Réseaux de neurones et deep learning : Utilisation et méthodologie

GEOFFREY DANIEL – CEA/IRFU/DAP

Réseaux de neurones et deep learning

À partir de mi-octobre

Séances de 30/40 mn suivies de 20 mn de discussion

- **Séance 1 : Introduction aux réseaux de neurones**
  - **Utilisations courantes du deep learning**
  - **Bases générales**
- Séance 2 : Architecture des réseaux, hyperparamètres et évaluation des performances
  - Comment construire mon réseau et adapter la phase d'apprentissage ?
  - Comment évaluer les performances de mon réseau de neurones ?
- Séance 3 : Construction de la base de données
  - Éléments méthodologiques sur la mise en place du problème à résoudre potentiellement par deep learning
  - Comment utiliser l'évaluation des performances pour améliorer la base de données et le réseau ?
- Séance 4 : Réseaux de neurones convolutifs
  - Introduction à des structures plus avancées

L’inévitable IA

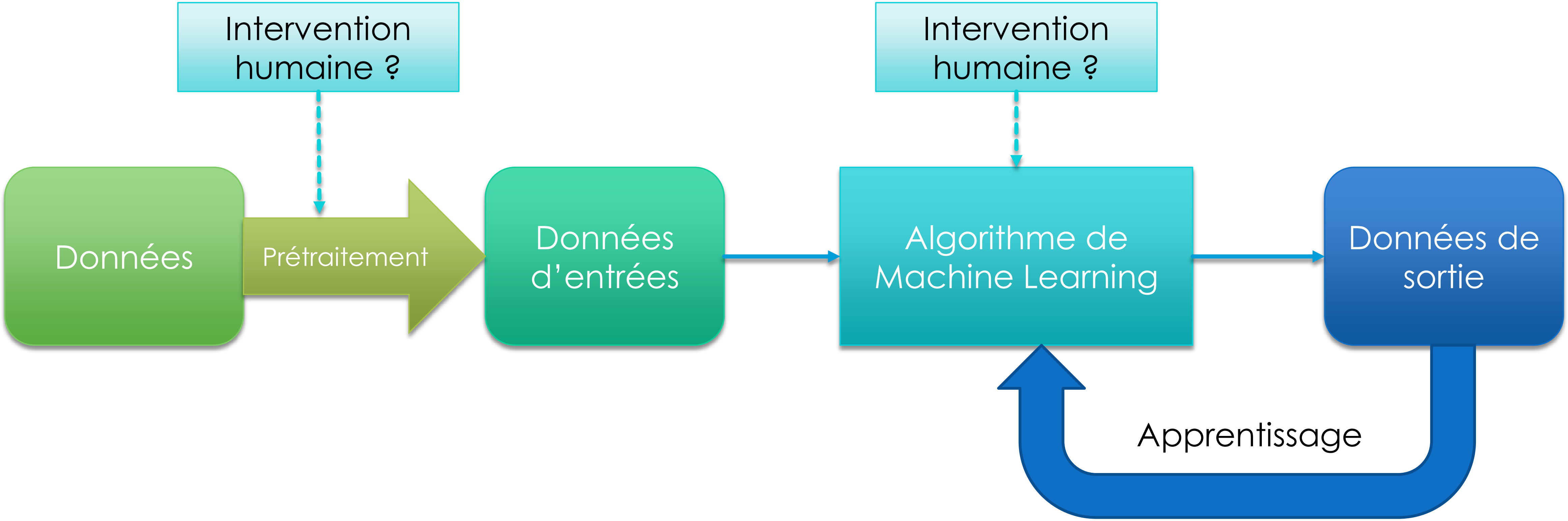
- Génération automatique de texte : rapports et synthèses
- Reconnaissance d’images : reconnaissance biométrique, classifications d’images sur les réseaux sociaux
- Agents virtuels : chatbots
- Reconnaissance automatique de la parole
- Automatisation robotisée
- Génération d’image, de musique

