

Apprentissage Profond - Partie 1

Redouane Lguensat

Master Data Engineering de l'EHTP

Avril 2020

Table des matières

1 Introduction

2 Réseaux de Neurones Artificiels

3 Apprentissage profond

Pourquoi un tel engouement?



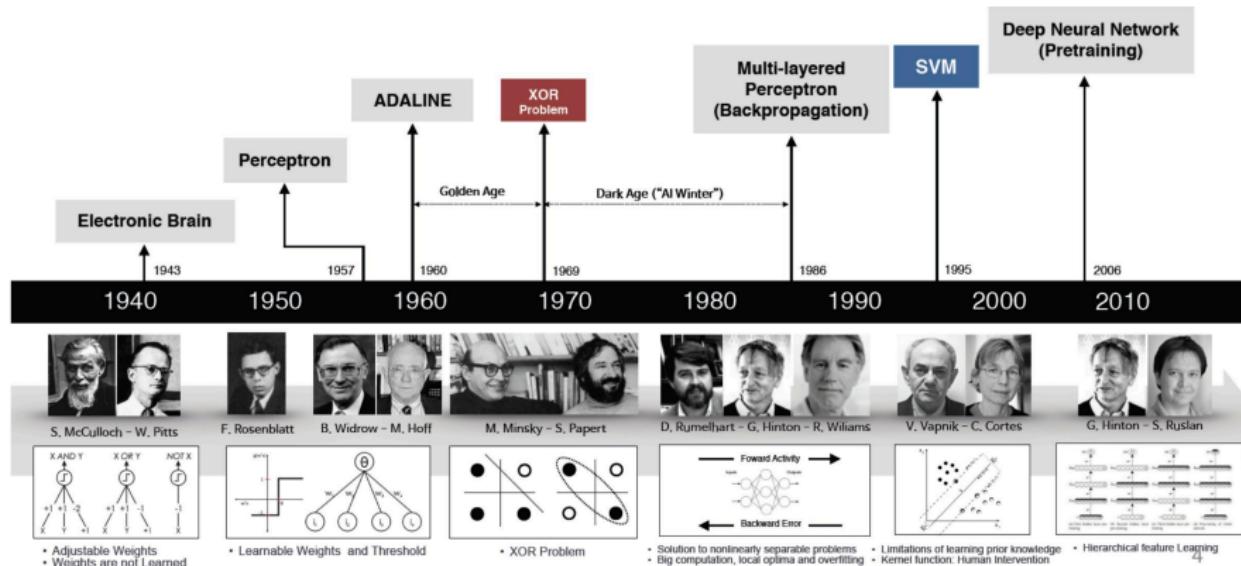
Source: <https://www.slideshare.net/Franrgo/gtam-vis-ban-the-new-cold-war-in-technology>

Impact socio-économique fort



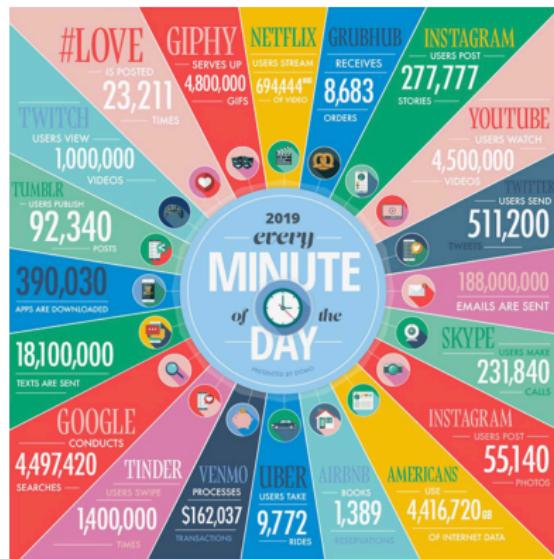
Découvertes majeures
en recherche scientifique

Un peu d'Histoire



Source: <https://www.slideshare.net/devview/251-implementing-deep-learning-using-cu-dnn>

Les catalyseurs de la révolution



<https://www.domo.com/learn/data-never-sleeps-7>

Volume de données grandissant



Avancées technologiques en capacité de calcul notamment sur cartes graphiques (GPU)



المدرسة الحسنية للأشغال العمومية
ECOLE HASSANIA DES TRAVAUX PUBLICS

Quelques applications de l'apprentissage profond

- Reconnaissance d'objets dans une image
- Traduction automatique
- Détection de contenu indésirable (ex: spam)
- Voiture autonome
- Assistance médicale
- Systèmes de recommandation
- Détection d'anomalies (ex: fraudes)
- Chatbots (agents conversationnels)
- Création d'oeuvre artistique
- Télédétection spatiale
- Jeux de table ou vidéo
- Transcription de la voix
- Génération de musique
- Recherche de nouveaux médicaments
- Détection de nouvelles galaxies
-

La liste est en pleine expansion!

Table des matières

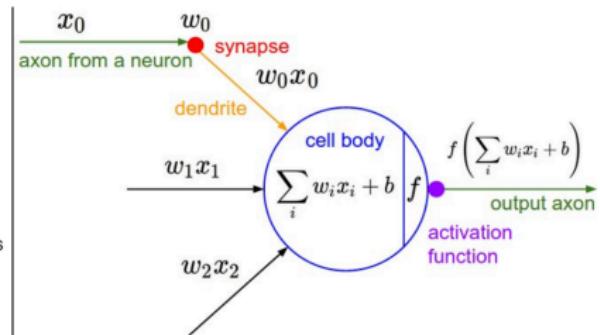
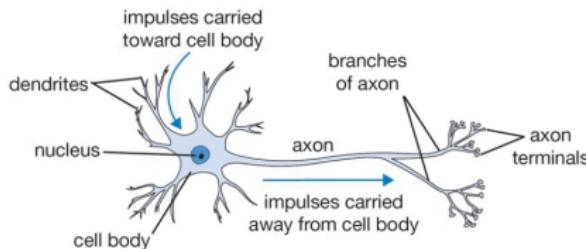
1 Introduction

