Réseaux de neurones profonds pour l'IA : Fondements théoriques et applications

Dominique Fourer

Université du Temps Libre (UTL)

27 mai 2025

Syllabus

Objectifs :

- Introduction à l'apprentissage profond
- Compréhension des concepts théoriques sous-jacents
- Exemples d'applications

Contenu

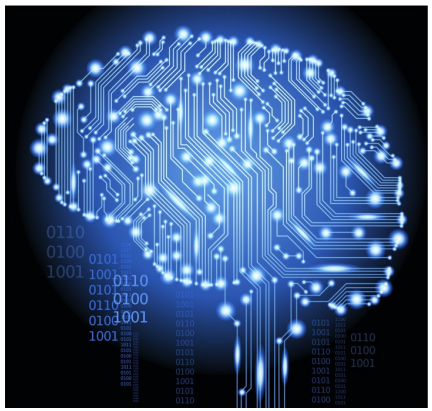
1. Introduction aux réseaux de neurones multicouches et leur entraînement
2. Les réseaux de neurones convolutifs
3. Quelques travaux récents et état de la recherche

Sources :

[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.
[2] Blaise Hanczar, Cours d'apprentissage profond. Univ. Évry Paris-Saclay.

Plan

Naissance de l'IA [Minsky, McCarthy, 1956]



IA : une tentative de définition...

Ensemble des théories et techniques capables de simuler certains traits de l'intelligence humaine.

The Logic Theorist [Newell, Shaw, Simon 1956] :
Résolution de problèmes et démonstration automatique de théorèmes.

Introduit 3 concepts :

- **Combinatoire** (i.e. exploration de graphe)
- **Heuristique** (solution calculable)
- **Liste de procédures** (e.g. λ-calcul, IPL, LISP [McCarthy, 58])

Algorithmes inspirés de la biologie

Algorithmes évolutionnaires

Inspirés de la théorie de l'évolution

- Stratégie d'évolution [Barricelli et al, 1954]
- Automates cellulaires [Neumann et al, 1966], (jeu de la vie [Conway, 1970])
- Algorithmes génétiques [Holland et al., 1973]

Intelligence distribuée "*swarm intelligence*"

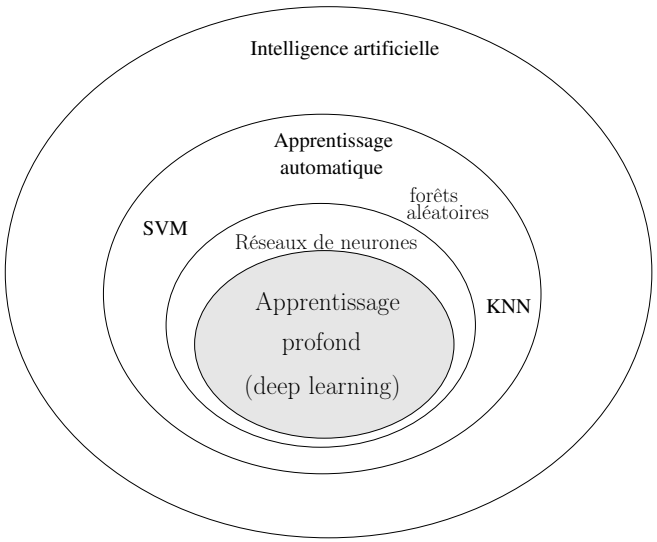
Repose sur la stigmergie [Grassé, 1959]

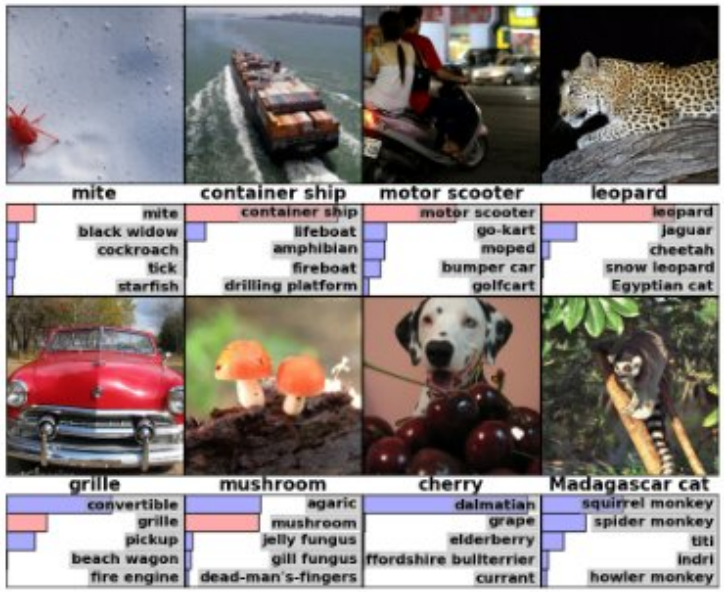
- Essaim particulaire [Kennedy, Eberhart, 1995]
- Algorithmes de colonies de fourmis [Ebling,Dorigo et al. 1989-2005]

Réseaux de neurones artificiels

- Neurone formel [McCulloch, Pitts, 1943] [Hebb, 1949]
- Perceptron [Rosenblatt, 1957], récurrent [Hopfield, 1982], multi-couche [Rumelhart, LeCun et al. 1986]
- Rétropropagation du gradient [Werbos, 1975] [Parker, LeCun, 1985] [Hinton, Williams, 1986]
- Réseaux convolutifs (CNN) [LeCun, 1998]

Le succès du *deep learning* depuis 2012





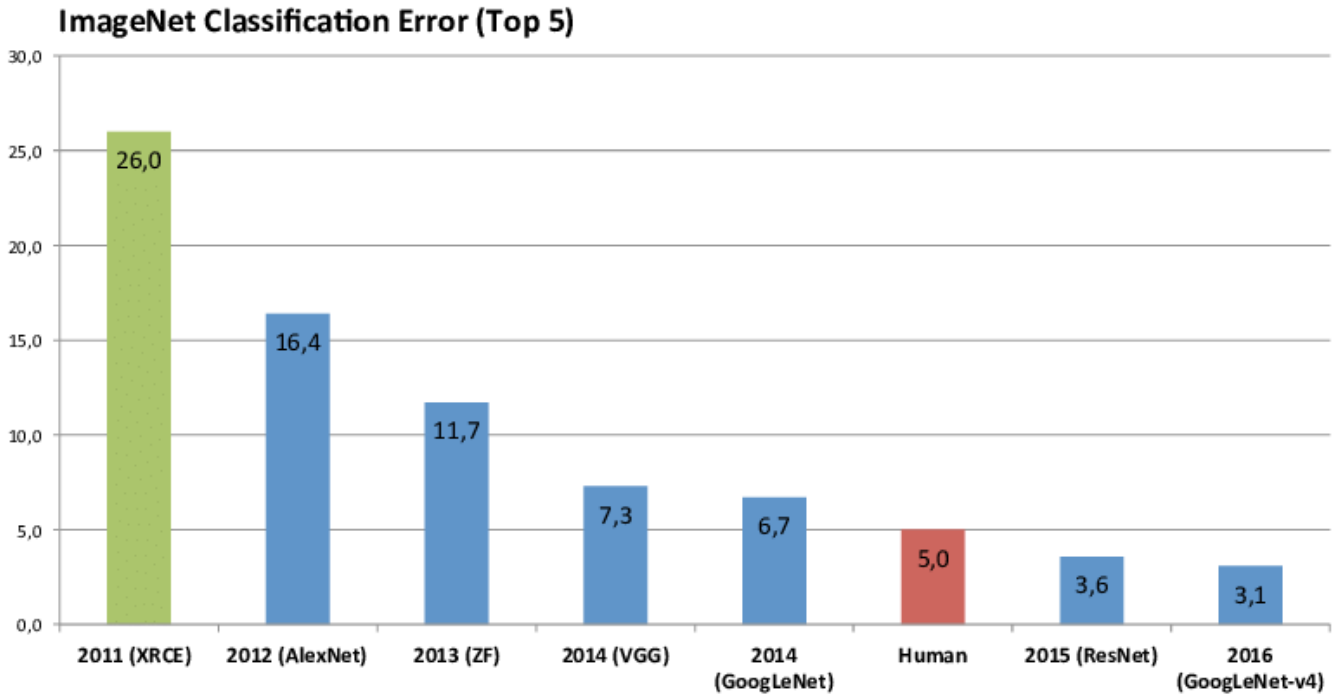
`http://image-net.org/challenges/LSVRC/`

≈ 1,2 millions d'images, 1000 classes [ILSVRC2012] [Krizhevsky et al. NIPS'12]

- Excellentes performances
- Grande masse de données annotées
- Parallélisation des algorithmes
- Moyens de calculs accessibles (e.g. GPU, clusters, etc.)

	Affiliation	Taux d'erreur	Description
1	U. Toronto	15,31 % (-10,8%)	**Deep CNN**
2	U. Tokyo	26,172 %	desc. + classif.
3	U. Oxford	26,979 %	desc. + classif.
4	Xerox/INRIA	27,058 %	desc. + classif.

Avantages du *deep learning*



- État de l'art en vision par ordinateur [Azizpour et al., 2016]
- Forte capacité de généralisation
- Apprend automatiquement une représentation discriminante (*deep features*)
- Surpasse l'humain dans de plus en plus de domaines (e.g. reconnaissance d'images, jeu d'échec, Alphago, Alphastar, etc.)
- Une infinité de domaines d'application (e.g. biomédical, finance, astronomie, reconnaissance vocale, etc.)

Exemple 1 : Reconnaissance d'objets

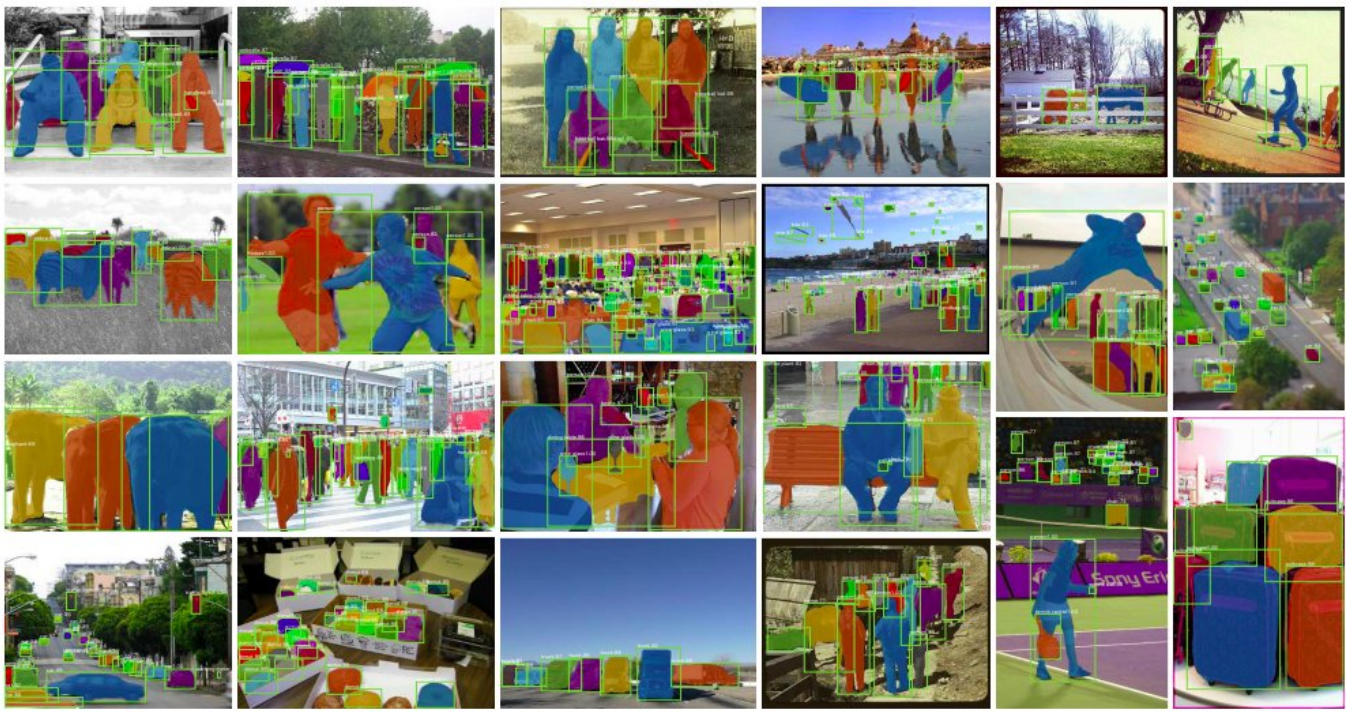
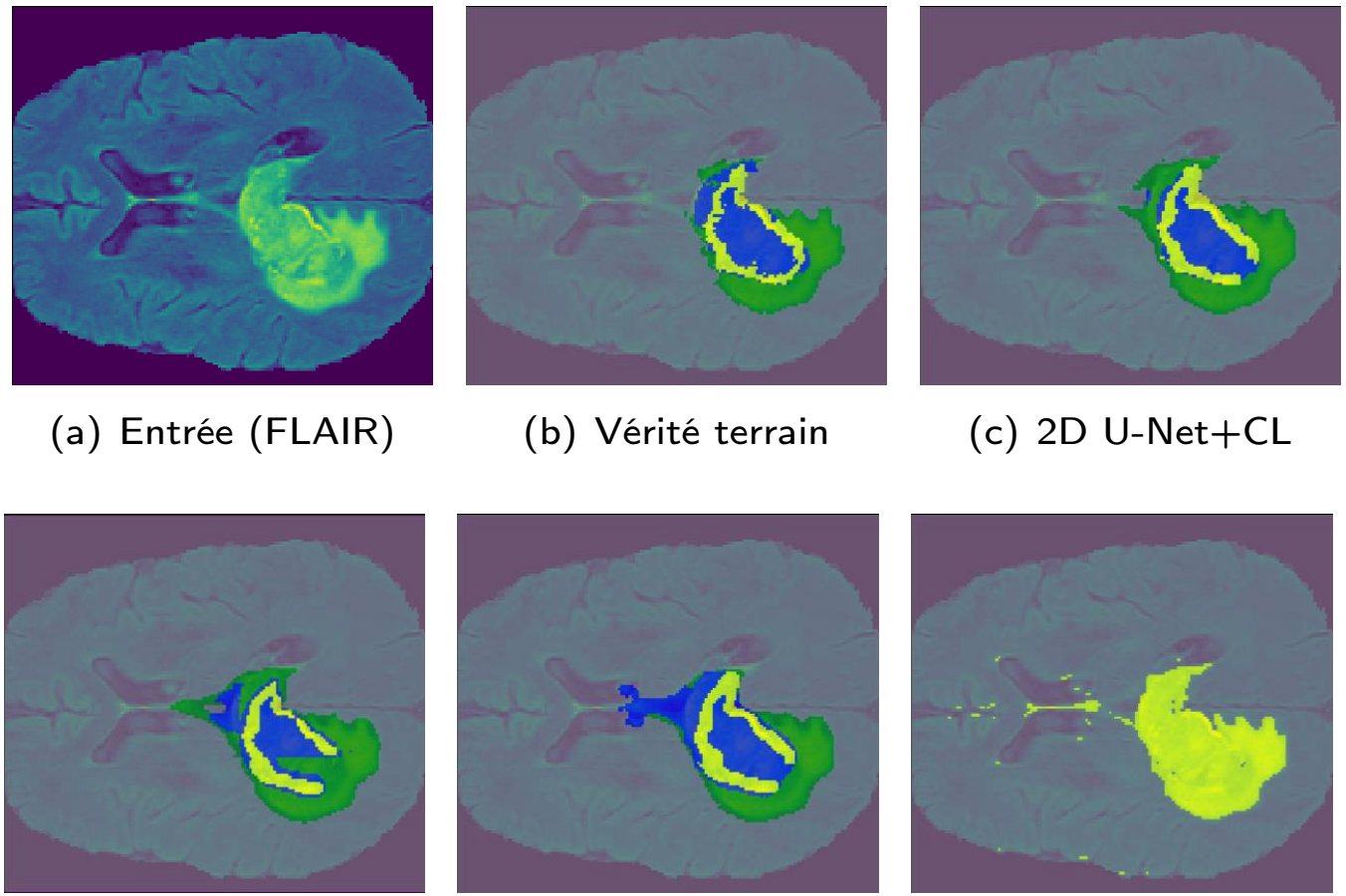


Figure : Mask-R-CNN avec un ResNet-101-FPN, 5 fps appliqué sur la base COCO

1

1. He, K., Gkioxari, G., Dollàr, P., & Girshick, R. (2017). Mask R-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 2961-2969).