L’inévitable IA

L’IA surpasse l’humain

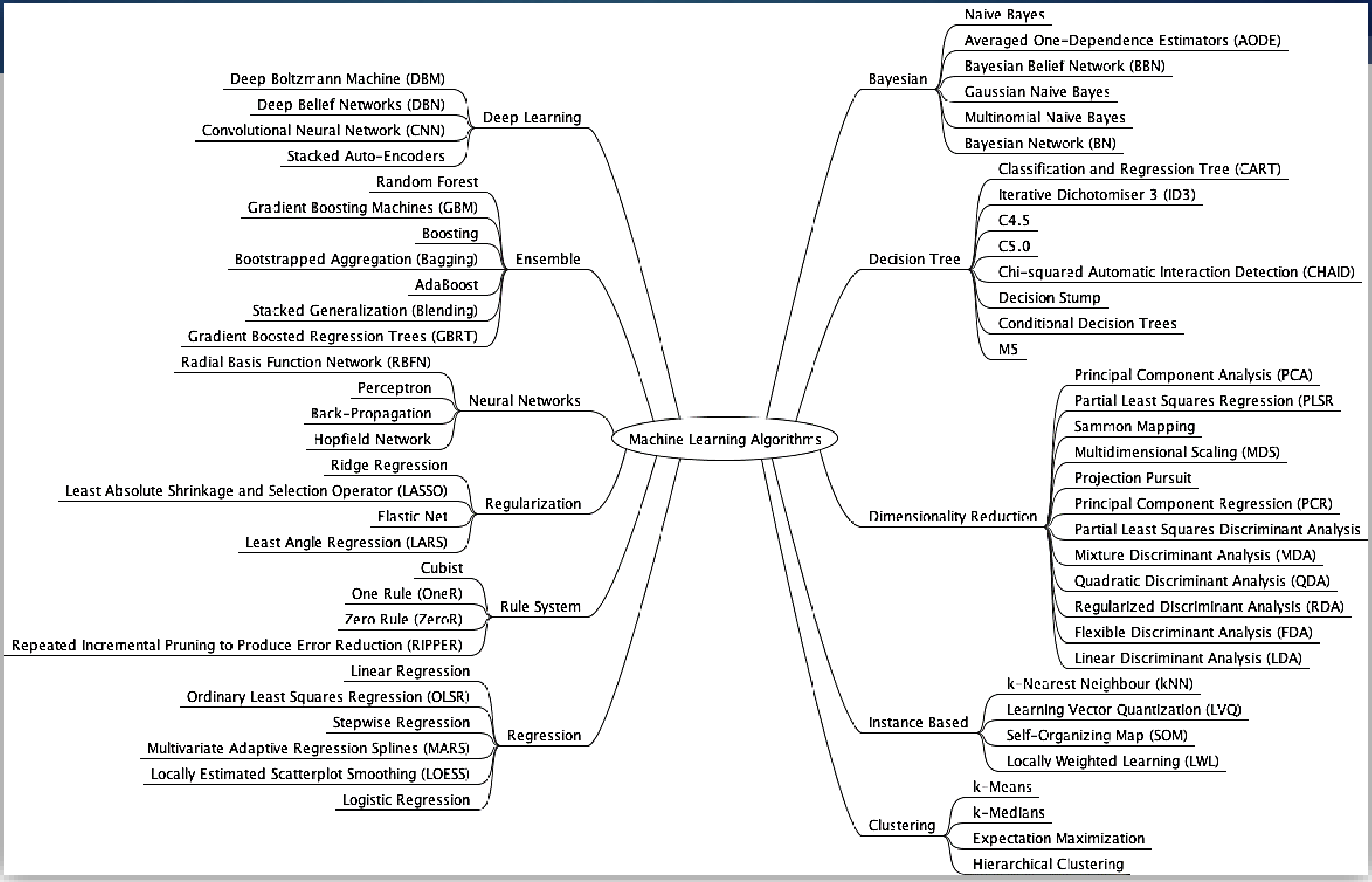
Google car

Philosophie du Machine Learning



La forêt du Machine Learning

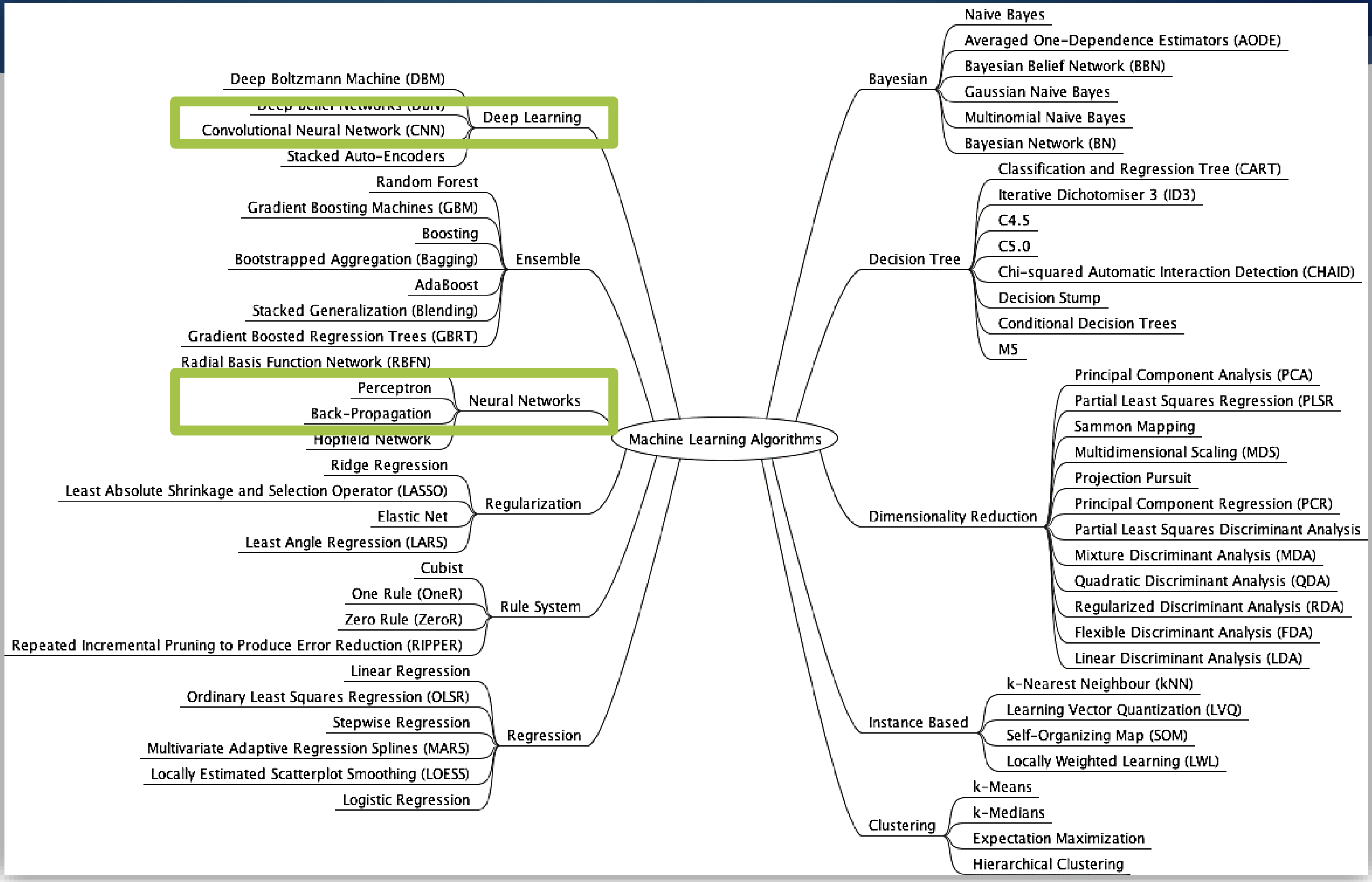
Et encore, ce n'est qu'une partie



Jason Brownlee
2013

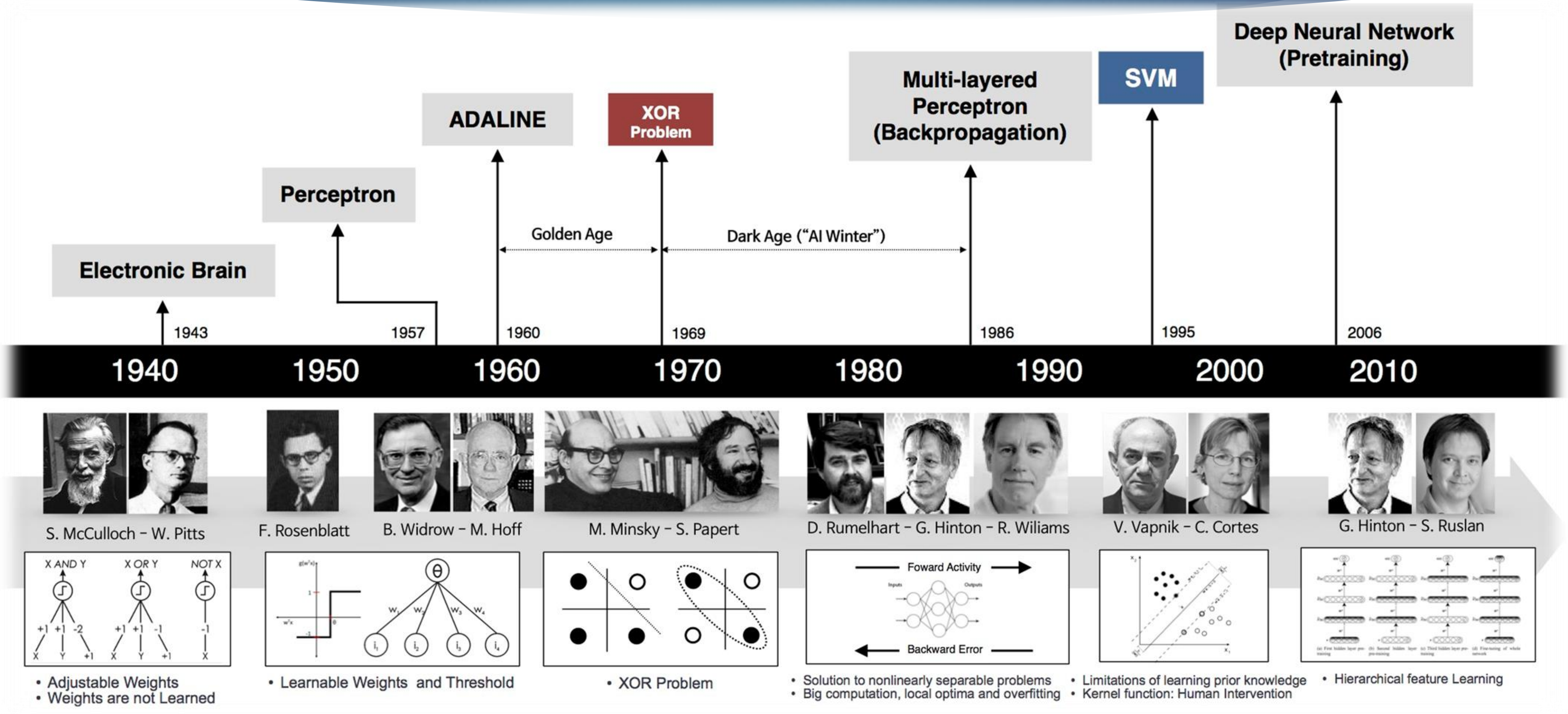
La forêt du Machine Learning

Et encore, ce n'est qu'une partie



Jason Brownlee