2 Réseaux de Neurones Artificiels

3 Apprentissage profond

Inspiration de la biologie

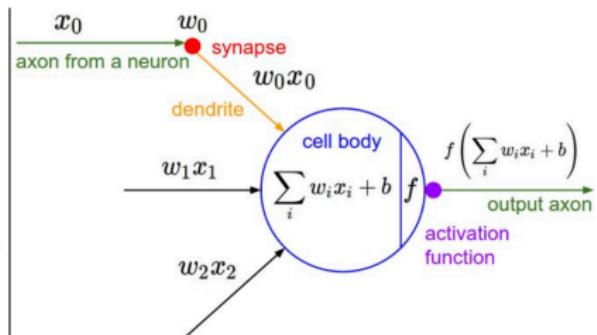
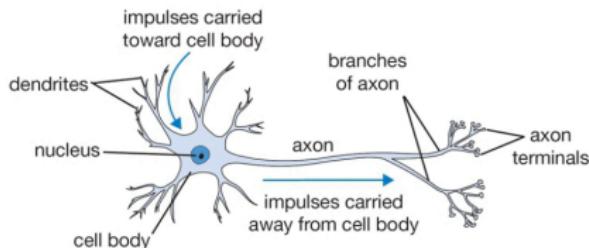
Neuron biologique vs neuron "artificial"



<https://cs231n.github.io/neural-networks-1/>

Inspiration de la biologie

Neuron biologique vs neuron "artificiel"



<https://cs231n.github.io/neural-networks-1/>

Les x_i sont les entrées du neurones, w_i sont appelés poids (*weights*) et b est un biais (*bias*). f est une fonction d'activation.

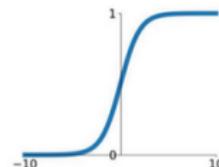
$$\text{sortie} = f\left(\sum_i w_i x_i + b\right)$$

Exemples classiques de fonctions d'activation

Servent à introduire de la nonlinéarité dans le modèle

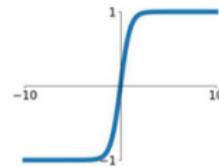
Sigmoid

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



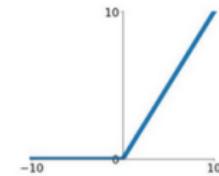
tanh

$$\tanh(x)$$



ReLU

$$\max(0, x)$$



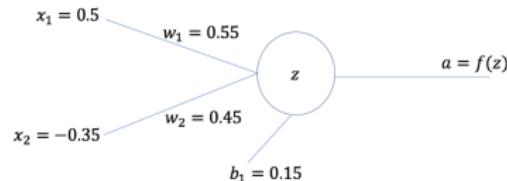
D'autres exemples de fonctions d'activation https://fr.wikipedia.org/wiki/Fonction_d%27activation

Un neurone artificiel en quelques lignes de code

```
from keras.models import Sequential
model = Sequential()
model.add(Dense(1, input_dim=5, activation='sigmoid'))
```

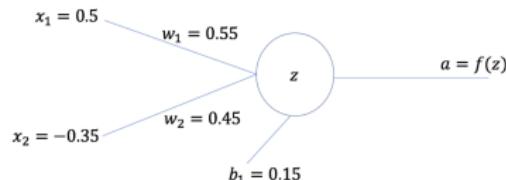
Exemple

Exercice 1: supposons que f est une sigmoïde, calculez a la sortie du neuron suivant:



Exemple

Exercice 1: supposons que f est une sigmoïde, calculez a la sortie du neuron suivant:



CO EHTP_MsDataEng.ipynb ☆

Fichier Modifier Affichage Insérer Exécution Outils Aide

+ Code + Texte

```
<> import numpy as np
     import math
```

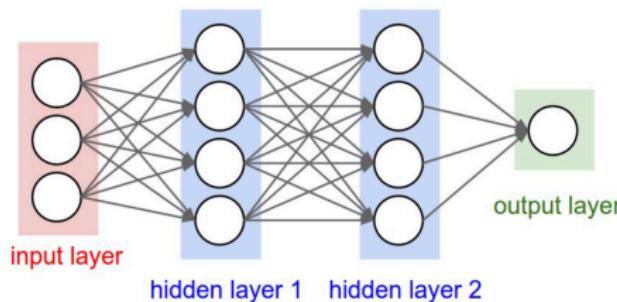
[2] def sigmoid(x):
 return 1 / (1 + math.exp(-x))

[3] sigmoid(0.55*0.5+(-0.35)*0.45+0.15)

0.5664790559676278

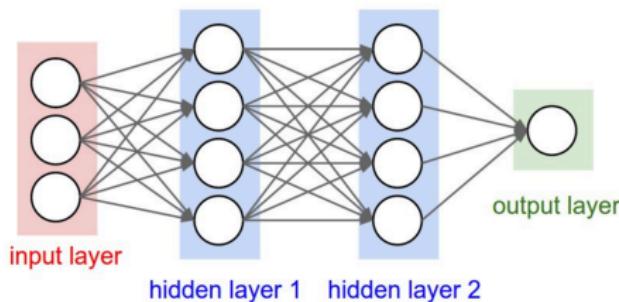
Réseaux de Neurones Artificiels

Les neurones artificiels peuvent être interconnectés et construire des **r  seaux de neurones artificiels**. Ces derniers peuvent aussi  tre compos s en couches (*layers*) de neurones. Un exemple tr s utilis  est le **perceptron multicouche** (*Multilayer perceptron (MLP)*).



