Réseaux de neurones artificiels

Historique et développements

Approches informatiques pour résoudre un problème :

- Approche algorithmique (programmation complète)
- Création des « moteurs d'inférence » (programme qui raisonne ; règles SI..ALORS.. ; système expert)
- Approche connexionniste : le réseau s'organise par apprentissage (pas de programmation)

Historique et développements

Caractéristiques de l'approche connexionniste (réseaux neuronaux)

- Calcul non-algorithmique
- Information et mémoire distribuée dans le réseau
- Architecture massivement parallèle (processeurs élémentaires interconnectés)
- Apprentissage par entraînement sur des exemples
- Inspiré du fonctionnement du cerveau

Historique et développements

- 1943 J.Mc Culloch et W.Pitts établissent le "modèle logique" du neurone qui ouvre la voie à des modèles techniques.
- 1949 D.Hebb élabore une théorie formelle de l'apprentissage biologique par modifications des connexions neuronales.
- 1957 F.Rosenblatt réalise le Perceptron, le premier modèle technique basé sur la modification des poids.
- 1960 B.Widrow réalise Adaline (Adaptive Linear Element),un réseau adaptatif de type perceptron.
- 1969 M.Minsky et S.Papert émettent des critiques et démontrent les limites des modèles neuronaux de type perceptron.
- La recherche s'arrête durant un peu plus d'une dizaine d'années.
- 1982 J.Hopfield (physicien) propose une nouvelle approche des réseaux neuronaux basée sur l'analogie avec les milieux à grand nombre de particules. Cela relance l'intérêt pour les réseaux neuronaux
- depuis 1984 : développement croissant du domaine connexionniste aussi bien en IA qu'en informatique.

Historique et développements

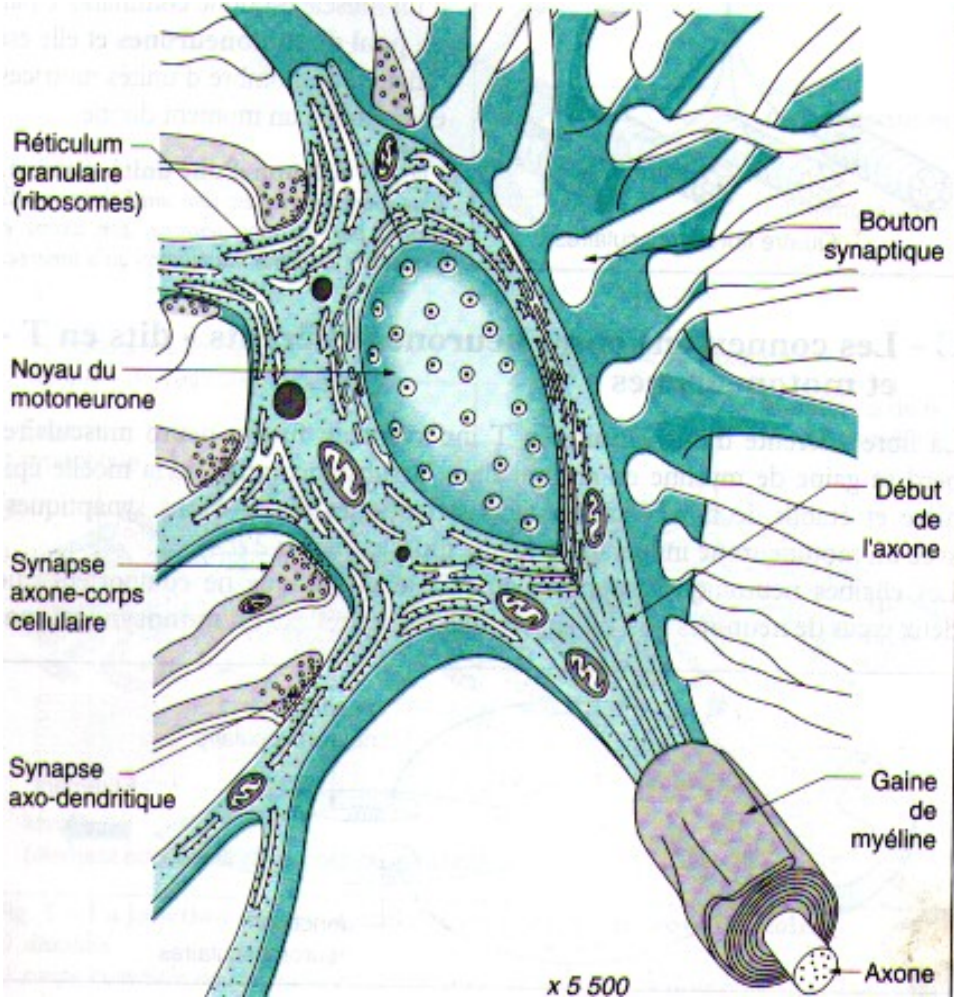
Applications

- Traitement des images
- Identification des signatures
- Reconnaissance des caractères (dactylos ou manuscrits)
- Reconnaissance de la parole
- Reconnaissance de signaux acoustiques (bruits sous-marins, …)
- Extraction d'un signal du bruit
- Contrôle de systèmes asservis non-linéaires (non modélisables)
- Robotique (apprentissage de tâches)
- Aide à la décision (domaine médical, bancaire, management, …)

Modélisation du neurone

Le neurone réel

- Dans un cerveau, il y a 10^12 neurones avec 10^3 à 10^4 connexions par neurone.
- Dendrite : récepteur des messages
- Corps : génère le potentiel d'action (la réponse)
- Axone : transmet le signal aux cellules suivantes
- Synapse : jonction axone - dendrite (plus ou moins passante)
- Neurone : élément autonome dépourvu d'intelligence

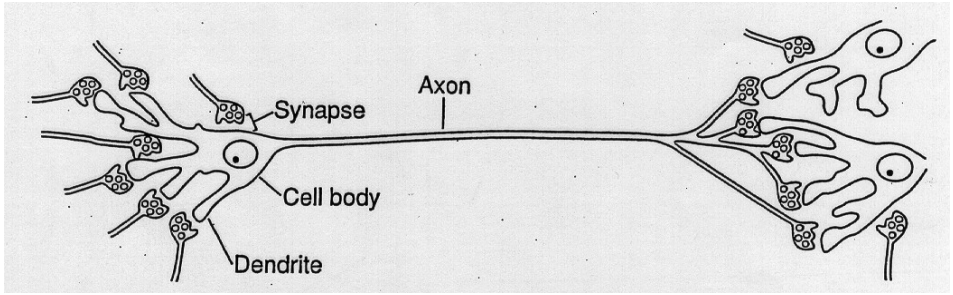


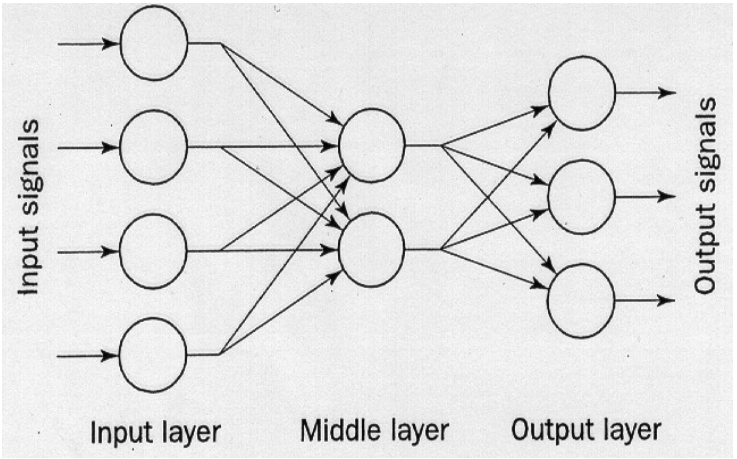
Réseau de neurones artificiel (RNA)

- Un modèle de calcul inspiré du cerveau humain.

- Cerveau humain :
  - 10 milliards de neurones
  - 60 milliards de connexions (synapses)
  - Un synapse peut être inhibant ou excitant.

- RNA :
  - Un nombre fini de processeurs élémentaires (neurones).
  - Liens pondérés passant un signal d'un neurone vers d'autres.
  - Plusieurs signaux d'entrée par neurone
  - Un seul signal de sortie