2013

Deep learning : un peu d'histoire



https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

Deep learning : l’avènement

ImageNet Classification Challenge Error

embedded VISION SUMMIT

Première utilisation du Deep learning dans la compétition

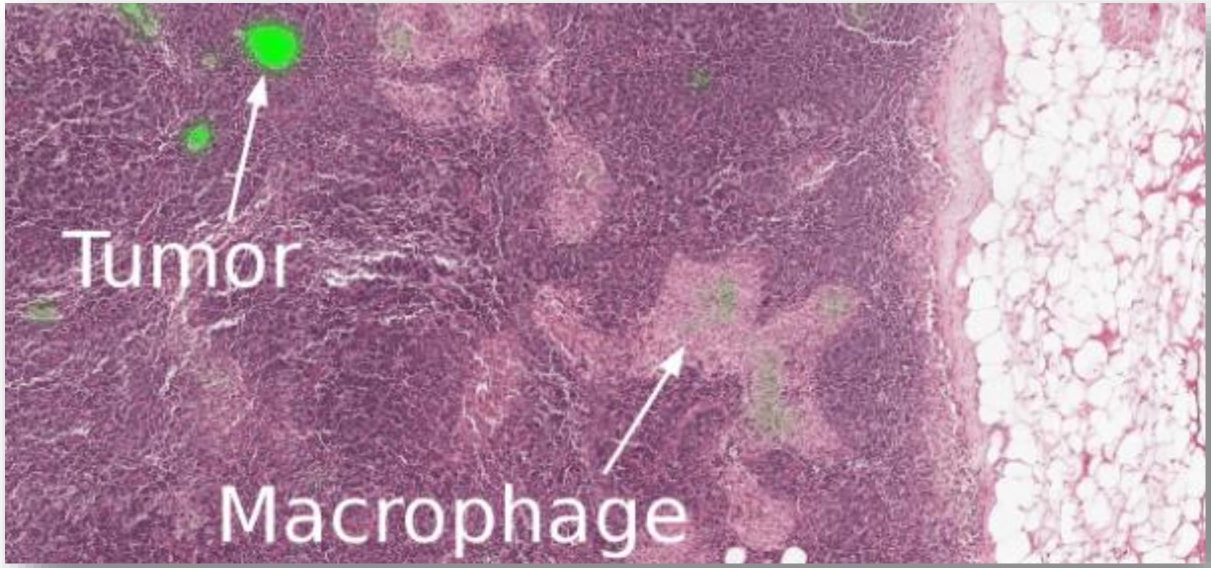
* Human Performance based on analysis done by Andrej Karpathy.
More details here.