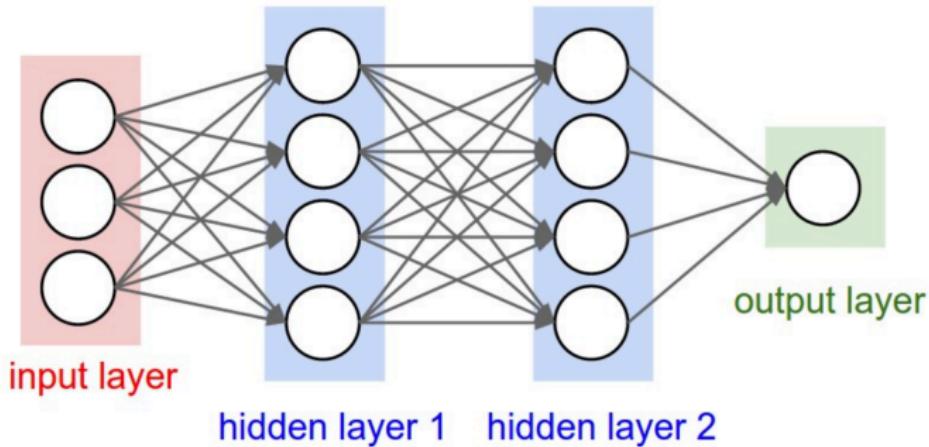
Réseaux de Neurones Artificiels

Les neurones artificiels peuvent être interconnectés et construire des **réseaux de neurones artificiels**. Ces derniers peuvent aussi être composés en couches (*layers*) de neurones. Un exemple très utilisé est le **perceptron multicouche** (*Multilayer perceptron (MLP)*).

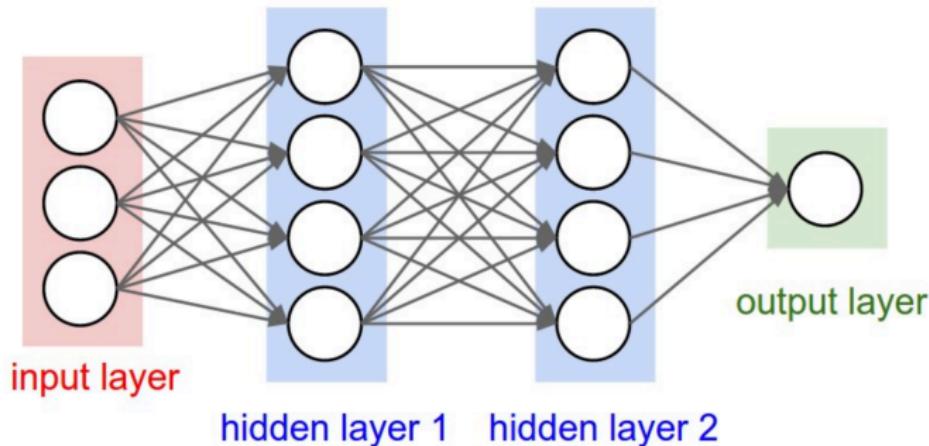


Le MLP est un réseau à propagation directe (*feed forward*), et il est complètement connecté (*fully connected*). Chaque neurone agrège de l'information en provenance de tous les neurones de la couche précédente.

Traduire un MLP en code Keras



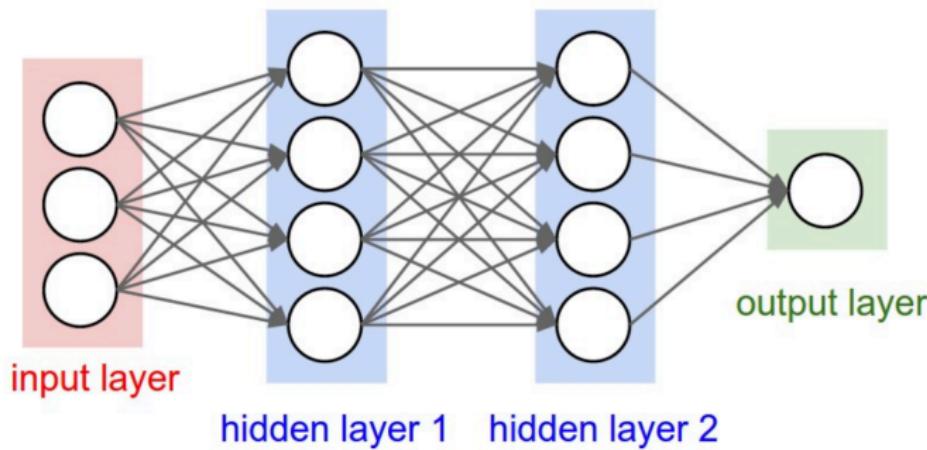
Traduire un MLP en code Keras



```
model = Sequential()  
model.add(Dense(4, activation='relu', input_shape=(3,)))  
model.add(Dense(4, activation='relu'))  
model.add(Dense(1))
```

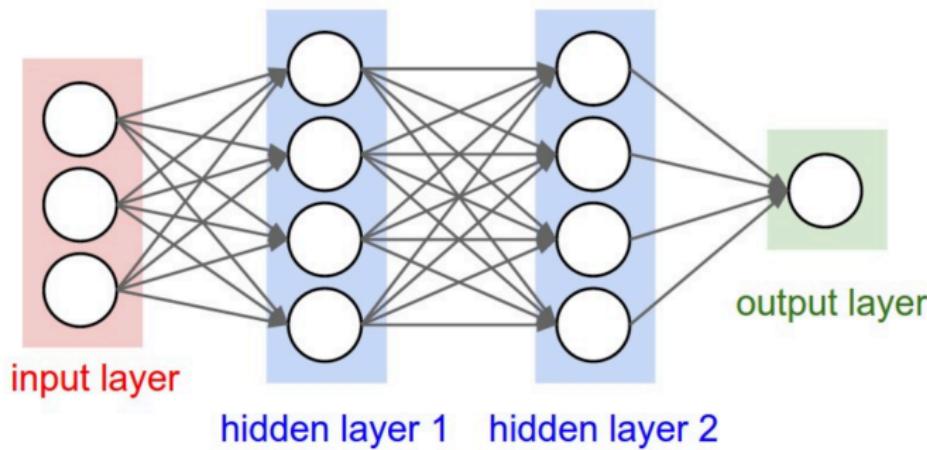
Entraîner des Réseaux de Neurones

Entraîner un réseau de neurones pour une tâche particulière revient à trouver les bons paramètres de ce réseaux: les poids w et les biais b de chaque neurone.