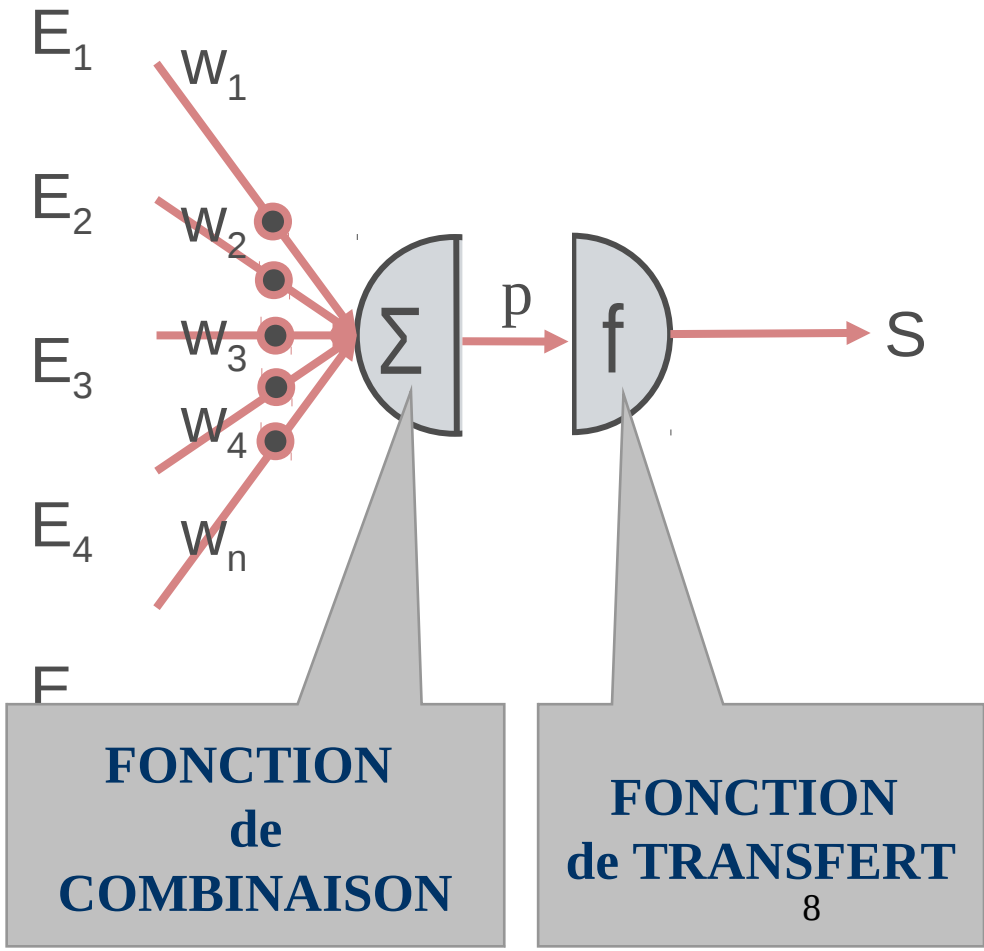


Cerveau	RNA
cellule (soma)	neurone
dendrites	entrées
synapses	poids
axon	sortie

Modélisation du neurone

Les éléments constitutifs du neurone artificiel

- Les entrées "**E**" du neurone proviennent soit d'autres éléments "processeurs", soit de l'environnement.
- Les poids "**W**" déterminent l'influence de chaque entrée.
- La fonction de combinaison "**p**" combine les entrées et les poids.
- La fonction de transfert calcule la sortie "**S**" du neurone en fonction de la combinaison en entrée.

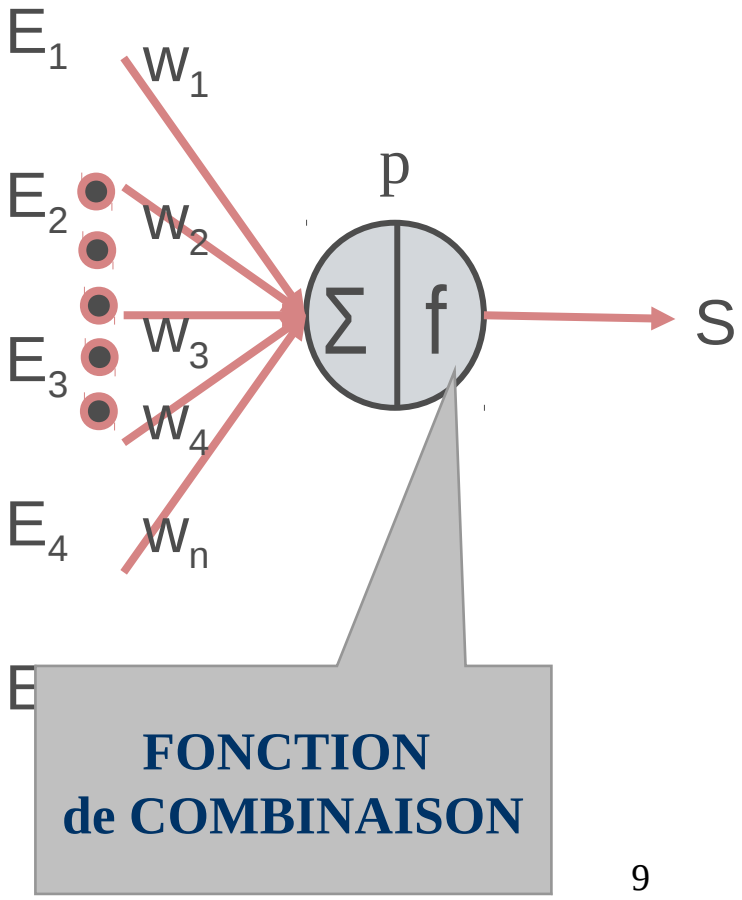


Modélisation du neurone

La Fonction de Combinaison calcule l'influence de chaque entrée en tenant compte de son poids. Elle fait la somme des entrées pondérées :

$$p = \sum W_i E_i$$

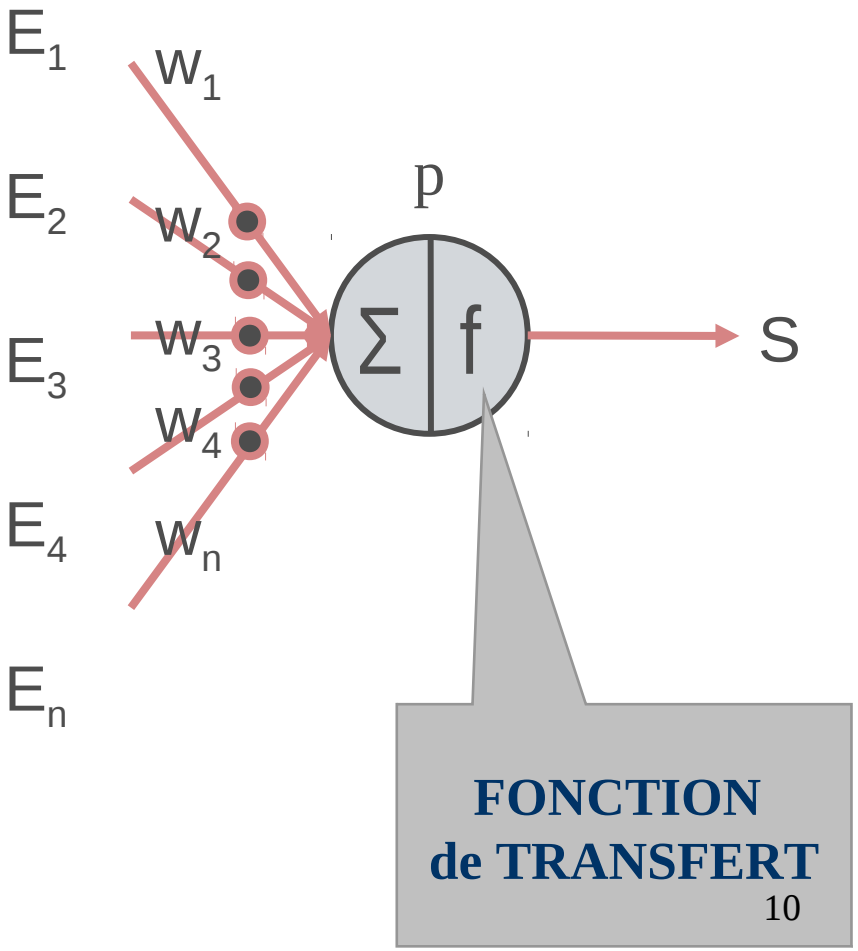
- W_i :
- Poids de la connexion à l'entrée i.
- E_i :
- Signal de l'entrée i.



Modélisation du neurone

La Fonction de Transfert détermine l'état du neurone (en sortie)

- Calcul de la sortie :
- ***S = f(p)***
- ou encore :
- ***S = f(∑ W_i E_i)***
- La fonction de transfert "**f**" peut avoir plusieurs formes.



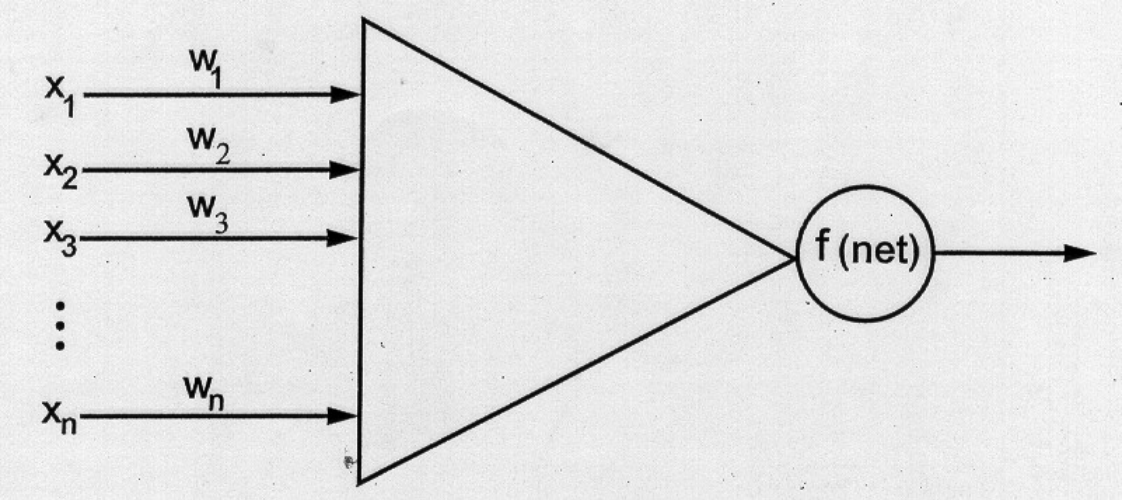
Modélisation du neurone

La fonction ‘f’ peut être de la forme :

- Fonction en échelon.
- Fonction linéaire par morceaux.
- Fonction dérivable (sigmoïde).

Comment un neurone calcule sa sortie ?