Google Copyright © 2017 Google 12

Applications du deep learning

https://www.re-work.co/blog/deep-learning-daniel-mcduff-affectiva

Reconnaissance d'émotions

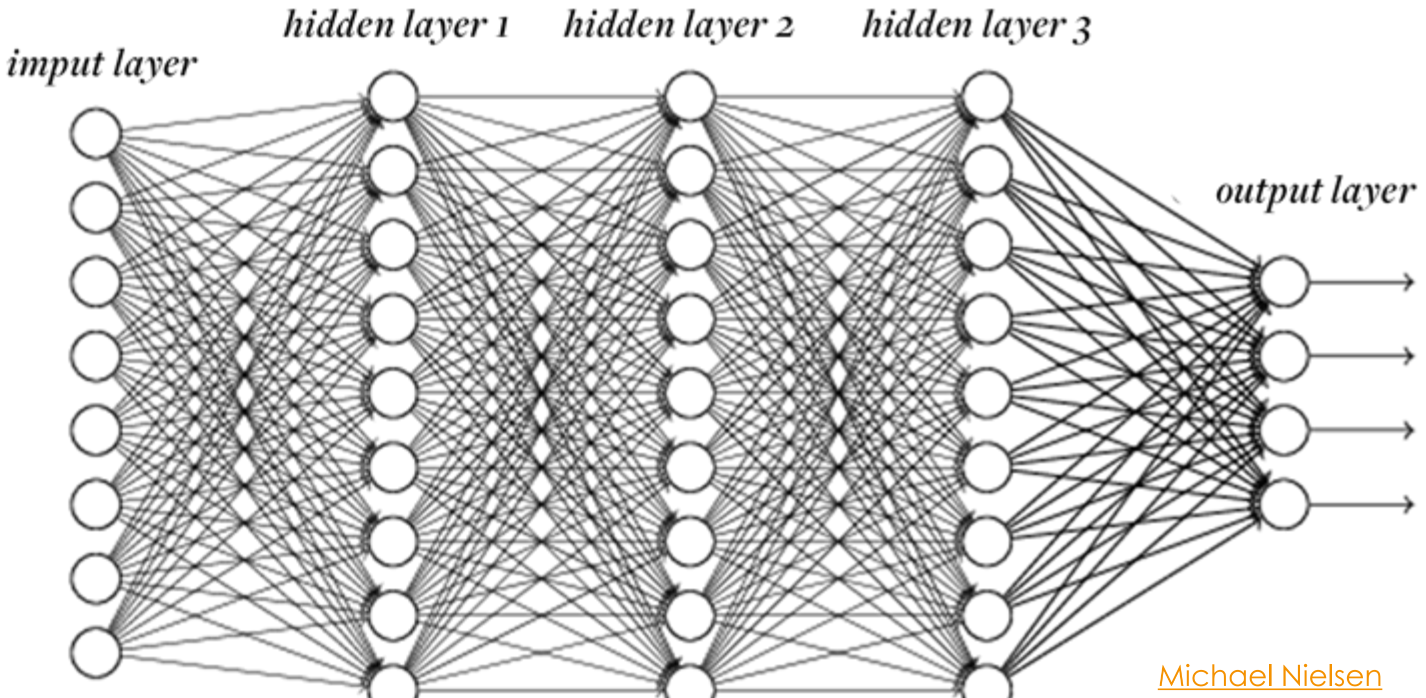


General News, Medical

Détection de cancers

Réseaux de neurones et deep learning

- Réseaux de neurones : Structure constituée d'un ensemble (couches) de briques élémentaires (neurones) effectuant chacune des opérations simples

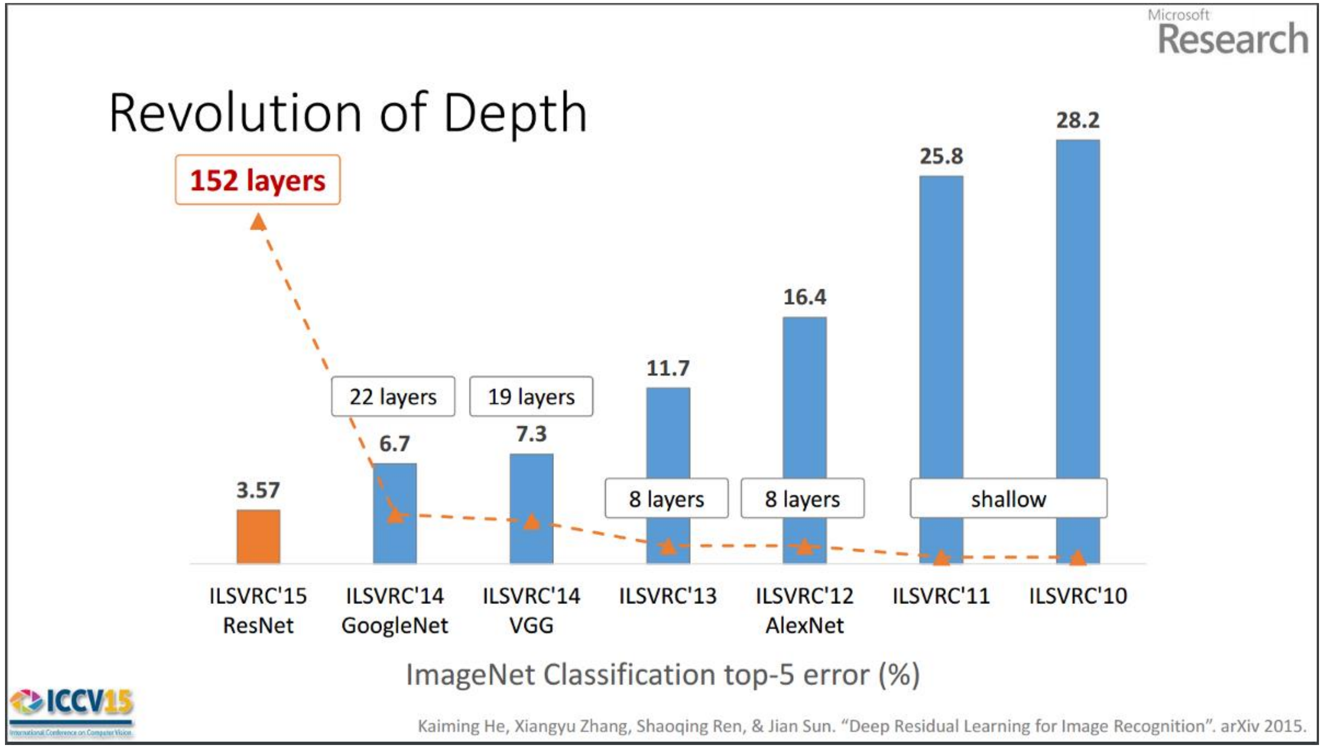


Classification :
$Output \in \{0,1\}$

Régression :
$Output \in \mathbb{R}, [0,1], \mathbb{R}^{+*}$ ...