Exemple 2 : Segmentation d'IRM



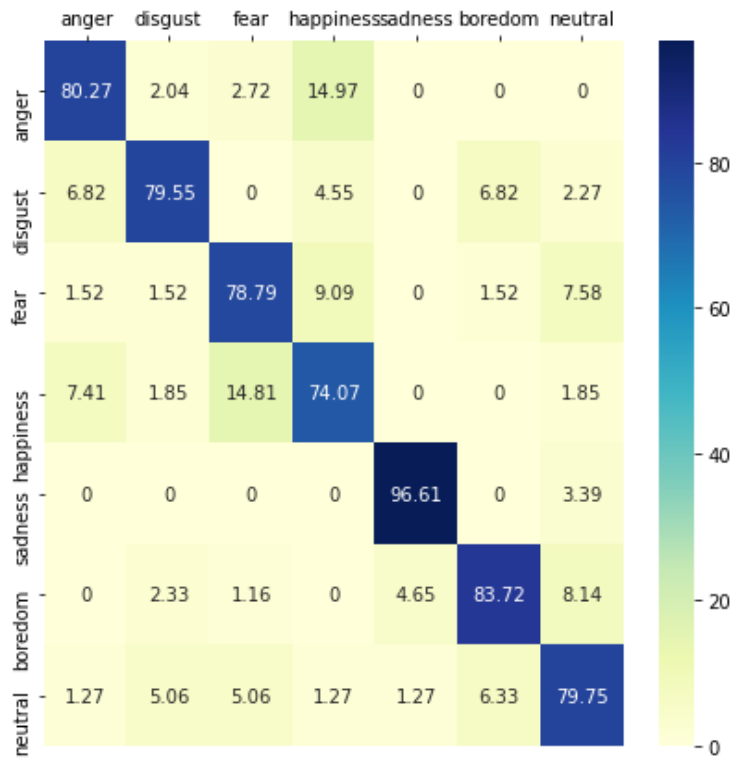
(a) Entrée (FLAIR) (b) Vérité terrain (c) 2D U-Net+CL

(d) 3D U-Net (e) Cascaded Network (f) K-Means

2. Ikram Brahim, Dominique Fourer, Vincent Vigneron and Hichem Maaref. Deep Learning Methods for MRI Brain Tumor Segmentation : a comparative study. Proc. IEEE IPTA 2019. Istanbul, Turkey

Exemple 3 : Reconnaissance des émotions dans la voix

(g) eNTERFACE05, STFT-Alex+RCS41 (Acc. 85.33%)



(h) EMO-DB, STFT-Alex+RCS19 (Acc. 81.82%)

3

3. Xia, S., Fourer, et al. (2022, May). Speech Emotion Recognition using Time-frequency Random Circular Shift and Deep Neural Networks. In Speech Prosody 2022.

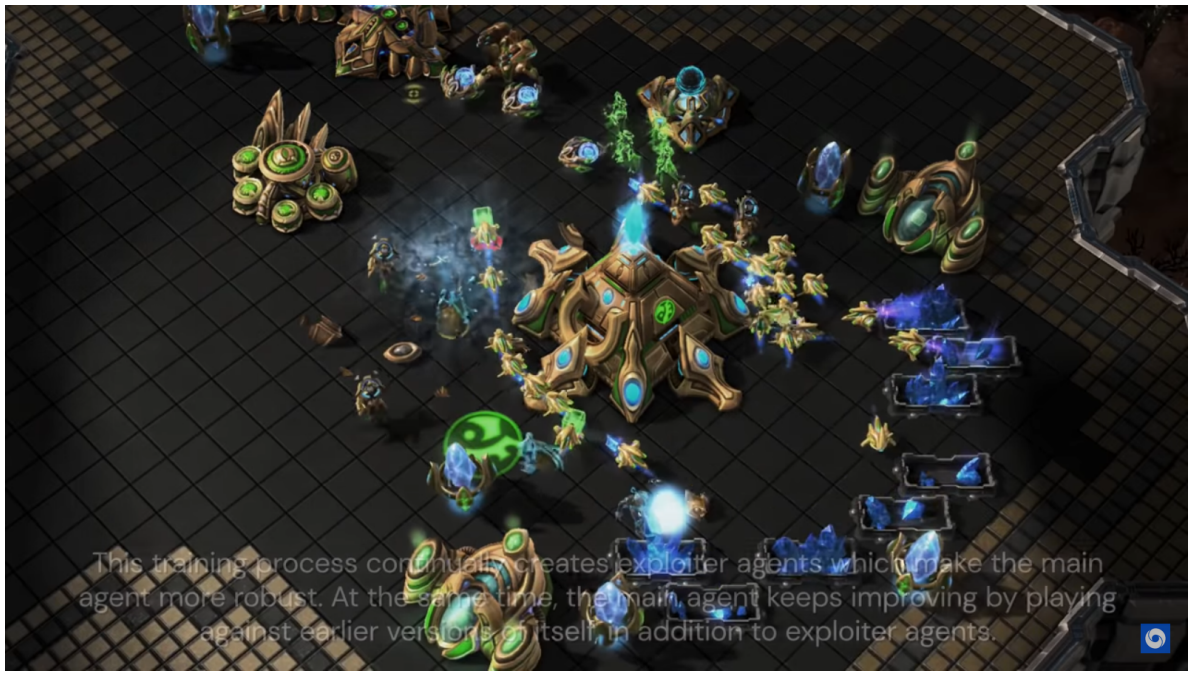
Exemple 4 : Transfert de style

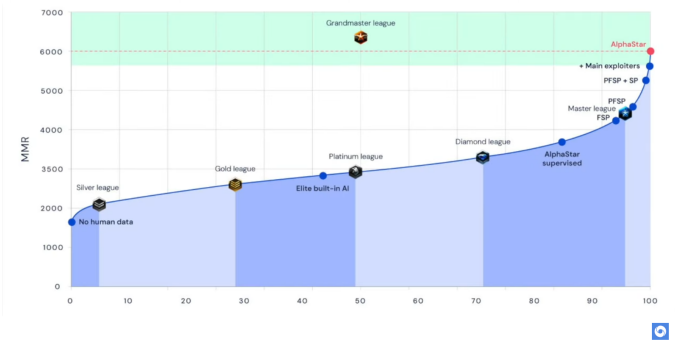
https://colab.research.google.com/github/tensorflow/hub/blob/master/examples/colab/tf2_arbitrary_image_stylization.ipynb

Exemple 5 : Apprentissage par renforcement (alphago)

https://www.deepmind.com

Exemple 6 : Apprentissage par renforcement (alphastar)





https://www.deepmind.com/open-source/alphastar-resources

Exemple 7 : Génération automatique de code (modèle de langage LLM)

Apparition des transformers en 2017 [4]

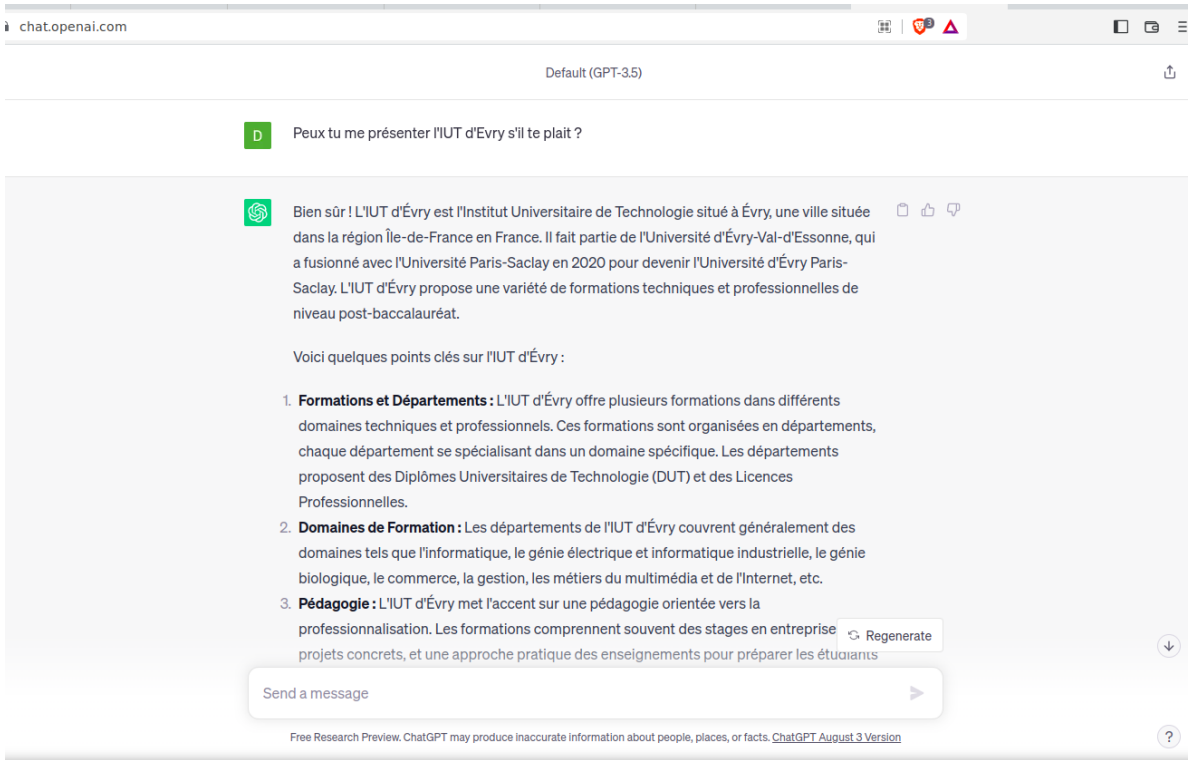
`https://alphacode.deepmind.com/`

4. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

ChatGPT (OpenAI, nov. 2022)...

GPT 1-3 (2018).

GPT : Generative Pre-trained Transformer [5]

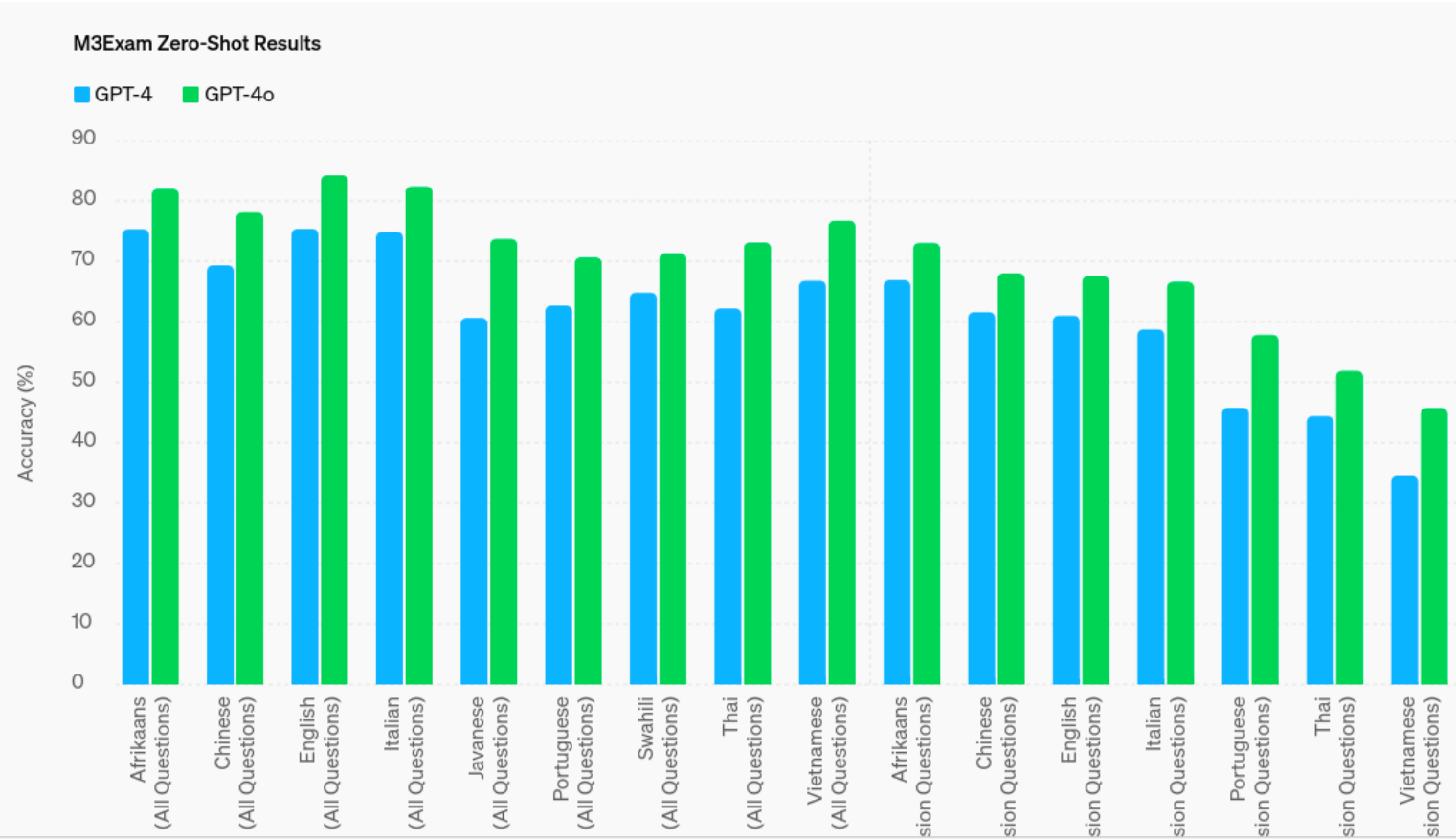


Concept

Agent conversationnel développé par OpenAI reposant sur les réseaux de neurones artificiels auto-attentifs, entraînés sur un volume massif de données (environ 175 milliards de paramètres pour GPT-3, plus de 500 milliards pour GPT-4).

5. https://chat.openai.com/

ChatGPT 4o (OpenAI, 2024)...



Version multimodale de Chat GPT (texte / audio / image)
`https://openai.com/index/hello-gpt-4o/`

Comparatif des principaux LLMs en 2025

Nom du modèle	Licence	Contexte	Modalité	# Paramètres	Accuracy
GPT-4	Propriétaire	32k	Texte	∼1T	Très élevée
GPT-4 Turbo	Propriétaire	128k	Texte	∼1T	Très élevée
GPT-4o	Propriétaire	128k	Multimodal	∼1T	Très élevée
Claude 3 Opus	Propriétaire	200k	Multimodal	∼200B	Très élevée
Claude 3 Sonnet	Propriétaire	200k	Multimodal	∼200B	Très élevée
Gemini 1.5 Pro	Propriétaire	1M	Multimodal	∼1T	Très élevée
Grok-3	Propriétaire	128k	Multimodal	314B	élevée
LLaMA 3 70B	Open (CC-BY-SA)	128k	Multimodal	70B	élevée
Mistral 7B	Open (Apache 2.0)	32k	Texte	7B	élevée
Mixtral 8x7B	Open (Apache 2.0)	32k	Texte	45B (MoE)	élevée
Falcon 180B	Open (TII)	4k	Texte	180B	élevée
Command R+	Semi-ouvert (CC-BY-NC)	256k	Texte	111B	élevée
Qwen 3	Open (Apache 2.0)	128k	Multimodal	235B	élevée
DeepSeek R1	Open (MIT)	128k	Texte	100B	élevée

`https://ollama.com/`

De nombreuses autres applications...

Actu IA : `https://www.actuia.com/`

Data IA : `https://www.dataia.eu/`

Paper with code : `https://paperswithcode.com/`

LLM : `https://ollama.com/`

axe IA (IBISC) : `https://ia.fourer.fr`

Apprentissage profond

Technologie de rupture en IA avec de nombreux champs d'application.

Principe

- Effectue multiple transformations non-linéaires ⇒ découvre les structures complexes
- Réseau de neurones de grande taille multi-couche ⇒ construit une représentation hiérarchique des caractéristiques sur plusieurs niveaux d'abstraction

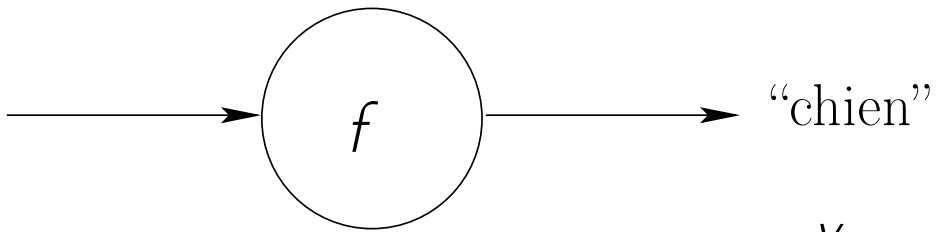
Quelques défis

- Interprétabilité des méthodes
- Validation des modèles (preuve)
- Contrôler la qualité des données et des modèles (systèmes critiques)
- Optimisation des architectures
- Sobriété énergétique (green IA, complexité algorithmique, ...)
- Apprentissage auto-supervisé,
- ...

Formulation du problème (classification)

Comment construire f qui fonctionne quelque soit x ?