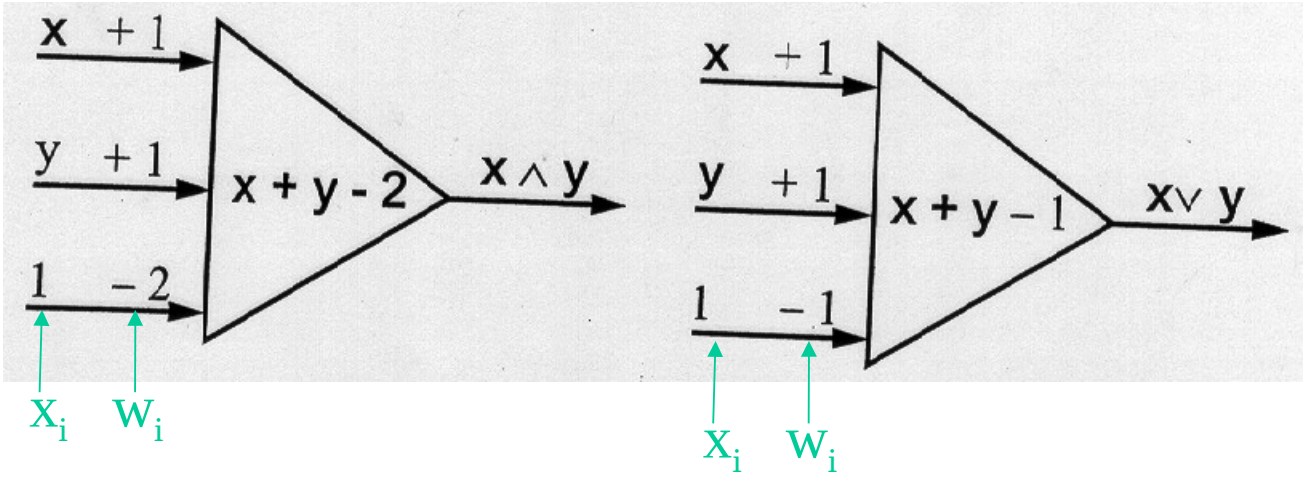
[McCulloch-Pitts, 1943]



- $net = \sum_{i=1}^{n} w_i x_i$
- f(net)= +1 si net ≥ 0, -1 sinon.
- C.à-d. : f(net) = sign(net)

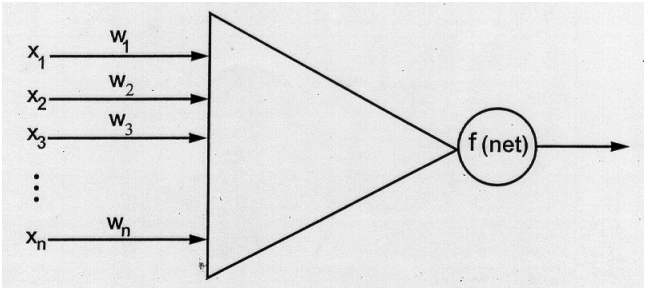
Comment un neurone calcule sa sortie ?

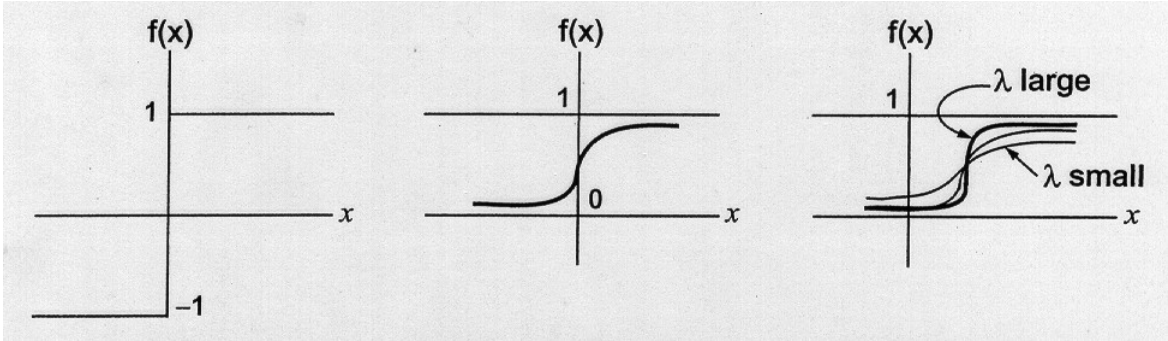
[McCulloch-Pitts, Exemples]



x	y	x+y-2	∧	x+y-1	∨
1	1	0	1	1	1
1	0	-1	-1	0	1
0	1	1	-1	0	1
0	0	-2	-1	-1	-1

Fonction d'activation sigmoïde



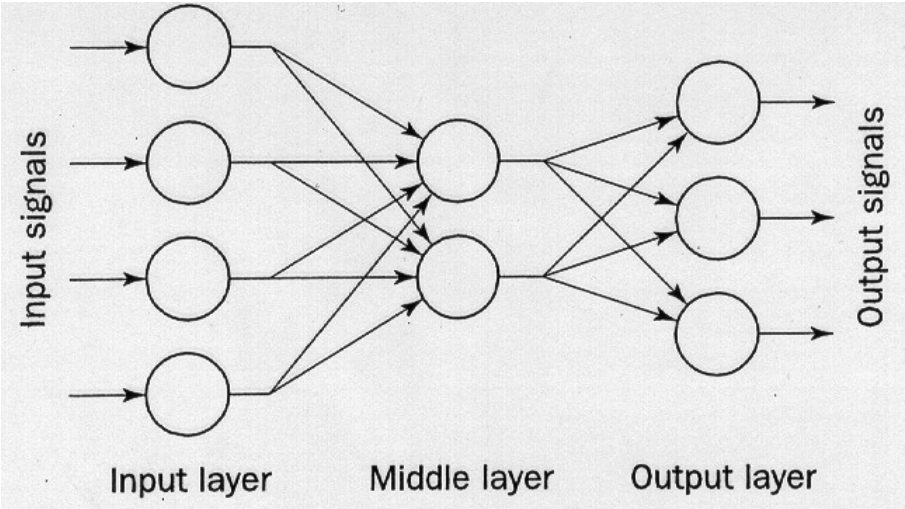


$$x = \sum_{i=1}^{n} w_i x_i$$

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

$f(x) = sign(x)$

Comment un RNA apprend ?



- Les liens entre les neurones ont des poids numériques.
- Un poids reflète la force, l'importance, de l'entrée correspondante.
- La sortie de chaque neurone est fonction de la somme pondérée de ses entrées.
- Un RNA apprend en ajustant ses poids itérativement jusqu'à ce que les sorties soient en accord avec les entrées.

Le perceptron

Le perceptron

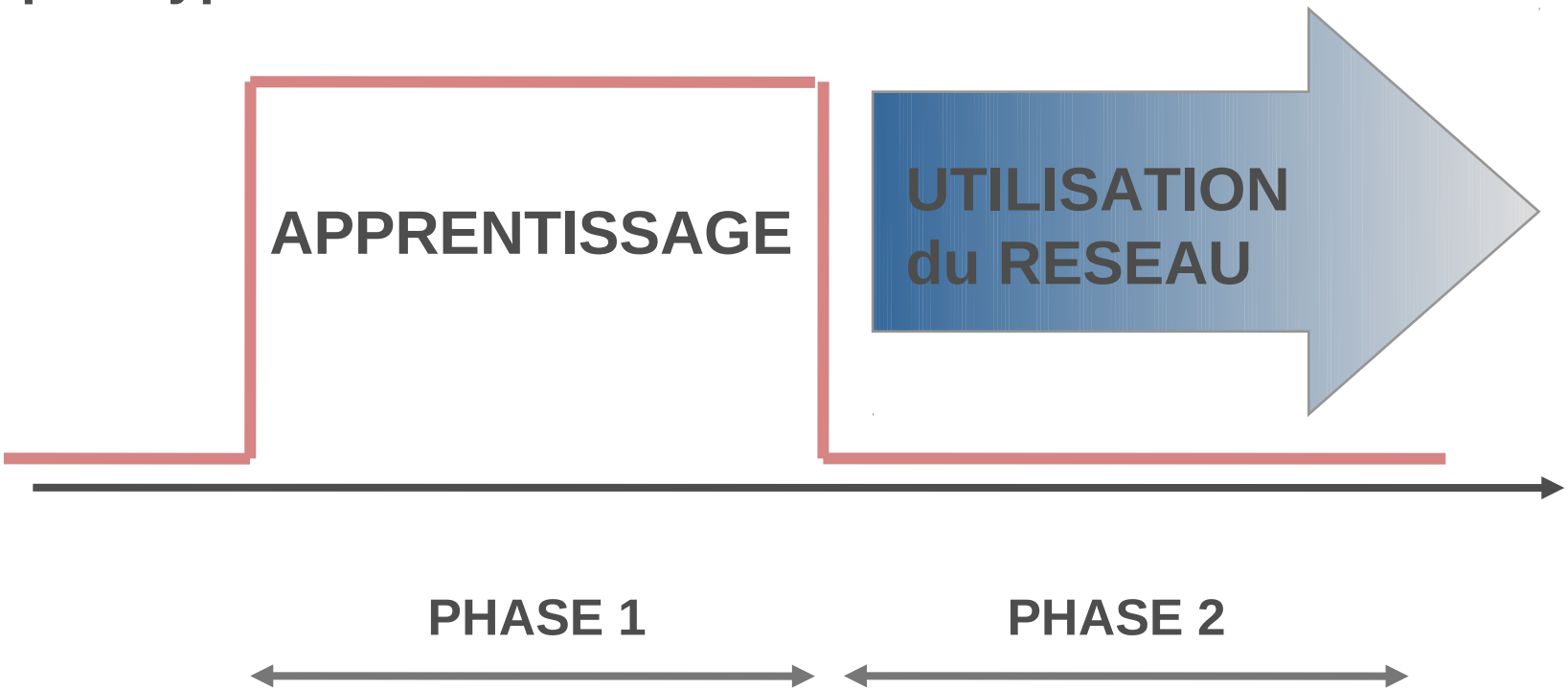
Les phases apprentissage-utilisation

- Phase 1: **APPRENTISSAGE**, le concept du Perceptron est basé sur un algorithme d'apprentissage dont l'objectif est de corriger les poids de pondération des entrées afin de fournir une activité (en sortie) en adéquation avec les éléments à apprendre.