Réseaux de neurones et deep learning

- Apprentissage **profond :** nombre de couches élevé



Mathématiquement : le calcul d'un neurone

Vocabulaire :

X : données d'entrée (ou couche précédente)

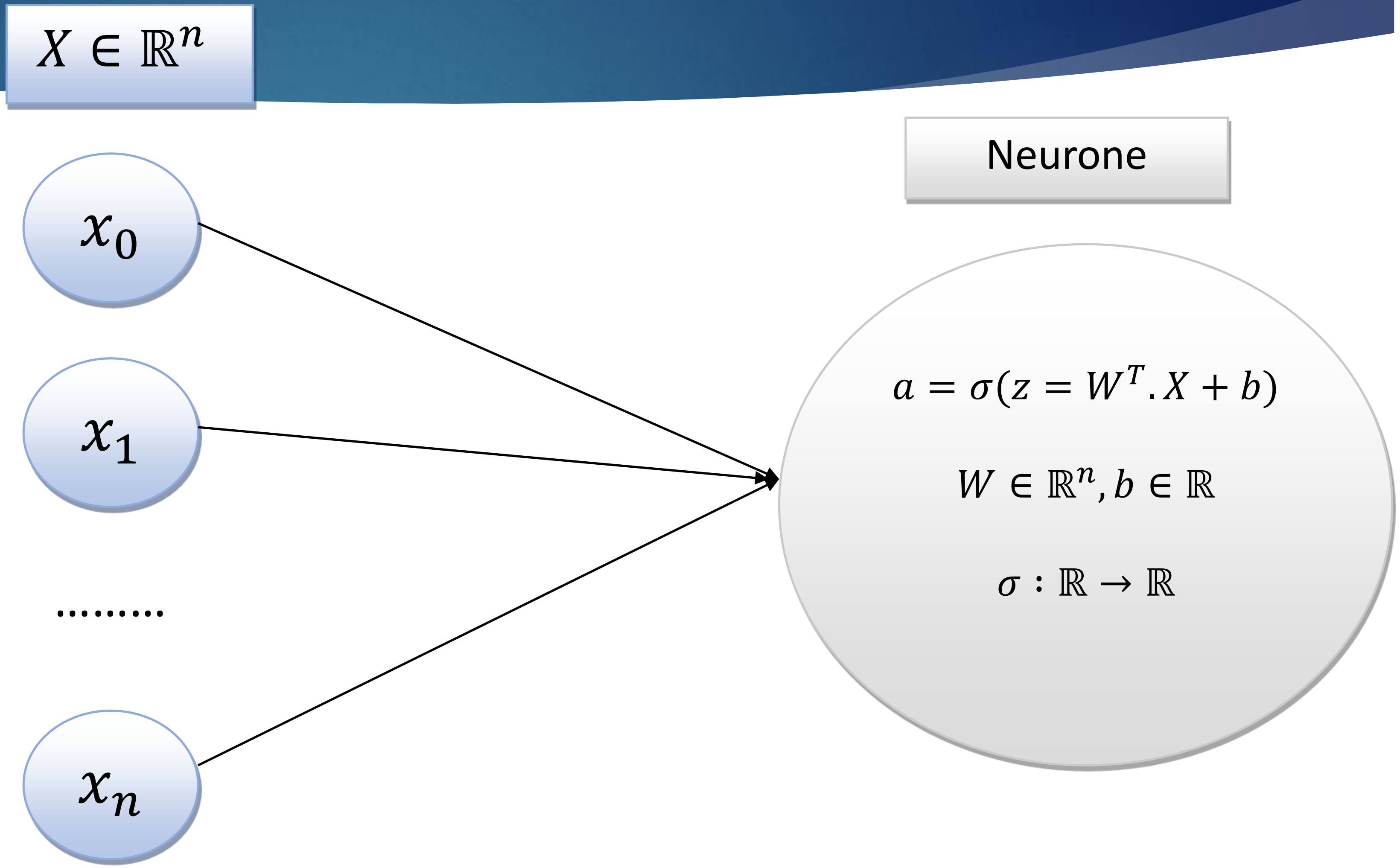
<u>W : poids (weight)</u>

<u>b : biais (bias)</u>

