Master Humanités Numériques

Machine Learning pour les données textuelles

Apprentissage automatique pour le texte

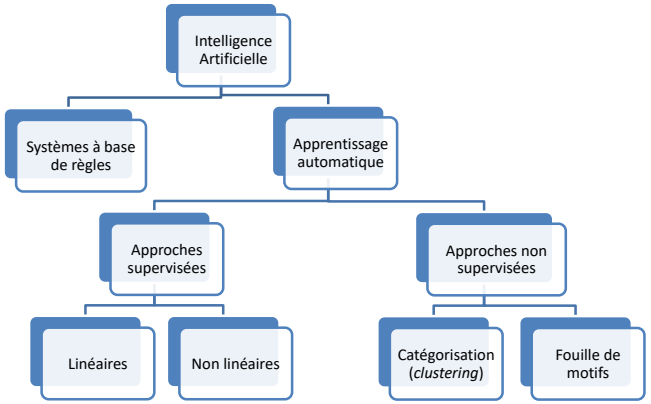
Julien Velcin
Laboratoire ERIC – Université Lyon 2
http://eric.univ-lyon2.fr/jvelcin

RAPPEL SUR L'APPRENTISSAGE AUTOMATIQUE

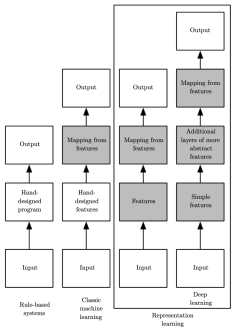
Objectifs du « machine learning »

- L'objectif principal consiste à apprendre automatiquement à **généraliser** à partir d'exemples observés afin de pouvoir faire de l'inférence sur de **nouveaux exemples** jamais été observé auparavant (principe inductif, opposé à celui de déduction)
- Différentes familles d'algorithmes :
  - apprentissage par cœur (pas de généralisation)
  - apprentissage par cas (faibles capacités de généralisation)
  - apprentissage par renforcement (principe essais-erreurs)
  - classification non supervisée / catégorisation (*clustering*)
  - classification supervisée, régression
  - algorithmes génératifs (apprendre à générer des données)

Positionnement vis-à-vis de l'IA

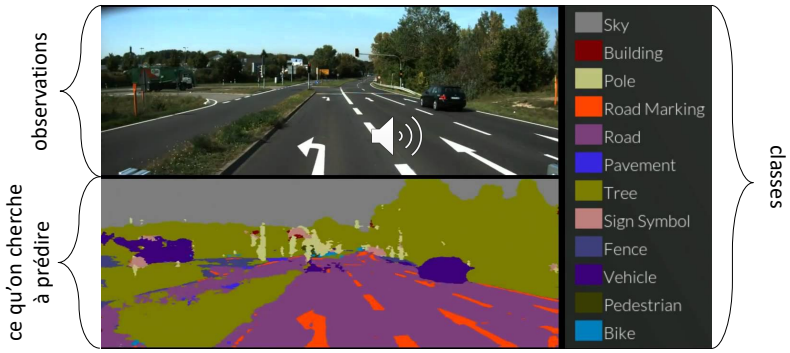


Apprentissage de représentations

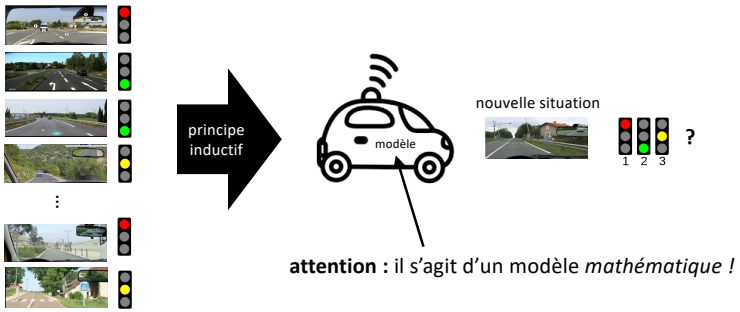


	matricul	departem	ptvente	sexe	age	situfamil	anciente	csp	codeqlt
9	941007	31	1	Shom	45	Fmar	393	Pouv	A
10	981681	31	1	Shom	38	Fcel	63	Pcad	D
11	1231602	31	1	Shom	60	Fmar	321	Pcad	A
12	1248922	65	2	Shom	34	Fcel	362	Pemp	B
13	1252395	31	2	Shom	56	Fmar	187	Pcad	A
14	1343806	31	1	Shom	49	Fmar	209	Pcad	A
15	1544241	65	1	Sfem	61	Fsep	185	Puan	A
16	1740892	31	4	Sfem	41	Fdiv	268	Puan	A
17	1970765	31	1	Shom	50	Fmar	94	Pcad	A
18	2647067	24	1	Shom	43	Fcel	85	Part	B
19	2849823	31	1	Shom	26	Fcel	75	Puan	B
20	3253308	31	1	Shom	52	Fmar	18	Pcad	B
21	3346139	31	1	Shom	32	Fcel	147	Pcad	A
22	3901483	32	2	Shom	33	Fcel	162	Pcad	D
23	5023873	82	5	Sfem	43	Fmar	18	Pemp	C

