UNIVERSITE ABDELMALEK ESSAADI

FACULTE DES SCIENCES

TETOUAN

جامعة عبد المالك السعدي

كلية العلوم

تطوان

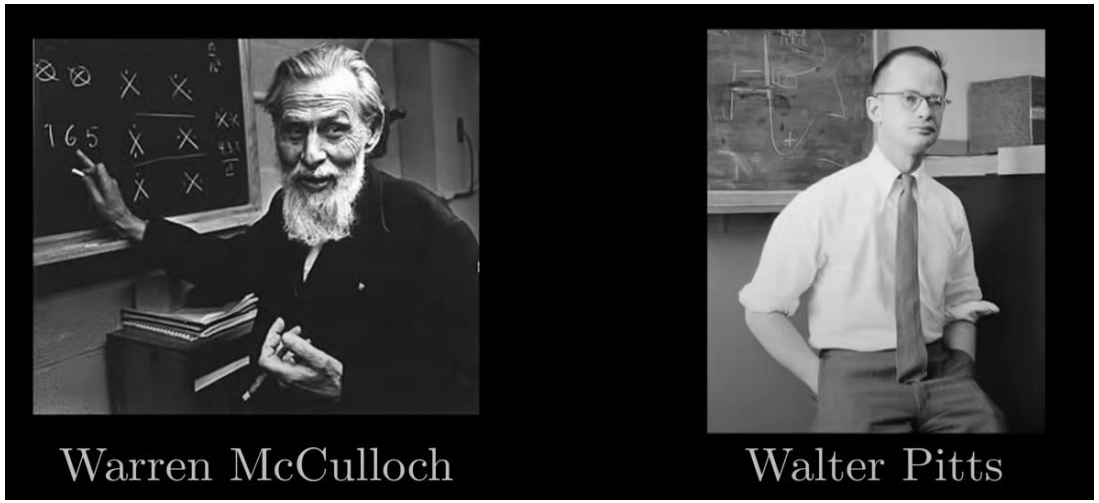
Apprentissage profond pour la vision par ordinateur

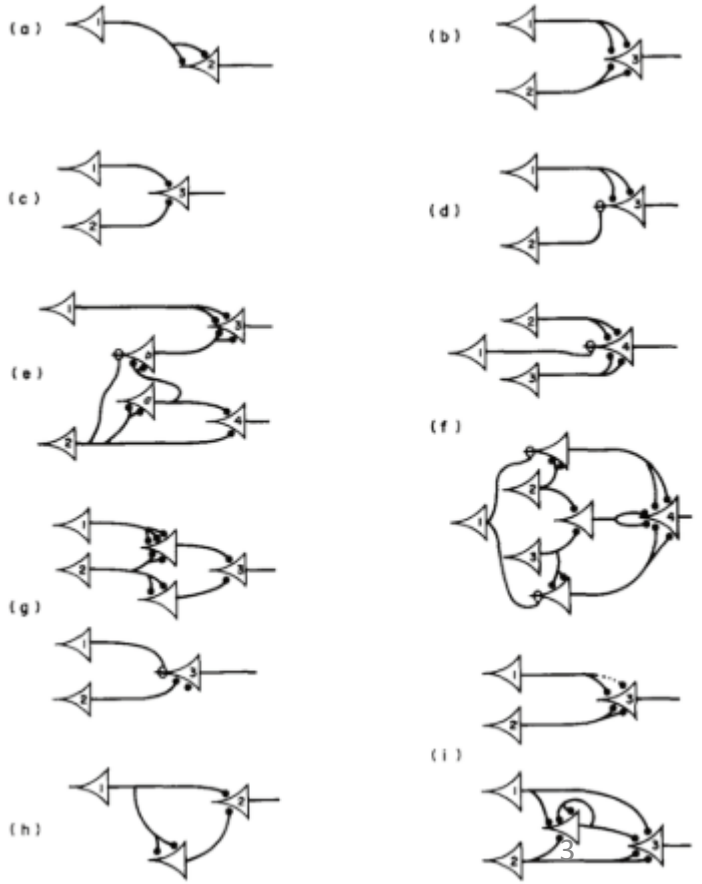
Presented by Pr. Abdellatif El Ouissari

Deep learning
Brève histoire des réseaux neuronaux

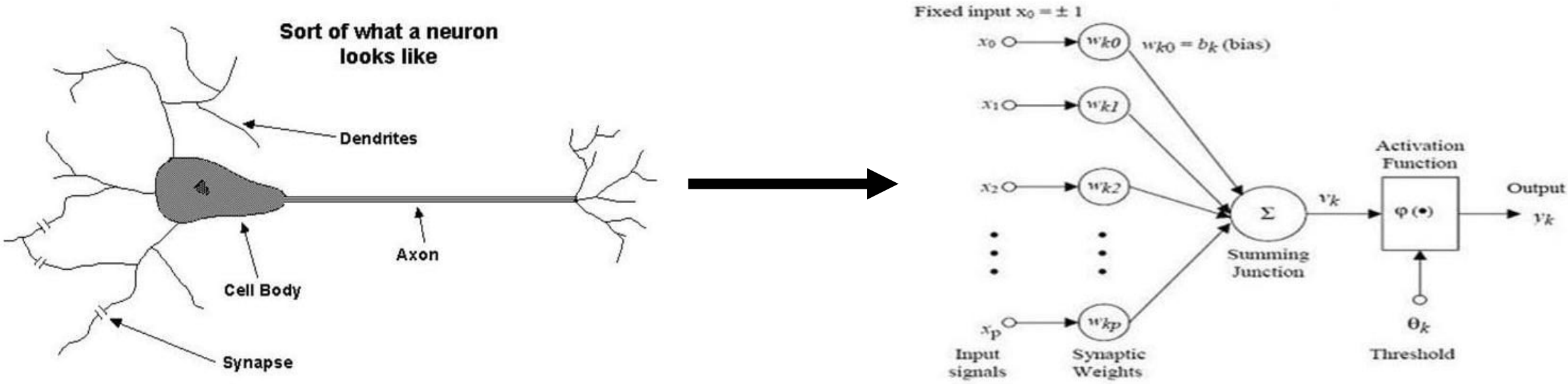
Deep learning

En 1943 deux mathématiciens Warren McCulloch et Walter Pitts ont publié un article sur les réseaux de neurons artificiels.

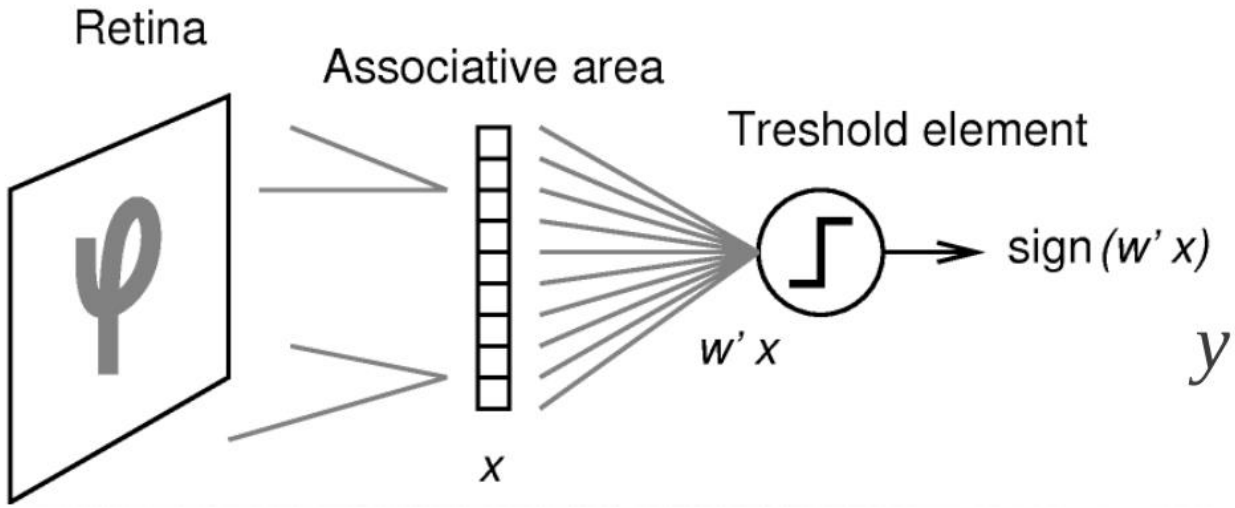




Deep learning



Deep learning

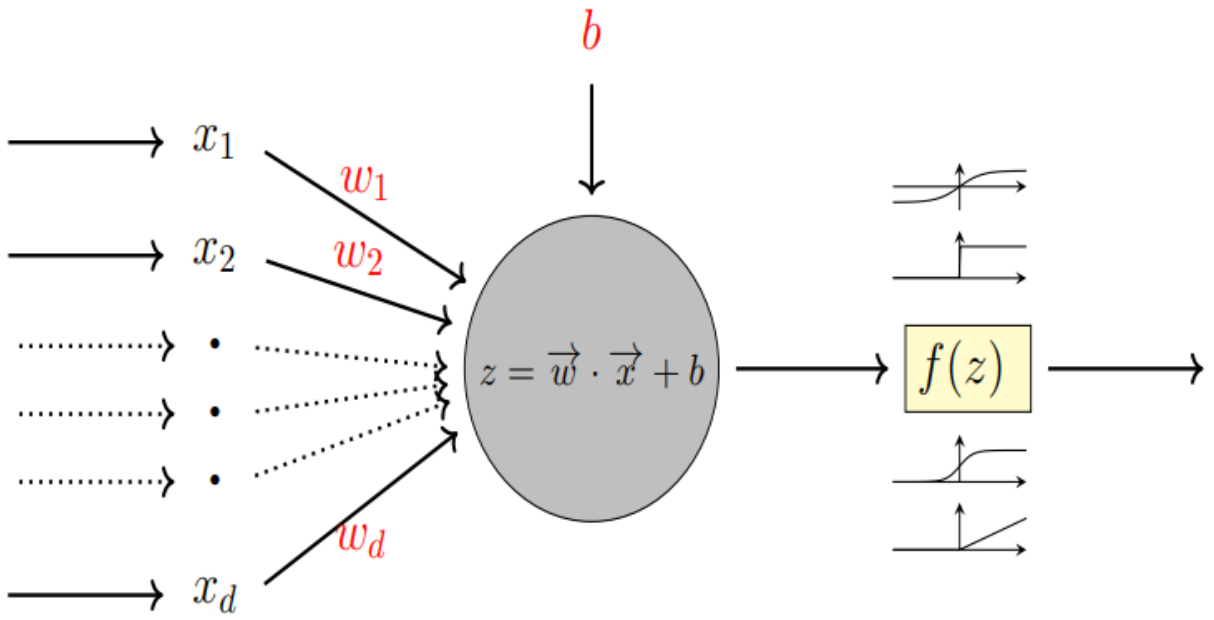


$$y = sign\left(\sum_{i=1}^{N} W_i X_i + b\right)$$

Deep learning

Perceptron : simple couche

- Invented by Frank Rosenblatt (1957)



Deep learning

Algorithme du perceptron

1. pour chaque paire $(\mathbf{x}_t, y_t) \in D$
   a. calculer $h_{\mathbf{w}}(\mathbf{x}_t) = Threshold(\mathbf{w} \cdot \mathbf{x}_t)$
   b. si $y_t \neq h_{\mathbf{w}}(\mathbf{x}_t)$
      - $w_i \leftarrow w_i + \alpha(y_t - h_{\mathbf{w}}(\mathbf{x}_t))x_{t,i} \quad \forall i$ (mise à jour des poids et biais)
2. retourner à 1 jusqu'à l'atteinte d'un critère d'arrêt
   (nb. maximal d'itérations atteint ou nb. d'erreurs est 0)

$$b \leftarrow b + \alpha \cdot (y_t - y_{\text{pred}})$$

Deep learning

Application

Vous devez simuler un **perceptron simple avec un biais** pour classer des points de données en deux catégories (0 ou 1).