- Phase 2: **UTILISATION**, une fois les exemples appris, chaque neurone active ou non sa sortie (en correspondance avec le domaine acquis), en fonction des éléments appliqués en entrée.

Les phases apprentissage-utilisation

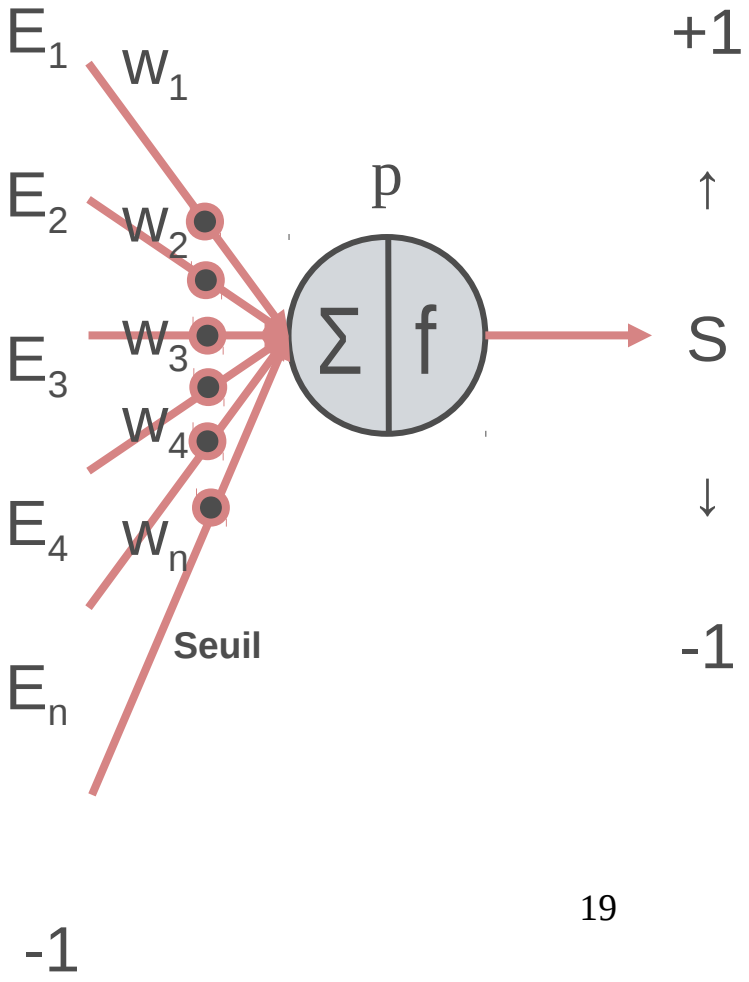
Toute utilisation du réseau doit être précédée d'une phase d'apprentissage durant laquelle on lui présente des exemples type.



L'algorithme d'apprentissage

Classes et séparabilité linéaire

- Le perceptron est un **classificateur linéaire**
- Il réalise une partition de son espace d'entrée (E1,…,En) en deux, ou plusieurs classes C1, …, Cm. séparables linéairement
- On considère deux classes :
  - C1 (S = +1)
  - C2 (S = -1)



L'algorithme d'apprentissage

Algorithme du perceptron (algorithme de principe)

```
1: INITIALISATION: W1 et W2:[-1;+1], SEUIL et PAS:[0;+1]
   Base d'apprentissage: (E1;E2)-> "Sortie correcte"
   ET logique:(+1;+1)->+1;(-1;+1)->-1;(-1;-1)->-1;(+1;-1)->-1
2: REPETER
3:   POUR chaque exemple de la base:(E1;E2)-> "S_correcte"
4:     Calcul de la sortie "S_calculée" pour chaque exemple:
4a:          Fonction de combinaison:(sa sortie"p": potentiel)
             "p" = (W1 x E1) + (W2 x E2) - SEUIL
4b:          Fonction d'activation:
             SI("p")>= 0  ALORS "S_calculée" = +1
                          SINON "S_calculée" = -1
5:     SI la sortie "S_calculée" est différente de la sortie
         "S_correcte" (ERREUR = "S_correcte" - "S_calculée")
       ALORS  Modification des poids:
5a:          W1(t+1)= W1(t) + ( E1 x PAS x ERREUR )
5b:          W2(t+1)= W2(t) + ( E2 x PAS x ERREUR )
6: Revenir à 4 pour recalculer la sortie
7: TANTQUE une ERREUR subsiste revenir à 4
```

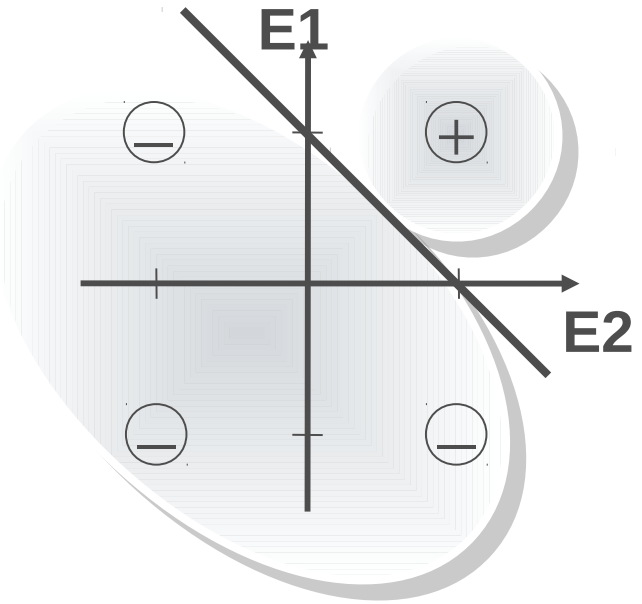
Apprentissage des fonctions logiques

Simulation de la fonction ET logique par apprentissage:

- Dans le cas de la fonction ET la séparation des deux classes se fait par une ligne droite:
  - **W1xE1 + W2xE2 - SEUIL = 0**

  Deux classes (deux réponses):
  - C1 (S = +1):
    - **Pour les entrées: (+1;+1)**
  - C2 (S = -1):
    - **Pour les entrées:**

    {(-1;-1); (+1;-1); (-1;+1)}



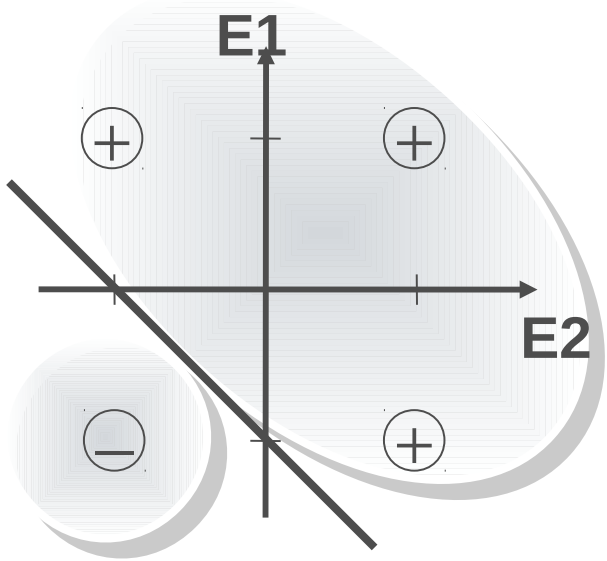
« ET » logique

Apprentissage des fonctions logiques

Simulation de la fonction OU logique par apprentissage:

- Dans le cas de la fonction OU, une droite permet toujours la séparation des deux classes.

- Pour deux classes :
  - C1 (S = +1):
    - **{(+1;+1); (+1;-1);(-1;+1)}**
  - C2 (S = -1):
    - **(-1;-1)**

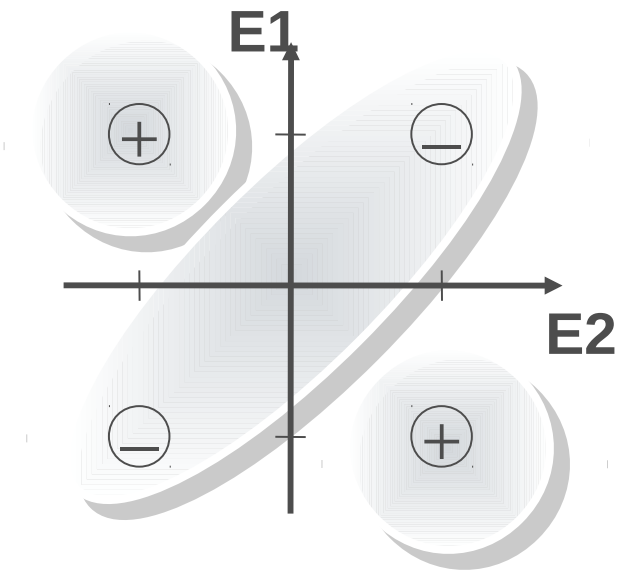


« OU » logique

Les limites du perceptron

Les limites du perceptron : la fonction logique OU exclusif.

- Dans le cas de la fonction **OU-EXCLUSIF**, la séparation des deux classes ne peut se faire par une droite mais par une courbe.
  - C1 (S = +1):
    - **{(-1;+1); (+1;-1)}**
  - C2 (S = -1):
    - **{(+1;+1); (-1;-1)}**



« OU-Exc » logique

Les limites du perceptron

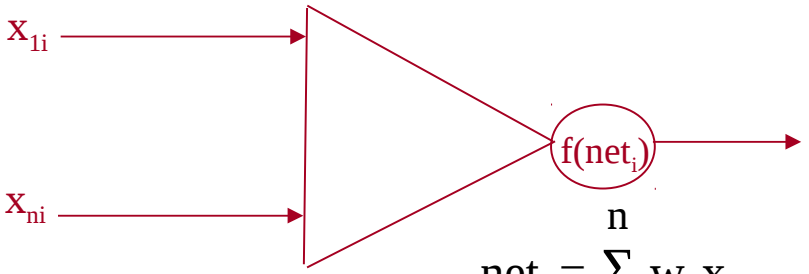
- Le perceptron est un classificateur linéaire.
- Il ne peut traiter les problèmes non linéaires du type OU EXCLUSIF, COMPARATEUR (et bien d'autres…).
- La structuration d'un réseau neuronal (constitué de plusieurs couches), et l'utilisation conjointe d'un algorithme d'apprentissage approprié vont permettre de pallier les limites identifiées ici.