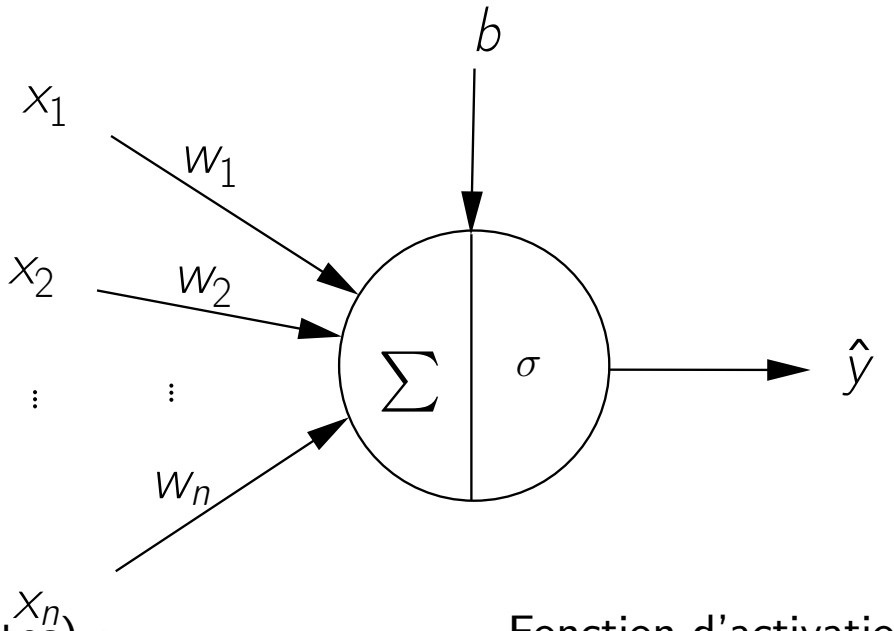
x



Difficultés

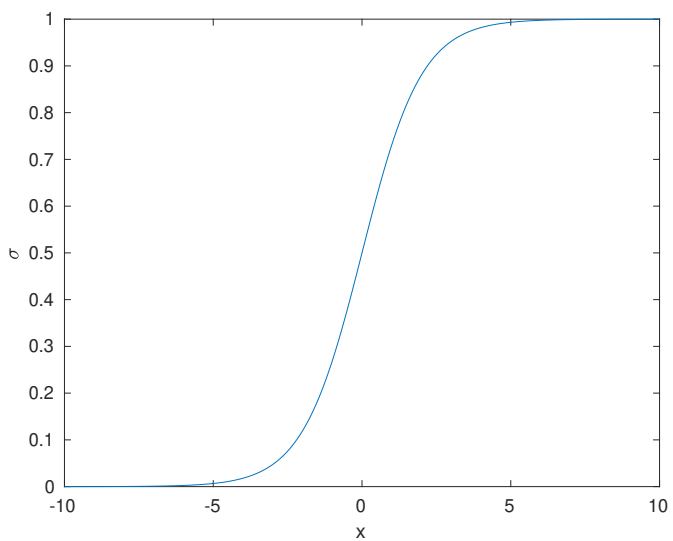
- $dim(x) \approx 10^6$ (fléau de la dimension [Bellman, 1961])
- x et y n'ont pas la même grandeur physique :
  x : image (tenseur), y : label (scalaire)
- f n'a pas été entraîné sur x et doit **généraliser** le problème

Neurone formel



- entrées (dendrites) : $\mathbf{x} = (x_1, x_2, ..., x_n)$
- poids : $w_1, w_2, ..., w_n$
- biais : $b = x_0 w_0$ avec $x_0 = 1$
- fonction d'agrégation : Σ
- fonction d'activation : $\sigma(x) = \frac{1}{1+e^{-x}}$
- sortie (axone) : $\hat{y} = \sigma\left(\sum_{i=1}^{n} x_i w_i + b\right)$

Fonction d'activation sigmoïde.

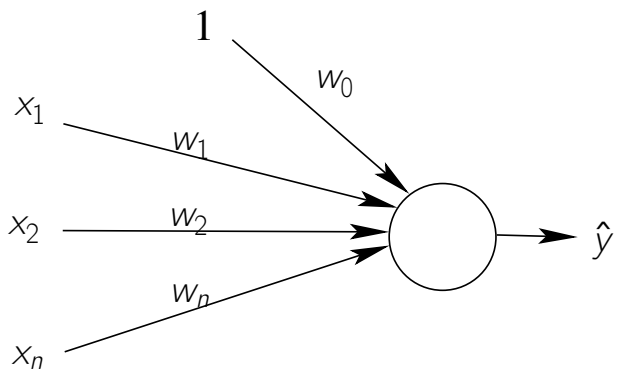


Perceptron [Rosenblatt, 1957] - gen1

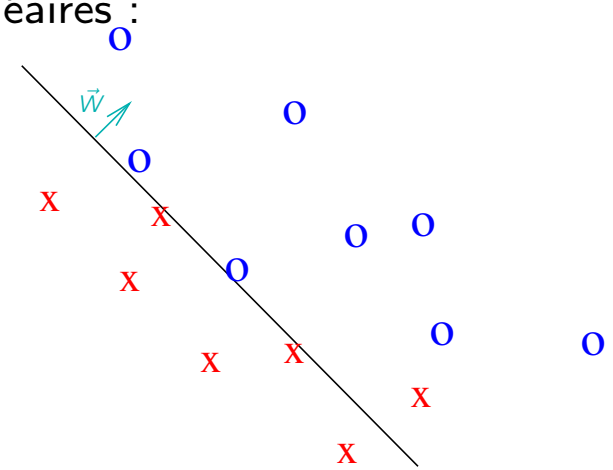
Sortie pour un seul neurone (n entrées) :

$$\hat{y} = \sigma\left(W\mathbf{x} + b\right) = \sigma\left((w_0, w_1, w_2, \cdots, w_n)(1, x_1, x_2, \cdots, x_n)^T\right) \quad (1)$$

Fonction d'activation (Heaviside) : $\sigma(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases}$



$\Rightarrow$ limité aux problèmes linéaires :

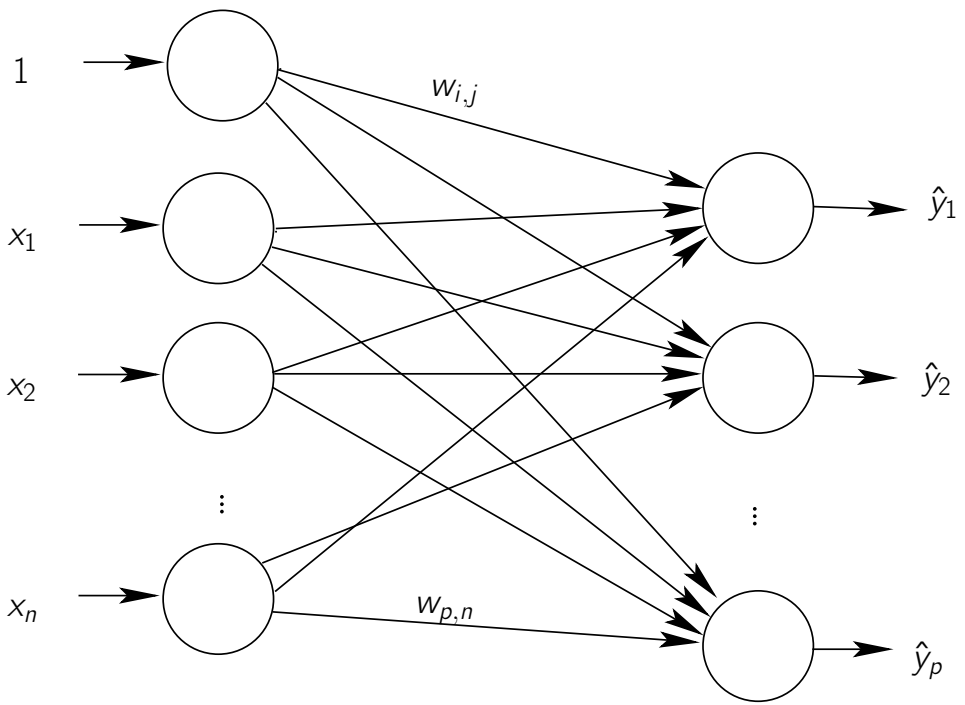


Cas d'une seule couche de p neurones

Écriture matricielle pour une couche de p neurones :

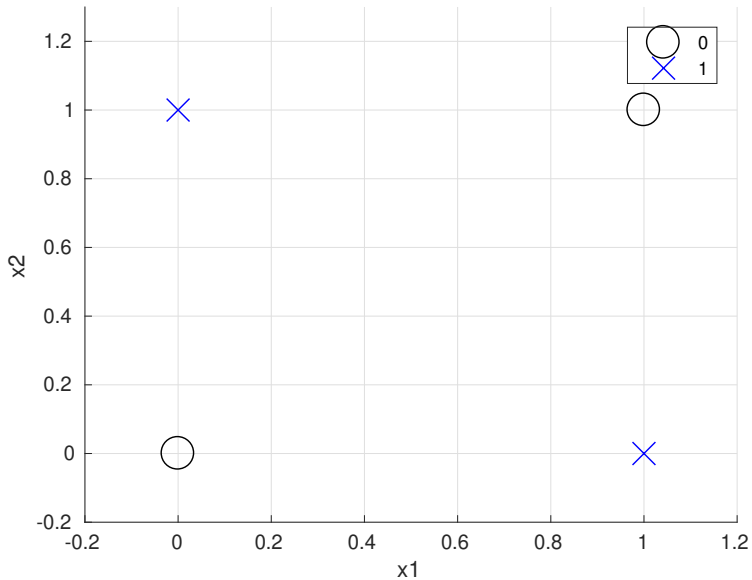
$$\begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_p \end{pmatrix} = \begin{pmatrix} \sigma(\tilde{y}_1) \\ \sigma(\tilde{y}_2) \\ \vdots \\ \sigma(\tilde{y}_p) \end{pmatrix} = \sigma \left(\begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ w_{p,1} & w_{p,2} & \cdots & w_{p,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{pmatrix} \right) \quad (2)$$

- Une couche d'entrée (n valeurs)
- Une couche de sortie (p valeurs)
- Pas de connexion entre les neurones d'une même couche



Problème avec la fonction XOR

xor	0	1
0	0	1
1	1	0



Impossible à traiter avec une seule couche de perceptrons [6].

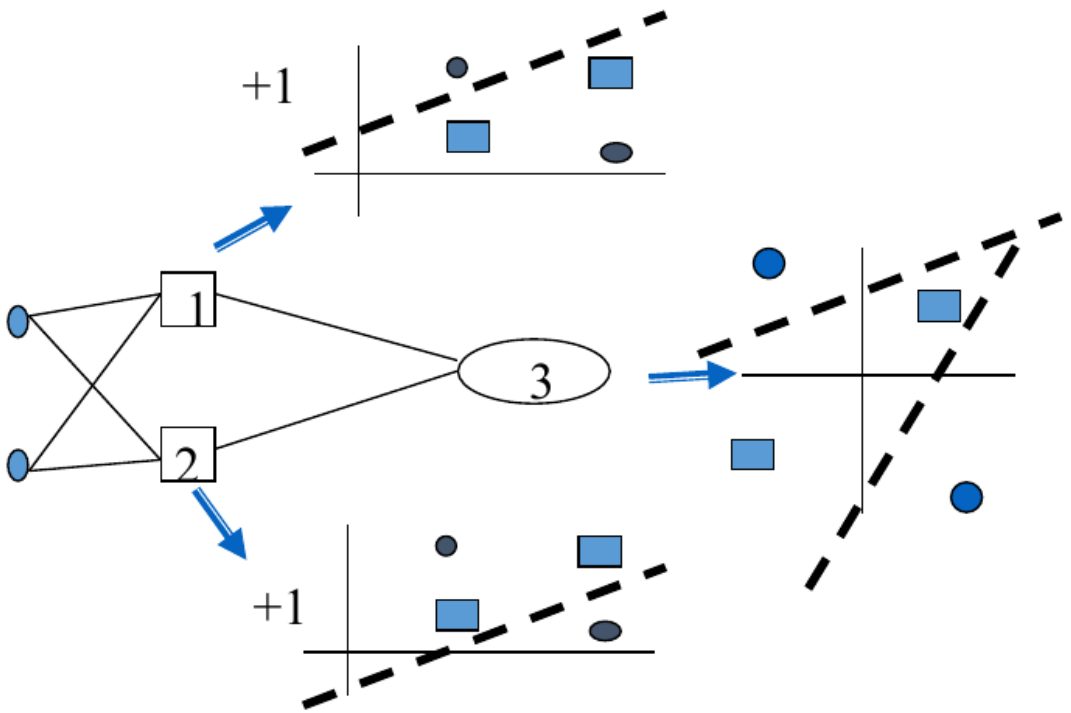
6. dans le cas d'une fonction d'activation classique

Problème avec la fonction XOR

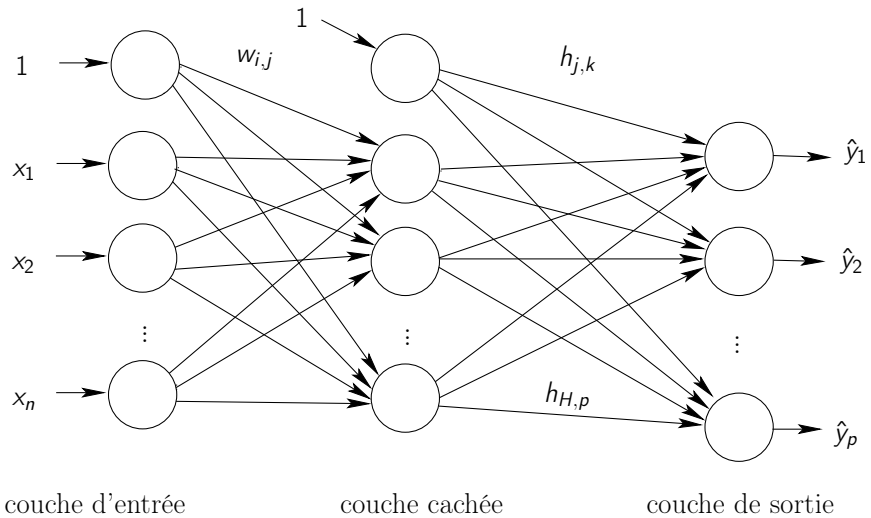
Solution : Combiner au moins 2 perceptrons avec une couche supplémentaire

et	0	1
0	0	0
1	0	1

ou	0	1
0	0	1
1	1	1



Perceptron multicouche (MLP) (années 80-90) - gen2



$$\hat{y}_k = h_{0,k} + \sum_{j=1}^{H} h_{j,k}\sigma\left(w_{0,j} + \sum_{i=1}^{n} w_{i,j}x_i\right) \quad ,\forall k \in \{1, 2, \cdots, p\} \qquad (3)$$

Propriétés

- Plusieurs couches de neurones dont **1** couche cachée
- Pas de connexion entre les neurones d'une même couche
- Couche d'entrée $\in \mathbb{R}^n$, couche cachée $\in \mathbb{R}^{H\times n}$ et couche de sortie $\in \mathbb{R}^p$

Limitations

- Difficultés pour entraîner les réseaux de grande taille (surapprentissage, minimum local, temps de calcul)
- Difficultés d'interprétation des modèles (effet boîte noire)
- Surapprentissage
- Performances inférieures à d'autres approches (SVM, random forest, boosting, etc.)
- Couche de sortie :
  - regression : un neurone par valeur prédite
  - classification : une neurone par classe

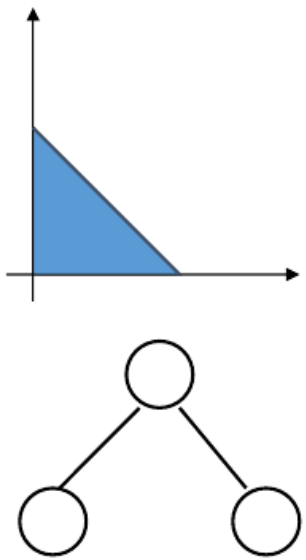
Exemple d'implémentation python

```python
import numpy as np

def sigma(x):
    return 1/(1 + np.exp(-x));

class FCLayer:
    # input_size = number of input neurons
    # output_size = number of output neurons
    def __init__(self, input_size, output_size):
        self.weights = np.random.rand(input_size, output_size) - 0.5
        self.bias = np.random.rand(1, output_size) - 0.5

    # returns output for a given input
    def forward_propagation(self, input_data):
        self.input = input_data
        self.output = sigma(np.dot(self.input, self.weights) + self.bias)
        return self.output
```

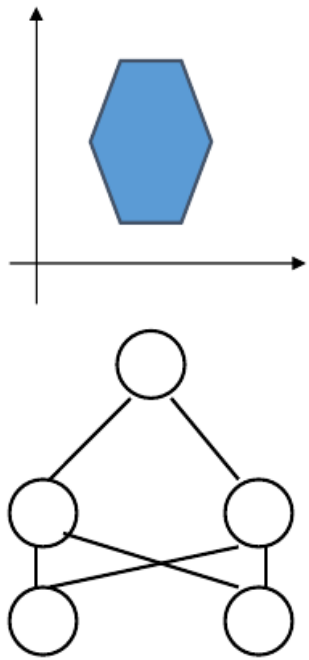
7

7. Fonction d'activation sigmoïde : $\sigma(x) = \frac{1}{1+e^{-x}}$

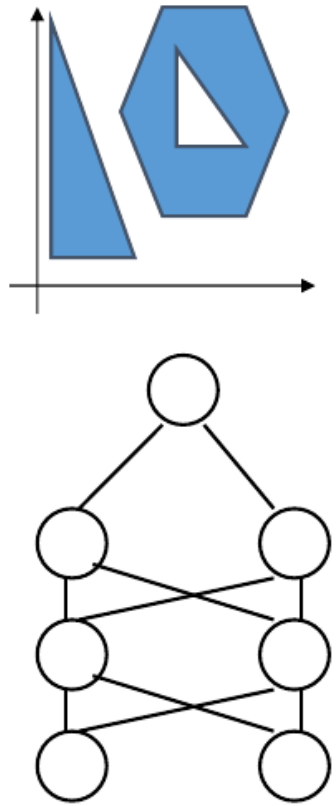
Réseau de neurones profond (années 2010) - gen3



1 couche



2 couches



3 couches...