Entraîner des Réseaux de Neurones

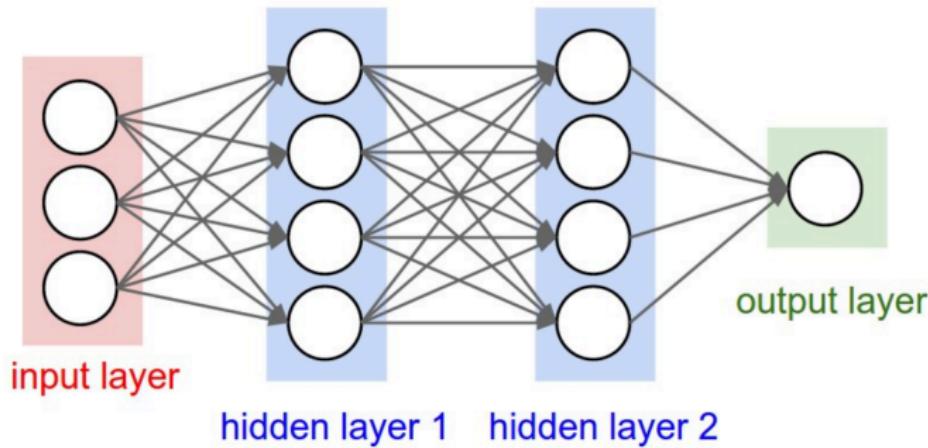
Entraîner un réseau de neurones pour une tâche particulière revient à trouver les bons paramètres de ce réseaux: les poids w et les biais b de chaque neurone.



Exercice 2: Le réseau illustré a combien de paramètres?

Entraîner des Réseaux de Neurones

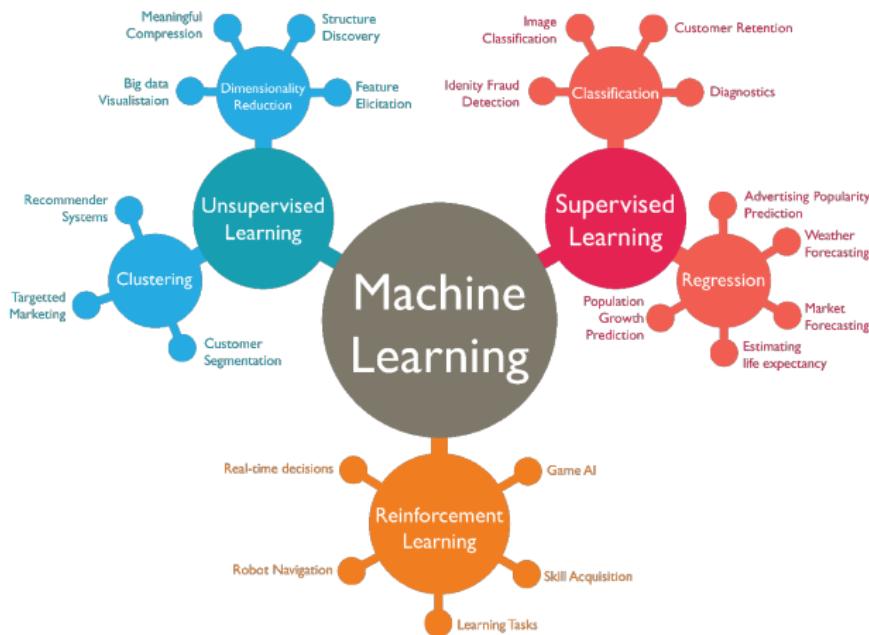
Entraîner un réseau de neurones pour une tâche particulière revient à trouver les bons paramètres de ce réseaux: les poids w et les biais b de chaque neurone.



Exercice 2: Le réseau illustré a combien de paramètres? **41**

Backpropagation + SGD: au cœur de l'entraînement des réseaux de neurones

Avant d'entraîner des réseaux de neurones, il est nécessaire de comprendre le cadre de la tâche souhaitée: apprentissage supervisé/non supervisé/par renforcement, problème de classification/régression/clustering etc..



Exemple courant: Classification multiclasse

Problème classique de l'apprentissage supervisé

Entrées (x_n)



Labels (y_n)

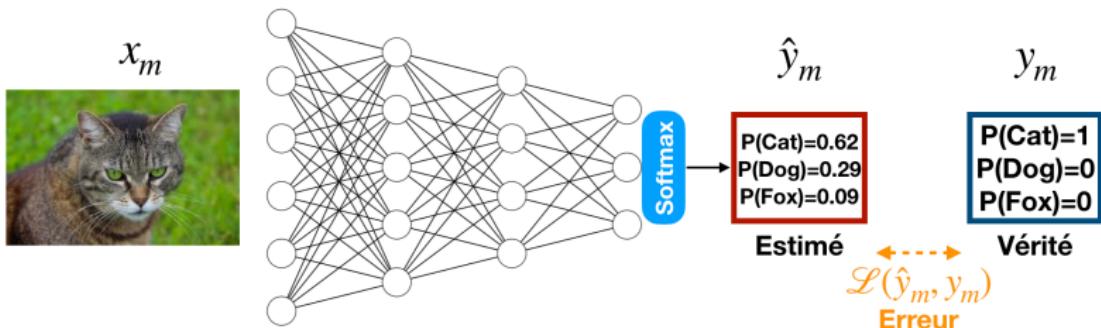
$P(\text{Cat})=1$
 $P(\text{Dog})=0$
 $P(\text{Fox})=0$

