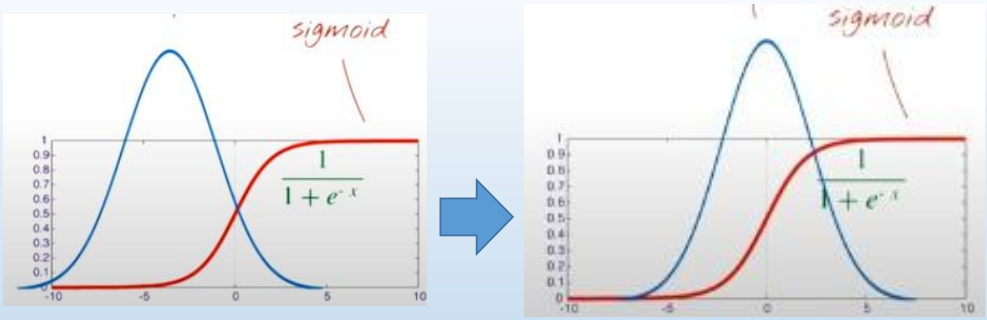
Gradient évanescent
Normalisation par lots

Plus on rajoute de couches, plus l'apprentissage par rétropropagation du gradient est difficile.
:



➔ ***Une réponse*** : ***Couche Batch Normalization***
Réduction du phénomène d'évanescence du gradient
Permet de maintenir les exemples d'un batch (lot) dans la zone non saturée d'une unité.
Par exemple, la zone proche de 0 dans la fonction tangente hyperbolique

Normalisation par lots



Couche Batch Normalization
Pour les sorties de chaque couche de neurones :
On calcule la moyenne et les écarts-types d'un lot
On fait l'opération suivante pour chaque sortie :

$$\frac{\text{Valeur sortie} - \text{valeur moyenne}}{\text{Écart type}}$$

Applicable aux ***sorties pondérées*** *avant la fonction d'activation*

Christian Pasco 54

7. Démonstrations - Exemples

Utilisation de Tensorflow

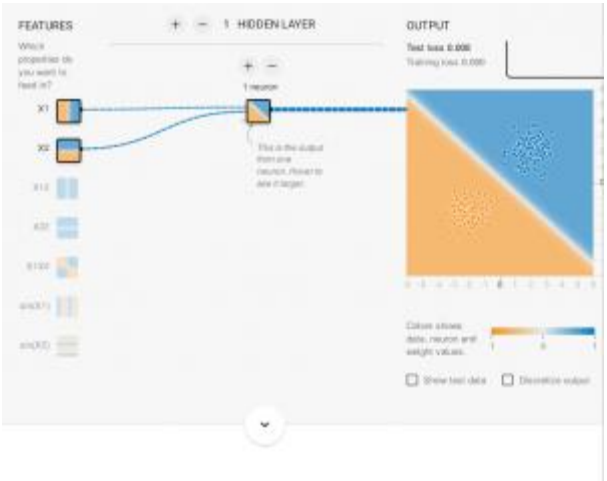
En pratique, on arrive à trouver des paramètres satisfaisants, mais il n'y a pas vraiment de cadre théorique pour formaliser tout cela, c'est affaire d'expérience :

- *choisir le bon nombre de neurones, le bon nombre de couches de neurones,*
- *quels calculs préliminaires ajouter en entrée ; par exemple multiplier les entrées pour augmenter les degrés de liberté permettant de faire le calcul*

Ce genre de techniques permet d'obtenir des résultats impressionnants en pratique, comme en reconnaissance de la voix ou d'objets dans une image

Voir les vidéos du cours de Yann Le Cun au Collège de France.

QUELQUES EXEMPLES



http://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=gauss&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=1&seed=0.76081&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false

Si on choisit de simplement classer deux populations, ici orange et bleue, qui sont déjà groupées en deux blocs facilement séparés par un ligne, alors c'est très facile : il suffit de trois neurones. En cliquant sur l'image on peut lancer la solution et observer l'ajustement des paramètres de chaque unités de calcul, ces pseudo neurones.

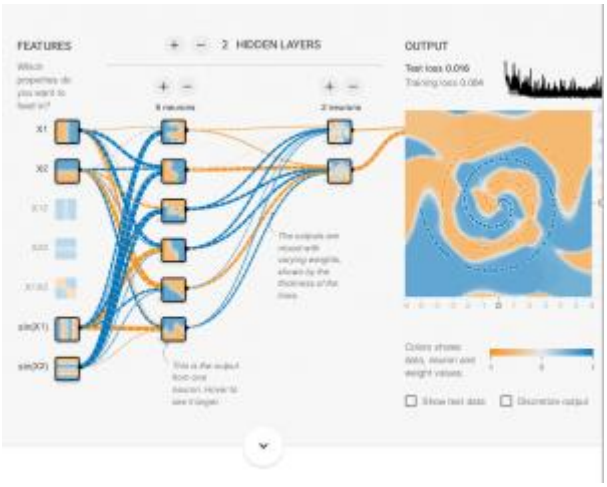
Les deux premiers neurones de la couche d'entrée calculent pour la position horizontale et verticale et le neurone de sortie combine ses informations pour faire une sortie oblique.

On parle de problème linéaire pour décrire un tel problème de classification dont la solution est une simple séparation par une surface plate.

http://playground.tensorflow.org/#activation=tanh&regularization=L2&batchSize=30&dataset=spiral&regDataset=reg-plane&learningRate=0.03&regularizationRate=0.001&noise=50&networkShape=8,8,8,8,8,8&seed=0.14280&showTestData=false&discretize=false&percTrainData=80&x=true&y=true&xTimesY=true&xSquared=true&ySquared=true&cosX=false&sinX=true&cosY=false&sinY=true&collectStats=false&problem=classification&initZero=false&hideText=false

Mais si les données sont complètement intermélées comme dans le cas de ces données en spirale alors il faut beaucoup plus de couches de calcul avec plus de neurones.

On voit alors comment au fil du calcul dans les couches de l'architecture, chaque neurone code pour un aspect de la forme à trouver, basique dans les couches basses, plus sophistiqué dans les couches plus hautes. Beaucoup de calculs pour une simple paire de spirales ! Mais ce qui est remarquable c'est que cette foule de calculs élémentaires a pu coder approximativement un objet non trivial, en s'adaptant aux données, sans avoir eu à donner des informations à priori sur ces formes spiralées.



http://playground.tensorflow.org/#activation=tanh&regularization=L2&batchSize=10&dataset=spiral&regDataset=reg-plane&learningRate=0.3&regularizationRate=0.001&noise=0&networkShape=6,2&seed=0.53189&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=true&cosY=false&sinY=true&collectStats=false&problem=classification&initZero=false&hideText=false

Faut-il tous ces neurones pour résoudre ce problème de classification de données ? Pas forcément : lors d'un atelier lycéen où des jeunes expérimentaient ce logiciel à travers l'interface Web, en expérimentant numériquement les solution possibles, deux d'entre eux Akmal et Amandine ont obtenu une solution plus longue à obtenir et moins stable qu'une solution gourmande en nombre de neurones, mais qui marche plutôt bien.

Et vous avez vous une solution encore plus astucieuse à proposer ?