Propriétés

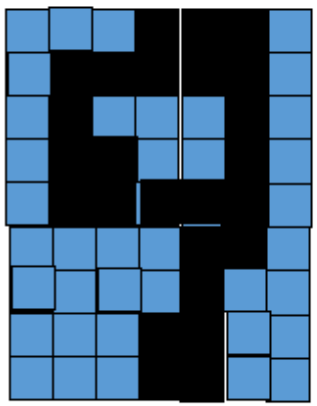
- Plusieurs couches cachées (nombre arbitraire)
- Fonction d'activation σ arbitraire (non linéaire et non polynomiale)

Pourquoi seulement maintenant ?

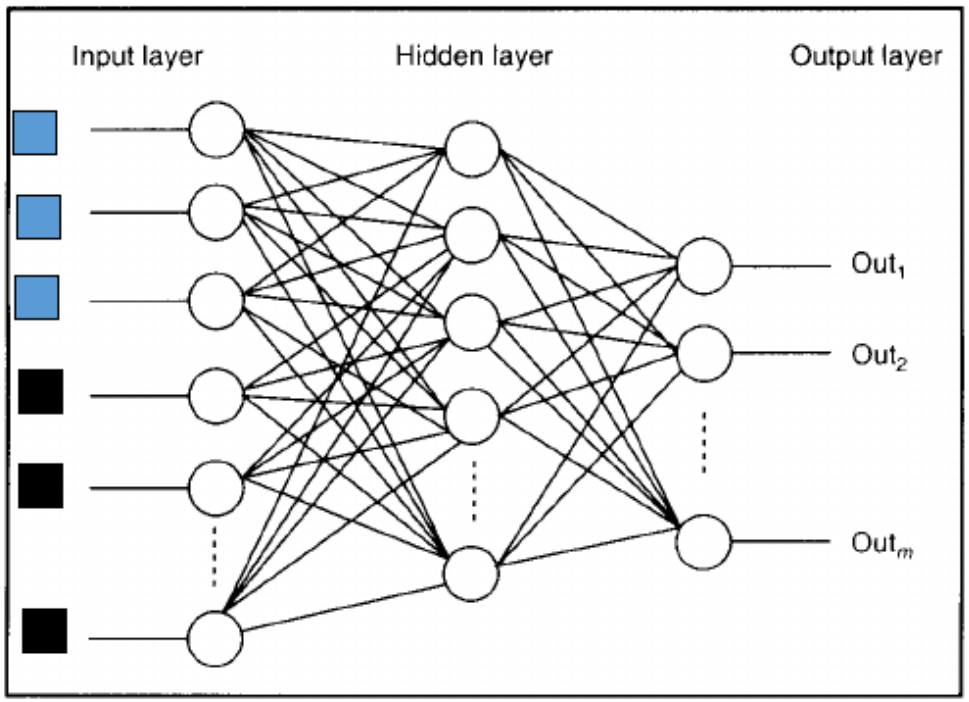
- Puissance de calcul (GPU, calcul parallèle, etc.)
- Données massives
- Algorithme de rétro-propagation (back-propagation) pour l'entraînement [Rumelhart, 86]
- Succès en traitement d'image (Lecun Net, 98)
- Imagenet (NIP'12) (rupture Technologique)

Qu'apprend un réseau de neurones ? 1/10

Figure 1.2: *Examples of handwritten digits from postal envelopes.*



Construction de nouvelles représentations



Problème

Reconnaissance de caractères manuscrits

Qu'apprend un réseau de neurones ? 2/10

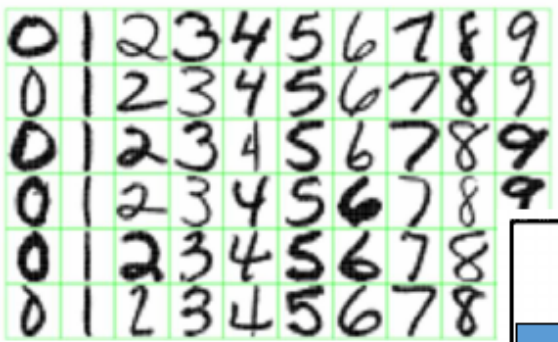
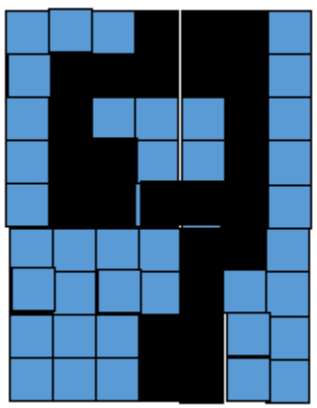
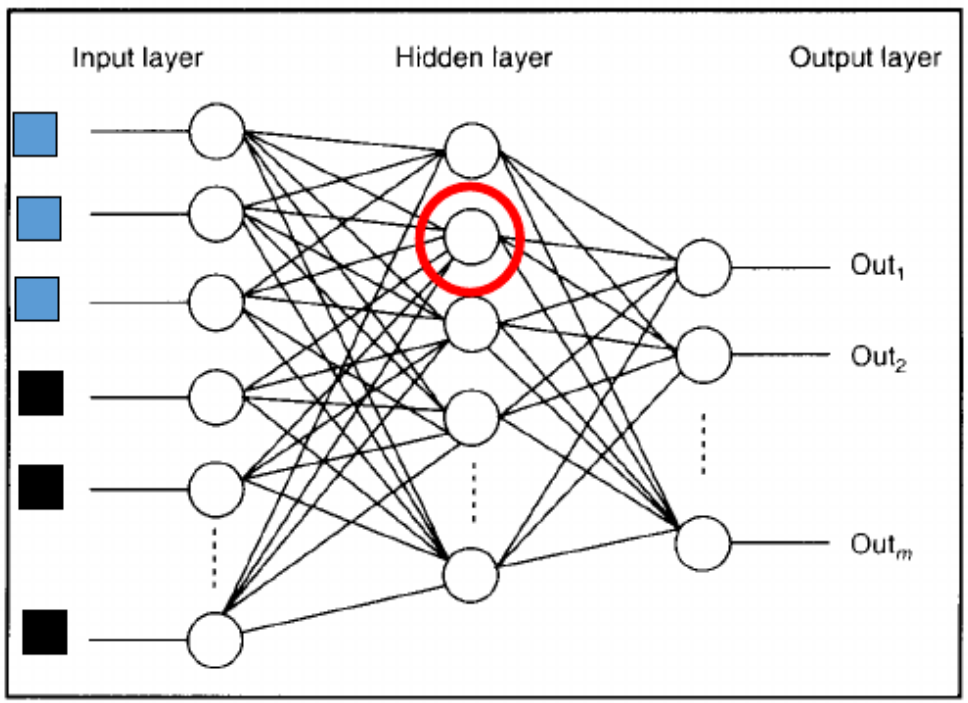


Figure 1.2: *Examples of handwritten digits from postal envelopes.*



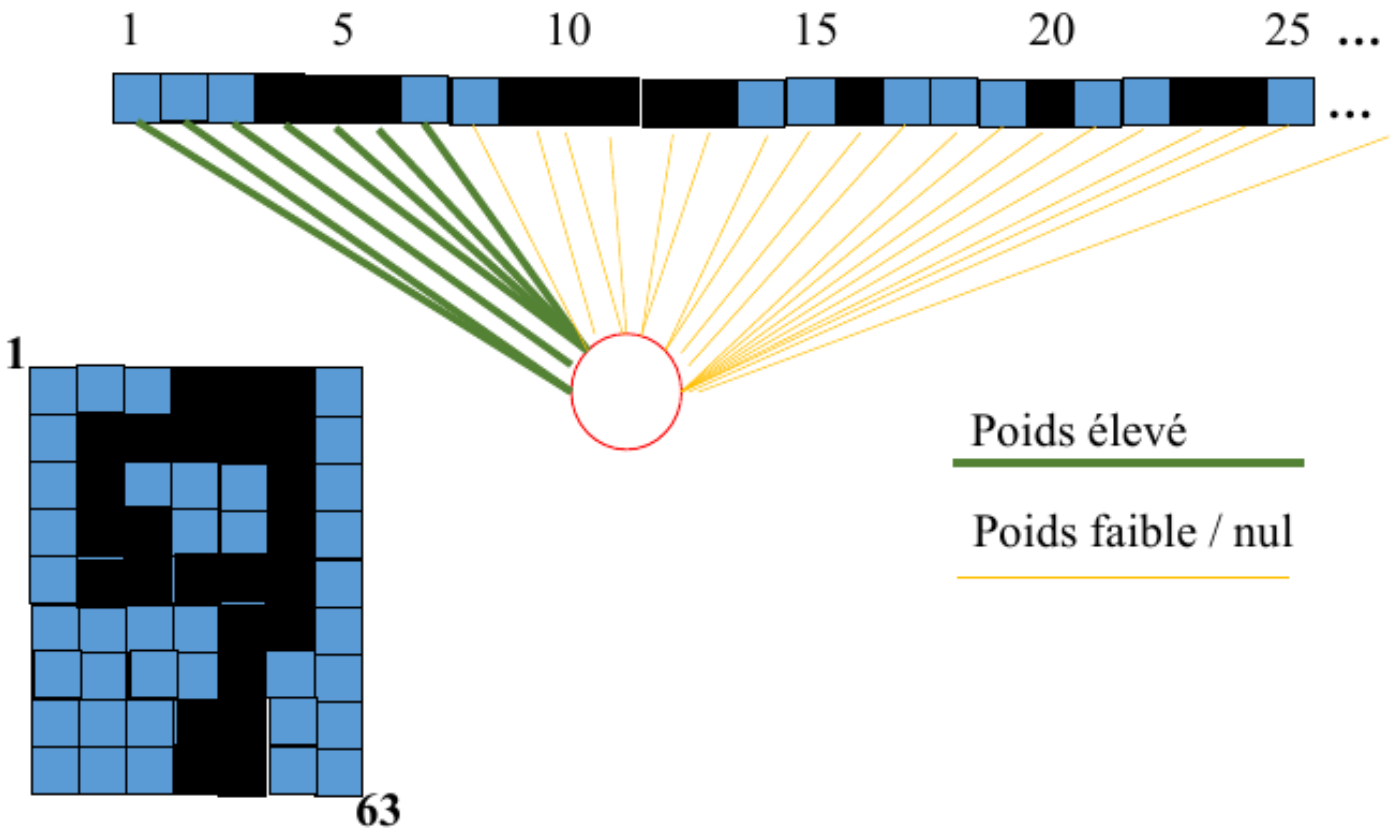
Que fait ce neurone?



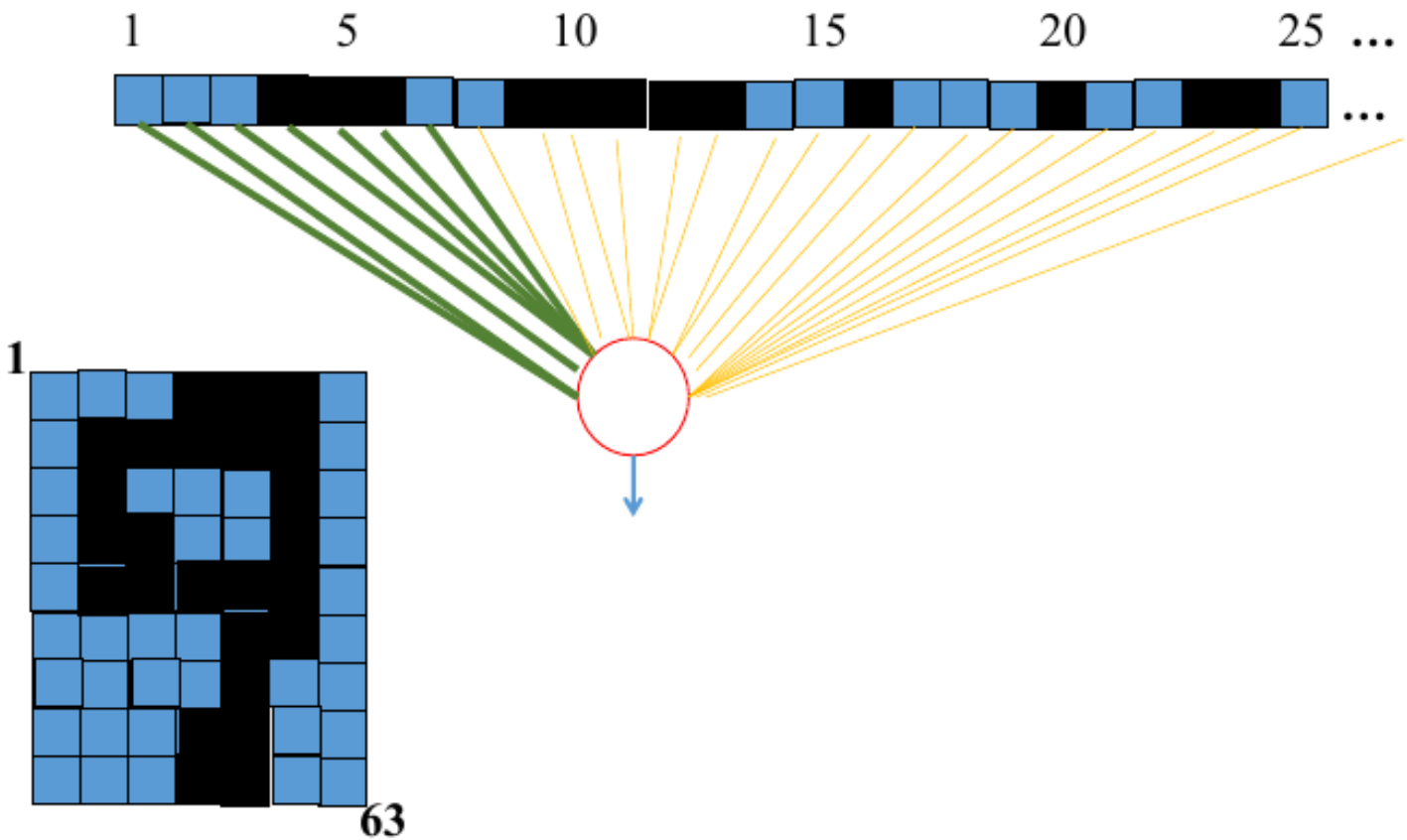
Qu'apprend un réseau de neurones ? 3/10

Question :

Que fait ce neurone ?