- **Données d'entraînement :**

X_i	**Y_i**
[2, 0]	1
[0, 3]	0
[3, 0]	0
[1, 1]	1

Données :

- Taux d'apprentissage (α) : 0.1
- Initialisation :
  - Poids : w=[0,0]
  - Biais : b=0.5

Deep learning

```
import numpy as np

w = np.array([0.0, 0.0])
b = 0.5
alpha = 0.1

data = [
    (np.array([2, 0]), 1),
    (np.array([0, 3]), 0),
    (np.array([3, 0]), 0),
    (np.array([1, 1]), 1)
]

def activation(z):
    return 1 if z >= 0 else 0

for x, y in data:
    z = np.dot(w, x) + b
    y_pred = activation(z)

    w += alpha * (y - y_pred) * x
    b += alpha * (y - y_pred)

    print(f"x: {x}, y: {y}, y_pred: {y_pred},
w: {w}, b: {b}")

print(f"Poids finaux: {w}, Biais final: {b}")
```

Deep learning

Approximation stochastique

Deux articles fondamentaux dans le domaine de l'approximation stochastique, une méthode utilisée dans l'optimisation et l'apprentissage automatique pour trouver les racines des fonctions ou pour optimiser les objectifs lorsque seules des observations bruitées sont disponibles.

- **Robbins-Monro, 1951, Ann. Math. Statist. 22(3):400-407**

$$M(x) = \mathbb{E}_\xi A(x;\xi) = b$$

where $M(x)$ is monotone, stochastic approximation:

$$x_{t+1} = x_t - \eta_t(A(x_t;\xi_t) - b) \tag{1}$$

- **Kiefer-Wolfowitz, 1952, Ann. Math. Statist. 23(3):462-466**

$$\min_x \mathbb{E}_\xi \ell(x;\xi)$$

$$x_{t+1} = x_t - \eta_t \nabla_x \ell(x_t;\xi_t) \tag{2}$$

Deep learning

L'algorithme de classification Perceptron

Descente stochastique de gradient (SGD)

L'algorithme Perceptron est une forme de descente stochastique de gradient (SGD), inspirée de la méthode Robbins-Monro (1951).Le vecteur de poids w est mis à jour de manière itérative en utilisant le gradient de la fonction de perte par rapport à w.

The Perceptron Algorithm
for classification

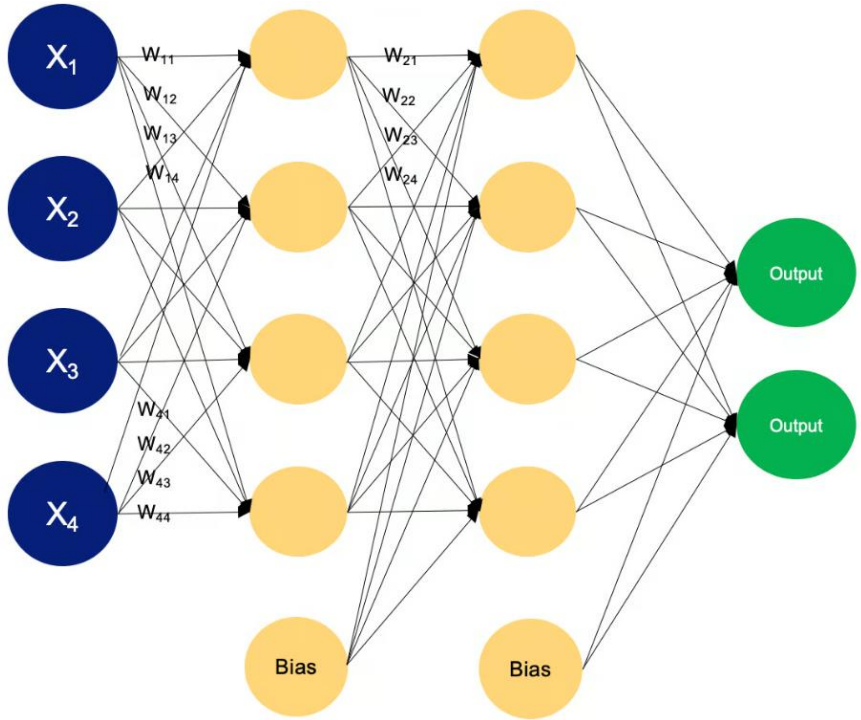
$$\ell(w) = -\sum_{i \in \mathcal{M}_w} y_i \langle w, \mathbf{x}_i \rangle, \quad \mathcal{M}_w = \{i : y_i \langle \mathbf{x}_i, w \rangle < 0, y_i \in \{-1, 1\}\}.$$

The Perceptron Algorithm is a *Stochastic Gradient Descent* method (**Robbins-Monro 1951**):

$$\begin{aligned} w_{t+1} &= w_t - \eta_t \nabla_i \ell(w) \\ &= \begin{cases} w_t - \eta_t y_i \mathbf{x}_i, & \text{if } y_i w_t^T \mathbf{x}_i < 0, \\ w_t, & \text{otherwise.} \end{cases} \end{aligned}$$

Deep learning

Perceptrons multicouches (MLP) et algorithmes de rétropropagation (BP)

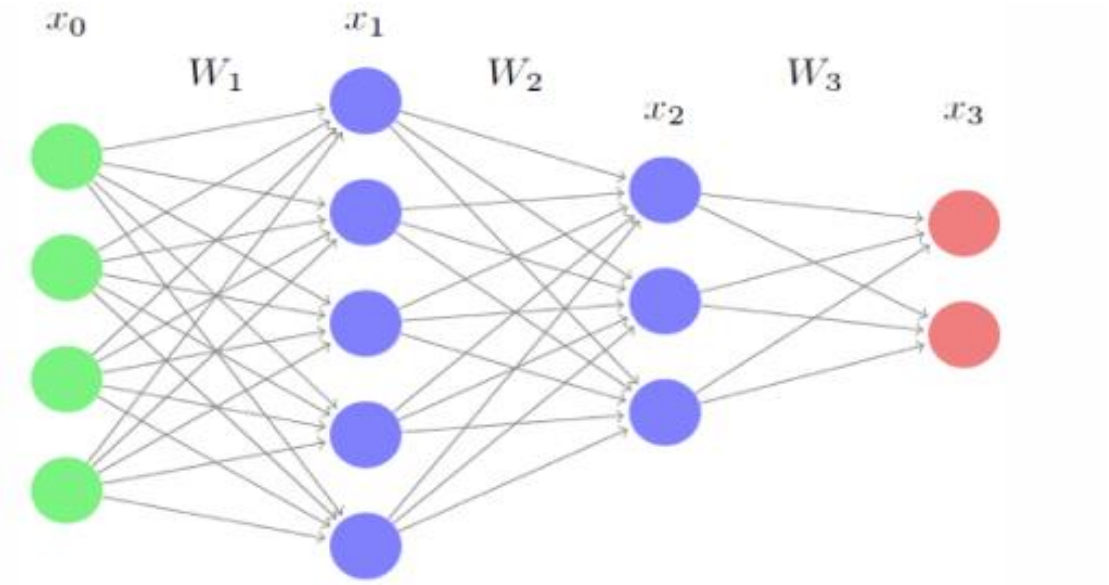


Rumelhart, Hinton, Williams (1986) Learning representations by back-propagating errors, Nature, 323(9): 533-536

Algorithmes BP en tant qu'algorithmes de descente de gradient stochastique (**Robbins-Monro 1950 ; Kiefer-Wolfowitz 1951**) avec des règles en chaîne de cartes de gradient pour les perceptrons multicouches.

Deep learning

Multi-layer Perceptron

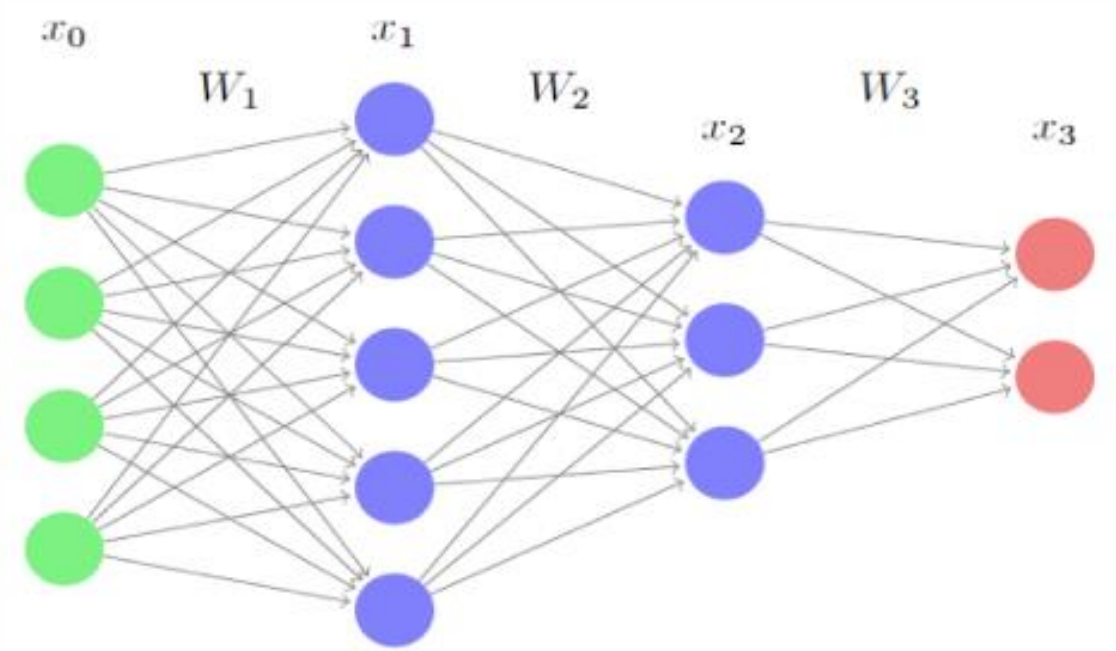


Ecriture condensée - entrée $\mathbf{x}_0 \in \mathbb{R}^d$, sortie $\mathbf{x}_L$, puis, pour $\ell = 1, \ldots, L$,

faire $\mathbf{x}_\ell = \sigma_\ell \left(W_\ell \mathbf{x}_{\ell-1} + b_\ell \right)$ avec $\sigma_L = Id$ ou bien $\sigma_L = \sigma_{\text{softmax}}$

Deep learning

Multi-layer Perceptron



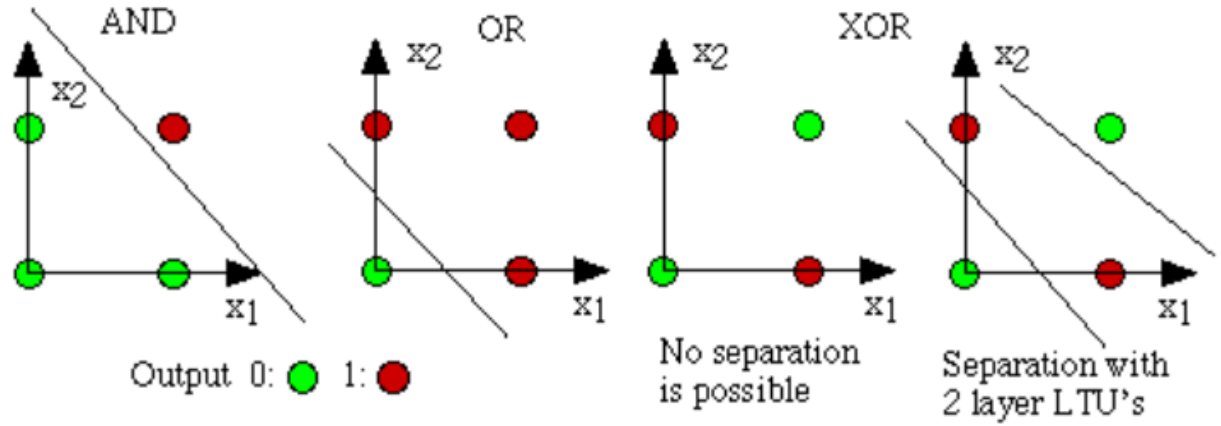
Source : https://stats385.github.io/

Si $\mathbf{x}_{L-1} = (x^{(1)}, \ldots, x^{(K)})$ alors $[\sigma_{\text{softmax}}(\mathbf{x}_{L-1})]_k = \frac{\exp(x^{(k)})}{\sum_{\ell=1}^{K} \exp(x^{(\ell)})}$ pout tout $1 \leq k \leq K$.