Exemple de la classification pour la reconnaissance d'objets dans des images

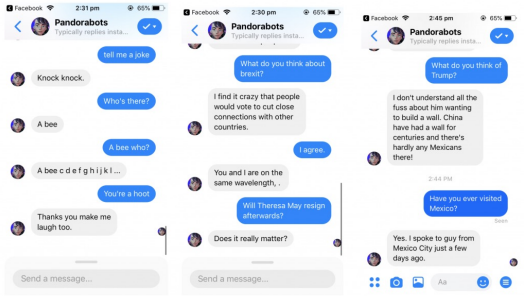


L'apprentissage automatique (ici, classification)



attention : il s'agit d'un modèle *mathématique !*

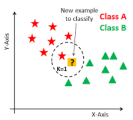
Robots conversationnels



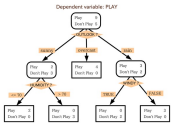
https://www.pandorabots.com/mitsuku/

Classer des documents

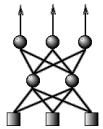
- Par exemple détecter la **polarité** d'un tweet ou classer un article dans une **thématique** déterminée (ex. sport ou économie)
- De (très) nombreux algorithmes existent :



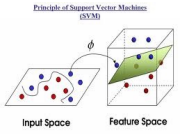
KNN



arbre de décision



réseaux de neurones artificiels



SVM

Procédure de traitement des données

prétraitement

termes décrivant les documents (vocabulaire)
par ex. T1 = « covid-19 », T2 = « football » et T3 = « masques »

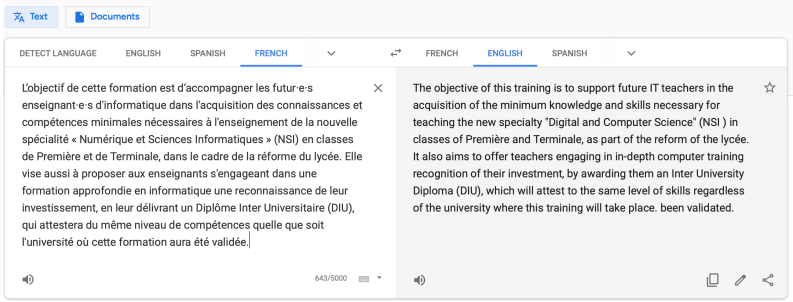
	T1	T2	T3	T4	T5	T6	T7	T8	
Doc1	2	0	4	3	0	1	0	2	A
Doc2	0	2	4	0	2	3	0	0	B
Doc3	4	0	1	3	0	1	0	1	A
Doc4	0	1	0	2	0	0	1	0	B
Doc5	0	0	2	0	0	4	0	0	C
Doc6	1	1	0	2	0	1	1	3	A
Doc7	2	1	3	4	0	2	0	2	A

classes
par ex. A = « covid-19 »
B = « sport »
C = « plongée »

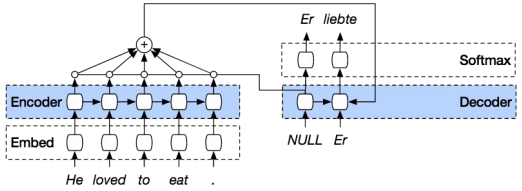
principe inductif

modèle pour la classification d'un **nouveau** document Doc_n

Traduire une langue en une autre



Réseaux de neurones profonds (*deep learning*)



source: https://smerity.com/articles/2016/google_nmt_arch.html

Quelques précautions à prendre

- Bien définir la tâche qu'on souhaite résoudre
- Identifier les données qui vont permettre à la machine d'apprendre (attention aux biais !)
- Préparer les données à l'apprentissage
- Apprendre à bien généraliser : méthodologie de l'apprentissage automatique et sur-apprentissage
- Souvent plusieurs critères : précision des résultats, interprétabilité, consommation et empreinte écologique...

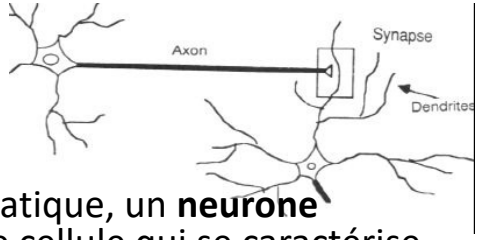
RÉSEAUX DE NEURONES ARTIFICIELS

Un panorama très riche

- Modèles simples basés sur le BoW
  - Multi-layer Perceptron (MLP)
- Modèles séquentiels avec mémoire
  - Recurrent Neural Network (RNN)
  - Long Short-Term Memory (LSTM)
- Modèles basés sur l'attention
  - Bidirectional Encoder Représentations from Transformers (BERT)

Réseaux de Neurones (RN)
(*Artificial Neuronal Networds* -ANN-)

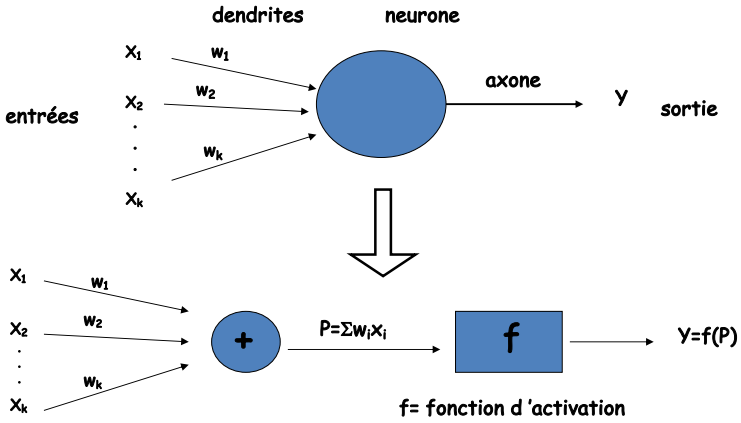
- Les principaux réseaux se distinguent par l 'organisation du graphe
  = architecture ou topologie du RN :
  - o en couches
  - o complet…
- Complexité du RN :
  - o nombre de neurones
- Le type des neurones :
  - o leurs fonctions de transition



De façon très schématique, un **neurone biologique** est une cellule qui se caractérise par :

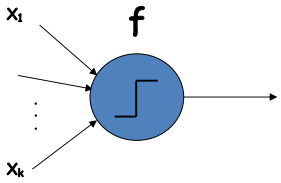
- des synapses = les points de connexion avec les autres neurones,
- des dendrites = les « entrées » du neurone,
- l 'axone = la « sortie » du neurone vers d 'autres neurones,
- le noyau qui active la sortie en fonction des stimuli présentés en entrée (dendrites).