$P(\text{Cat})=0$
 $P(\text{Dog})=1$
 $P(\text{Fox})=0$

$P(\text{Cat})=0$
 $P(\text{Dog})=0$
 $P(\text{Fox})=1$

Les labels sont écrit en format *one-hot encoding*, et pourraient être vus comme des probabilités

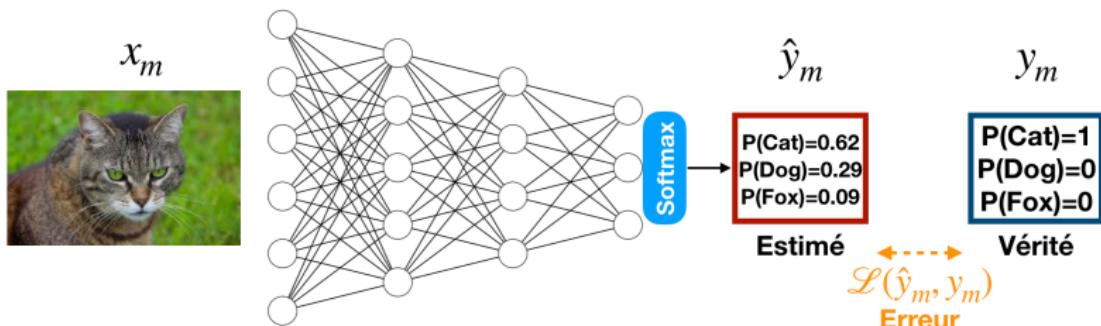
Passons un exemple x_m de la base de données d'entraînement à travers le réseau puis comparons la sortie avec \hat{y}_m la vérité y_m



Pour les problèmes de classification multiclasse, le softmax est utilisé comme fonction d'activation de la dernière couche. Il sert à ramener les sorties à des probabilités $\mathbb{P}(\text{classe}|\text{entrée})$:

$$\text{softmax}(z_j) = \frac{e^{z_j}}{\sum_i e^{z_i}}$$

L'information qui guide un réseau de neurones supervisé est l'erreur entre son estimation et la vérité. L'erreur est calculée à travers une fonction de coût *loss function* \mathcal{L} .



Dans le cas de la classification multiclass nous utilisons l'entropie croisée catégorique (*categorical cross-entropy*):

$$\mathcal{L}_{\text{cross-entropy}}(\hat{\mathbf{y}}, \mathbf{y}) = - \sum_i y_i \log(\hat{y}_i)$$

Rétropropagation du gradient (Backprop)

L'algorithme le plus populaire pour entraîner des réseaux de neurones est celui de la rétropropagation du gradient. L'idée est de définir une fonction de coût \mathcal{L} puis mettre à jour les paramètres en utilisant un algorithme d'optimization tel que la descente de gradient (*Gradient Descent*).

Entraîner un réseaux de neurones revient à:

- Au début, initialiser les paramètres du réseau
- Passer un échantillon (ou un batch) de données d'apprentissage par le réseau pour obtenir les valeurs de sortie.
- Calculer l'erreur (la fonction coût) commise par le réseau
- Utiliser le backprop pour calculer les dérivées de la fonction de coût par rapport à tous les paramètres du réseau.
- Mettre à jour tous les paramètres du réseau
- Passer un autre batch de données et refaire les étapes jusqu'à convergence.

Backprop avec SGD

Pour pouvoir utiliser un grand nombre de données d'apprentissage, l'algorithme d'optimization le plus utilisé avec le backprop est la **descente de gradient stochastique** (SGD)

En prenant un batch de n données d'entrées/sorties $x^{(i:i+n)}; y^{(i:i+n)}$ et un taux d'apprentissage η (*learning rate*) , la descente de gradient se fait sur les poids et biais:

$$w = w - \eta \cdot \nabla_w \mathcal{L} \left(w; x^{(i:i+n)}; y^{(i:i+n)} \right)$$

$$b = b - \eta \cdot \nabla_b \mathcal{L} \left(b; x^{(i:i+n)}; y^{(i:i+n)} \right)$$

Exemples de fonction de coût

Les fonctions de coût dépendent de l'application souhaitée, en voici quelques exemples assez classiques:

- Classification
 - Entropie croisée binaire (cas spécial de la CE vue avant)

$$BCE = -\frac{1}{n} \sum_{i=0}^n (y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i))$$

- Régression:
 - Erreur quadratique moyenne ou Erreur absolue moyenne:

$$MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n} \quad MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n}$$

Important: les fonctions de coût doivent être différentiables pour permettre l'utilisation du backprop.

Importance du choix du learning rate

Le learning rate est probablement l'hyperparamètre le plus important pendant l'entraînement

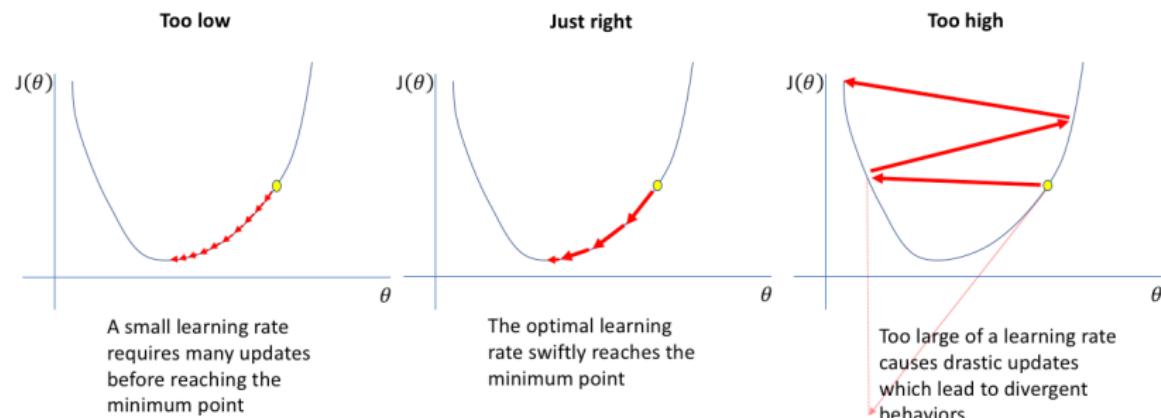


Image: J. Jordan

Importance du choix du learning rate

Le learning rate est probablement l'hyperparamètre le plus important pendant l'entraînement