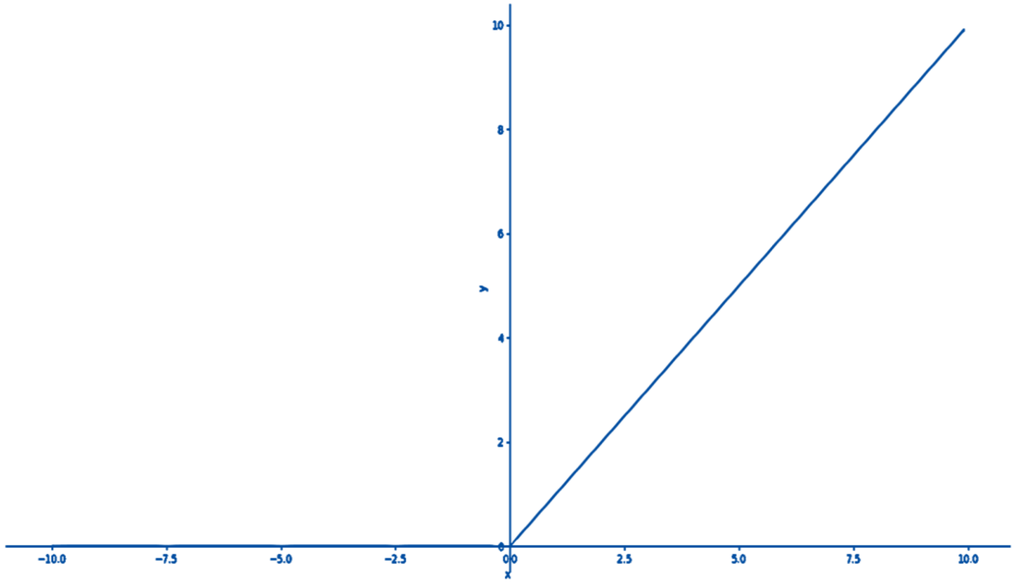
a : output du neurone

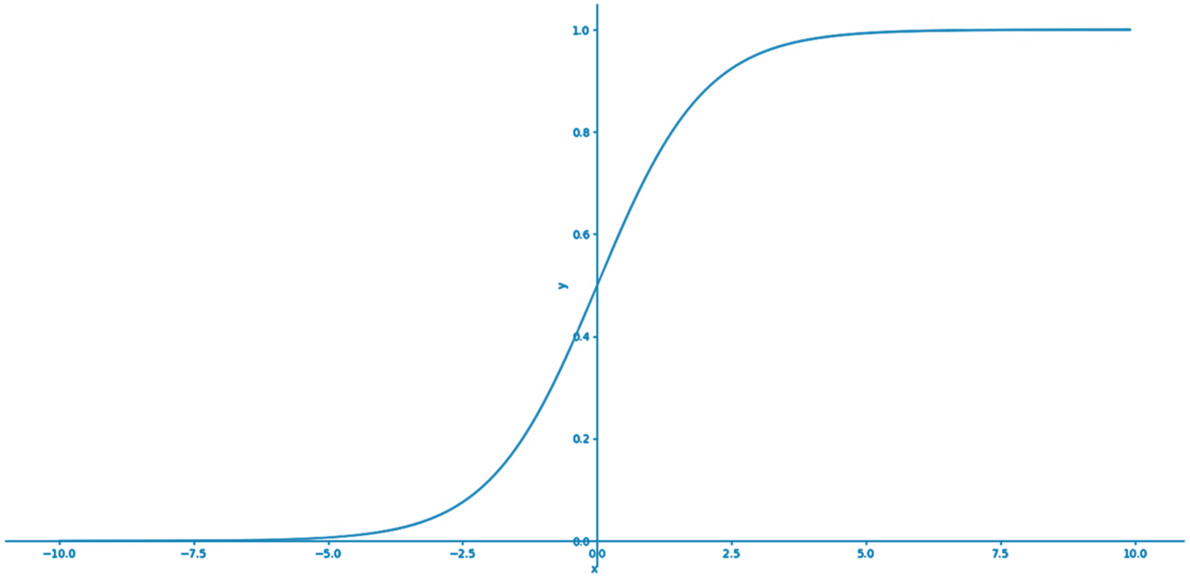
σ : fonction d'activation

z : calcul intermédiaire



La fonction d’activation



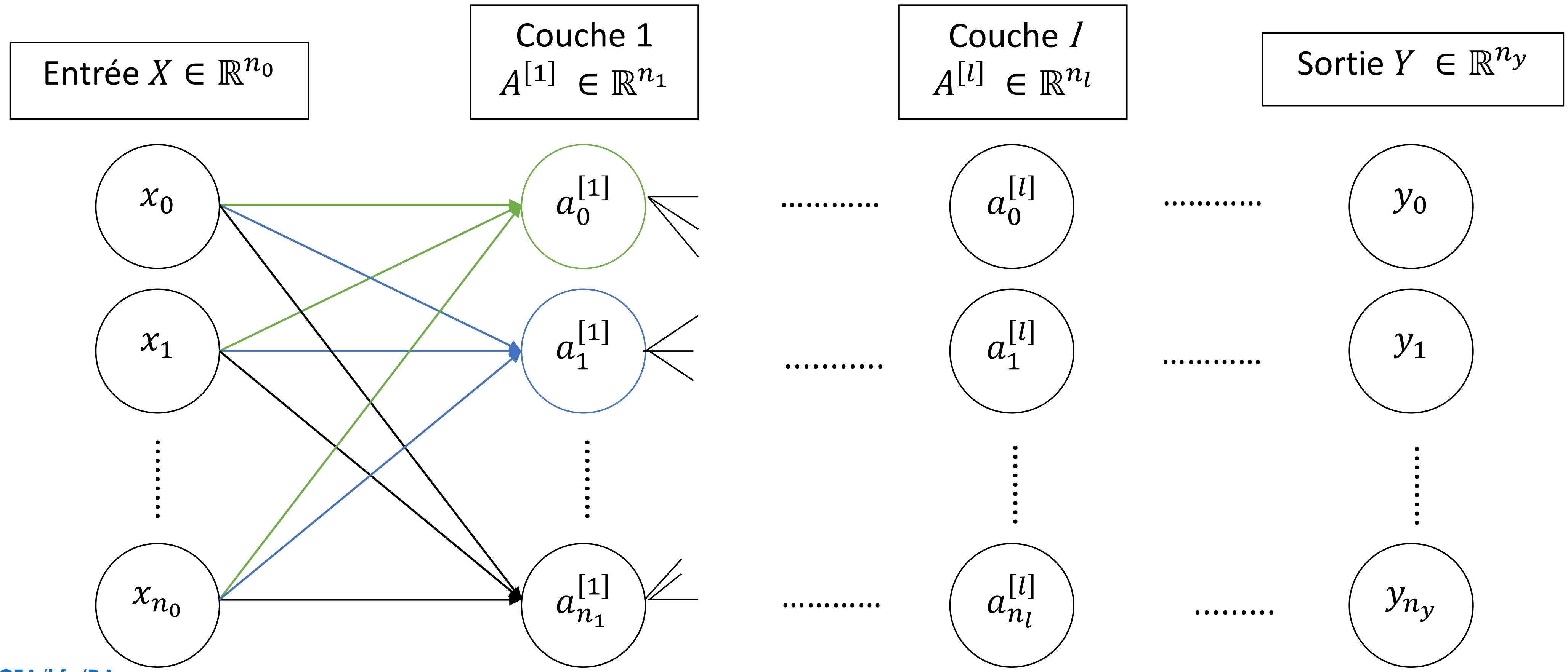


$ReLU(x) = \max(x, 0)$: La plus utilisée
Simplicité de calcul, gradient non évanescent