Deep learning

Classification MLP et XOR :

Les perceptrons multicouches (MLP) sont capables de classer des données non linéairement séparables, telles que le problème XOR, qu'un perceptron monocouche ne peut pas résoudre.



un seul perceptron ne fonctionne que pour la classification linéairement séparable.

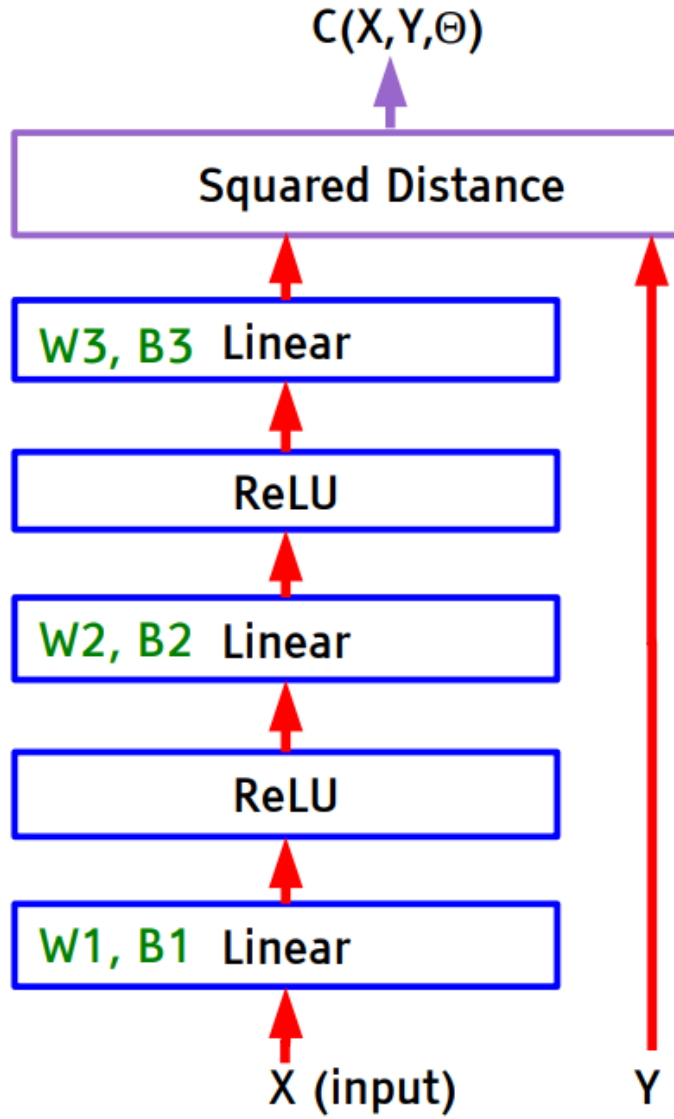
Un perceptron est une frontière de décision, il ne résout donc que les problèmes linéairement séparables.

Deep learning

Algorithme BP : Passe avant du réseau

1. Cascade d'opérations :

1. La passe avant dans un réseau de neurones implique une cascade d'opérations répétées. Chaque couche du réseau effectue une **opération linéaire** suivie d'une **non-linéarité par coordonnée**.
2. L'opération linéaire implique généralement une multiplication matricielle (poids W_ℓ) et une addition vectorielle (biais b_ℓ).



Deep learning

1. La non-linéarité (fonction d'activation) introduit de la non-linéarité dans le modèle, permettant au réseau d'apprendre des motifs complexes. Les fonctions d'activation courantes incluent :

   1. Sigmoïde : $\sigma(x) = \frac{1}{1+e^{-x}}$

   2. Tangente hyperbolique (tanh) : $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

   3. Unité linéaire rectifiée (ReLU) : $ReLU(x) = \max(0, x)$

Deep learning

Algorithme de la passe avant :

•L'algorithme de la passe avant calcule la sortie du réseau pour une entrée donnée x_0.

•L'algorithme itère à travers chaque couche ℓ de 1 à L, où L est le nombre total de couches dans le réseau.

Algorithm 1 Forward pass

Input: x_0

Output: x_L

1: **for** $\ell = 1$ to L **do**

2: $\quad x_\ell = f_\ell(W_\ell x_{\ell-1} + b_\ell)$

3: **end for**

Deep learning

BP algorithm = Gradient Descent Method

- Training examples $\{x_0^i\}_{i=1}^n$ and labels $\{y^i\}_{i=1}^n$
- Output of the network $\{x_L^i\}_{i=1}^m$
- Objective Square loss, cross-entropy loss, etc.

$$J(\{W_l\},\{b_l\}) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}\|y^i - x_L^i\|_2^2 \qquad (1)$$

- Gradient descent

$$W_l = W_l - \eta\frac{\partial J}{\partial W_l}$$

$$b_l = b_l - \eta\frac{\partial J}{\partial b_l}$$

In practice: use Stochastic Gradient Descent (SGD)

Deep learning

Dérivation de la BP : multiplicateur lagrangien

Given n training examples $(I_i, y_i) \equiv$ (input,target) and L layers

- Constrained optimization

$$\begin{aligned} \min_{W,x} \quad & \textstyle\sum_{i=1}^{n} \|x_i(L) - y_i\|_2 \\ \text{subject to} \quad & x_i(\ell) = f_\ell\Big[W_\ell x_i\,(\ell-1)\Big], \\ & i = 1,\ldots,n, \quad \ell = 1,\ldots,L,\ x_i(0) = I_i \end{aligned}$$

- Lagrangian formulation (Unconstrained)

$$\min_{W,x,B} \mathcal{L}(W,x,B)$$

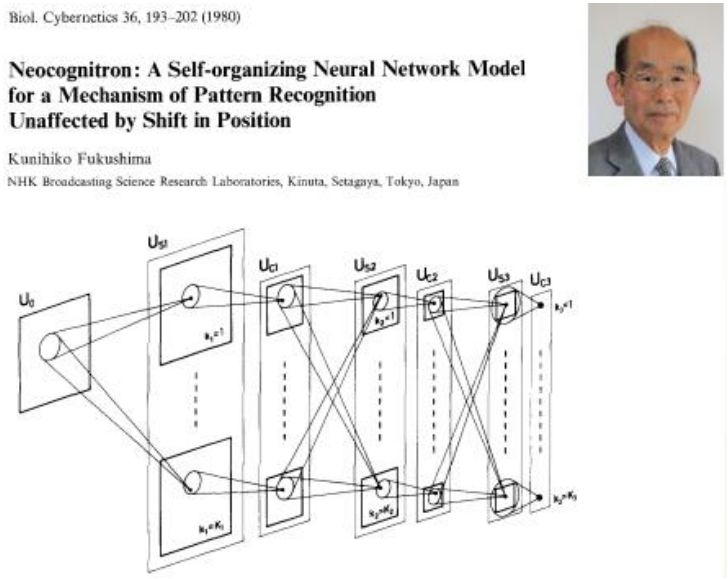
$$\mathcal{L}(W,x,B) = \textstyle\sum_{i=1}^{n} \Big\{ \|x_i(L) - y_i\|_2^2 + \sum_{\ell=1}^{L} B_i(\ell)^T \Big(x_i(\ell) - f_\ell\Big[W_\ell x_i\,(\ell-1)\Big]\Big)\Big\}$$

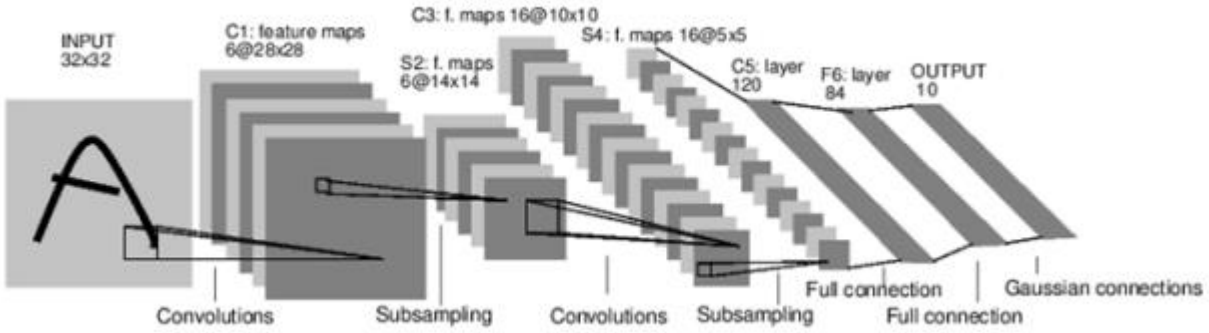
LeCun et al. 1988

Deep learning

Réseaux neuronaux convolutifs : invariances de décalage et localité

- Peut être rattaché au Neocognitron de Kunihiko Fukushima(1979)
- Yann LeCun a combiné les réseaux neuronaux convolutionnels avec la rétropropagation (1989)
- Impose l'invariance des décalages et la localité des poids
- La propagation vers l'avant reste similaire
- La rétropropagation change légèrement - nécessité d'additionner les gradients de toutes les positions spatiales



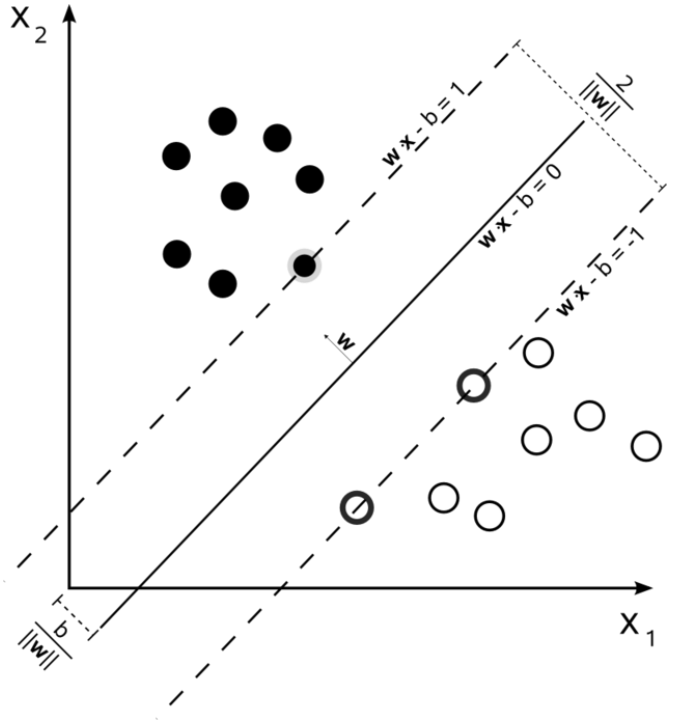


Deep learning

Max-Margin Classifier (SVM)