Apprentissage d'un perceptron avec une fonction d'activation sigmoïde

Régression logistique

- **Idée**: plutôt que de prédire une classe, prédire une probabilité d'appartenir à la classe 1

$$p(y=1|\mathbf{x}) = h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w}\cdot\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}$$



o_i : sortie du neurone i

$_{di}$: sortie désirée pour le neurone i

c : pas d'apprentissage : $0 \le c \le 1$

Δw_{ji} : ajustement du poids de l'entrée j au neurone i.

$$net_i = \sum_{j=1}^{n} w_{ji}x_{ji}$$

$$o_i = f(net_i) = \frac{1}{1+e^{-\lambda net_i}}$$

$f'(net_i) = \lambda \times f(net_i) \times (1-f(net_i))$

Algorithme d'apprentissage

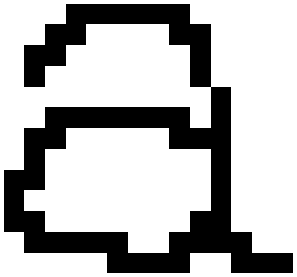
Pour chaque donnée d'entraînement, incrémenter le poids w_{ji} par :

$$\Delta w_{ji} = c(d_i - o_i) f'(net_i) x_{ji}$$

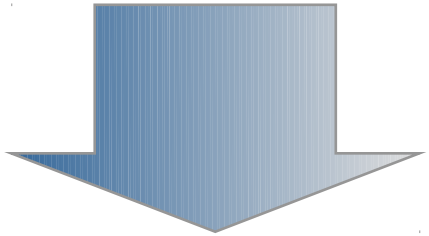
Exemple d'apprentissage

Apprentissage supervisé (ex: OCR)

- Association imposée entre un vecteur d'entrée (forme multidimensionnelle) et un vecteur de sortie (la réponse désirée).



- L'erreur est calculée à chaque essai afin de corriger les poids.



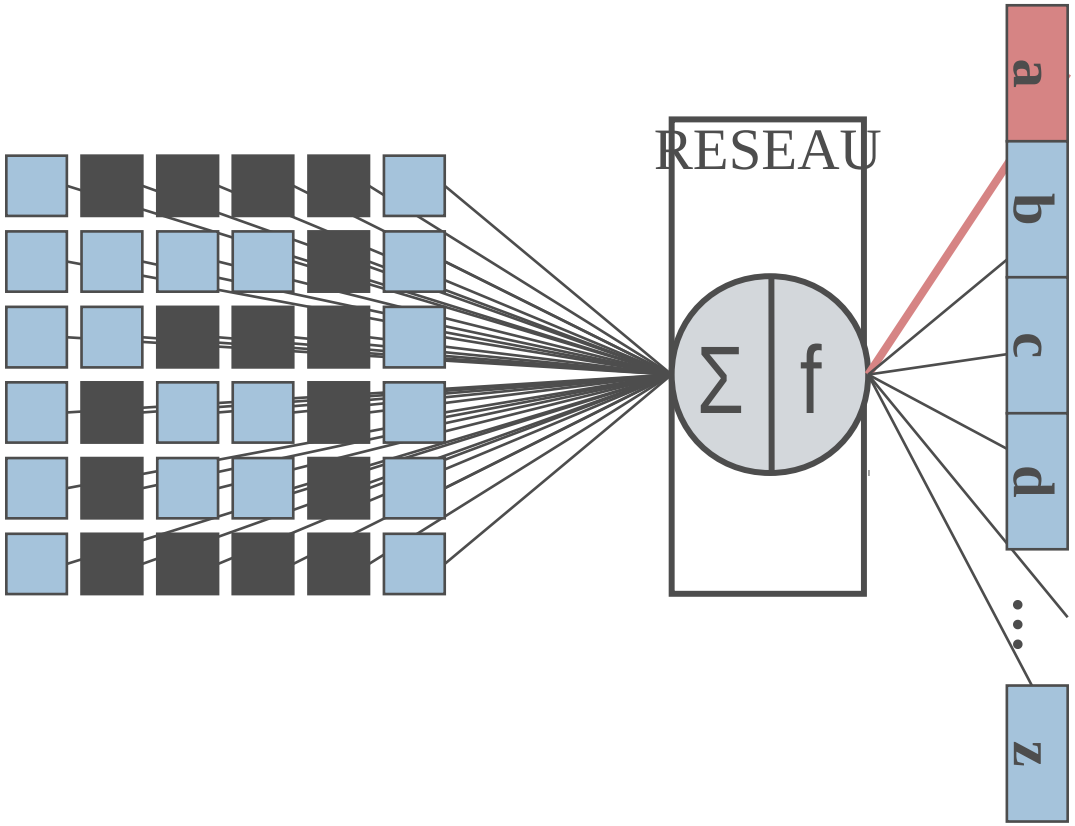
- Les poids sont modifiés jusqu'à l'erreur minimale, voire aucune erreur.

a

Exemple d'apprentissage

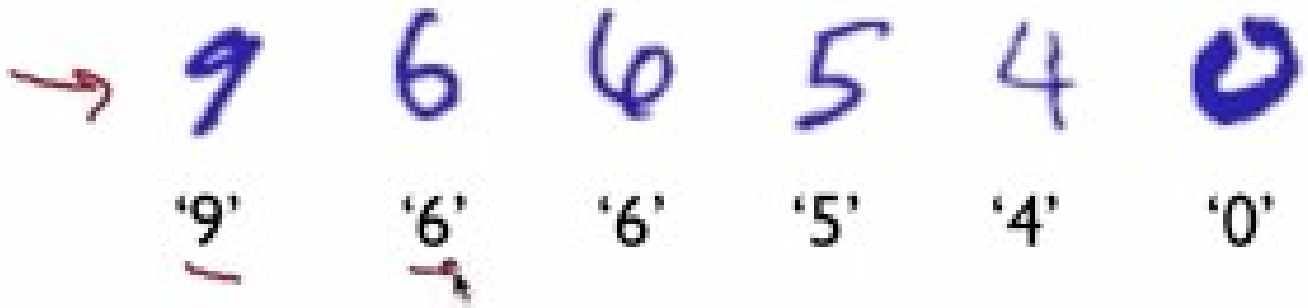
Apprentissage supervisé (ex: OCR)

- La réponse attendue est le "a". Un seul et unique vecteur de sortie doit être activé.



Autre exemple : chiffres manuscrits

on fournit à l'algorithme des **données d'entraînement** ...



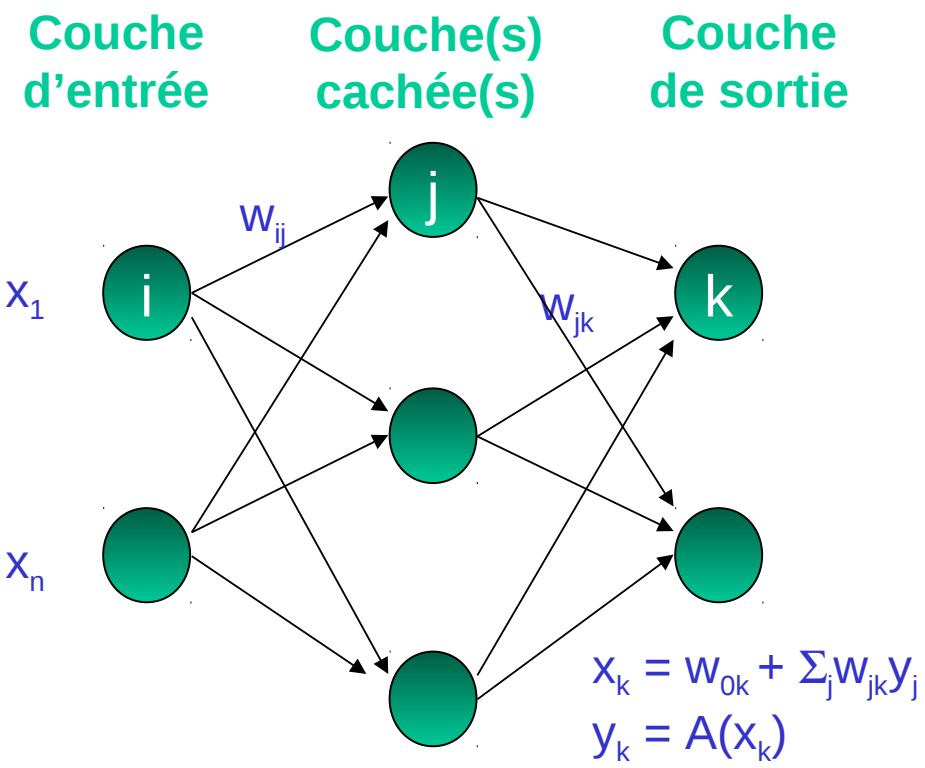
'9' '6' '6' '5' '4' '0'

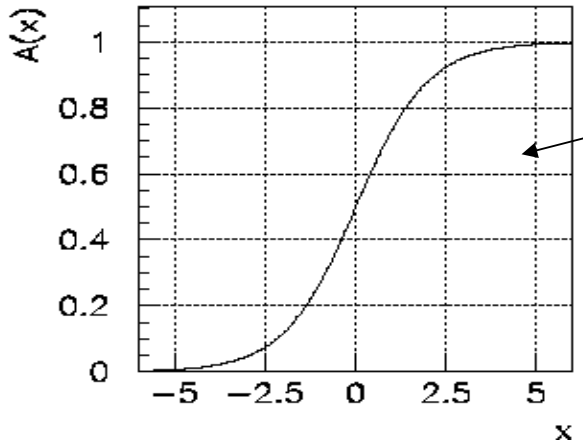
... et l'algorithme retourne un «programme» capable de **généraliser** à de nouvelles données

? ? ? ? ? ?