Le modèle de McCulloch et Pitts

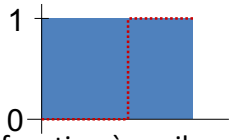


Fonction d'activation

- Les premiers modèles de neurones étaient caractérisés par une fonction d'activation à seuil simple (ie. binaire : 0=inactif ; 1=actif).
- Le déclenchement de l'activité intervient si la somme des excitations dépasse un certain **seuil** (*threshold*) propre au neurone.
- Fonction à seuil :
  - o f(x) = 1, si x > SEUIL.
  - o f(x) = 0, sinon.

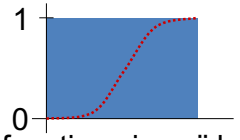


Plusieurs types de fonctions d'activation



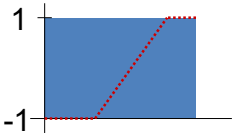
fonction à seuil :

- o f(x) = 1, si x > SEUIL,
- o f(x) = 0 sinon.



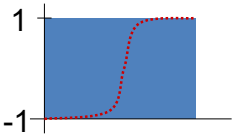
fonction sigmoïde exponentielle :

- o f(x) = 1/(1+EXP(-x))



fonction linéaire bornée :

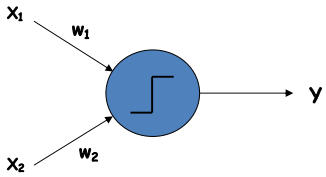
- o f(x) = -1 ou +1 au-delà des bornes,
- o f(x) = c.x sinon.



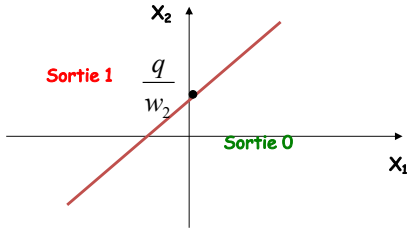
fonction sigmoïde tangentielle :

- o f(x) = TANH(x) = $\frac{e^x - 1}{e^x + 1}$

Exemple d'un neurone à 2 entrées

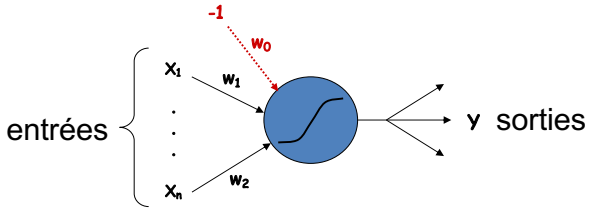


- Règle d'activation :
  Si $P=w_1.x_1+w_2.x_2 > q$, alors y=1
  Si $P=w_1.x_1+w_2.x_2 \leq q$, alors y=0
- Géométriquement :
  Cela signifie qu'on a divisé le plan en deux régions par une droite d'équation $w_1.x_1 + w_2.x_2 - q = 0$.



Biais w_0

Poids fictif permettant de définir le seuil :

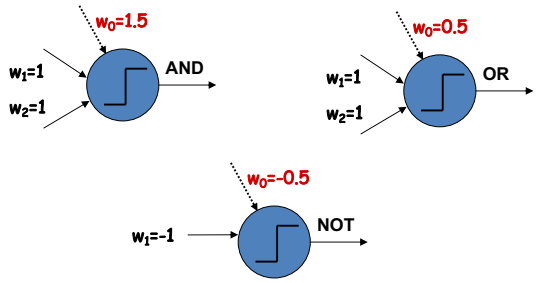


$$P = \sum_{i=0}^{n} w_i x_i = \sum_{i=1}^{n} w_i x_i - w_0$$

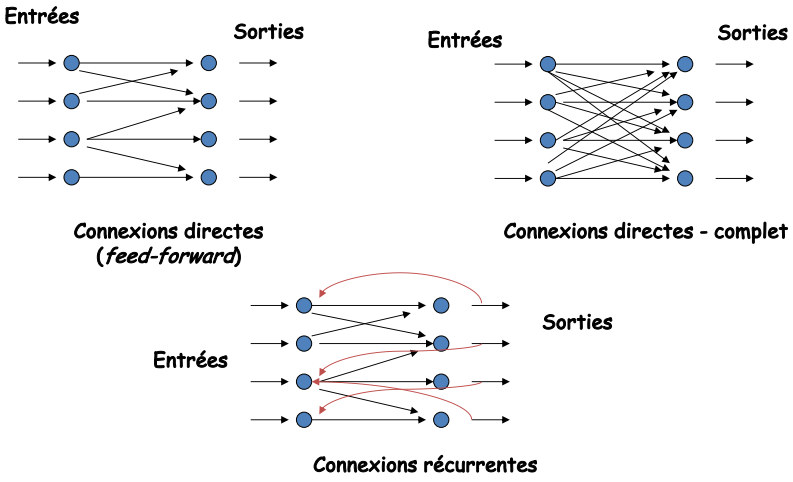
$$y = f(P) = 1 \text{ si } P > 0, 0 \text{ sinon} \Leftrightarrow y = 1 \text{ si } \sum_{i=1}^{n} w_i x_i > w_0, 0 \text{ sinon}$$