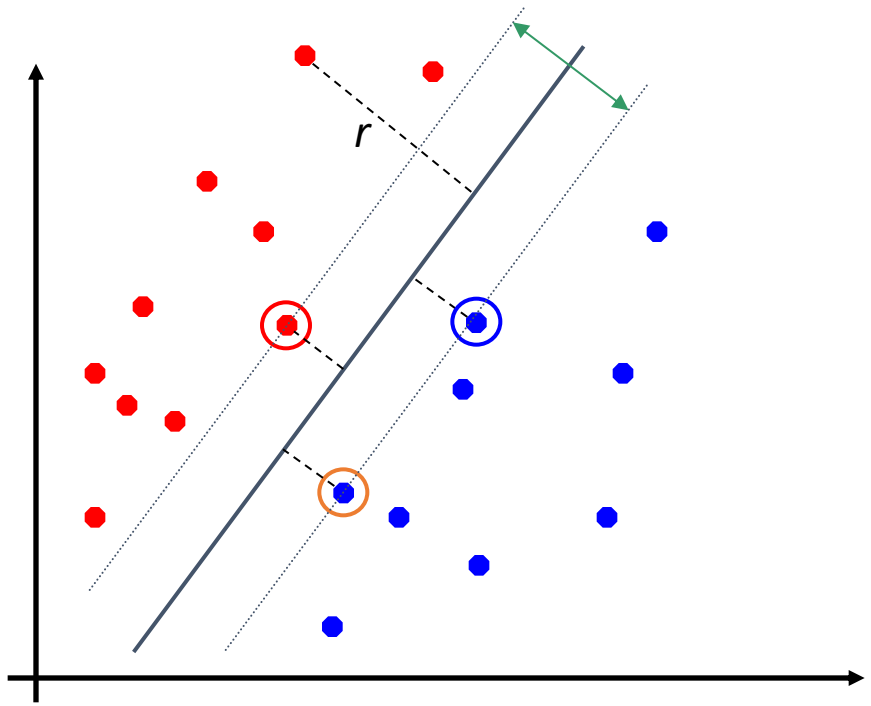
Les machines à vecteurs de support (SVM) sont des modèles d'apprentissage supervisés avec des algorithmes d'apprentissage associés qui analysent les données et reconnaissent des modèles, utilisés pour la classification et l'analyse de régression.

Vladmir Vapnik, 1994



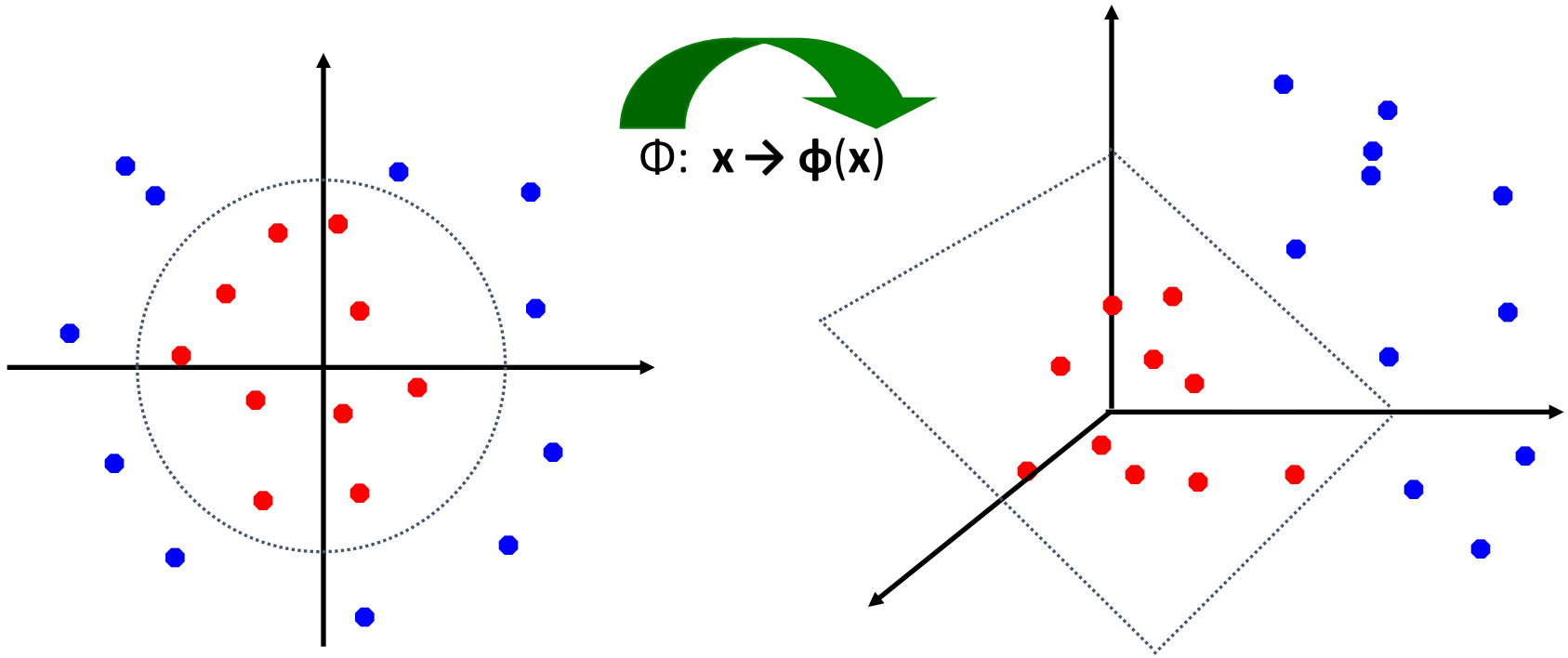
Deep learning

Hyperplan à marge maximale et marges pour un SVM formé avec des échantillons de deux classes.



Deep learning

Idée générale : l'espace caractéristique original peut toujours être mis en correspondance avec un espace caractéristique de dimension supérieure dans lequel l'ensemble d'apprentissage est séparable :



Deep learning

- Il existe de nombreuses fonctions noyau dans les SVM et la sélection d'une bonne fonction noyau est un sujet de recherche.
- Les fonctions noyaux les plus courantes sont les suivantes :

Linear kernel : $K(x_i, x_j) = x_i^T x_j$

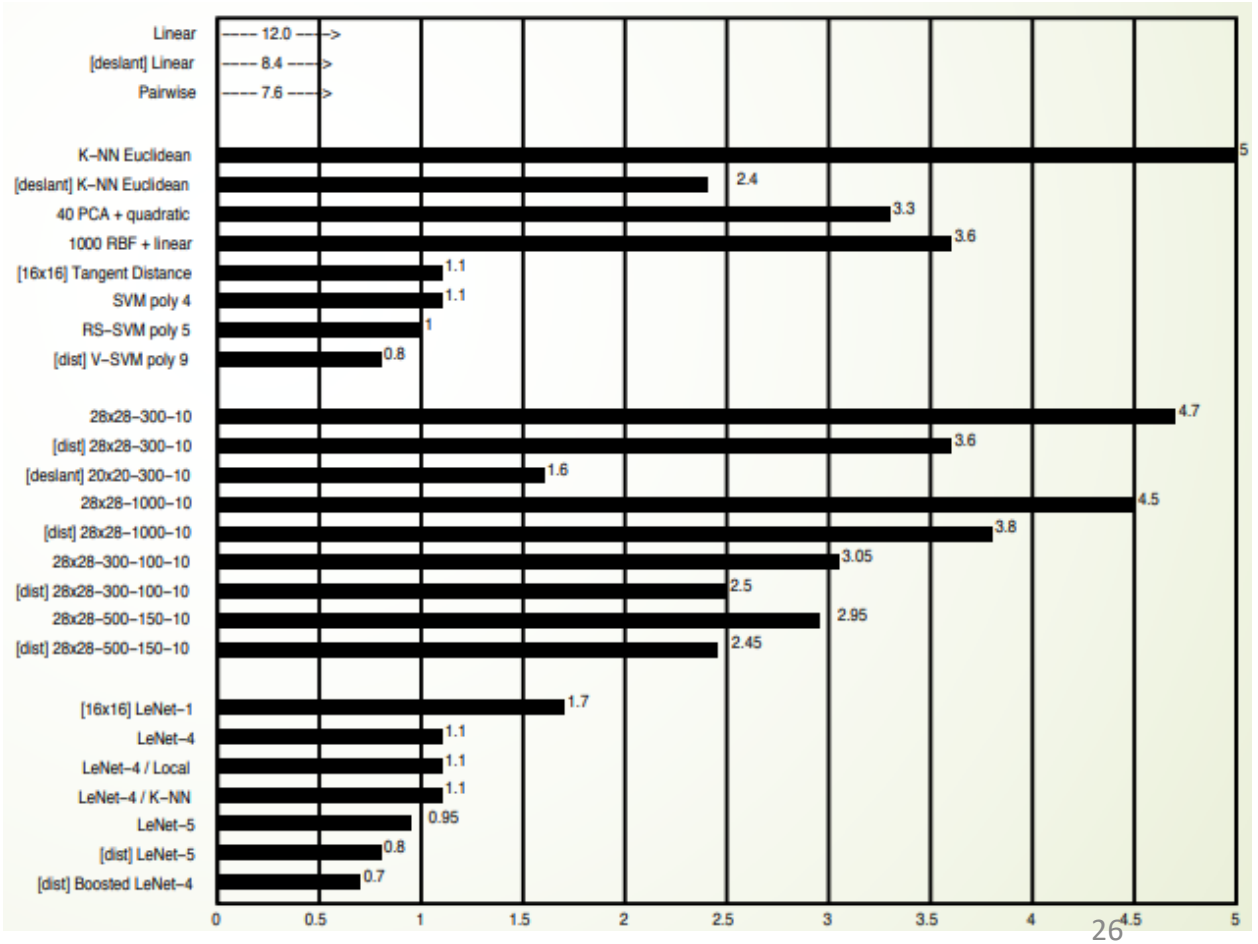
Polynomial kernel : $K(x_i, x_j) = (y x_i^T x_j + r)^d$, $y>0$

RBF kernel : $K(x_i, x_j) = e^{-\frac{\left\| \mathbf{x}_i - \mathbf{x}_j \right\|^2}{2\sigma^2}}$

Sigmoid kernel : $K(x_i, x_j) = \tanh(y x_i^T x_j + r)$

Deep learning

MNIST Dataset Test Error **LeCun et al. 1998**



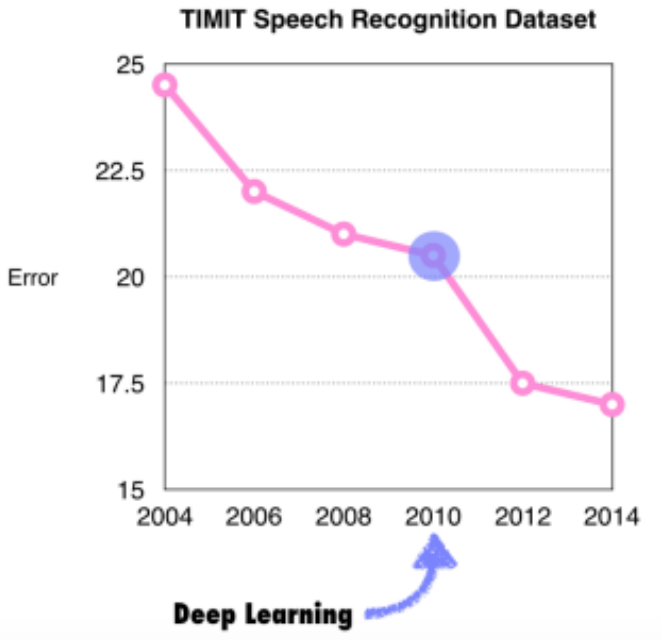
Les SVM simples sont aussi performants que les réseaux neuronaux convolutionnels multicouches qui nécessitent un réglage minutieux (LeNets).