Perceptron MultiCouche (MLP)

Le perceptron multi-couches





- Neurones organisés en couches
  - 0, 1, 2… couches cachées
  - Réseau $n_1 - n_2 - n_3$
- Information se propage dans un seul sens (« feed-forward »)
- La fonction A(x) (fonction d'activation) est
  - linéaire ou non pour le (ou les) neurone(s) de la couche de sortie
  - non-linéaire pour les neurones cachés, de type $1/(1+e^{-x})$ (« sigmoïde »)
- Les paramètres ***w*** s'appellent les poids (« weight »)
- **y** = mlp(**x**, ***w***)

La détermination des poids *w*

- Les paramètres ***w*** sont déterminés de façon itérative durant une phase d'apprentissage afin de résoudre un problème donné

- Méthode:
  1. Boucler sur les exemples d'un lot d'apprentissage
  2. Comparer pour chaque exemple la sortie du réseau et la sortie désirée (« apprentissage supervisé »)
  3. Modifier après chaque exemple les poids du réseau (transparent suivant)
  4. Recommencer en 1 « plusieurs fois »

- Après la phase d'apprentissage, le réseau doit être capable de généraliser (donner une réponse correcte sur un exemple nouveau)

Modifier les poids du réseau

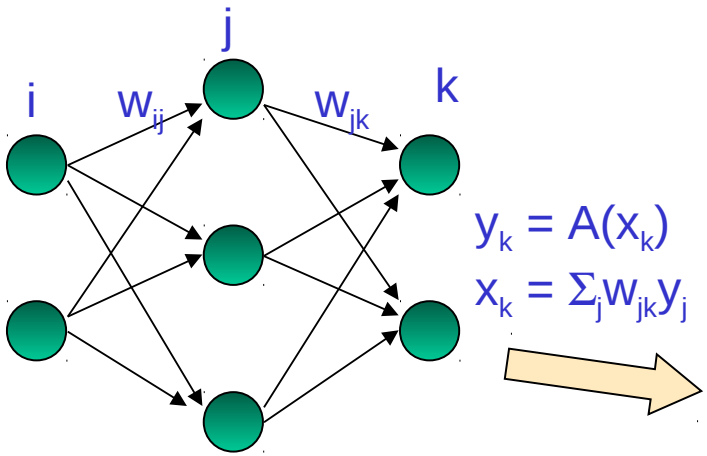
- La méthode pour modifier les poids du réseau est souvent appelée « méthode de la rétropropagation de l'erreur »

La rétropropagation de l'erreur

1. **Présenter un exemple au réseau, calculer la sortie y_k, comparer à la réponse attendue**

$$d_k \rightarrow \delta_k = A'(x_k)(y_k - d_k)$$

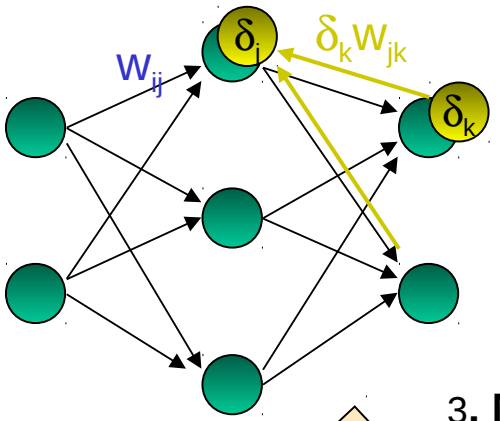
Poids initiaux aléatoires petits autour de 0 (par exemple uniformes entre -0.5 et 0.5)



$$y_k = A(x_k)$$
$$x_k = \Sigma_j w_{jk} y_j$$

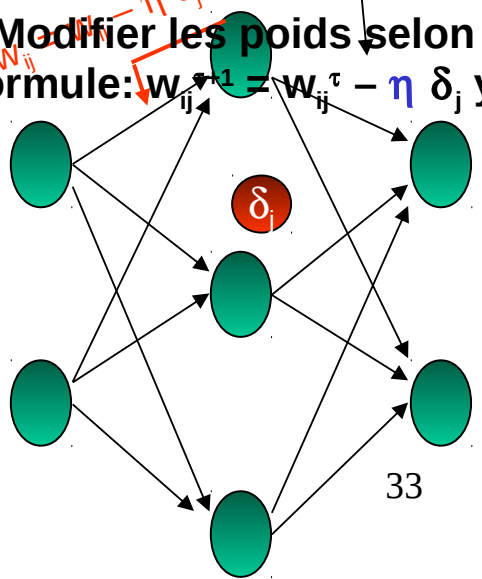
2. Ca**lculer $\delta_j = A'(x_j) \Sigma_k \delta_k w_{jk}$**

« Rétropropagation de l'erreur »



Paramètre d'apprentissage < 1, à optimiser selon le problème

3. **Modifier les poids selon la formule: $w_{ij}^{\tau+1} = w_{ij}^{\tau} - \eta\ \delta_j\ y_i$**



Exemple dans une image

- Pour une image x = (x1,…,xk) en entrée et un ensemble de poids synaptiques w on commence par définir une fonction de coût C qui mesure l'écart entre les prédictions yj(x) des neurones de sortie et les valeurs cibles tj spécifiées dans l'ensemble d'entraînement :

$$C(\mathbf{x};\mathbf{w}) = \sum_{j=0}^{9} \left(y_j(\mathbf{x}) - t_j\right)^2 \qquad (2)$$

- Naturellement yj(x) est une fonction extrêmement compliquée de la configuration x en entrée et des poids w, spécifiée par l'itération de (1). Ce que l'on cherche à minimiser en principe c'est la somme ou la moyenne CMSE(w) de ces erreurs sur toutes les configurations x de l'ensemble d'entraînement E :

$$C_{\mathrm{MSE}}(\mathbf{w}) = \sum_{\mathbf{x}\in\mathcal{E}} C(\mathbf{x};\mathbf{w}) \qquad (3)$$

Exemple dans une image :Méthode de la rétropropagation de l'erreur : explications

- On cherche à minimiser la somme ou la moyenne des erreurs sur toutes les configurations x de l'ensemble d'entraînement E

- Une descente de gradient consiste à calculer la direction dans l'espace des poids w dans laquelle la décroissance de la somme des erreurs est maximale.

- Si tout se passe bien, c.à.d. qu'il n'y a pas de minima locaux, on s'approchera du minimum par itérations successives de petites corrections apportées à w :

$$\mathbf{w}^{(\text{new})} = \mathbf{w}^{(\text{old})} - \alpha \nabla C_{\text{MSE}}\left(\mathbf{w}^{(\text{old})}\right) \qquad (4)$$

- Le coefficient α permet d'ajuster la vitesse d'apprentissage, le cas échéant dynamiquement.

- Dans la pratique cependant, le calcul de la somme sur toutes les configurations $x \in E$ est impossible car beaucoup trop coûteux en temps de calcul.

Exemple dans une image : Méthode de la descente de gradient stochastique (SGD)

- approximer à chaque étape le gradient $\nabla CMSE(w)$ par le gradient $\nabla C(x;w)$ d'un seul échantillon x tiré au hasard dans l'ensemble d'entraînement E

- Il faut alors calculer toutes les composantes $\partial C(x;w)/(\partial wij)$ du gradient $\nabla C(x;w)$. On utilise alors l'algorithme de retropropagation de l'erreur

- on montre en effet que les dérivées par rapport aux wij de la couche n – 1 sont calculables dès lors celles de la couche n ont déjà été calculées

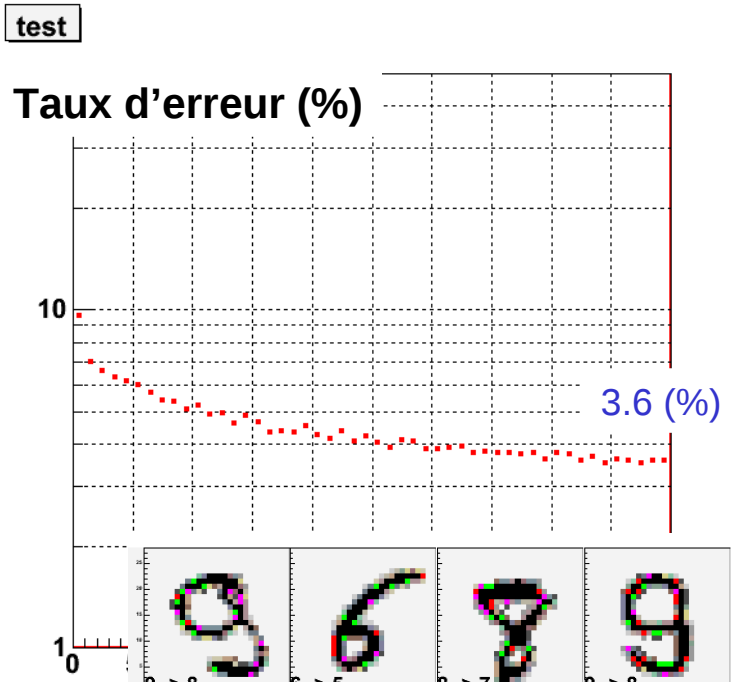
Un exemple grandeur nature: un réseau qui apprend à lire