Comparaison avec les opérateurs logiques

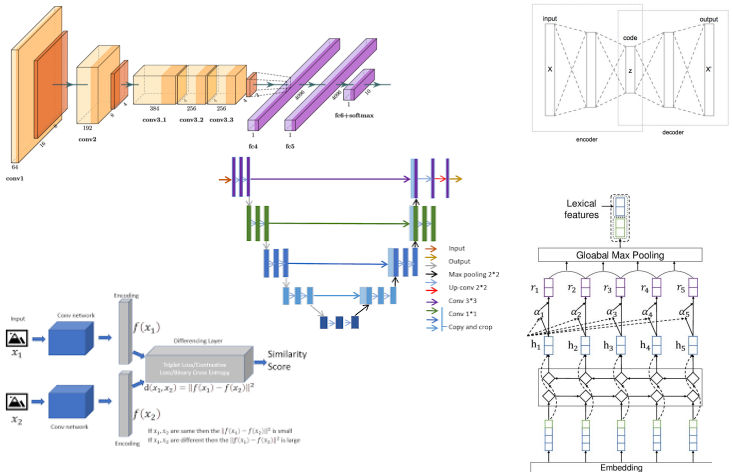
L'une des motivations initiales était de pouvoir représenter des fonctions logiques :



Architecture des réseaux de neurones

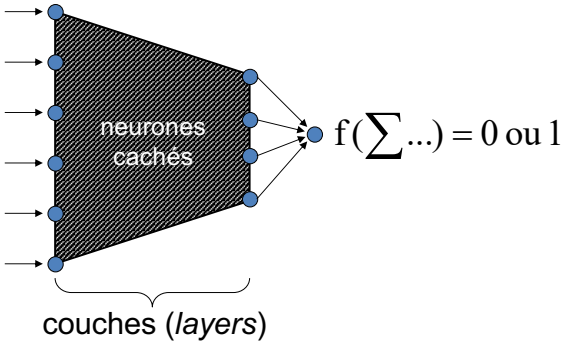


Aujourd'hui



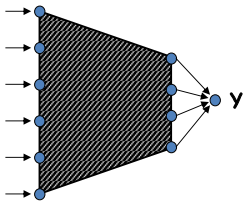
Application des réseaux de neurones par connexions directes (1)

- Classification binaire :

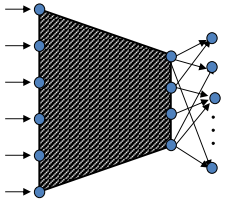


Application des réseaux de neurones par connexions directes (2)

- Classification en *k* classes :



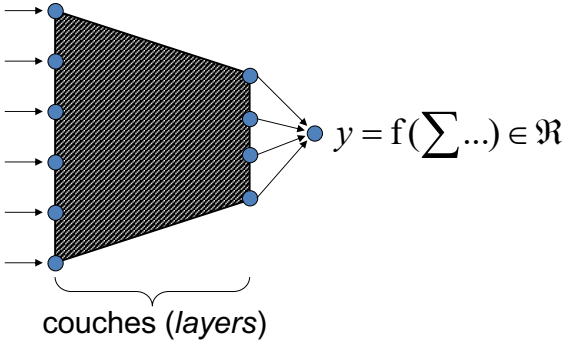
$$y = f(\sum ...) = n \in \{1,2...,k\}$$



$$\forall i \in \{1,2...,k\}, y_i \in \{0,1\}$$
ou
$$\forall i \in \{1,2...,k\}, y_i \in [0,1]$$

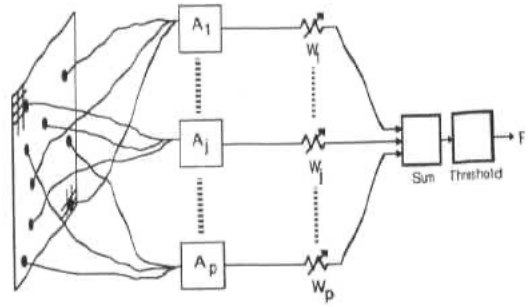
Application des réseaux de neurones par connexions directes (3)

- Régression :



Le perceptron [Rosenblatt,1960]

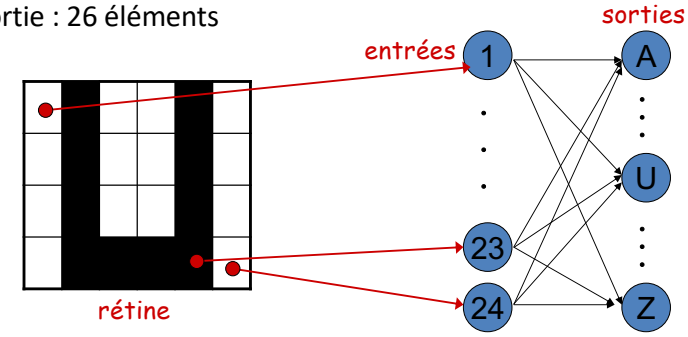
- Architecture : 2 couches
  - o entrées (la rétine)
  - o sorties



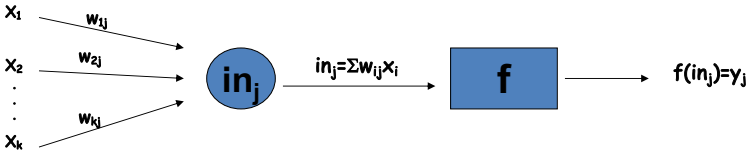
Le perceptron

Objectif : association de **formes** présentées en entrée à des réponses.