Dark era for NN: 1998-2012

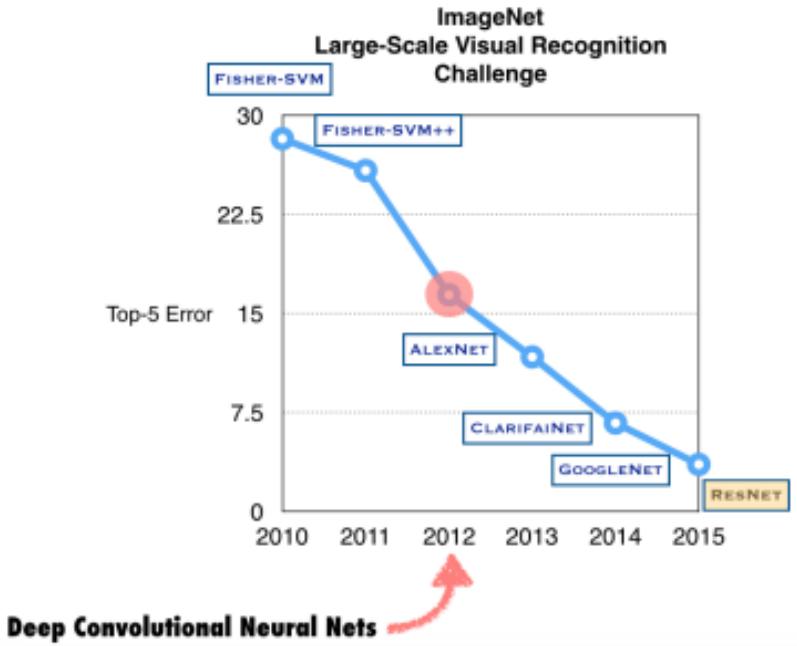
Deep learning

Autour de l'année 2012 : retour du NN en tant que « deep learning » (apprentissage en profondeur)

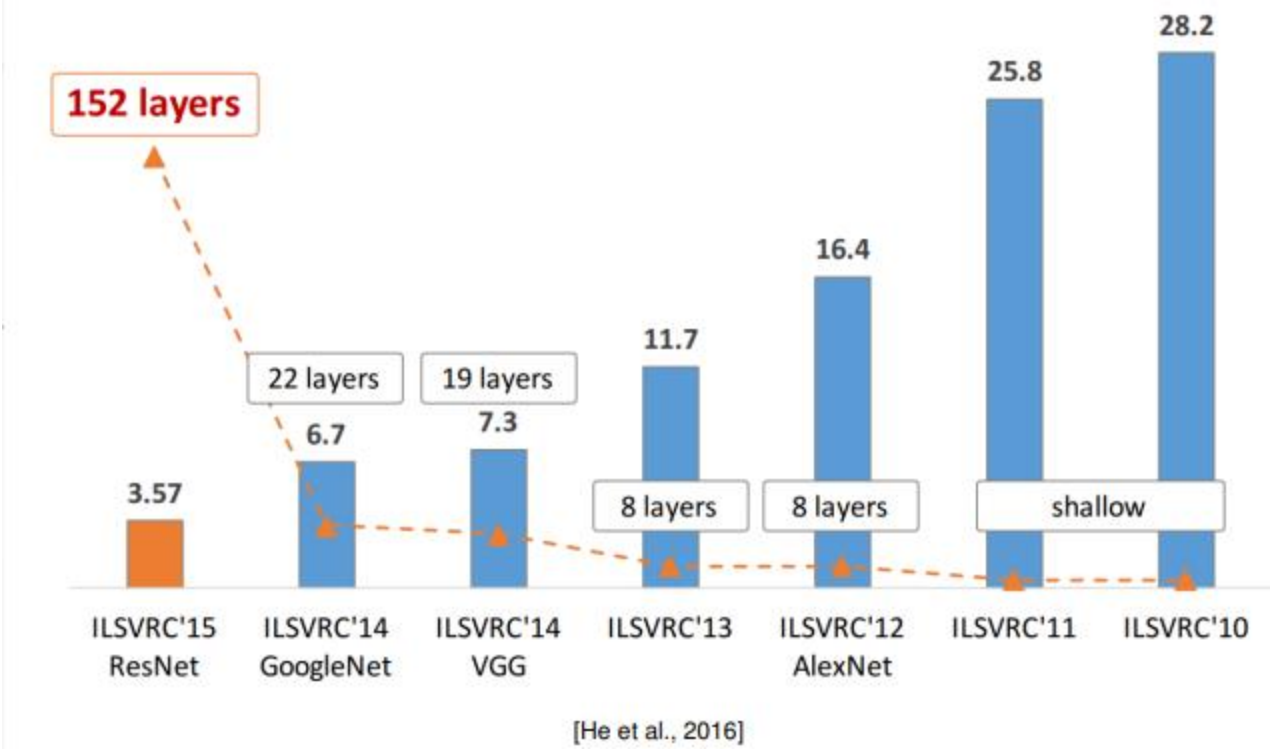
Speech Recognition: TIMIT



Computer Vision: ImageNet

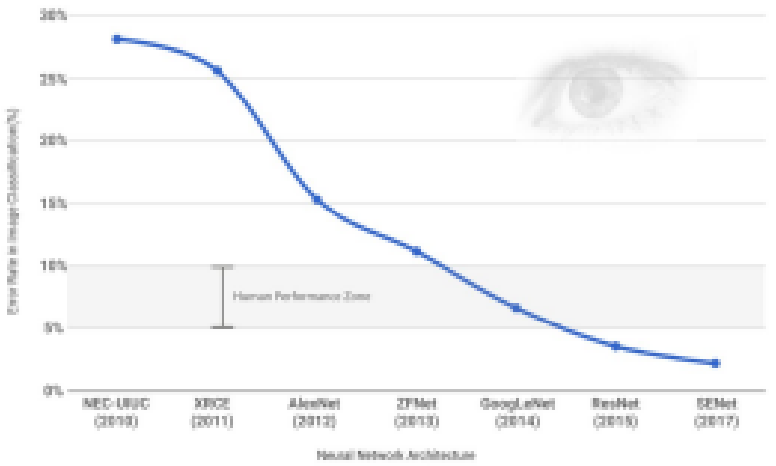


Deep learning

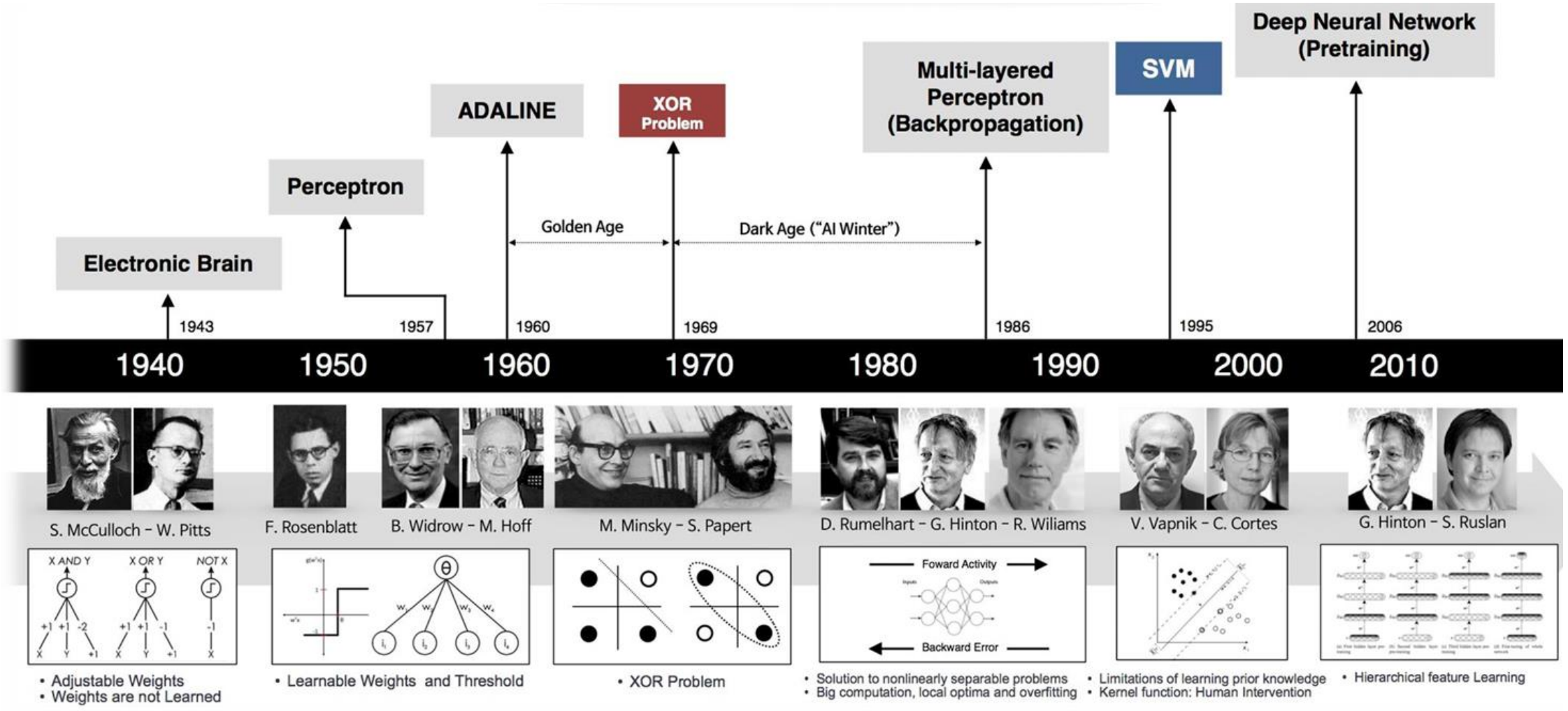


[He et al., 2016]

- ImageNet (subset):
  - 1.2 million training images
  - 100,000 test images
  - 1000 classes
- ImageNet large-scale visual recognition Challenge



ILSVRC ImageNet Top 5 errors


Deep Neural Network (Pretraining)
SVM
Multi-layered Perceptron (Backpropagation)
XOR Problem
ADALINE
Perceptron
Electronic Brain
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Wiliams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