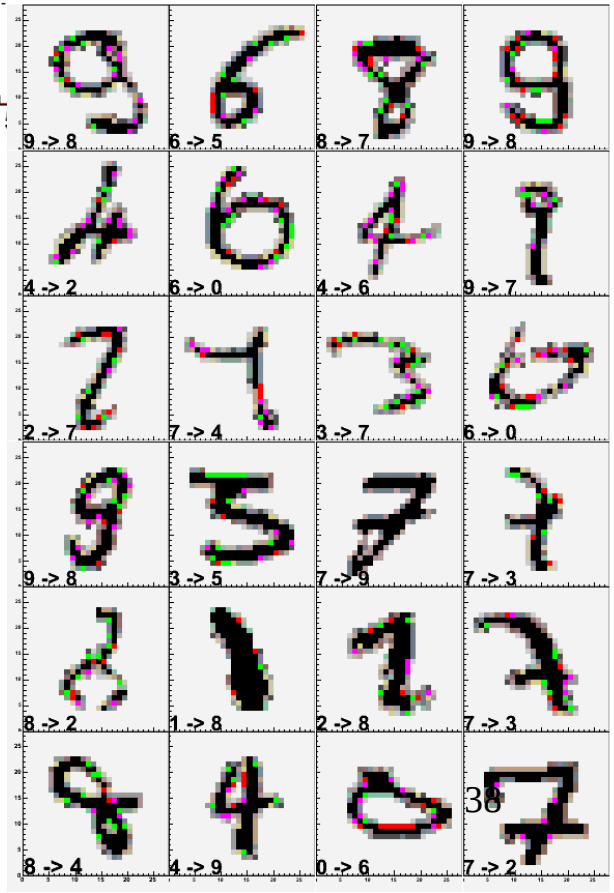
- Reconnaissance de chiffres manuscrits
- Base de données de 60000 + 10000 chiffres manuscrits (taille 28 x 28 pixels) disponible sur: http://yann.lecun.com/exdb/mnist/
- Entrées = 784 pixels, valeurs entre 0 et 255 (ramenées entre 0 et 1 en entrée du réseau)
- Classification dans 10 catégories → 10 neurones de sortie

Résultats

- Exemple: réseau 784 – 300 – 10 → ~ 238000 paramètres !

- Taux d'erreur:
  - On attribue à chaque exemple de test la classe du neurone de sortie qui donne la plus grande valeur
  - Taux d'erreur ~ 3.6% après 50 époques (~ 2.5 heures)
  - Erreurs sur des cas difficiles, mais aussi sur des exemples faciles…





Des exemples de chiffres mal classés

Résultats (2)

- Classification linéaire: taux d'erreur = 12%
- MLP: 2.5 – 5%
- Réseaux spécifiques (utilisent le caractère 2d du problème): 0.7 – 1%

Remarque:

Ce genre de techniques est utilisé par les banques pour la lecture automatique des chèques, la poste pour la lecture des adresses, …

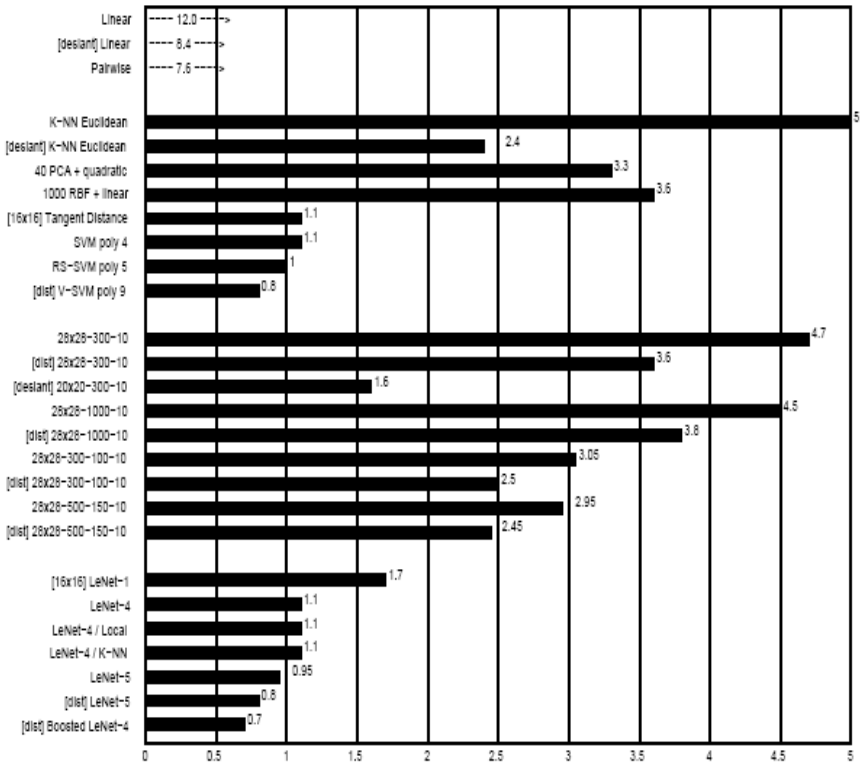


Fig. 9. Error rate on the test set (%) for various classification methods. [deslant] indicates that the classifier was trained and tested on the deslanted version of the database. [dist] indicates that the training set was augmented with artificially distorted examples. [16x16] indicates that the system used the 16x16 pixel images. The uncertainty in the quoted error rates is about 0.1%.

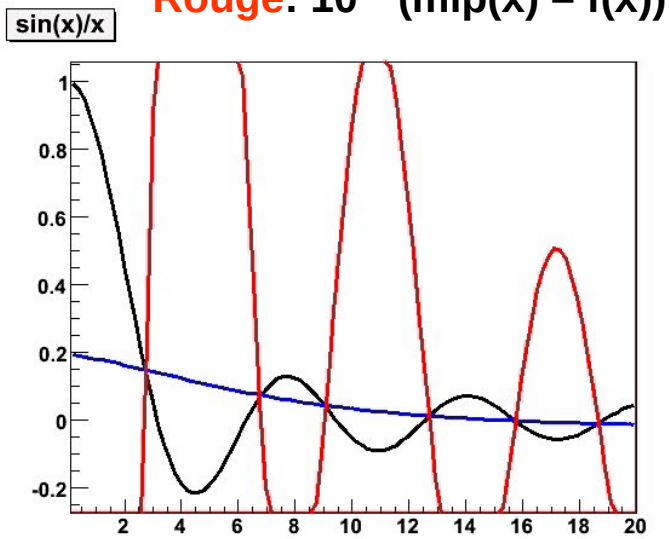
Exemple de sin(x)/x

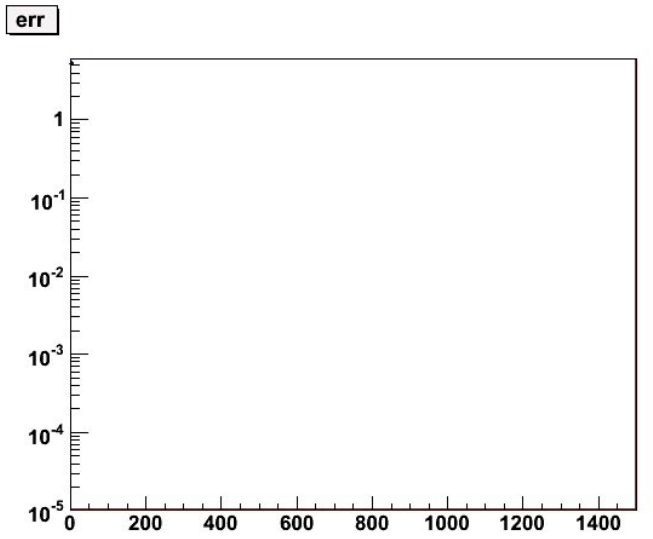
Noir: f(x) = sin(x)/x
Bleu: mlp(x)
Rouge: 10 * (mlp(x) – f(x))

- Approximation de la fonction f(x) = sin(x)/x par un réseau 1-6-1 dans le domaine 0 – 20

- Lot d'apprentissage = 100 points équidistants, lot de test = 100 points décalés des précédents

- On constate que:
  - L'ajustement commence par le plus haut pic (le réseau « apprend » d'abord le trait le plus caractéristique)
  - Il y a des paliers dans l'apprentissage (« ravins » dans la fonction d'erreur)





En résumé

- Un perceptron multi-couches peut être entraîné à « reconnaître » des formes (écriture manuscrite, signal / bruit de fond, …)
- Un MLP avec une couche cachée et un neurone de sortie linéaire est une combinaison linéaire de fonctions sigmoïdes

- Il sélectionne un sous-espace de l'espace des variables par combinaison logique de demi-espaces

- Les poids sont fixés de façon itérative pendant la phase d'apprentissage

Résumé

Perceptron : principe de séparabilité linéaire

NEURONE = Combinaison + Transfert

Rétro-propagation de l'erreur : estimation de l'erreur en couches cachées

Fonction Combinaison :

$$\sum Wi\ Ei$$

Fonction Transfert : en échelon, ou linéaire par morceaux, ou dérivable

Quelques exemples

- Regression linéaire et quadratique : prédire le prix d'une pizza en fonction de son diamètre en inches

- Un MLP avec une couche cachée et un neurone de sortie linéaire est une combinaison linéaire de fonctions sigmoïdes

Inconvénients des RN

- Le temps d'entraînement d'un RN croît rapidement lorsque le nombre de couches augmente.
  - C'est d'ailleurs l'une des raisons pour lesquelles à partir des années 1990 les RN ont été remisés au profit d'autres algorithmes non-linéaires moins gourmand en ressources comme les SVM.
- sur-apprentissage