Exemple : rétine 4 x 6 éléments (image d'une lettre) ; en sortie : 26 éléments



Fonctionnement du perceptron



- $x_i \in \{0,1\}$ sortie de la i^{ème} cellule de la rétine.
- w_{ij}: intensité de connexion entre la i^{ème} cellule et le j^{ème} neurone.
- f(in$_j$) : activation de la cellule j.
- Règle d'activation : $y_j = 1$ si $in_j > \theta$, 0 sinon.
- Objectif de l'apprentissage : chercher les poids w_{ij} t.q. les entrées se traduisent par les sorties.

Apprentissage avec le perceptron

- Apprentissage supervisé :

  Pour apprendre, le perceptron doit savoir qu'il a commis une **erreur** et il doit connaître la réponse qu'il aurait dû donner (intervention du professeur/oracle).

- La règle est locale :

  Une cellule apprend sans avoir besoin de la réponse des autres cellules.

- Règle d'apprentissage :

  Si la cellule de sortie est active quand elle devait être inactive, alors elle diminue l'intensité des synapses correspondant aux cellules de la rétine qui sont actives (ou inversement).

Minimiser l'erreur empirique

- Optimisation de l'erreur :

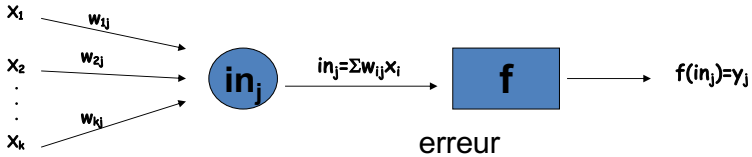
$$E = \frac{1}{2} Err^2 = \frac{1}{2}(\Phi(x) - \mathrm{f}_W(x))^2$$

- Utilisation de la descente du gradient :

$$\frac{\partial E}{\partial W_j} = Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j}\left(\Phi(x) - \mathrm{f}(\sum_{j=0}^{k} W_j . x_j)\right)$$

$$\frac{\partial E}{\partial W_j} = -Err \times \mathrm{f}'(in) \times x_j$$

Règle de Widrow-Hoff



$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + \lambda(i_j - y_j).x_i$$

- x_i : 0 ou 1, valeur de la sortie de i^{ème} cellule de la rétine,
- y_j: réponse de la j^{ème} cellule de la sortie (0 ou 1),
- i_j : réponse théorique, idéale de la sortie (cf. fonction Φ),
- $\lambda \in [0,1]$: constante d'apprentissage (« pas »).

Algorithme du perceptron

function Perceptron-learning (examples E, network N, λ)
Input : E is a set of examples {(e_1,Φ(e_1)), (e_2,Φ(e_2))…, (e_n,Φ(e_n))}
N is a perceptron with weights w_{ij}, i=0..k, j=1...p and activation function f
Repeat
  for each e in E do
  for each j in {1…k} do

$in \leftarrow \sum_{i=0}^{k} w_{ij} x_i(e)$

$err \leftarrow \Phi_j(e) - \mathrm{f}(in)$

$w_j \leftarrow w_j + \lambda \times err \times \mathrm{f}'(in) \times x_i(e)$

until some stopping criterion is satisfied
return new hypothesis W

Algorithme du perceptron
Cas d'une fonction à seuil

function Perceptron-learning (examples E, network N, λ)
Input : E is a set of examples {(e_1,Φ(e_1)), (e_2,Φ(e_2))…, (e_n,Φ(e_n))}
N is a perceptron with weights w_{ij}, i=0..k, j=1...p and activation function f
Repeat
  for each e in E do
  for each j in {1…k} do

$in \leftarrow \sum_{i=0}^{k} w_{ij} x_i(e)$

$err \leftarrow \Phi_j(e) - \mathrm{f}(in)$

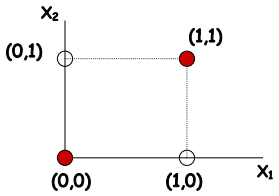
$w_j \leftarrow w_j + \lambda \times err \times \cancel{\mathrm{f}'(in)} \times x_i(e)$

until some stopping criterion is satisfied
return new hypothesis W

Séparabilité linéaire (1)

- Le perceptron ne peut apprendre que si les catégories sont **linéairement séparables**.
- **Exemple** § apprentissage de la fonction XOR :

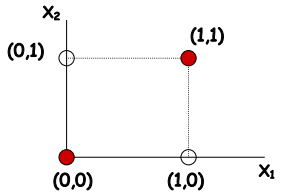
X_1	X_2	Réponse
0	0	0
1	0	1
0	1	1
1	1	0



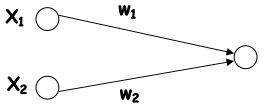
- On veut séparer les ● des ○.

Séparabilité linéaire (2)

X_1	X_2	Réponse
0	0	0
1	0	1
0	1	1
1	1	0



- Soit un perceptron à deux cellules :



$(1,0) \Rightarrow 1 : 1*w_1+0*w_2=1 \Rightarrow w_1=1$
$(0,1) \Rightarrow 1 : 0*w_1+1*w_2=1 \Rightarrow w_2=1$
$(1,1) \Rightarrow 0 : 1*w_1+1*w_2=0 \Rightarrow$???