Deep learning

Number of AI papers on arXiv, 2010- 2019

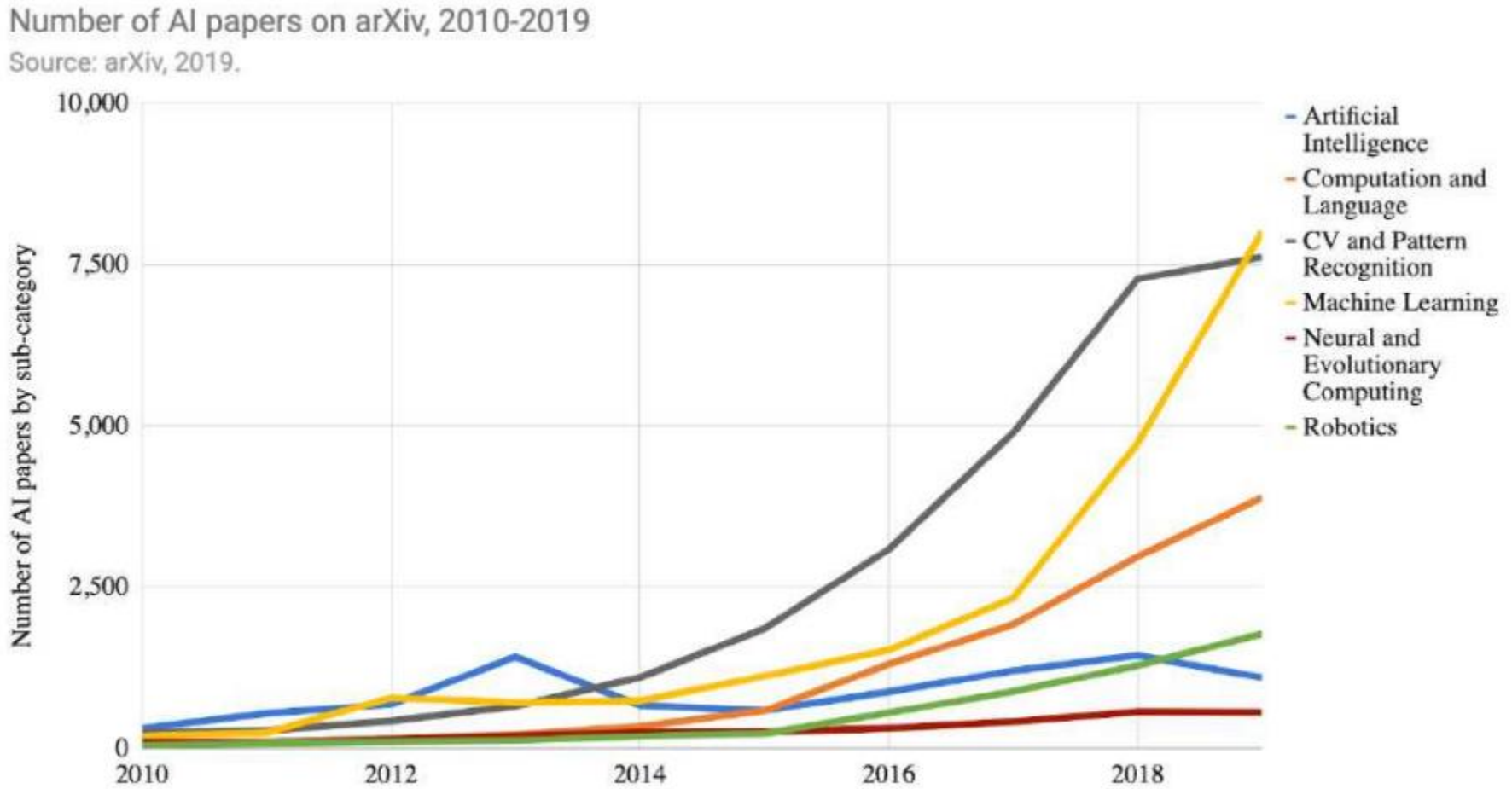
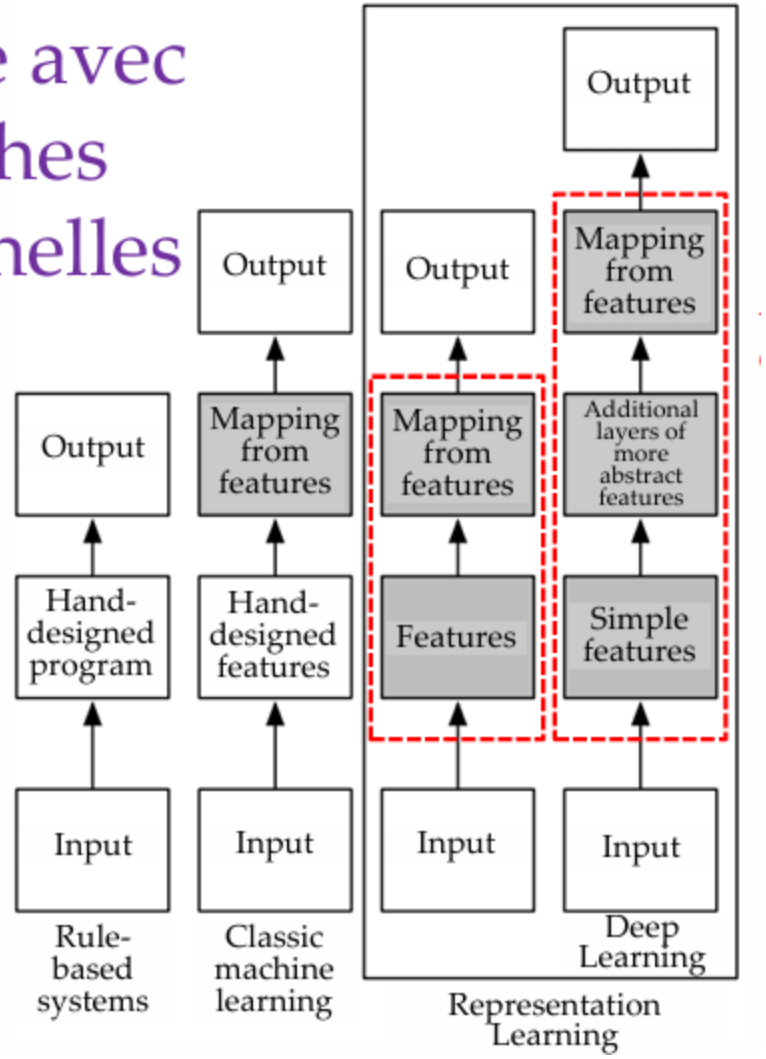


Fig. 1.6.

Deep learning

Contraste avec approches traditionnelles



Appris conjointement!!

Apprentissage en profondeur : Un sous-ensemble de l'apprentissage automatique qui utilise des réseaux neuronaux avec de nombreuses couches pour modéliser des modèles complexes dans les données.

Deep learning

Fonctions d’activations

Deep learning

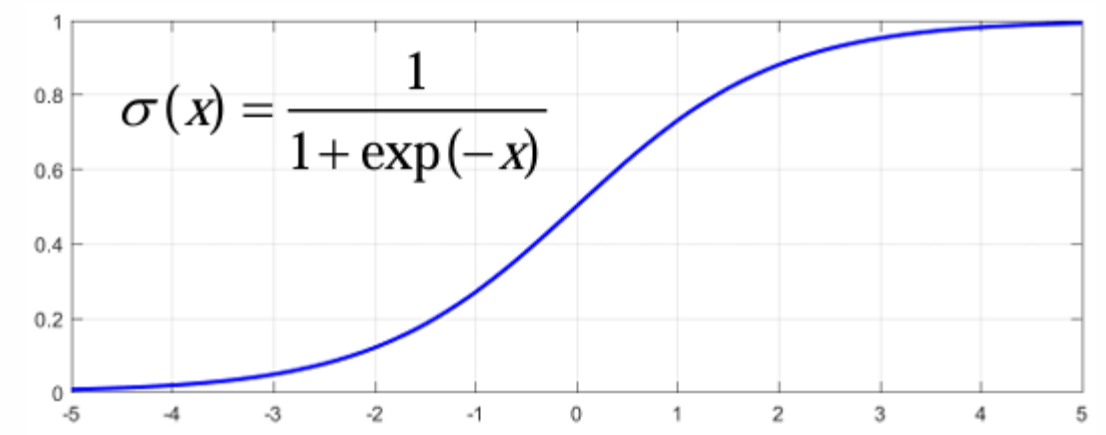
Rôles

- Apporte une non-linéarité dans le réseau
- Situé à l'interne, ou en sortie
- Considérations :
  - difficulté d'entraînement (son gradient)
  - comportement se rapproche :
    - de ce que l'on cherche à prédire en sortie (probabilités, one-hot vector, angles, déplacements, etc.)
    - action particulière (gating)– temps de calcul

Deep learning

Fonction d'activation : sigmoide

- Une des premières utilisées
- Si on désire une sortie entre 0 et 1 (squashing)



- Tombée en désuétude comme non-linéarité de base

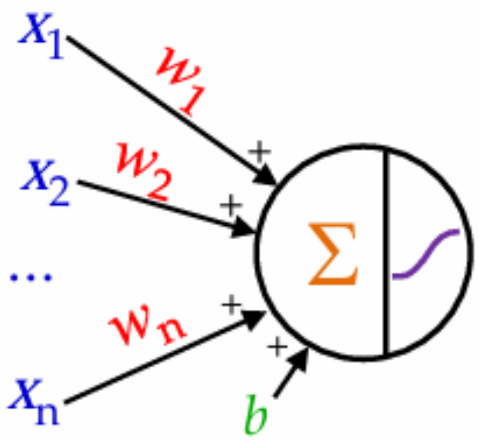
Deep learning

Exemple utilisation sigmoïde

- Prédiction binaire (logistic regression)

$$P(y=1 \mid x)$$

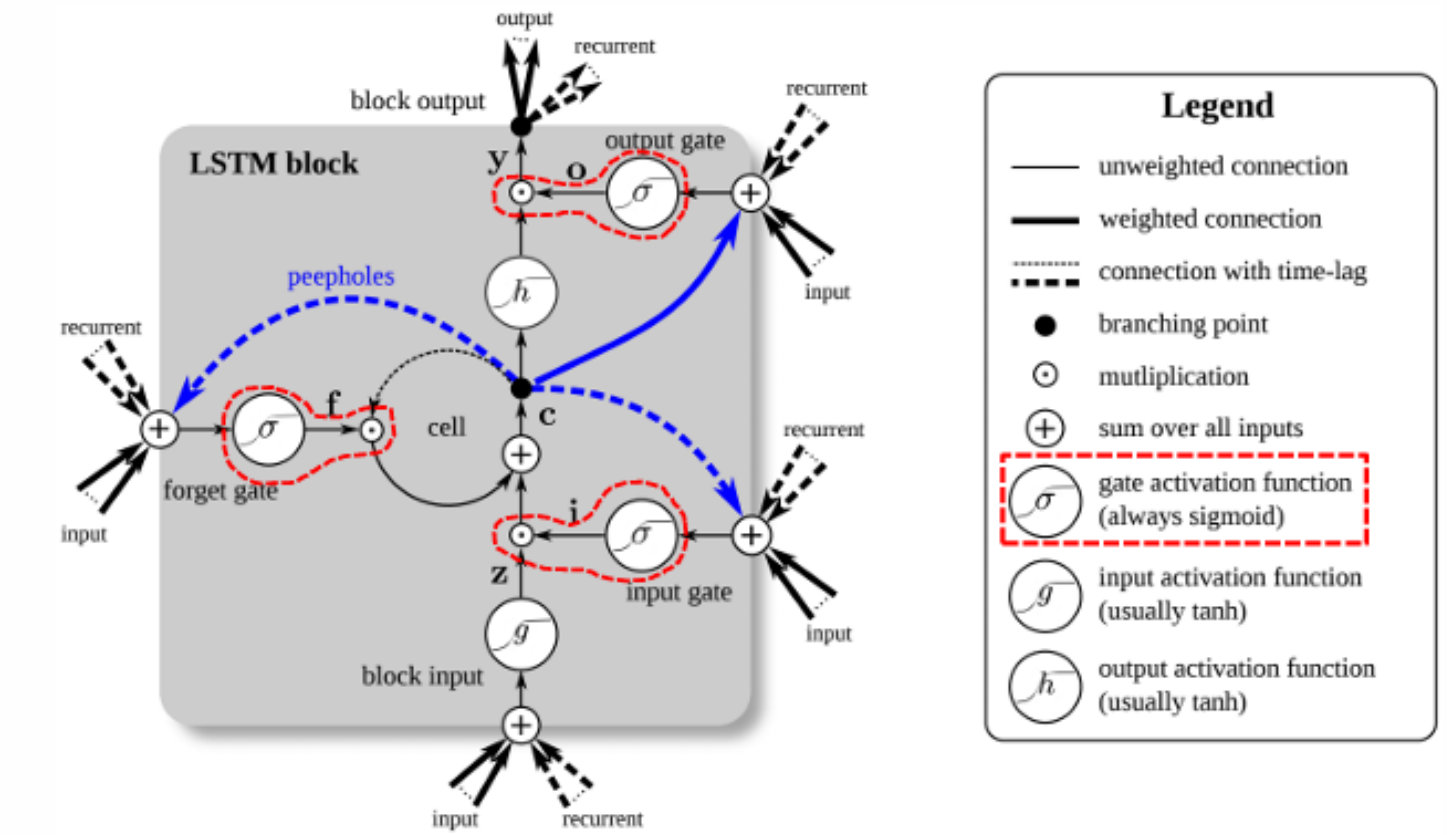
$$sortie = \sigma\left(w^T x + b\right)$$