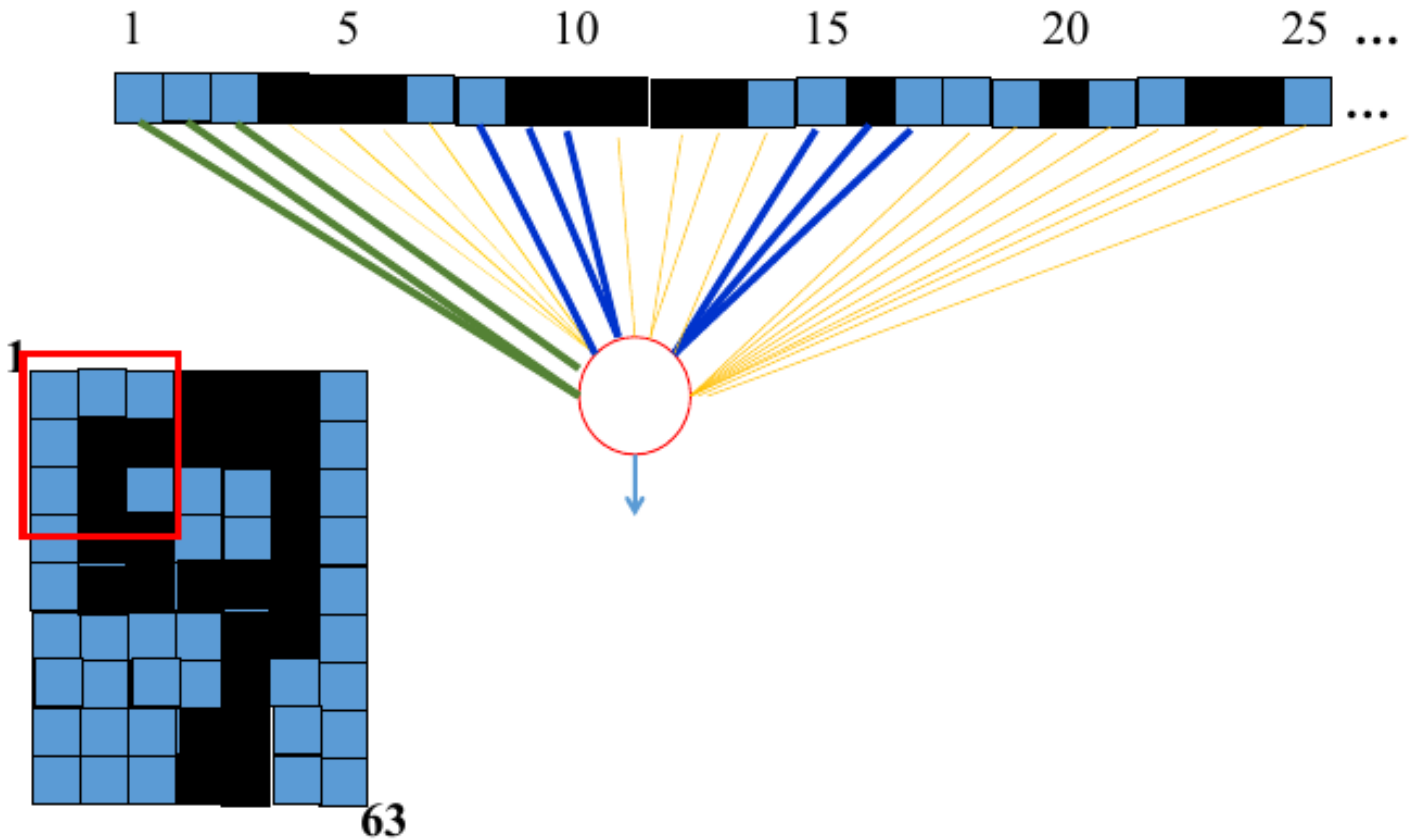
Qu'apprend un réseau de neurones ? 4/10



Sortie du neurone

Considère une ligne horizontale sur la première ligne, ignore le reste de l'image

Qu'apprend un réseau de neurones ? 5/10



Sortie du neurone

Considère la zone supérieure gauche de l'image et ignore le reste

Qu'apprend un réseau de neurones ? 6/10

Question

Que devrait détecter un bon réseau de neurones

Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

Qu'apprend un réseau de neurones ? 7/10

1

Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

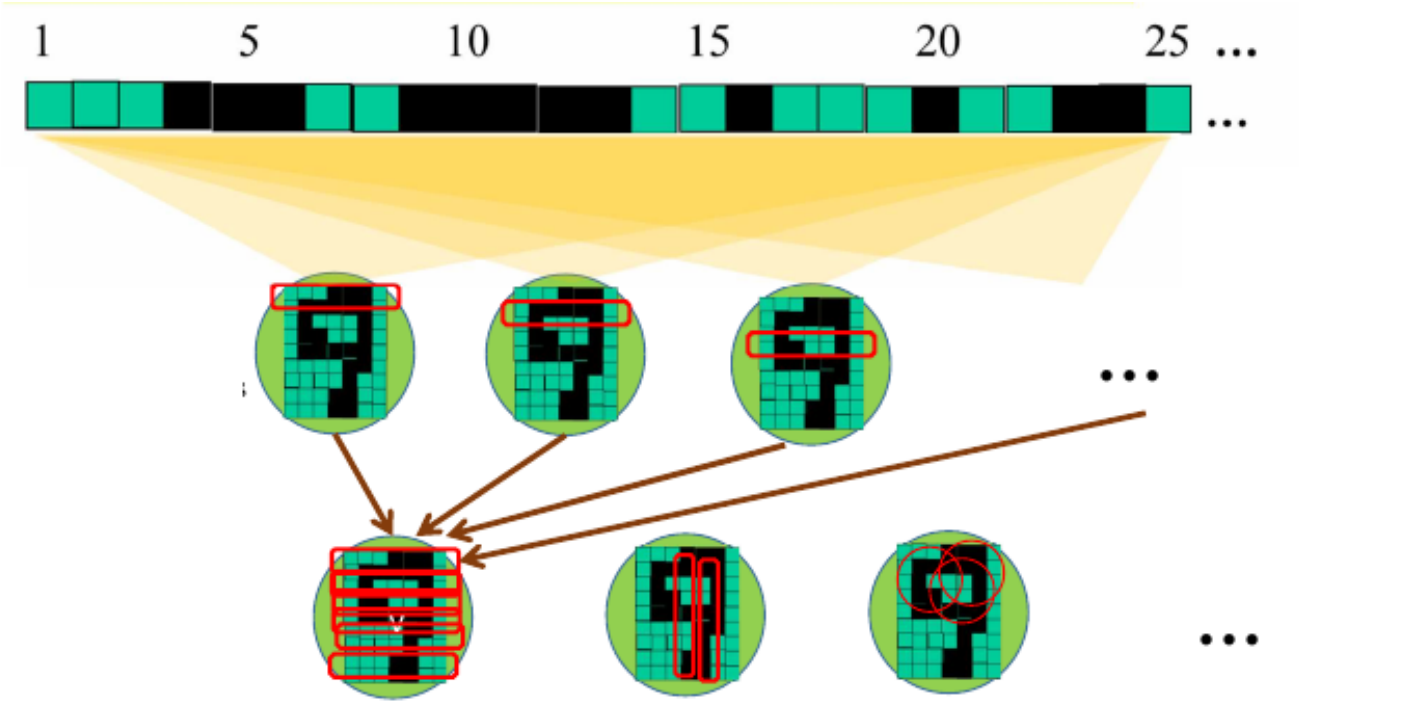
Qu'apprend un réseau de neurones ? 8/10

Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

Qu'apprend un réseau de neurones ? 9/10

Problème de l'invariance de position ?
Les neurones sont liés à des endroits spécifiques de l'image

Qu'apprend un réseau de neurones ? 10/10



Représentation hierarchique :

- Les premières couches détectent des motifs à des endroits spécifiques
- les couches suivantes recherchent des motifs plus abstraits (lignes, cercles, etc.)

Fonction d'activation

Objectif

- Conditionner la sortie d'un neurone pour éviter la dispersion des valeurs (eg. négliger les réponses non significatives et atténuer les valeurs aberrantes).
- Introduit de la non-linéarité dans le modèle et permet d'apprendre des formes structures complexes
- Permet de se placer dans les conditions où le théorème d'approximation universelle est vérifié (activation non polynomiale)

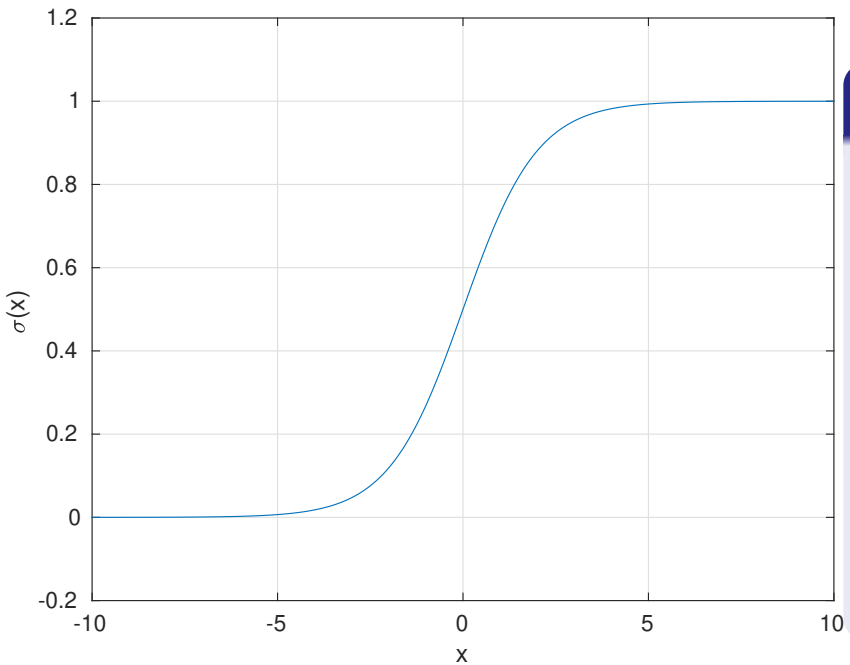
Quelques exemples :

- Relu : $\text{Relu}(x) = \max(0, x)$
- Tangente hyperbolique : $\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$
- sigmoïde : $\sigma(x) = \frac{1}{1+e^{-x}}$
- softmax : $\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$
- etc.

Fonction sigmoïde

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x)) \tag{5}$$



Propriétés (problèmes)

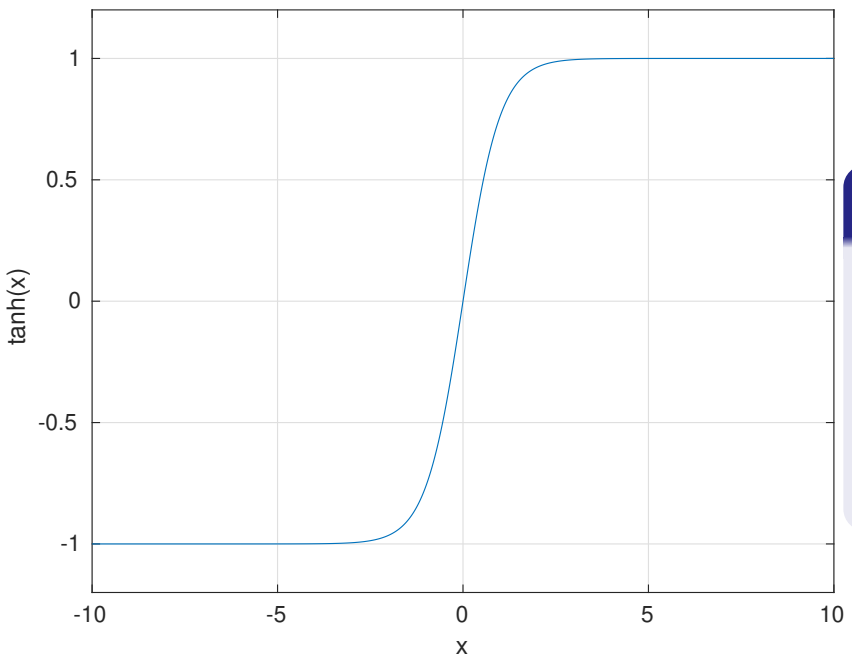
- Compresse les valeurs dans $[0; 1]$
- Non nulle pour $x = 0$
- Saturation des neurones (annule le gradient)
- Complexité de la fonction exponentielle

Fonction tangente hyperbolique

$$\tanh(x) = \frac{\text{sh}(x)}{\text{ch}(x)} = 2\sigma(2x) - 1 = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{6}$$

avec $\sigma(x)$ la fonction sigmoïde.

$$\frac{d}{dx}\tanh(x) = 1 - \tanh(x)^2 \tag{7}$$

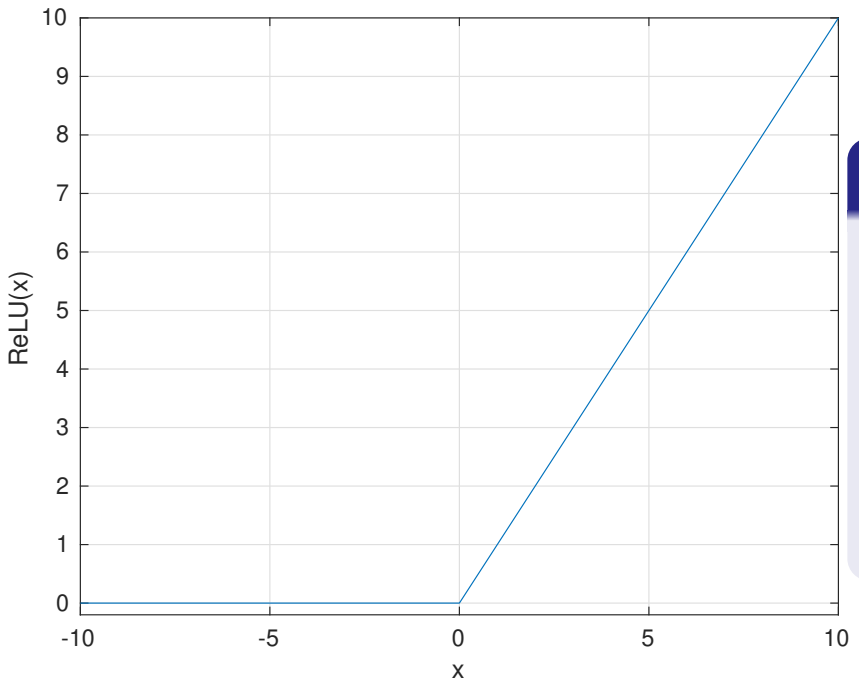


Propriétés (problèmes)

- Compresse les valeurs dans $[-1; 1]$
- Saturation des neurones (annule le gradient)

Fonction d'unité linéaire rectifiée

$$\text{ReLU}(x) = \max(0, x) \tag{8}$$



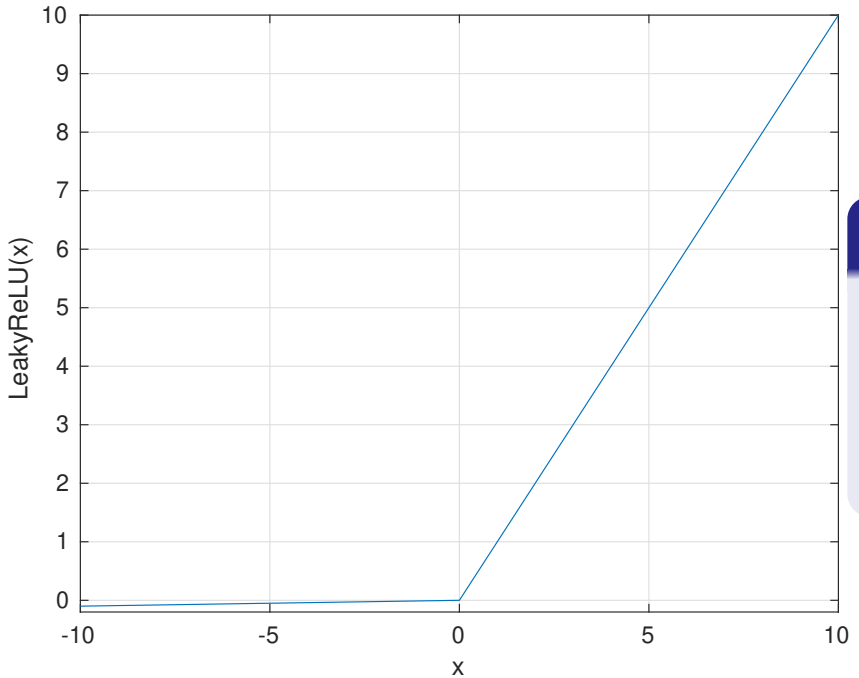
Propriétés (problèmes)

- Pas de saturation pour $x \geq 0$
- Calcul rapide
- Gradient nul dans la partie négative (saturation pour $x < 0$)

Fonction d'unité linéaire rectifiée fuyante

$$\text{LeakyReLU}(x) = \max(\alpha x, x) \tag{9}$$

avec $\alpha = 10^{-2}$ dans la version initiale LeakyReLU ou n'importe quelle valeur (PReLU)



Propriétés (problèmes)

- Pas de saturation
- Calcul rapide
- Gradient toujours non nul

Entraînement

But

L'entraînement du réseau consiste à trouver les paramètres de chaque neurone qui minimisent une fonction de coût J :

$$(w^*, b^*) = \arg\min_{w,b} J(w, b) \tag{10}$$

Algorithme de descente de gradient

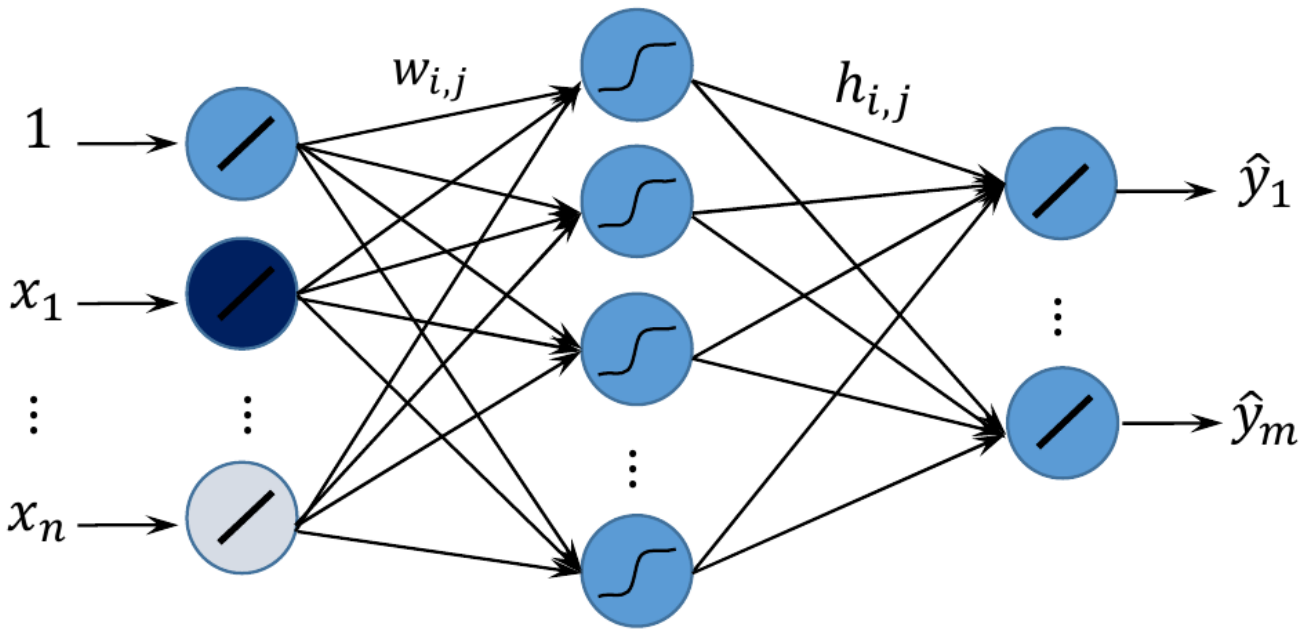
Suppose que J est convexe et différentiable.