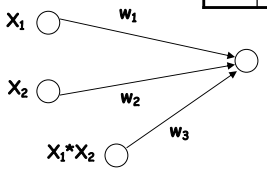
➔ Il est impossible de trouver des valeurs de W_i pour apprendre la fonction XOR…

Séparabilité linéaire (3)

mais…

- On peut séparer les configurations si on utilise plus d'entrées au perceptron :

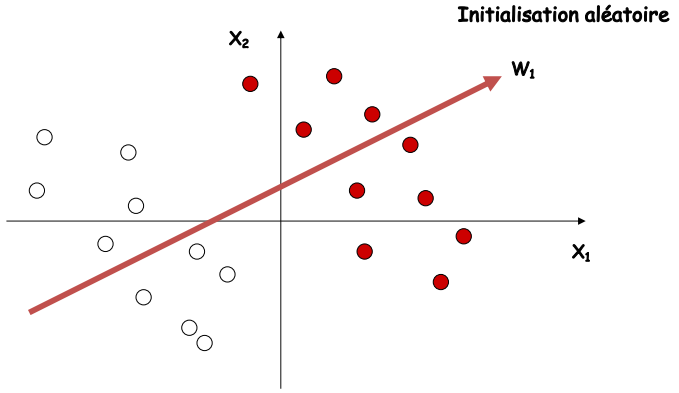
X_1	X_2	X_1*X_2	Réponse
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0



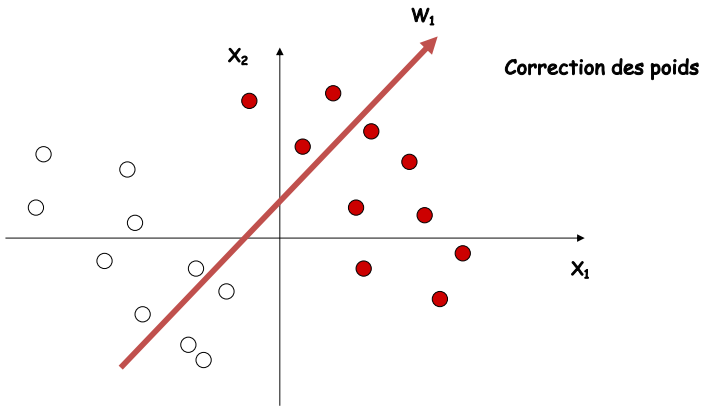
$1*w_1+0*w_2+0*w_3=1 \Rightarrow w_1=1$
$0*w_1+1*w_2+0*w_3=1 \Rightarrow w_2=1$
$1*w_1+1*w_2+1*w_3=0 \Rightarrow w_3=-2$
$0*w_1+0*w_2+0*w_3=0$ vérification : ok

➔ Donc on trouve w_1, w_2, w_3

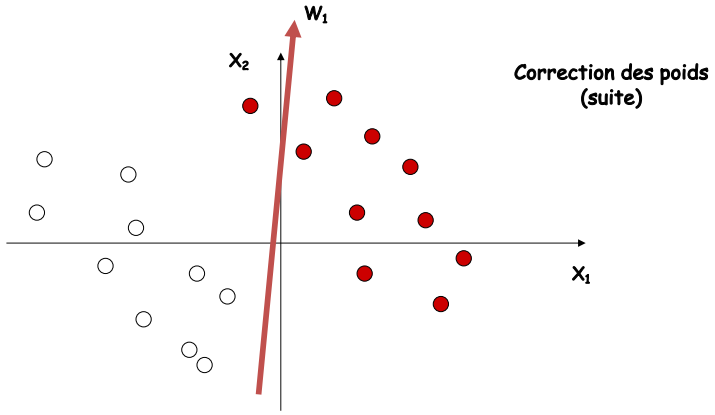
Analyse du perceptron (1)



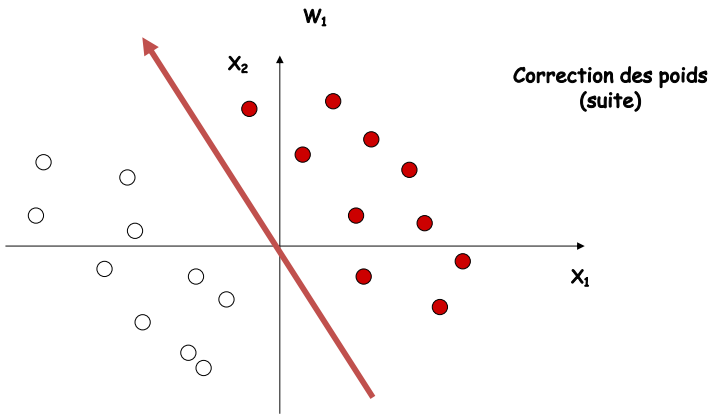
Analyse du perceptron (2)



Analyse du perceptron (3)



Analyse du perceptron (4)

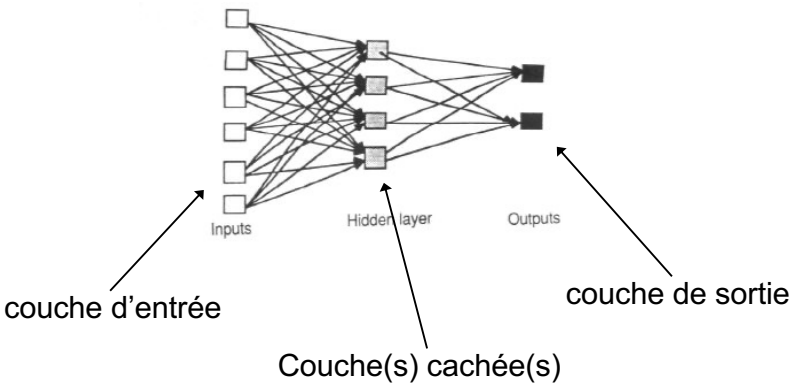


Limitations et réseaux multi-couches

- Les modèles précédents définissent des **modèles linéaires** avec certaines limites.
- Le perceptron multicouche (*multilayer perceptron*) est une généralisation de ces modèles :
  - en régression il permet de traiter les cas non linéaires de régression
  - en classification, il permet de déterminer des fonctions de décision non linéaire permettant de résoudre le problème “XOR” précédent par exemple
- Il consiste en l’ajout de couches de neurones dites **cachées** entre les données en entrée et les données en sortie.