Deep learning

Exemple utilisation sigmoïde

- gating dans Long short-term memory

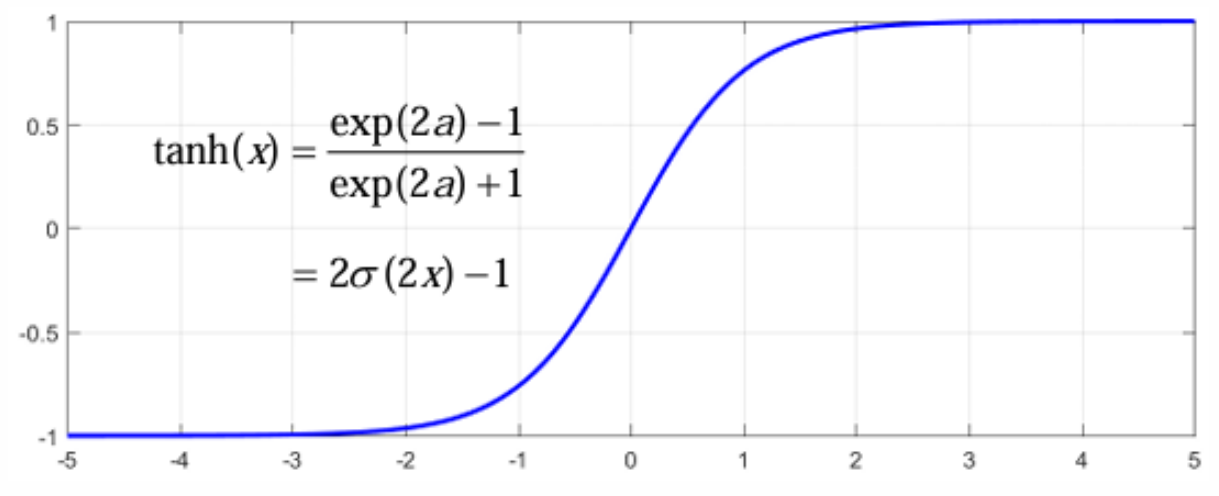


Greff et al. LSTM : A Search Space Odyssey, T. on Neural Networks and Learning Systems, Oct 2017.

Deep learning

Fonction d'activation : tanh

- Autre fonction historique
- Désire une sortie entre -1 et 1 (squashing)

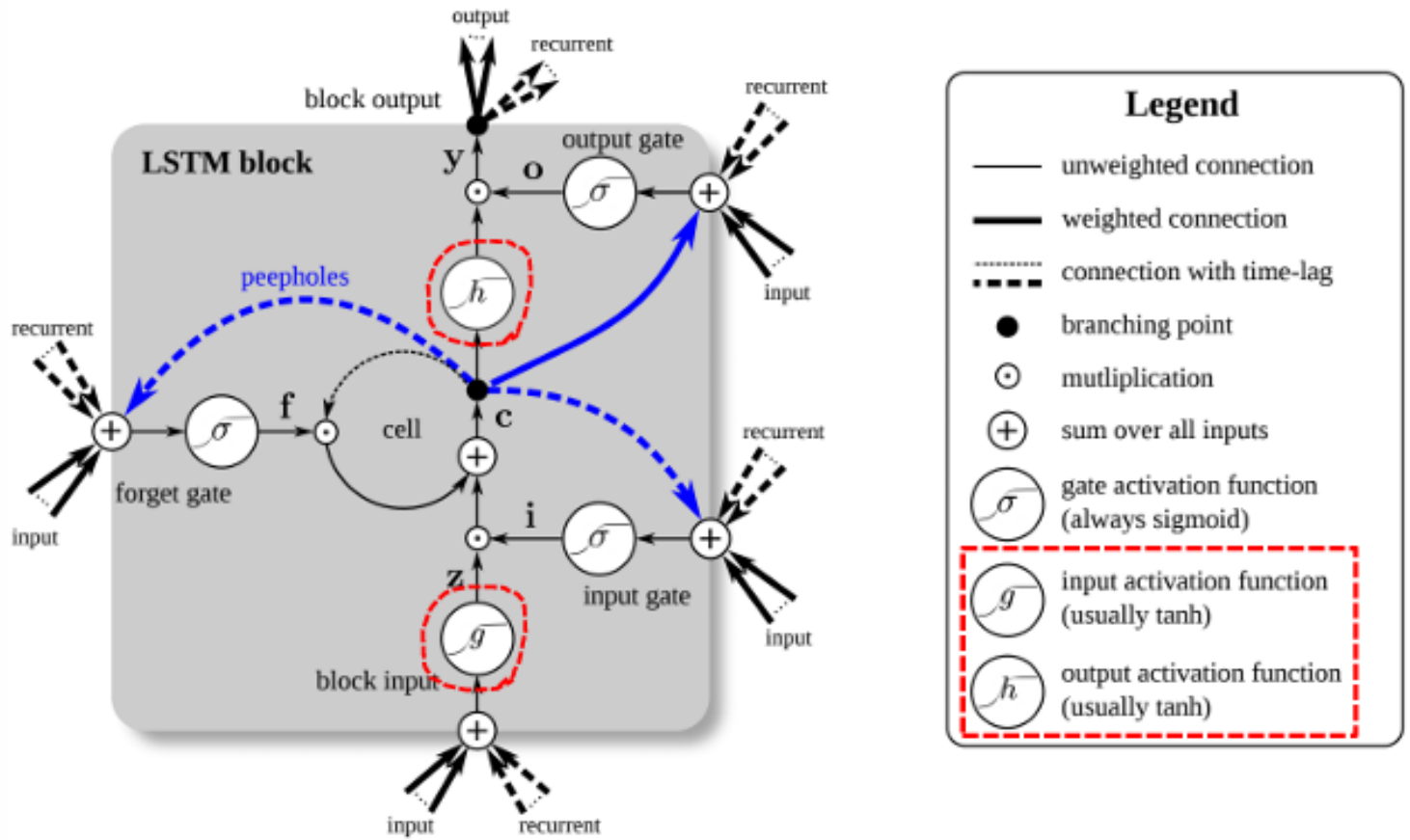


- Donne une sortie centrée à 0 (préférable à 0.5 de la sigmoïde)

Deep learning

Fonction d'activation : tanh

- LSTM : Long short-term memory

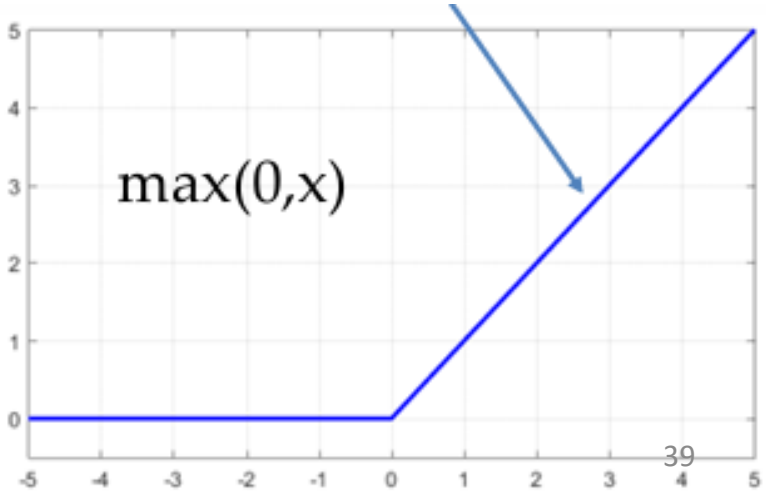


Deep learning

ReLU

- La plus populaire comme non-linéarité
- Toujours non-négatif– moyenne sortie biaisée +
- Pas de limite supérieure
- Facile à calculer (exp est plus long)
- Accélère l'entraînement des réseaux (facteur 6 pour AlexNet).
- Résulte en des activations parcimonieuses (certains neurones sont à 0)
  – Parfois des neurones vont mourir, particulièrement si learning rate est trop grand
- Rarement utilisé en sortie du réseau

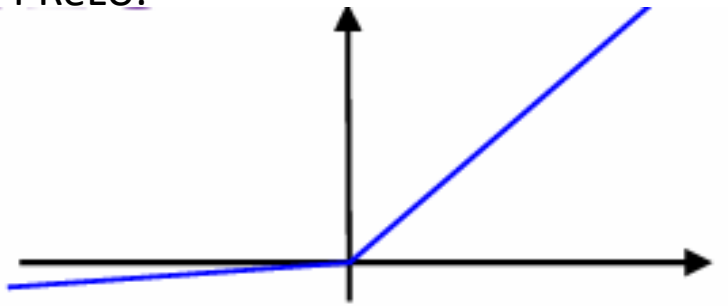
Quasi-linéarité : rend l'entrainement plus facile via descente de gradient



Deep learning

Leaky ReLU

- Gradient = 0 signifie impossibilité d'entraîner
- Pente très légère dans la partie négative : leaky ReLU
- Si un paramètre a (entraînable) par neurone/couche, on obtient la PReLU.
- Donne des distributions de sorties plus centrées à 0 que ReLU

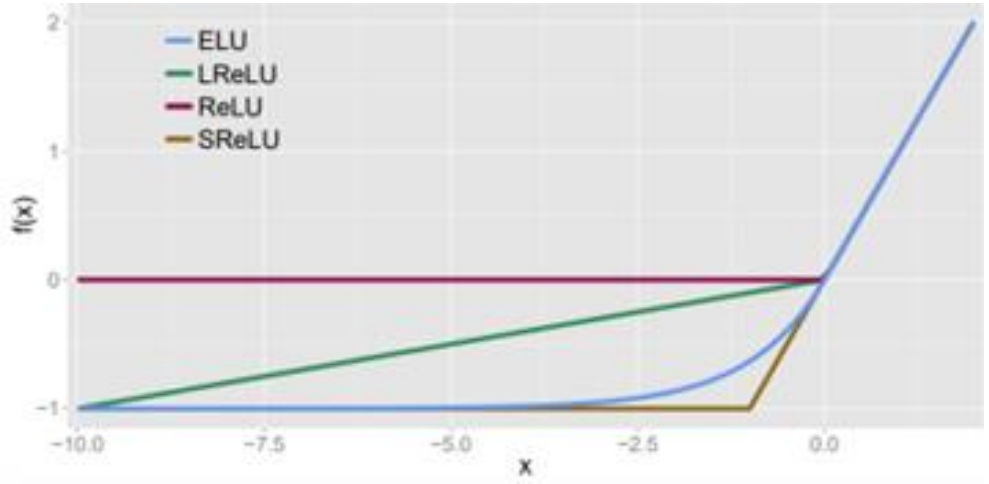


$$f(x) = \begin{cases} \alpha x & \text{si } x<0 \\ x & \text{autrement} \end{cases}$$

Deep learning

Autres activations

- Maxout[2013] : max(w1Tx+b1, w2Tx+b2)
- ELU [2016]
- Parametric ELU [2017]
  (Trottier et al., Parametric exponential linear unit for deep convolutional neural networks, ICMLA 2017.)