1. Initialiser aléatoirement w et b et fixer μ le pas d'apprentissage
2. Répéter :
   - Calculer $\nabla J = \left(\frac{\partial J}{\partial w}, \frac{\partial J}{\partial b}\right)$
   - Mettre à jour $w \leftarrow w - \mu \frac{\partial J}{\partial w}$ et $b \leftarrow b - \mu \frac{\partial J}{\partial b}$

   Tant que $||\nabla J|| > \epsilon$ (ou s'arrêter si $J(w, b)$ ne diminue pas)

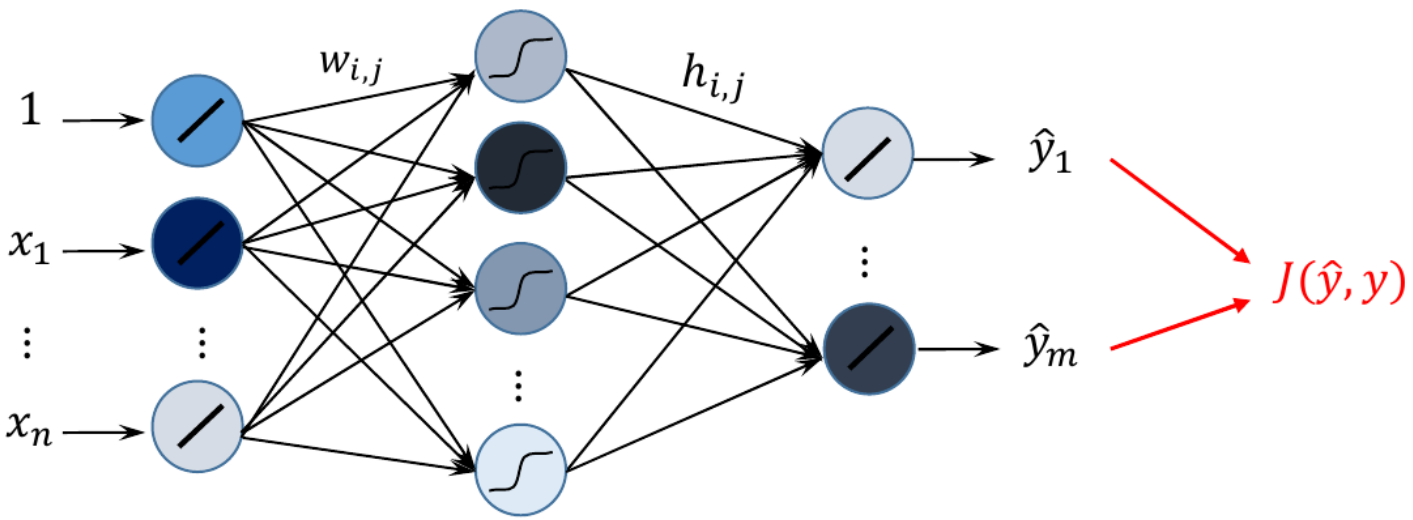
Algorithme de rétropropagation (backpropagation) du gradient [Werbos, Parker, Lecun, Hinton, Williams, 75-86]

Propagation d'un exemple (x, y) avec $x \in \mathbb{R}^n$ et $y \in \mathbb{R}^m$



Algorithme de rétropropagation du gradient [Werbos, Parker, Lecun, Hinton, Williams, 75-86]

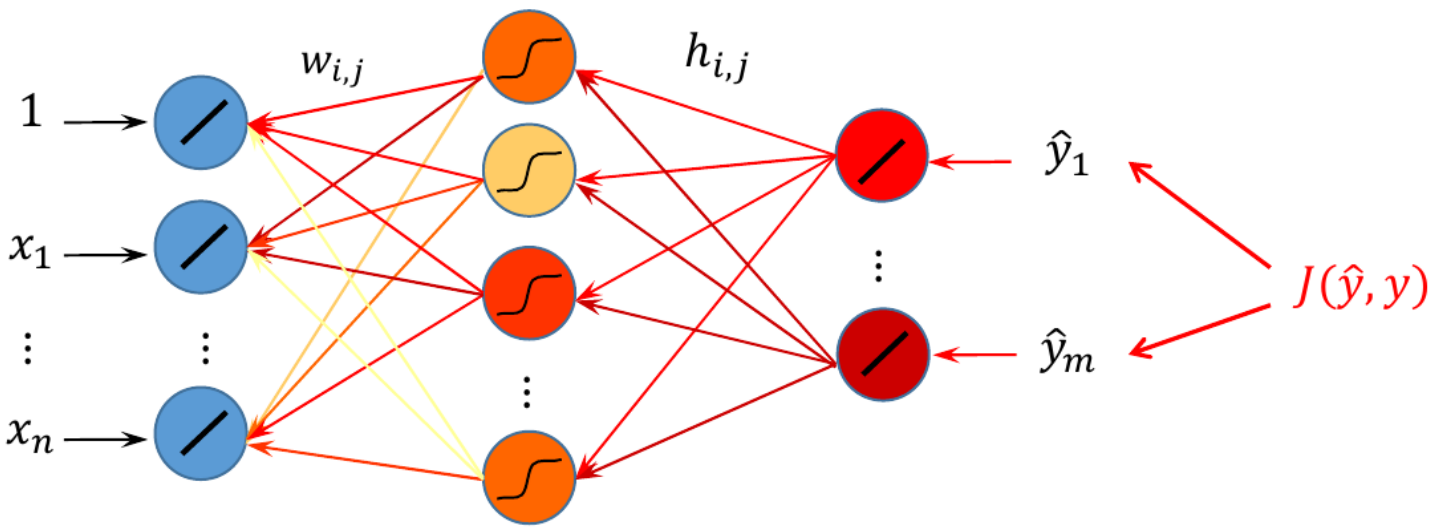
Estimation de l'erreur de prédiction $J(\hat{y}, y)$



Algorithme de rétropropagation du gradient [Werbos, Parker, Lecun, Hinton, Williams, 75-86]

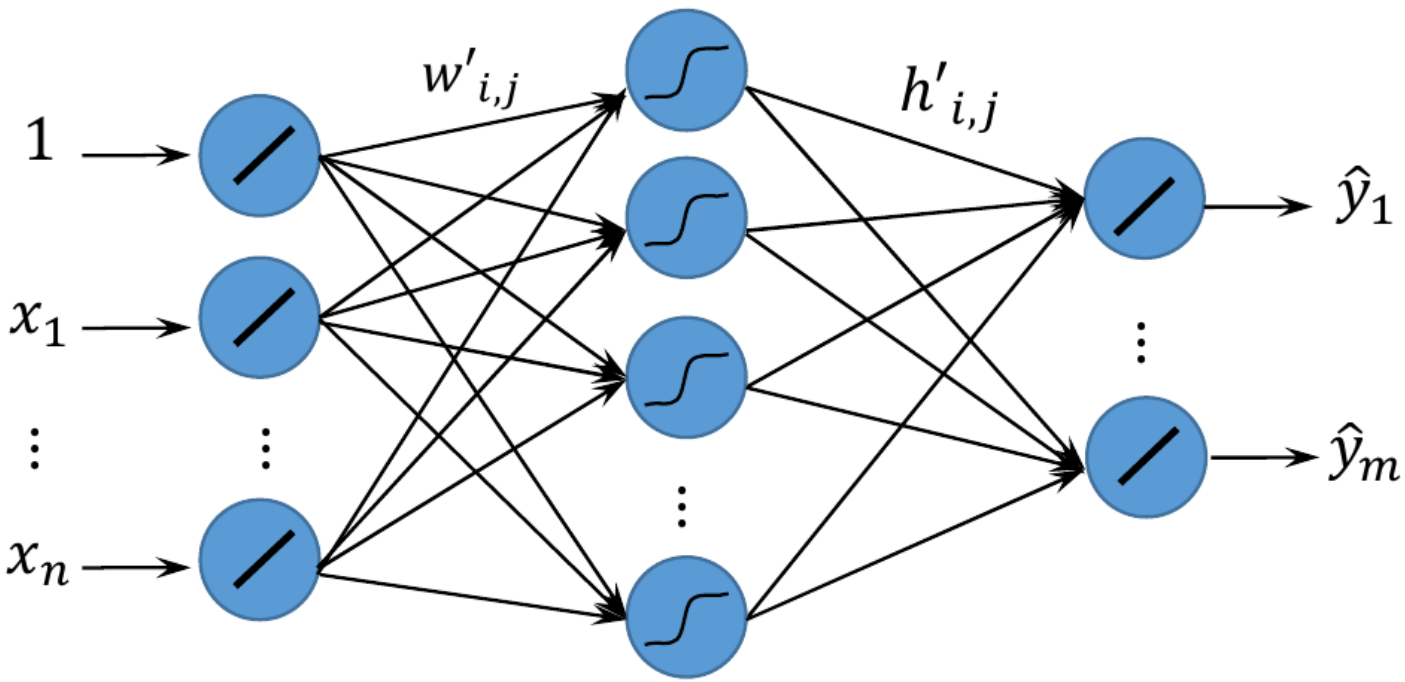
Calcul des gradients et mise à jour des poids :

$$w'_{i,j} = w_{i,j} - \mu \frac{\partial J}{\partial w_{i,j}} \qquad h'_{i,j} = h_{i,j} - \mu \frac{\partial J}{\partial h_{i,j}} \tag{11}$$



Algorithme de rétropropagation du gradient [Werbos, Parker, Lecun, Hinton, Williams, 75-86]

Propagation avant d'un autre exemple puis rétropropagation jusqu'à convergence



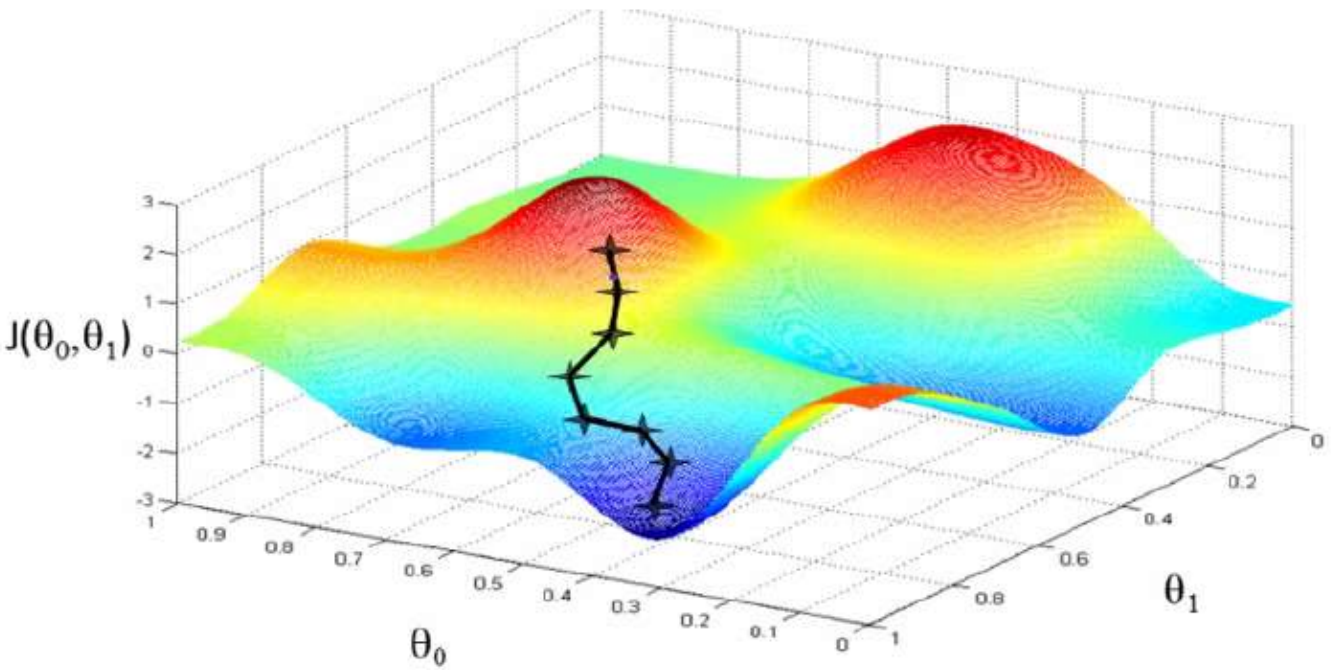
La descente de gradient

Algorithme :

- Initialiser les poids W
- Répéter :

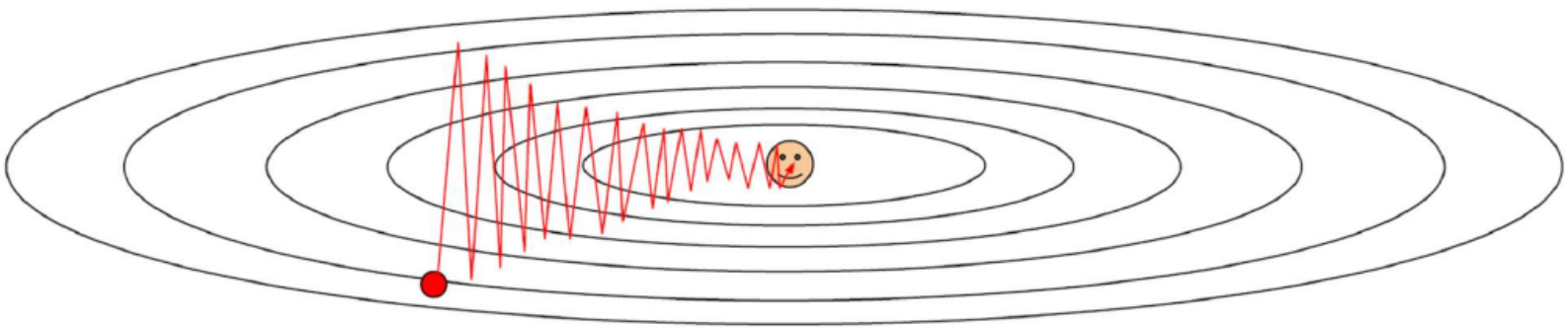
$$W' \leftarrow W - \mu \frac{\partial J}{\partial W} \qquad (12)$$

tant que J diminue.