$Sigmoïde(x) = \frac{1}{1+e^{-x}}$: Pour la classification entre 0 et 1 en output

Et d’autres : $\tanh(x)$ (variante de la sigmoïde), $softmax(x) = \frac{e^{-z}}{\sum_{z_0 \in output} e^{-z_0}}$ (multi-classes exclusives),…

Et maintenant, un réseau



Théorème d'approximation universelle

- Soit $f : [0,1]^n \rightarrow [0,1]^m$. Pour tout $\epsilon > 0$, il existe un réseau de neurones à une seule couche intermédiaire RN tel que $\|f - RN\|_\infty < \epsilon$
- Cela signifie que toute fonction bornée peut être approximée par un réseau de neurones.

- Condition nécessaire pour le fonctionnement des réseaux de neurones
  - Montre l'intérêt des réseaux de neurones
- Condition non suffisante en pratique :
  - Le théorème ne dit rien sur le nombre de neurones : en fait, pour un réseau monocouche, énormément de neurones peuvent être nécessaires selon la fonction f à approximer

La fonction de coût

- **Évaluer** la qualité de la prédiction sur un jeu de données connues
- Notation : θ, paramètres du réseau (poids + biais) ; $\hat{Y}(\theta) = \left(\hat{\mathrm{y}}_{\mathrm{ij}}(\theta)\right)_{\mathrm{ij}}$, output du réseau j pour l'exemple i ; $Y = \left(y_{ij}\right)_{ij}$, données réelles pour la valeur j du vecteur de sortie associé à l'exemple i
- Loss function :

$$L(\theta) = f\left(\hat{Y}(\theta), Y\right)$$

- Exemples de fonctions de coût :
  - Distance euclidienne (au carré) : $f\left(\hat{Y}(\theta), Y\right) = \left\|\hat{Y}(\theta) - Y\right\|_2^2 = \sum_{ij}\left(\hat{y}_{ij}(\theta) - y_{ij}(\theta)\right)^2$
  - Binary cross-entropy, pour la classification 0 ou 1 : $f\left(\hat{Y}(\theta), Y\right) = \sum_{ij}\left(y_{ij}\ln\left(\hat{y}_{ij}(\theta)\right) + \left(1 - y_{ij}\right)\ln\left(1 - \hat{y}_{ij}(\theta)\right)\right)$
  - Distance en norme 1 : $f\left(\hat{Y}(\theta), Y\right) = \left\|\hat{Y}(\theta) - Y\right\|_1 = \sum_{ij}\left|\hat{y}_{ij}(\theta) - y_{ij}(\theta)\right|$
  - Et d'autres…

La fonction de coût : utilité

- Sur l'apprentissage :
  - La fonction de coût doit être minimisée : $\theta_{optimaux} = \underset{\theta}{\operatorname{argmin}} \left(f\left(\hat{Y}_{learning}(\theta), Y_{learning}\right)\right)$
  - Fonction non convexe !!! Minima locaux possibles, mais :
    - Plusieurs minima locaux aussi « bons » vis-à-vis de la fonction de coût (Yann Le Cun)
    - On peut tomber dans un mauvais minimum, mais ceci est rare : différentes techniques permettent d'éviter cela (dropout, régularisation… voir séance 2)

- Monitoring de l'apprentissage :
  - On peut vérifier que la fonction de coût décroît bien à chaque itération sur notre jeu de données (voir séances 2 et 3)

L'apprentissage

- Supposons que nous avons un jeu de données d'entrées X_i et de sorties Y_i et un réseau de neurones avec les paramètres $\theta = (W, B)$ qui prédit les sorties $\hat{Y}_i$ à partir des données X_i
- Minimisation de la fonction de coût L par descente de gradient (itérations) :

$$\theta \coloneqq \theta - \lambda \nabla \mathrm{L}(\theta)$$

λ est le taux d'apprentissage : valeur définie ou adaptée à chaque itération pour assurer la convergence

- Intérêt des réseaux de neurones : le gradient de la fonction de coût L se calcule « facilement », par succession de calculs élémentaires appelé backpropagation

Calcul du gradient : backpropagation

- Pour chaque poids $w_k^{[l]}$ et biais $b_k^{[l]}$ du neurone k de la couche l, on veut calculer pour chque exemple i :

$$\frac{\partial L_i}{\partial w_k^{[l]}} ; \frac{\partial L_i}{\partial b_k^{[l]}}$$

- Pour la dernière couche n :

$$\frac{\partial L_i}{\partial w_k^{[n]}} = \frac{\partial L_i}{\partial \widehat{Y_{i_k}}} \frac{\partial \widehat{Y_{i_k}}}{\partial w_k^{[n]}} ; \frac{\partial L_i}{\partial b_k^{[n]}} = \frac{\partial L}{\partial \widehat{Y_{i_k}}} \frac{\partial \widehat{Y_{i_k}}}{\partial b_k^{[n]}}$$

$$\widehat{Y_{i_k}} = \sigma\left(z_k^{[n]} = {w_k^{[n]}}^T a^{[n-1]} + b_k^{[n]}\right)$$

$$\frac{\partial \widehat{Y_{i_k}}}{\partial w_k^{[n]}} = \frac{\partial \widehat{Y_{i_k}}}{\partial z_k^{[n]}} \frac{\partial z_k^{[n]}}{\partial w_k^{[n]}} = \sigma'\left(z_k^{[n]}\right) a^{[n-1]} \in \mathbb{R}^{[m_{n-1}]} ; \frac{\partial \widehat{Y_{i_k}}}{\partial b_k^{[n]}} = \frac{\partial \widehat{Y_{i_k}}}{\partial z_k^{[n]}} \frac{\partial z_k^{[n]}}{\partial b_k^{[n]}} = \sigma'\left(z_k^{[n]}\right) \in \mathbb{R}$$