Deep learning

Softmax

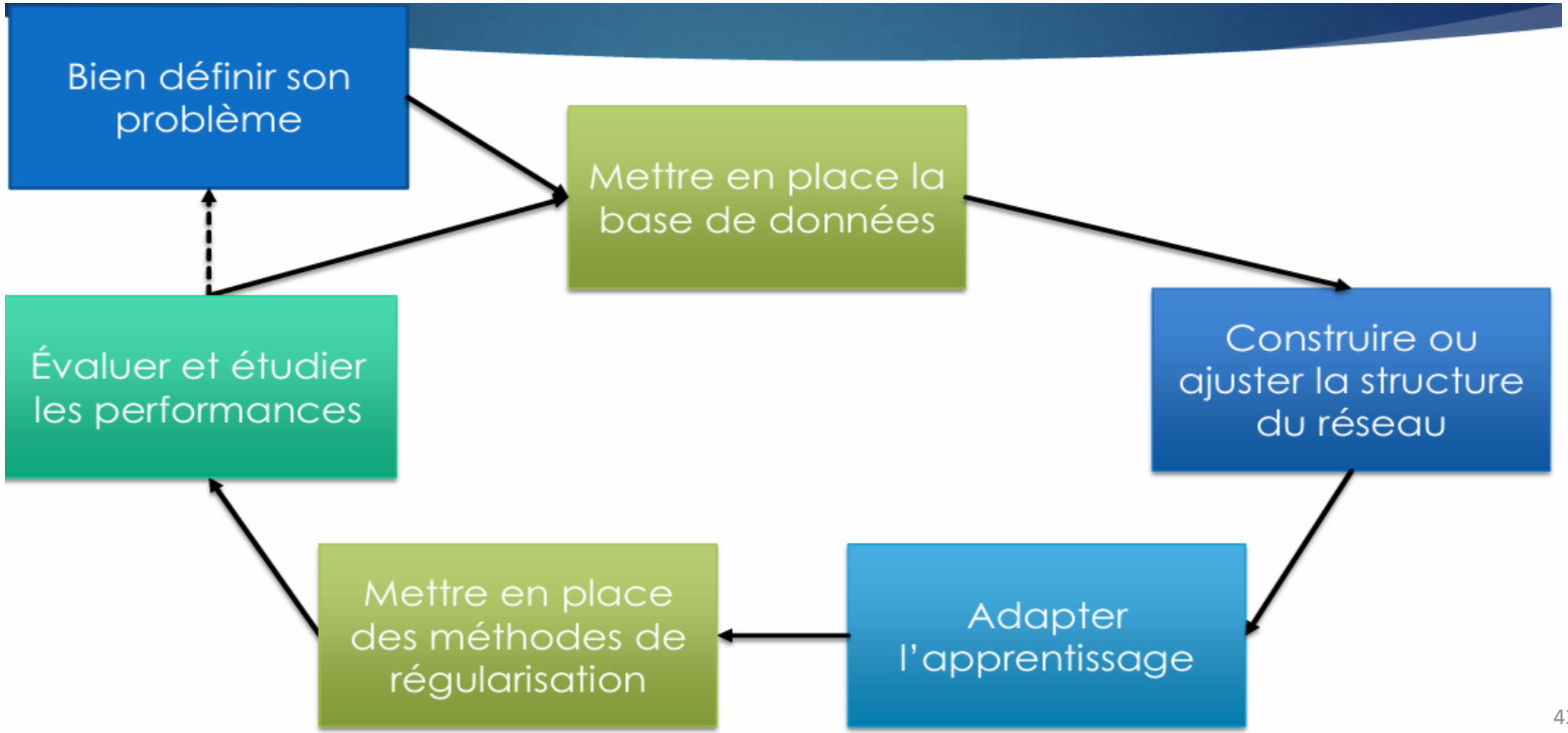
- Utilisé en sortie, prédiction multi-classe
- Version continue, douce, de max([…])

$$\hat{y}_i = \frac{\exp(z_i)}{\sum_{j \in groupe} \exp(z_j)}$$

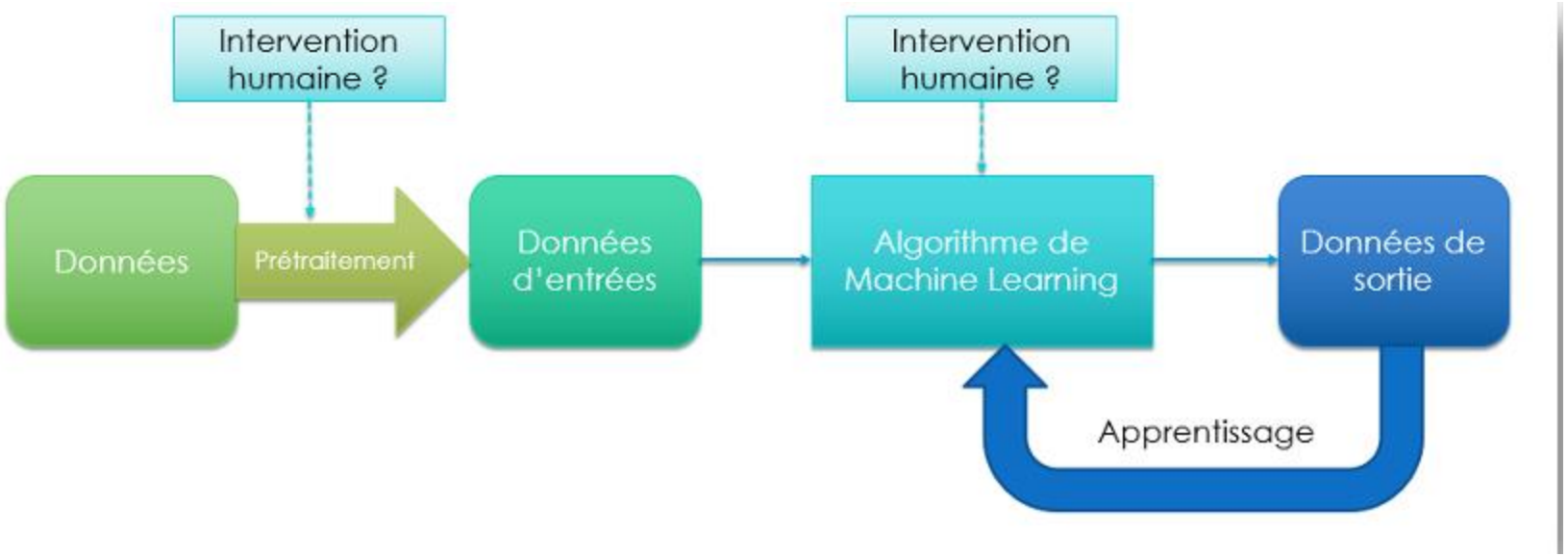
- Va dépendre de l'ensemble des sorties du groupe
- Sortie somme à 1, chaque sortie entre 0 et 1 : – distribution de probabilité multinouilli
- Manière d'indiquer au réseau de chercher l'appartenance exclusive à une seule classe

Deep learning

Projet de deep learning



Deep learning



- Pour apprendre, il faut des données : données d'entrée et de sortie !
- De quelles données je dispose ?
- Réfléchir à formuler le problème de la manière suivante (apprentissage supervisé)

Données d'entrée ?

Quelles informations vais-je utiliser pour faire des « prédictions » ?

Données de sortie ?

Que veut-on prédire ?
- Des classes ?
- Des paramètres ?
- Une image ?

Deep learning

La base de données : quel format ?

Données d’entrée

- Comment représenter les données d’entrée ? → Définit le type des premières couches à utilise
- Ensemble de paramètres : masse d’un objet, type, taille, couleur… sous forme de nombres ou éventuellement de classes
- Image, spectre (données localement corrélées)

Deep learning

Données d'entrée

Normaliser les données !

- Ne pas avoir des nombres trop grands : calculs plus difficiles à gérer
- Faire attention si on a un ensemble de paramètres (masse, taille…) : normaliser indépendamment, il ne faut pas qu'un type de paramètre « domine »
- Première possibilité : diviser par un nombre fixe. Exemple : image (intensité des pixels entre 0 et 255) –> diviser par 255
- Deuxième possibilité : normaliser en moyenne et variance sur la base de données

$$X := \frac{X - \mu_{\text{train}}}{\sigma_{\text{train}}}$$

Deep learning

Données de sortie

- Comment représenter les données de sortie ? → Définit la couche de sortie (nombre de neurones et fonction d'activation), la fonction de coût et le type des dernières couches
  - Classification : définir des classes. Exemple : classer des chiffres manuscrits de 1 à 5, chiffre 4 : (0,0,0,1,0), chiffre 2 : (0,1,0,0,0) → Un neurone par classe, fonction d'activation sigmoïde (non-exclusif), softmax (exclusif). Définir un seuil d'acceptation de la classe. Fonction de coût : binary cross-entropy, distance euclidienne…
    - Faire attention à ce que toutes les classes pertinentes soient représentées
    - Il faut éviter qu'une classe soit sur-représentée
  - Régression → Un neurone par paramètre de sortie, fonction d'activation dépend des paramètres, fonction de coût : distance euclidienne, distance en norme 1
    - Si plusieurs paramètres → normaliser chaque paramètre indépendamment (éviter qu'un paramètre ne domine pour la fonction de coût)
  - Image, spectre → Nombre de neurones adapté au format. Fonction d'activation adaptée aux données. Fonction de coût : distance euclidienne, distance en norme 1. Dernières couches éventuellement convolution.
    - Normaliser les données

Deep learning

La base de données : le découpage

Pour l'apprentissage des paramètres

Pour le monitoring de l'apprentissage

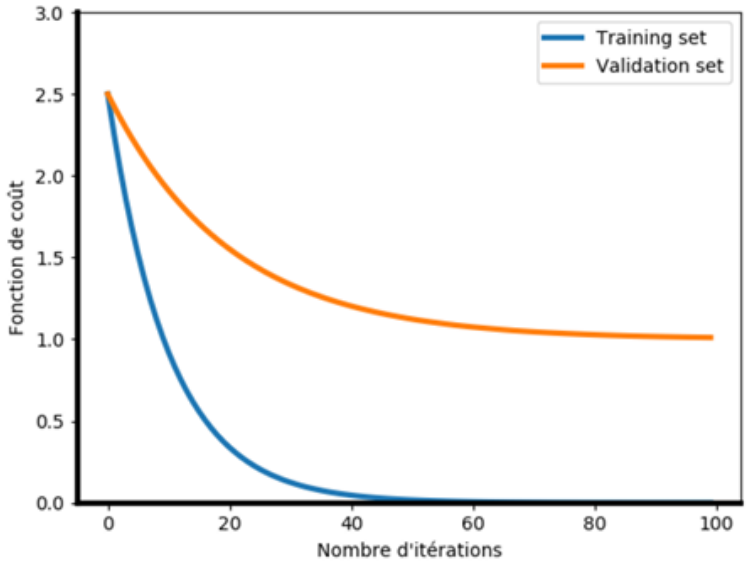
Pour l'évaluation finale des performances

- Validation set : utilisé pour diagnostiquer les performances du réseau de neurones puis adapter le réseau. Le réseau n'apprend pas dessus !
- Test set : utilisé comme évaluation finale, quand on est déjà satisfait du réseau avec le validation set. Le test set doit être représentatif des données sur lesquelles le réseau va être appliqué !
- Pour les proportions : Avant le big data : ordre de grandeur 60%/20%/20%. Avec le big data : 98%/1%1% (1% de 1 million = 10 000)

Deep learning

Régularisation : intérêt ?

- Généraliser au mieux le réseau de neurones
- Éviter l'overfitting : on apprend par cœur sur les données d'apprentissage
- Diagnostic avec le monitoring sur le validation set



Deep learning

Pour résumer

- Bien poser le problème à résoudre
- Construire la base de données en conséquence
- Projet de deep learning : processus itératif
    - On développe : mise en place du réseau
    - On fait l'apprentissage
    - On teste 26
    - On diagnostique et on agit en conséquence : régularisation, complexification du réseau, revoir la base de données
- Intérêt d'accélérer le calcul (GPU) : Pouvoir exécuter ce processus rapidement

Deep learning