Problème d'optimisation 1

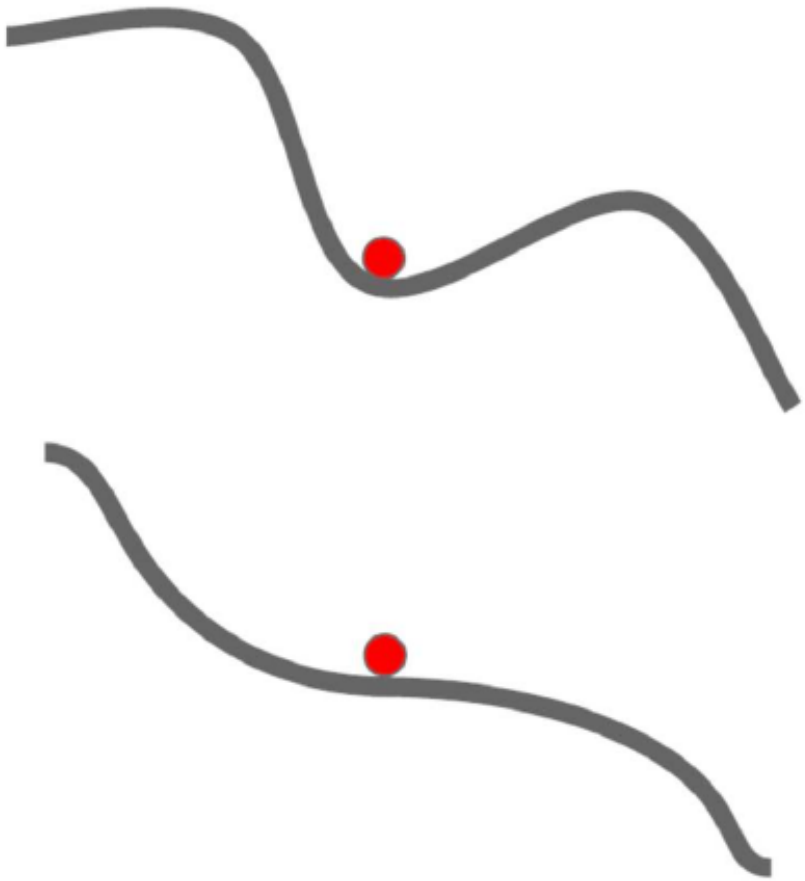
Le gradient converge lentement quand la fonction de coût change rapidement dans une dimension et lentement dans une autre



Problème d'optimisation 2

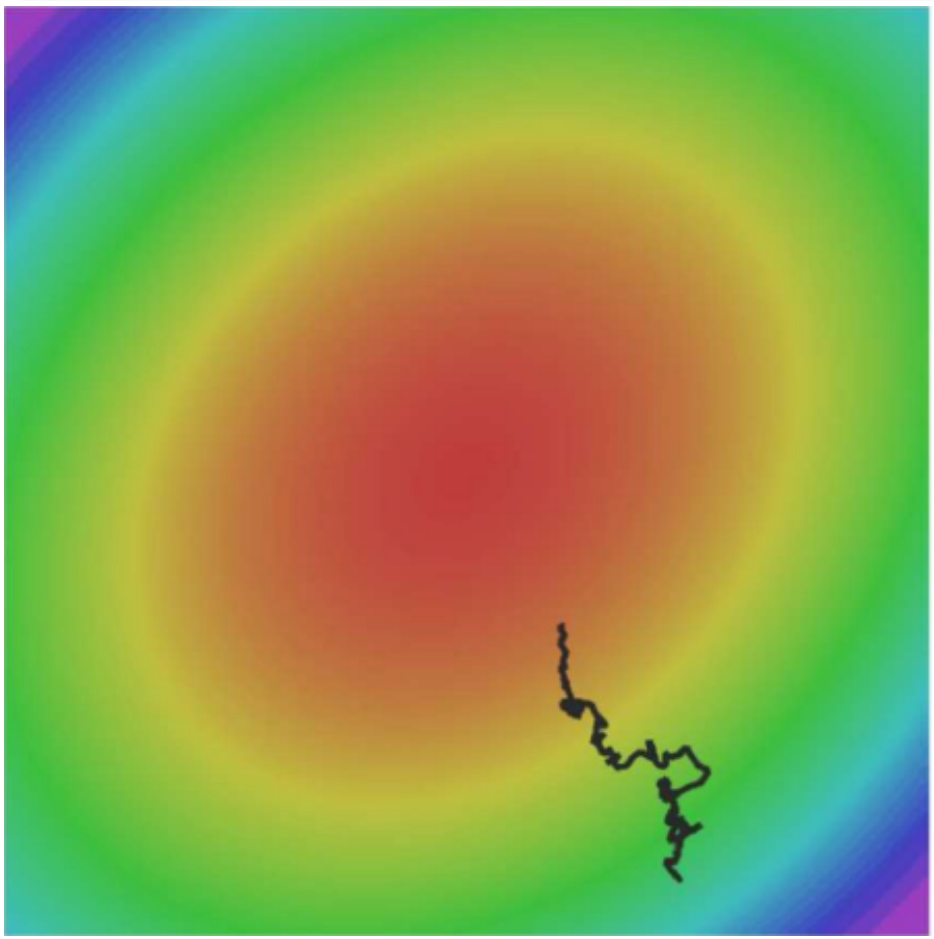
L'espace des paramètres est très grand

- peu de problèmes de minimum local
- nombreux points-selles *(saddle point)*



Problème d'optimisation 3

Problème de bruit sur les mini-lots



Momentum

Solution

Ajouter de l'inertie à la descente de gradient

Algorithme :

- Initialiser les poids W
- $V \leftarrow 0$
- Répéter :

$$V \leftarrow \rho V - \frac{\partial J}{\partial W} \tag{13}$$

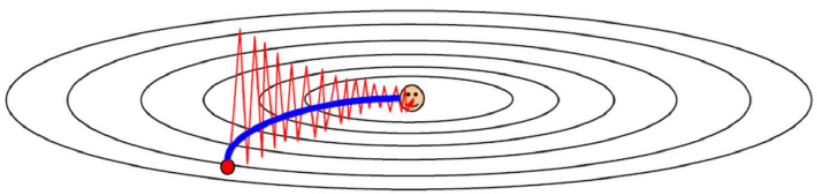
$$W \leftarrow W - \mu V \tag{14}$$

tant que J diminue.

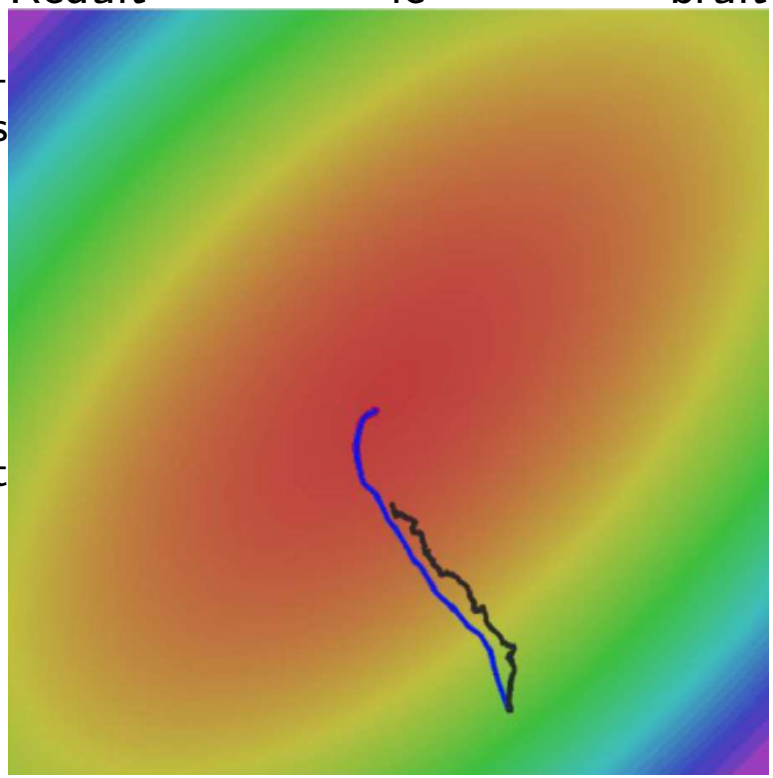
Momentum

Aide à sortir des minima locaux et des points-selles

Accélère la descente de gradient



Réduit le bruit



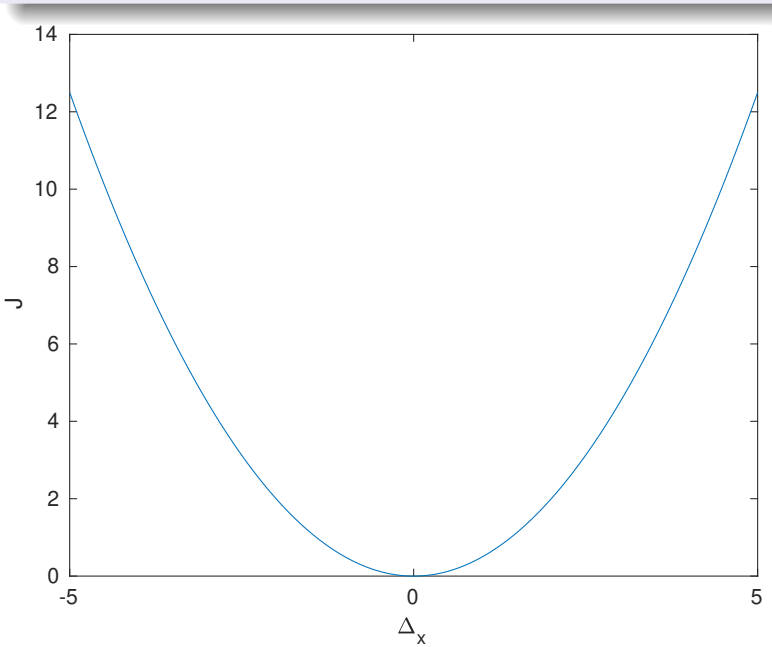
Fonction de coût

- Cette fonction permet de définir un objectif d'optimalité à atteindre avec les données d'entrée.
- Idéalement cette fonction est différentiable (donc continue) et convexe.

L'erreur quadratique moyenne

Utilisée pour les problèmes de régression

$$J(\hat{y}, y) = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 = \frac{1}{2n}||\hat{y} - y||_2^2 \tag{15}$$



Taux d'erreur de classification binaire

Utilisée pour les problèmes de classification (1 - Accuracy) :

$$J(\hat{y}, y) = \frac{1}{n} \sum_{i} (1 - \delta(\hat{y}_i - y_i)) \tag{16}$$

avec $\delta(x) = \begin{cases} 1 & \text{si} x = 0 \\ 0 & \text{sinon} \end{cases}$.

Retourne une valeur dans $[0, 1]$.

Régularisation de la fonction de coût

Objectif

- Ajoute de l'information sur la solution recherchée a priori
- Limite les valeurs des poids appris pour limiter le sur-apprentissage
- Réduits les problèmes d'explosion du gradient (*exploding gradient*)

Principe

Ajout d'un terme à la fonction de coût :

$$J'(W) = J(W) + \lambda\Omega(W) \tag{17}$$

Régularisation L2

$$\Omega(W) = \frac{1}{2}||W||_2^2 \tag{18}$$

Mise à jour des poids :

$$W \leftarrow W - \mu(\lambda W + \nabla_W J(W) \tag{19}$$

Vocabulaire de l'apprentissage profond

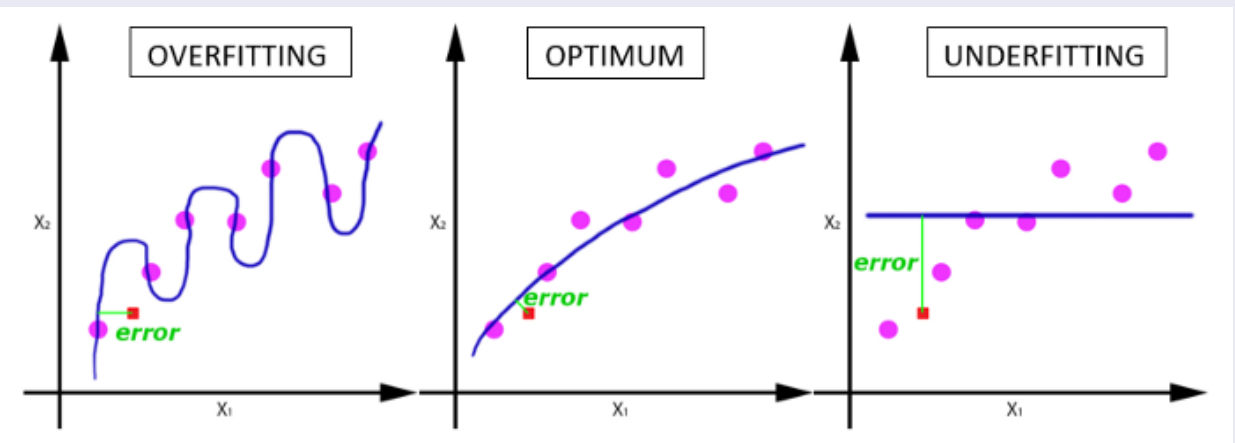
Époque (Epoch)

Situation lorsque l'algorithme d'apprentissage a utilisé entièrement le jeu données pour l'entraînement des paramètres (feedforward et backpropagation). En apprentissage profond, il est courant d'utiliser plusieurs *epochs* jusqu'à la convergence de l'algorithme (critère d'arrêt).

Mini-lot (mini-batch) et itérations

Un mini-lot est un sous-ensemble de la base de données d'entraînement traité en une fois à chaque itération de l'algorithme de descente de gradient. Ainsi pour un jeu de données de taille N et une taille de mini-lot de b nous obtenons le nombre total d'itérations par *epoch* : $n_{iter} = \lceil \frac{N}{b} \rceil$

Sous-apprentissage / Sur-apprentissage



Initialisation d'un réseau de neurones

Initialisation aléatoire pour l'entraînement

- L'initialisation va affecter la vitesse de convergence de l'algorithme de descente de gradient
- Une initialisation avec des poids nuls ne fonctionne pas la plupart du temps
- Le choix de l'initialisation est lié à la fonction d'activation
- Il est courant de choisir une initialisation aléatoire : $w_{i,j} \sim \mathcal{N}(0, \sigma^2)$ avec $\sigma \in]0; 1[$ pas trop petit ni trop grand

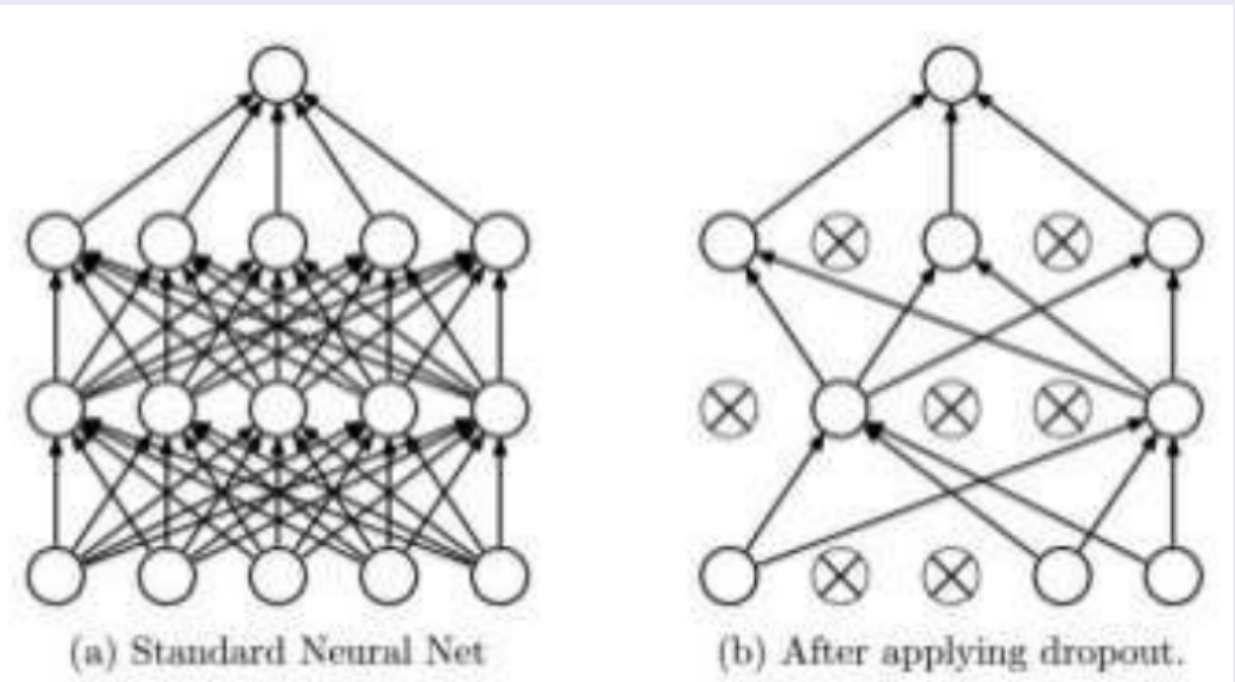
Plusieurs propositions :

- initialisation Xavier [Glorot et al. 2010] : $\sigma = \frac{1}{\sqrt{n}}$ (avec n la taille de l'entrée) $\Rightarrow$ bons résultats avec activation tanh()
- [*Heetal*.2015] : $\sigma = \frac{1}{\sqrt{n/2}} = \sqrt{\frac{2}{n}} \Rightarrow$ bons résultats avec activation ReLU

Ajout de bruit (dropout)

Principe

- Annule aléatoirement l'activation de certains neurones avec une probabilité p (0,2 pour l'entrée, 0,5 pour les couches cachées)
- Oblige le réseau à apprendre plusieurs solutions (améliore la robustesse)
- Aide à résoudre les problèmes de gradient nul (*vanishing gradient*)



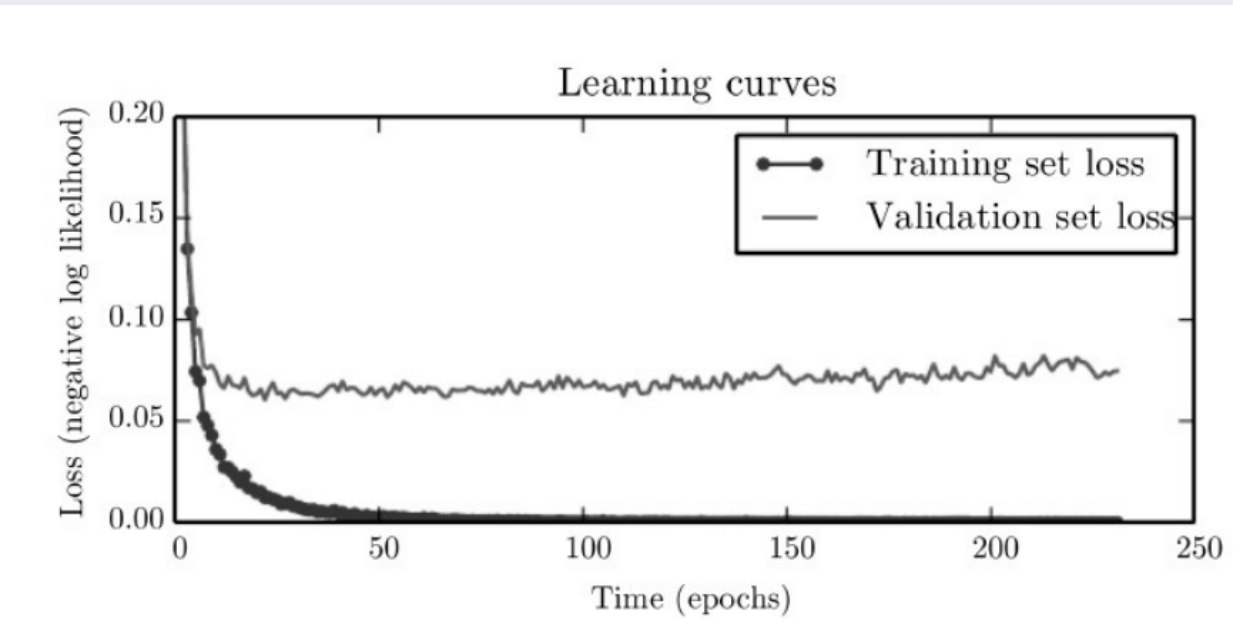
(a) Standard Neural Net

(b) After applying dropout.