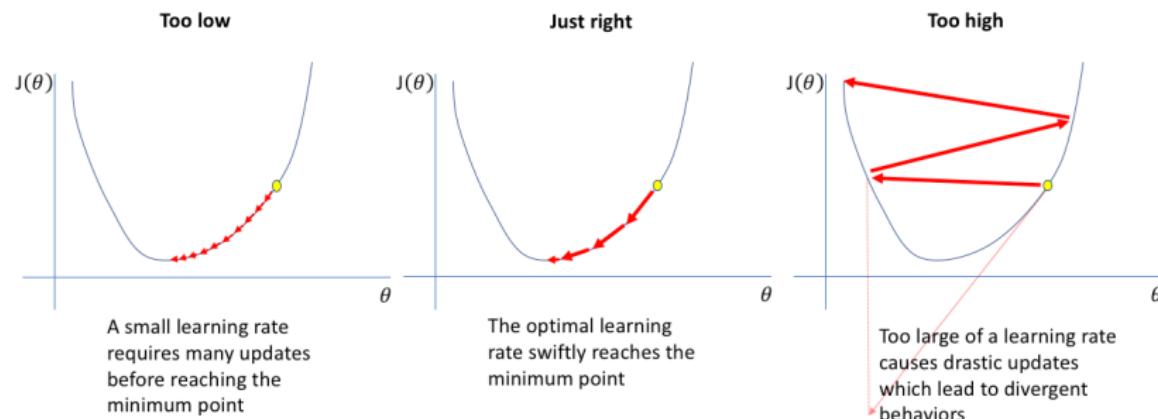
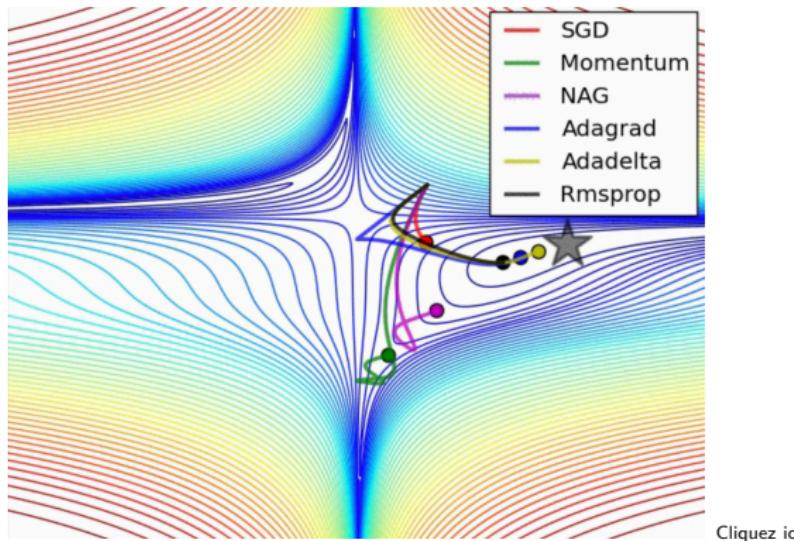


Image: J. Jordan

Solution pratique: learning rate adaptatif

Comparaison des variantes du SGD

Excellent travail de review par Sébastien Ruder



En pratique "Adam" est largement utilisé

Backprop avec SGD en pratique avec Keras

Heureusement vous n'avez pas à tout coder! quelques ligne suffisent

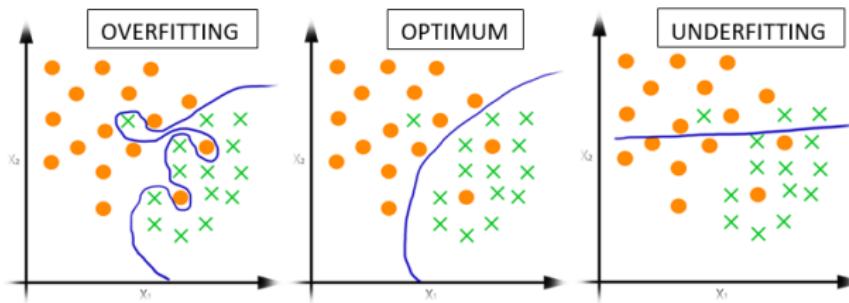
```
model = Sequential()
model.add(Dense(5, activation='relu', input_shape=(6,)))
model.add(Dense(4, activation='relu'))
model.add(Dense(3, activation='softmax'))

model.compile(loss='categorical_crossentropy',
               optimizer='adam')
model.fit(X_train, y_train, epochs=20, batch_size=128)
```

Attention au surapprentissage/sous-apprentissage

Un bon réseau de neurones performe aussi bien sur les données d'apprentissage que sur les données de test, c'est ce qu'on appelle la capacité de généralisation du réseau.