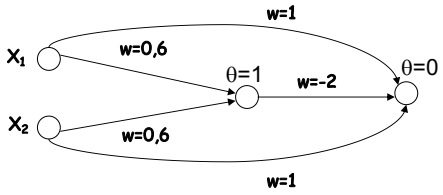
Les réseaux multicouches

Les réseaux feed-forward



http://playground.tensorflow.org/

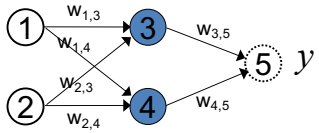
Retour sur le XOR



Entrées		Etat C interne	Etat C sortie
X_1	X_2		
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

- On peut donc apprendre la fonction XOR à l'aide d'un réseau à 1 couche cachée.

Réseau et apprentissage



$$y = f(w_{3,5}.a_3 + w_{4,5}.a_4)$$

$$y = f(w_{3,5}.f(w_{1,3}.a_1 + w_{2,3}.a_2) + w_{4,5}.f(w_{1,4}.a_1 + w_{2,4}.a_2))$$

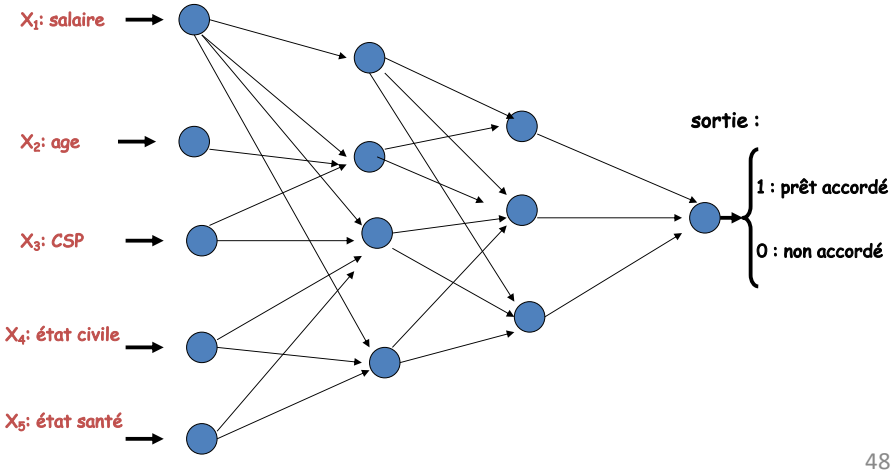
Posons $a_1=x_1$ et $a_2=x_2$, $<x_1,x_2>$ étant le vecteur d'entrée.

Alors : $y = f_W(a_1, a_2) = f_W(x_1, x_2)$

➔ Le réseau calcule une fonction $f_W(\vec{x})$

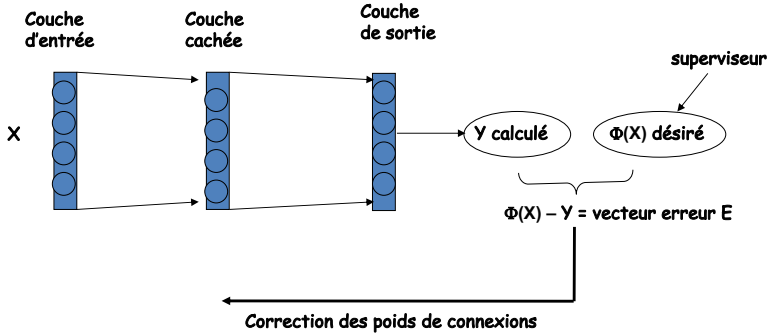
Modifier W équivaut à **changer** la fonction calculée.

Exemple de problème d'apprentissage



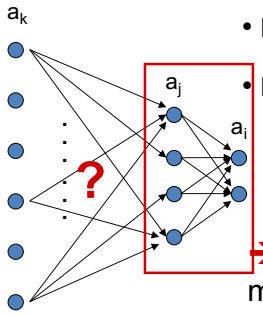
Règle d'apprentissage

- Algorithme de rétropropagation du gradient (*back-propagation*) :



Approche intuitive

cas du Single Layer Perceptron (SLP)



- Erreur des sorties : Err_i
- Erreur modifiée : $\Delta_i = Err_i \times f'(in_i)$
- Règle de mise à jour des poids :

$$w_{ji} \leftarrow w_{ji} + (\lambda \times a_j \times \Delta_i)$$

➔ Pour la couche de sortie, la règle de mise à jour ne change pas.

Mais comment mettre à jour les poids antérieurs ?