Calcul du gradient : backpropagation

$$\frac{\partial L_i}{\partial w_k^{[n]}} = \frac{\partial L_i}{\partial \widehat{Y_{i}}_k} \sigma'\left(z_k^{[n]}\right) a^{[n-1]} ; \frac{\partial L_i}{\partial b_k^{[n]}} = \frac{\partial L_i}{\partial \widehat{Y_{i}}_k} \sigma'\left(z_k^{[n]}\right)$$

- Exemple avec :

$$L_i = \left(\hat{Y}_i(\theta) - Y_i\right)^2$$
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

- On obtient :

$$\sigma'\left(z_k^{[n]}\right) = \sigma\left(z_k^{[n]}\right)\left(1 - \sigma\left(z_k^{[n]}\right)\right) = \widehat{Y_{i}}_k\left(1 - \widehat{Y_{i}}_k\right) \quad \text{Propriété du sigmoïde}$$
$$\frac{\partial L}{\partial \widehat{Y_{i}}_k} = 2\left(\widehat{Y_{i}}_k - Y_{i_k}\right)$$

Calcul du gradient : backpropagation

- Pour les autres couches $l \neq n$: de manière récursive

En rouge : par forward pass
En bleu : par récursivité

$$\frac{\partial L_i}{\partial w_k^{[l]}} = \frac{\partial L_i}{\partial a_k^{[l]}} \frac{\partial a_k^{[l]}}{\partial w_k^{[l]}} = \frac{\partial L_i}{\partial a_k^{[l]}} \sigma'\left(z_k^{[l]}\right) a^{[l-1]} \,;\, \frac{\partial L_i}{\partial b_k^{[l]}} = \frac{\partial L_i}{\partial a_k^{[l]}} \frac{\partial a_k^{[l]}}{\partial b_k^{[l]}} = \frac{\partial L_i}{\partial a_k^{[l]}} \sigma'\left(z_k^{[l]}\right); \left(\text{pour } l = 1, a^{[0]} = X\right)$$

$$\frac{\partial L_i}{\partial a_k^{[l]}} = \sum_j \frac{\partial L_i}{\partial a_j^{[l+1]}} \frac{\partial a_j^{[l+1]}}{\partial a_k^{[l]}}$$

$$a_j^{[l+1]} = \sigma\left(z_j^{[l+1]} = \sum_{k'} \left(w_j^{[l+1]}\right)_{k'} a_{k'}^{[l]} + b_j^{[l+1]}\right) \Rightarrow \frac{\partial a_j^{[l+1]}}{\partial a_k^{[l]}} = \left(w_j^{[l+1]}\right)_k \sigma'\left(z_j^{[l+1]}\right)$$

Ajustement des paramètres

- On connaît $\frac{\partial L_i}{\partial w_k^{[l]}}$ et $\frac{\partial L_i}{\partial b_k^{[l]}}$ pour chaque exemple i
- Finalement :

$$w_k^{[l]} := w_k^{[l]} - \lambda \frac{1}{N_{exemples}} \sum_i \frac{\partial L_i}{\partial w_k^{[l]}}$$
$$b_k^{[l]} := b_k^{[l]} - \lambda \frac{1}{N_{exemples}} \sum_i \frac{\partial L_i}{\partial b_k^{[l]}}$$

- On peut ne travailler simultanément que sur des sous-ensembles de la base de données (mini-batch), cela peut accélérer les calculs (voir séances 2 et 3)

Résumé des points importants

- Réseaux de neurones : calculs élémentaires $a = \sigma(z = W^T.X + b)$

- Chercher les paramètres $\theta = (W, b)$ qui minimisent une fonction de coût sur la base de données d'apprentissage : $\theta_{optimaux} = \underset{\theta}{\operatorname{argmin}} \left(f\left(\hat{Y}_{learning}(\theta), Y_{learning}\right)\right)$

- Apprentissage par descente de gradient : $\theta \coloneqq \theta - \lambda \nabla \mathrm{L}(\theta)$

Pour la suite

- Il n’est pas obligatoire de coder tous ces calculs soi-même ! Il existe des librairies qui font directement cela.

- Séance suivante :
  - Méthodologie pour mettre en place une architecture
  - Savoir évaluer son architecture et comment l’améliorer
  - Mise en place de méthodes de régularisation pour éviter l’overfitting
  - Savoir évaluer un réseau de neurones pour la classification

Quelques ressources

- Cours en ligne :
  - Coursera, spécialisation deep learning : https://www.coursera.org/specializations/deep-learning

  Quiz pour s'entraîner et exercices de programmation (Python) : fortement recommandé pour ceux qui veulent vraiment faire du deep learning

  - Cours de Yann Le Cun au Collège de France sur l'apprentissage profond (vidéos) : https://www.college-de-france.fr/site/yann-lecun/course-2015-2016.htm

  - Open Course MIT (1ère vidéo, les autres sont normalement proposées à la suite par Youtube) https://www.youtube.com/watch?v=TjZBTDzGeGg

Quelques ressources

- Vidéos Youtube :
  - Science Étonnante : Le deep learning (présentation générale)

  https://www.youtube.com/watch?v=trWrEWfhTVg
  - 3blue1brown : Calcul des réseaux de neurones illustrés (4 vidéos) :

  https://www.youtube.com/watch?v=aircAruvnKk (Introduction aux ANN)

  https://www.youtube.com/watch?v=IHZwWFHWa-w (Descente de gradient)

  https://www.youtube.com/watch?v=Ilg3gGewQ5U (Backpropagation version friendly)

  https://www.youtube.com/watch?v=tIeHLnjs5U8 (Backpropagation calculs)
  - Science4all : Playlist Intelligence artificielle, presque 50 vidéos sur l'IA (parfois un peu « philosophiques » et « sociologiques », d'autres plus techniques, réseaux de neurones à partir de la vidéo 40)

  https://www.youtube.com/watch?v=DrjkjPVf7Bw&list=PLtzmb84AoqRTl0m1b82gVLcGU38miqdrC

  (vidéos 42 et 43 non accessibles dans la playlist, mais toujours accessibles depuis la chaîne)