- Utiliser un nombre très faible de paramètres peut causer un sous apprentissage: le réseau est mauvais en training et en test
- Un nombre très grand de paramètres peut causer un surapprentissage: le réseau est excellent en training et mais mauvais en test



Combattre l'overfitting: régularisation des poids

En pratique, on tombe souvent dans le cas de l'overfitting. Les principales techniques pour le combattre sont:

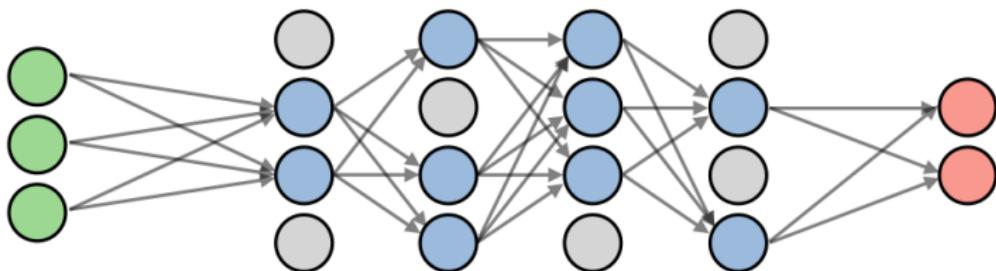
- La régularisation des poids (*weight regularization*): consiste à ajouter un terme d'erreur dans la fonction de coût qui évite de se retrouver avec des poids trop élevés (petite différence en entrée résulte en une grande différence en sortie). Deux exemples classiques sont la régularisation L2 et L1.

$$\text{L2 : } \mathcal{L} + \lambda \cdot \|W\|_2^2 \quad \text{L1 : } \mathcal{L} + \lambda \cdot \|W\|_1$$

W sont les poids (et/ou biais) du réseau. λ est le coefficient de régularisation, c'est un hyperparamètre à tuner qui contrôle le niveau de régularisation du réseau

Combattre l'overfitting: le dropout

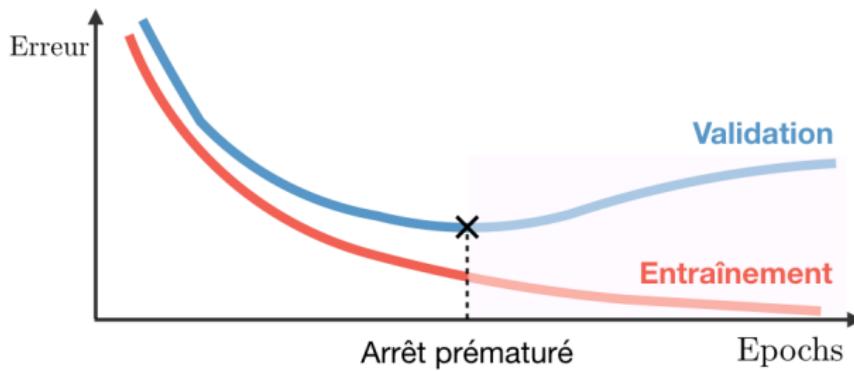
- Le dropout: consiste à "tuer" aléatoirement des neurones dans un réseau de neurones avec une probabilité $p > 0$. Ceci force le modèle à éviter de trop s'appuyer sur un ensemble particulier de neurones et de mieux généraliser.



CS230

Combattre l'overfitting: L'arrêt prématué

- L'arrêt prématué (*early-stopping*) est probablement la manière la plus simple pour combattre l'overfitting, elle consiste à suivre les erreurs en training et validation et à stopper l'entraînement dès que le loss de validation atteint un plateau ou commence à augmenter.



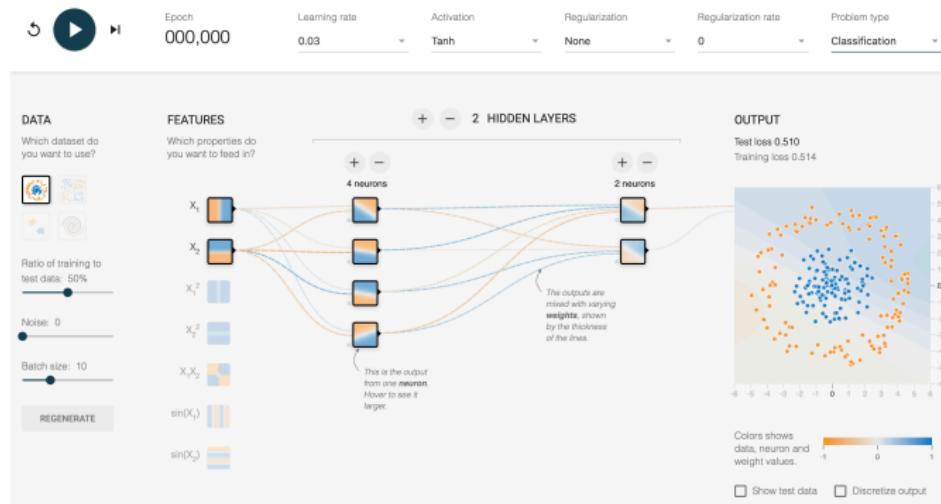
CS230



المدرسة الحسنية للشغاف العمومية
ECOLE HASSANIA DES TRAVAUX PUBLICS

Outil pédagogique intéressant

Terrain de jeu sur navigateur par Tensorflow (Google)