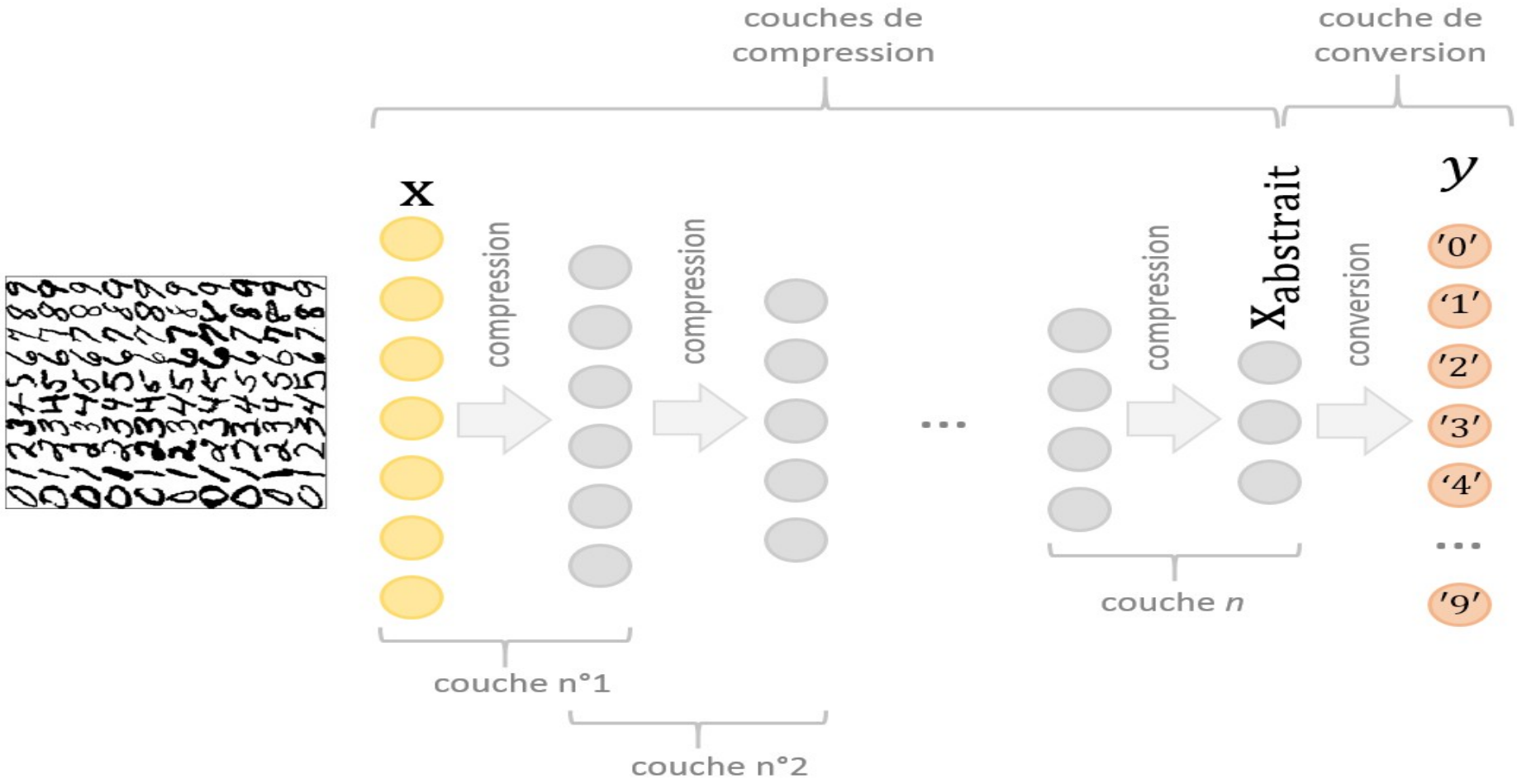
Introduction au Deep Learning

- Idées de Hinton :

- Elaborer un modèle génératif, c.à.d. un modèle pour la distribution de probabilité conjointe p(y,x) des images x et des labels y.
- Autrement dit, on cherche à construire un RN capable d'apprendre puis de générer simultanément les images x et les labels y avec une distribution de probabilité proche de celle observée dans un ensemble d'entraînement.
- Une fois le RN entraîné on pourra l'utiliser d'une part pour reconnaître des images, c.à.d. associer un chiffre y à une image x (on cherche le y qui maximise p(y|x)) ou, inversement, pour générer des images x correspondant à un chiffre y (on échantillonne p(x|y))
- http://www.cs.toronto.edu/~hinton/adi/index.htm

Introduction au Deep Learning

L'architecture de RN utilisée pour réaliser le modèle génératif (Deep Belief Network) de Hinton et al est :



Introduction au Deep Learning

L'architecture de RN utilisée pour réaliser le modèle génératif de Hinton et al est :

1) La première partie est constituée d'un ensemble de couches de compression qui convertissent les données entrées x en une représentation abstraite $X_{abstrait}$.

2) La seconde partie convertit cette représentation en labels de classification y.

3) La première partie a pour objectif d'apprendre la distribution des données x présentées en entrée sans tenir compte des labels y.

Elle est constituée d'une succession de couches dont chacune contiendra une représentation plus abstraite (ou compressée) que la précédente.

Pour un RN chargé de classifier des images (exemple de MINST) :

- La première couche stockera les niveaux de gris de l'image (l'équivalent de la rétine),

- la seconde contiendra, par exemple, un encodage des lignes ou des zones de contraste de l'image,

- la troisième détectera l'existence de certaines formes géométriques simples comme des cercles,

- la quatrième identifiera certains agencements particuliers de ces figures comme celles qui représentent un '8' formé de deux cercles juxtaposés et ainsi de suite.

Introduction au Deep Learning

- Le rôle de la seconde partie du DBN est de convertir la représentation abstraite et obscure Xabstrait en labels y utilisables par exemple dans le cadre d'un apprentissage supervisé.
- Dans l'exemple des digits cette représentation sera constituée d'une couche de sortie de dix neurones, un neurone par digit. Cette conversion peut être réalisée au moyen d'une couche logistique entraînée par une SGD classique.

- L'entraînement du DBN est considéré comme achevé lorsque la performance du DBN évaluée sur un ensemble de validation distinct de l'ensemble d'entraînement ne progresse plus significativement. Cette seconde étape est appelée le fine-tuning, elle est généralement beaucoup plus lente que l'initialisation.

Introduction au Deep Learning

- Comment implementer les couches de la première section.

- La principale caractéristique attendue de cette brique est sa capacité à apprendre rapidement une distribution de probabilité spécifiée empiriquement par un jeu d'exemples.

- Plusieurs techniques : une des premières utilisé par Hinton et al. est Restricted Boltzman Machines (RBM) en 2006.

- RBM fut exploiter dans l'algorithme dit de Contrastive Divergence de Hinton et al.

RBM : qu'est ce que c'est ?

- Les RBM sont des RN stochastiques, qui peuvent être entraînés sur un mode non supervisé pour générer des échantillons selon une distribution de probabilité complexe spécifiée par des exemples (l'ensemble d'images de digits p.ex.).

- Alors que les neurones des RN ont des niveaux d'activation xi déterministes compris entre 0 et 1,
- les neurones des RBM sont des variables aléatoires binaires.
- Elles sont réparties sur deux couches.
- L'une de ces couches est appelée la couche visible, c'est elle qui encodera les exemples de l'ensemble d'entraînement E.
- L'autre est appelée couche cachée, c'est elle qui stockera une forme compressée ou abstraite des données de la couche visible.

ALGORITHME DE CONTRASTIVE DIVERGENCE (CD)

- Les RBM utilise une version particulière de Monte-Carlo Markov-Chain MCMC dite avec échantillonnage de Gibbs pour échantillonner une distribution de probabilité multivariée $p\theta$.

- La distribution est sensée être proche de la distribution expérimentale dans E on peut accélérer la convergence de la MCMC en l'initialisant avec un échantillon tiré de E.

- Ensuite, plutôt que d'attendre patiemment la convergence de la MCMC, on arrête les itérations après un petit nombre d'étapes. Parfois, après une seule étape.

- On on commet évidement une erreur puisqu'on ne génère pas exactement $p\theta$.

- L'expérience montre cependant que les itérations de la SDG finissent par lisser cette erreur et que cette approximation grossière reste suffisante pour faire converger la fonction de coût vers un minimum approximatif utile.

Implémentations

- Deep Learning ne repose pas sur les seules avancées conceptuelles de Hinton et al. mais aussi sur des avancées technologiques (les GPU).

- en 2014 : Construction d'un cluster de 64 GPU conçu spécifiquement pour l'entraînement d'un Convolutionnal Neuronal Network (CNN) à 18 couches de neurones pleinement interconnectées, soit 212.7M de poids synaptiques (w).

Implémentations

- Codage d'un réseau de neurones ou d'un DBN
- Theano, un projet de l'université de Montréal,
- Torch, une solution maintenue par des experts œuvrant chez de grands acteurs du web (Google, Facebook ou Twitter)
- Caffe, une initiative de l'université de Berkeley
- Alternatives qui fonctionnent en sur-couche :
    - Pylearn2, une librairie Python reposant sur Theano
    - sickit-learn en Python
    - Python Keras, sur base de Theano
    - Tensorflow de Google

Resumer

- Un DBN consiste en un empilement de RBM et une couche logistique supplémentaire qui converti le contenu de la couche cachée du dernier RBM en labels de classification

L'entraînement d'un DBN complet procède alors selon les étapes suivantes :

1) la RBM n°1 est entraînée par CD avec les échantillons v de l'ensemble d'entraînement E présentés sur sa couche visible

2) une fois entraînée, cette première RBM fera office de convertisseur/compresseur. A chaque échantillon v de l'ensemble d'entraînement E elle associe une configuration h de neurones de la couche cachée. On construit h en échantillonnant les distributions $p\theta(hi \mid v)$ avec les paramètres θ appris à l'étape précédente. On peut montrer que les configurations h ainsi obtenues sont des versions compressées des v. Les h sont utilisées comme échantillons d'entraînement présentés à la couche visible de la RBM n°2

3) on itère 1. et 2. sur toutes les RBM du DBN. Une fois toutes les RBM entraînées, la phase d'initialisation est terminée. On dispose alors d'un mécanisme complexe de conversion des v en une représentation abstraite/compressée sur la couche cachée de la dernière RBM.

4) on peut alors utiliser cette représentation abstraite en entrée d'une couche logistique que l'on entraînera de manière classique avec une SGD supervisée basée sur une fonction de coût CNLL de type maximum de vraisemblance. C'est l'étape dite de fine-tuning