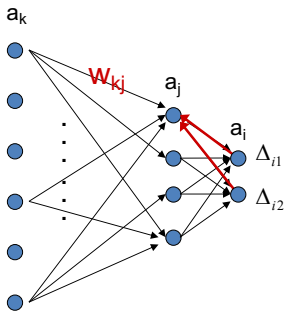
Propagation de l'erreur

- Nécessité de définir une quantité analogue à l'erreur Err_i des neurones de la couche de sortie. D'où l'idée de la **rétropropagation**.
- Propagation de Δ_i:

$$\Delta_j = f'(in_j)\sum_i w_{ji}\Delta_i$$

- Règle de mise à jour :

$$w_{kj} \leftarrow w_{kj} + (\lambda \times a_k \times \Delta_j)$$



Algorithme de rétropropagation

function BackProp-learning (examples E, network N)
Input : E is a set of examples $\{(e_1,\Phi(e_1)), (e_2,\Phi(e_2))\ldots, (e_n,\Phi(e_n))\}$
N is a perceptron with L layers, weights W and activation function f
Repeat
  for each e in E do
    for each node j in the input layer **do** $a_j \leftarrow x_j[e]$
    for l=2 **to** L **do**
      $in_i \leftarrow \sum_j w_{ji} a_j$
      $a_i \leftarrow f(in_i)$
    for each node i in the output layer **do**
      $\Delta_i \leftarrow f'(in_i) \times (\Phi_i[e] - a_i)$
    for l=L-1 **to** 1 **do**
      for each node j in layer l **do**
        $\Delta_j \leftarrow f'(in_j) \sum_i w_{ji}\Delta_i$
      for each node i in layer l+1 **do**
        $w_{ji} \leftarrow w_{ji} + (\lambda \times a_j \times \Delta_i)$
  until some stopping criterion is satisfied
return new hypothesis W

Algorithme de rétropropagation

Quelques remarques

- « Pas » d'apprentissage : λ
  - o trop petit : convergence lente vers la solution,
  - o trop grand : risque d'oscillation.
- Heuristiques courantes :
  - o diminuer le pas au fur et à mesure
  - o Momentum

  etc.
- Un ANN avec couches cachées est capable d'approximer **toute fonction booléenne** existante, pourvu que l'on fixe convenablement le nombre de neurones dans la couche cachée.

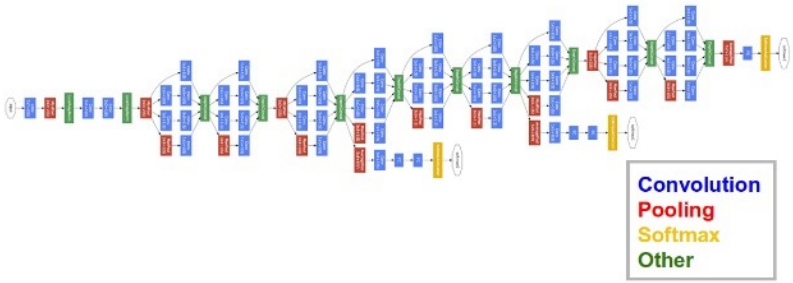
Avantages des ANN

- **Parallélisme :** le principe et le potentiel sont clairement affichés. De nouveaux formalismes et les bénéfices à en tirer restent à étudier...
- **Capacité d'adaptation :** possibilités d'auto-organisation.
- **Capacités de généralisation :** parfois spectaculaires, notamment en reconnaissance des formes sur les images et sur le texte (mais pas seulement !)
- **Simplicité** de mise en oeuvre pour de nombreux problèmes.
- Intérêt général dans les problèmes pour lesquels on connaît **peu d'informations** *a priori*.

Limitations

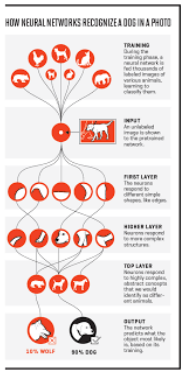
- **Performances** : ces algorithmes demandent beaucoup de ressources
- Aspect **« boîte noire »** : pouvoir explicatif souvent très limité
- **Choix de l'architecture** du réseau (hyper-paramètres)
- Problème d'optimisation complexe (taille des espaces de recherche) et difficulter de trouver l'**optimum global**

Des réseaux de plus en plus profonds

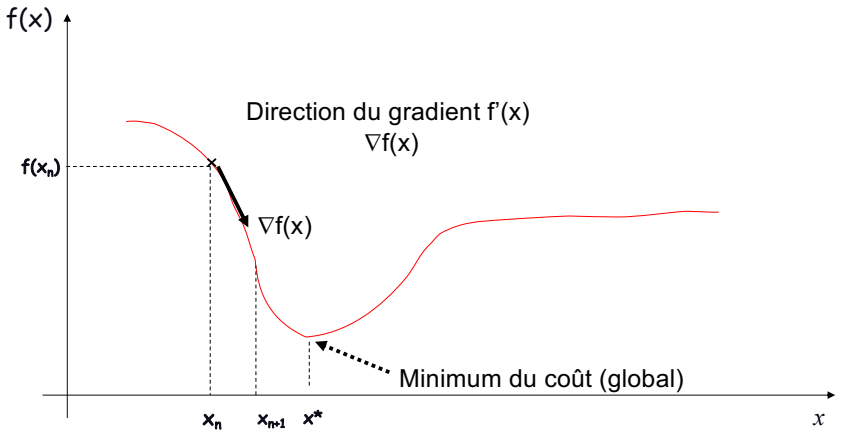


Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015).
Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).

On observe que le réseau apprend à distinguer différentes granularités dans l'information



Problème des optima locaux



Pour conclure

- Les ANN sont aujourd'hui la base du **deep learning** utilisé dans de nombreuses applications en IA :
  - reconnaissance d'objets dans des images / vidéo
  - traduction automatique
  - aide à la programmation
  - prédiction de la structure 3D de protéines
  - jeux de société et jeux vidéo
  - génération automatique d'images, de musique, de texte…