Cliquez ici

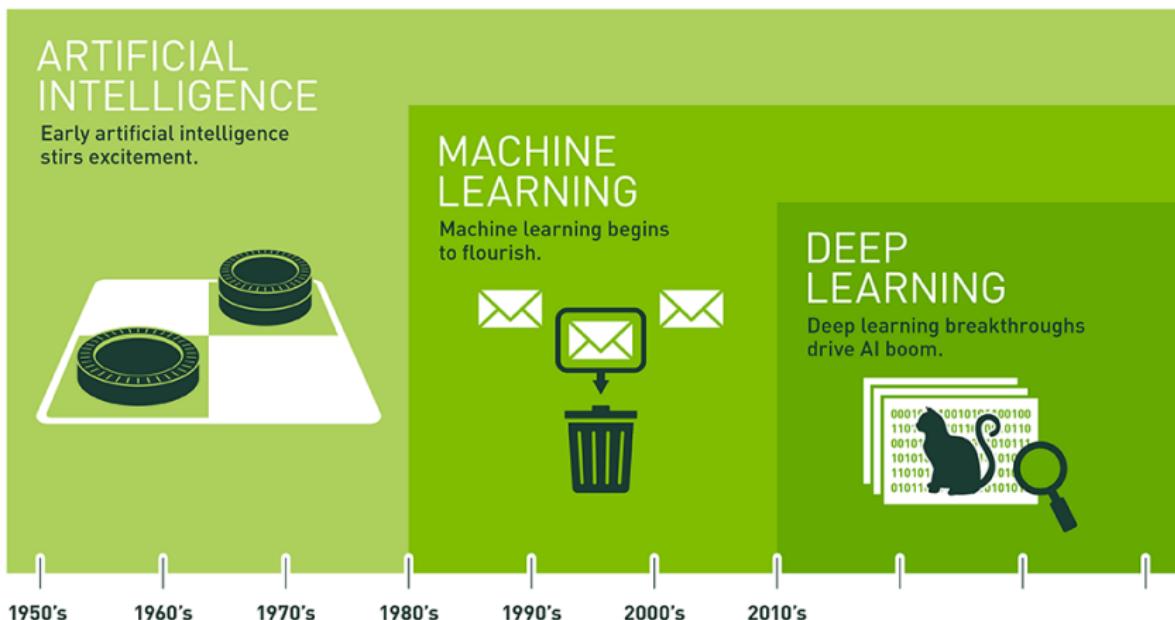
Table des matières

1 Introduction

2 Réseaux de Neurones Artificiels

3 Apprentissage profond

Un sous domaine du Machine Learning



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

Apprentissage profond

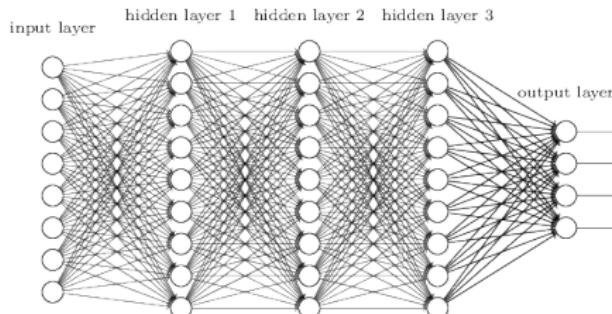
Slide tiré d'une présentation de Prof. Geoffrey Hinton (un des héros principaux de la renaissance des réseaux de neurones)

What was actually wrong with backpropagation in 1986?

- We all drew the wrong conclusions about why it failed.
The real reasons were:
 1. Our labeled datasets were thousands of times too small.
 2. Our computers were millions of times too slow.
 3. We initialized the weights in a stupid way.
 4. We used the wrong type of non-linearity.

Apprentissage profond

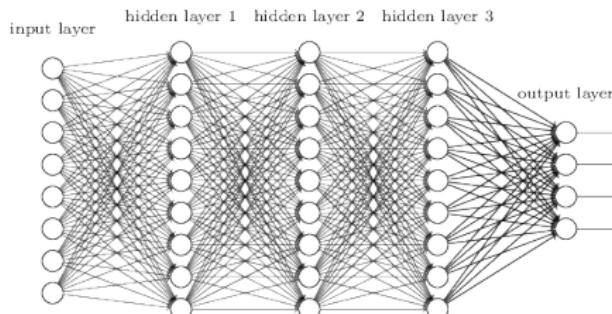
- C'est l'idée de l'utilisation de capacités de calculs avancées et de données massives (Big Data) pour entraîner des réseaux de neurones profond avec une multitude de couches.



Credits: M. Nielsen

Apprentissage profond

- C'est l'idée de l'utilisation de capacités de calculs avancées et de données massives (Big Data) pour entraîner des réseaux de neurones profond avec une multitude de couches.



Credits: M. Nielsen

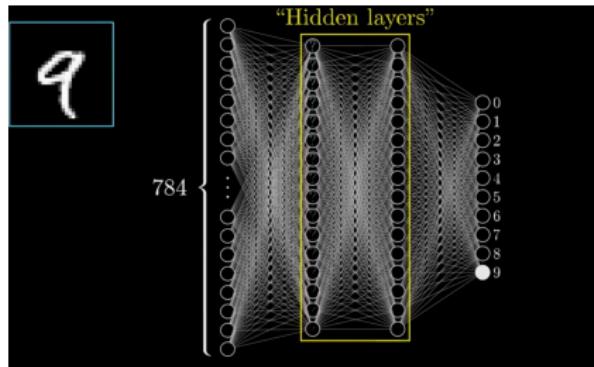
- "At which problem depth does Shallow Learning end, and Deep Learning begin? Discussions with DL experts have not yet yielded a conclusive response to this question." - Jürgen Schmidhuber



MNIST: Le "Hello world" de l'apprentissage profond

Dataset d'images en noir et blanc de chiffres manuscrits qui regroupe 60000 images d'apprentissage et 10000 images de test, 28 pixels de côté. Historiquement utilisé pour comparer les algorithmes de machine learning sur le problème de reconnaissance de chiffres manuscrits (utilisé par exemple pour les codes postaux).

Maintenant considéré comme problème "résolu" mais toujours utile pédagogiquement.



3Blue1Brown

MNIST: Le "Hello world" de l'apprentissage profond

Tutoriel sur Colaboratory ([Cliquez ici](#))

Prochain cours: Partie 2

L'apprentissage profond c'est surtout aussi le design et l'entraînement d'architectures adaptés à l'application désirée (classification, régression, etc.) et au type de données (signal, image, vidéo, etc..).
Nous verrons tout cela au prochain cours.

Des questions?