Arrêt anticipé (Early stopping)

Principe

- Surveillance de l'apprentissage (eg. évolution de la fonction de coût)
- Arrêt dès que le réseau commence à faire du sur-apprentissage



Augmentation de données (data augmentation)

Principe

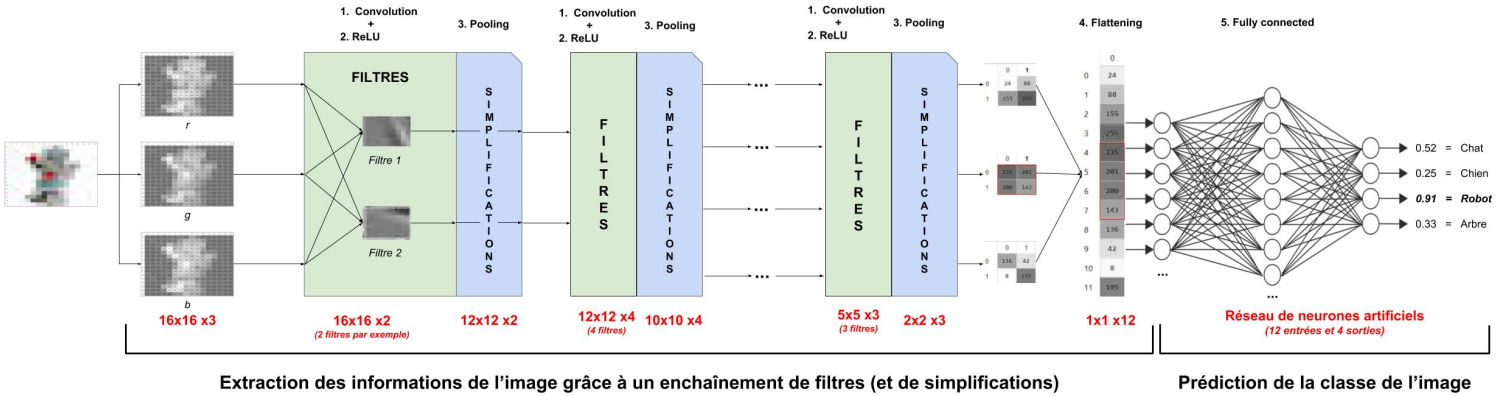
- Augmente la taille du jeu d'apprentissage pour rendre le modèle appris plus robuste aux perturbations
- À partir du jeu de donnée d'entraînement, créer de nouveaux exemples en appliquant certaines transformations pour simuler les perturbations naturelles des données et diversifier les exemples :
  - signal : ajout de bruit, re-échantillonnage, permutations circulaires, transposition, bruits structurés, etc.
  - image : rotations, homothéties, permutations, etc.

$\Rightarrow$ Augmente la robustesse au bruit du modèle appris

$\Rightarrow$ Évite le sur-apprentissage

$\Rightarrow$ Régularisation de l'apprentissage (attention de ne pas biaiser les données en apprenant les perturbations)

Principe des ConvNet (CNNs)



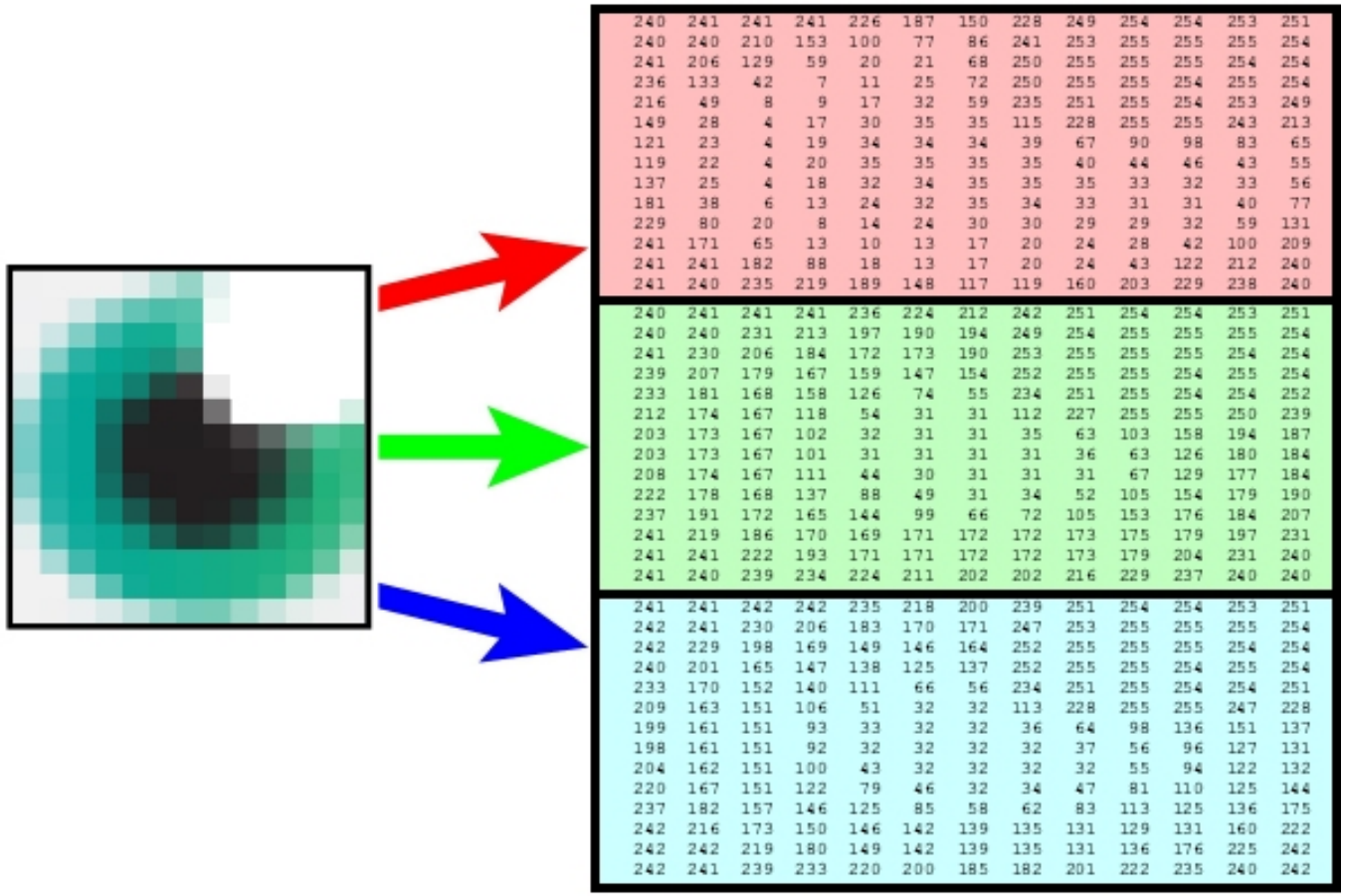
Principe

- On remplace la fonction neuronale par un produit de convolution avec l'entrée (feedforward)
- Les poids appris correspondent aux coefficients du filtre (*kernel*)
- On peut fixer la taille du chaque filtre et du pas de déplacement (stride)
- La rétropropagation du gradient fonctionne aussi comme pour le perceptron
- On peut intégrer au réseau des couches de transformation de l'entrée (eg. pooling, upsampling, etc.)

Motivation

- Interactions parcimonieuses : moins de connexions entre les neurones et une taille des filtres inférieure à la celle de l'entrée
- Calcul de représentations pertinentes de l'entrée robustes au bruit (invariantes pour certaines transformations)

Exemple : Traitement d'image par CNN



- Traitement 2D simultané des $c = 3$ canaux (*i.e.* RGB)
- Une convolution 2D distincte par canal (kernel de dimension $w_k \times h_k \times c$)
- On applique comme pour le perceptron une fonction d'activation sur la sortie $\sigma()$

Déplacement du filtre

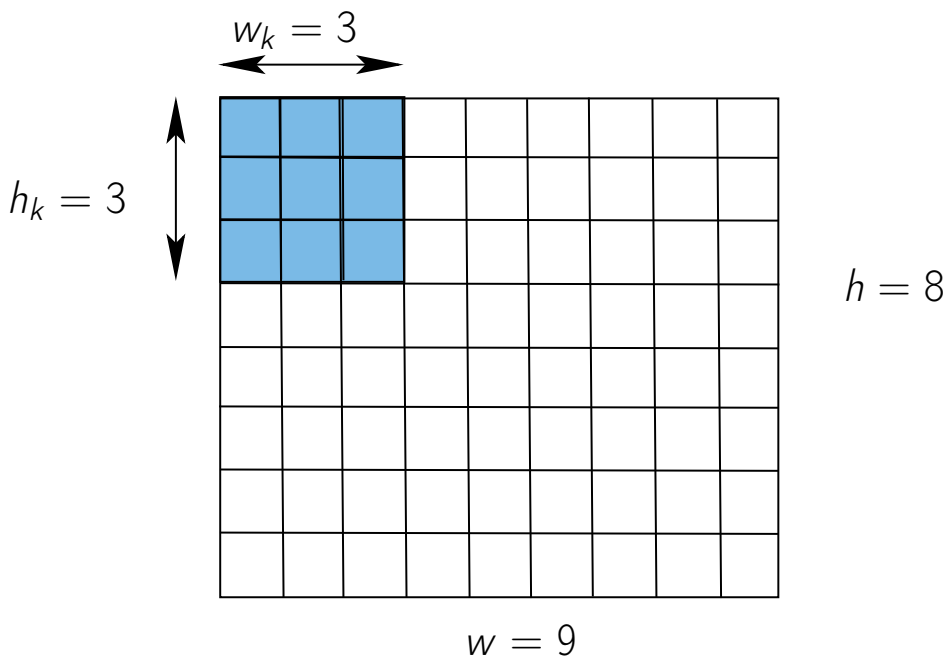
Paramètres

- Dimension de l'image $h \times w$ sur c canaux
- Dimensions du filtre $h_k \times w_k$ sur c canaux
- Pas d'avancement (*stride*) s

Taille de la sortie :

$$\hat{h} = \left\lfloor \frac{h - h_k}{s} \right\rfloor + 1 \quad \hat{w} = \left\lfloor \frac{w - w_k}{s} \right\rfloor + 1 \tag{20}$$

exemple :

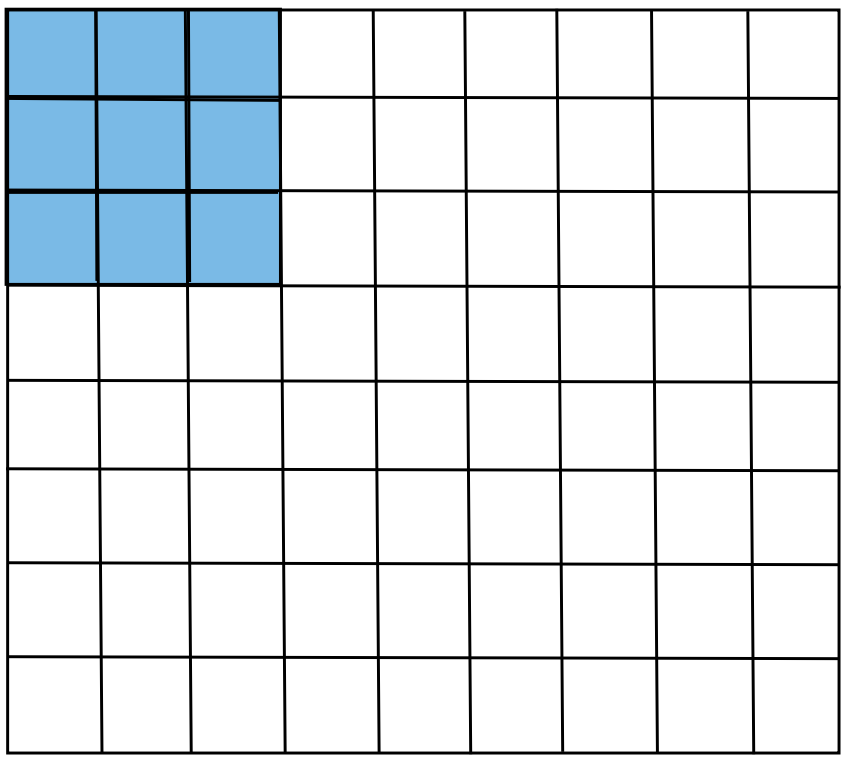


Exemple CNN 2D

Paramètres

- Dimension de l'entrée ($h \times w \times c$) : $h = 8$, $w = 9$, $c = 1$
- Dimensions du filtre $3 \times 3 \times 1$
- Pas d'avancement (*stride*) $s = 1$

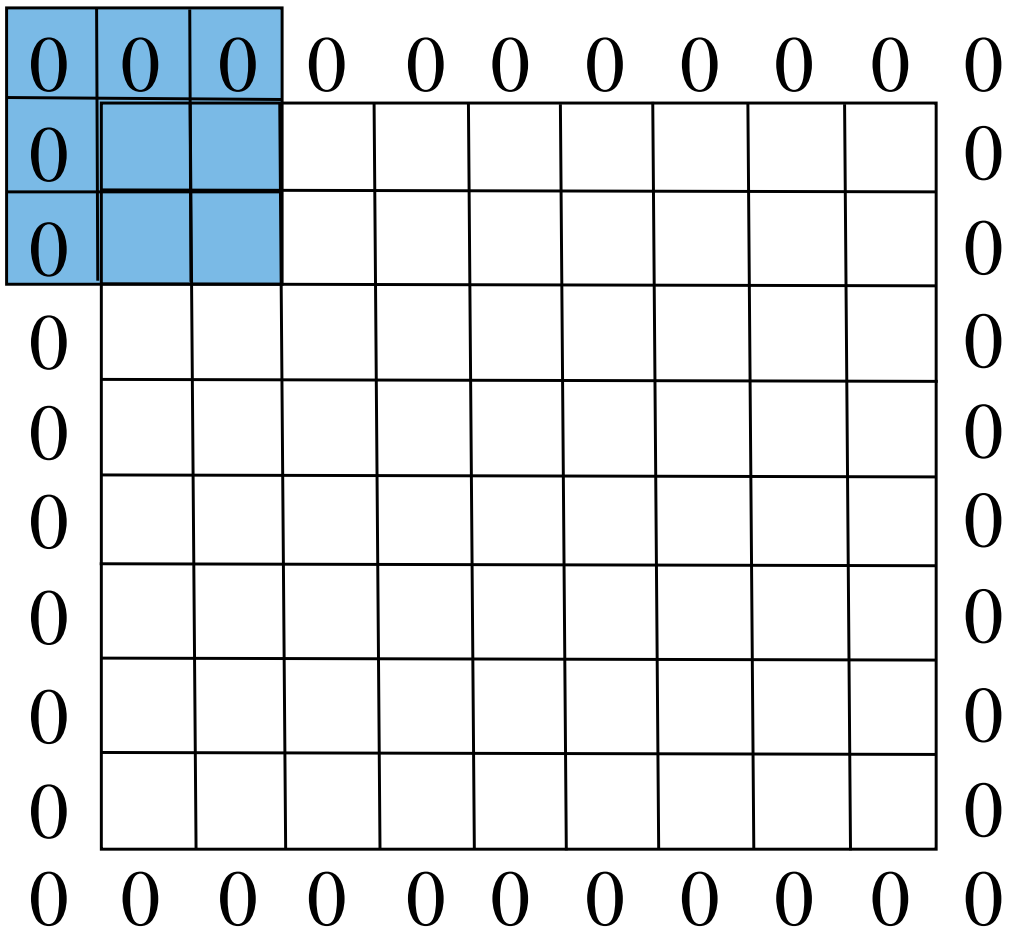
Taille de la sortie : $\hat{h} = 6$, $\hat{w} = 7$, $\hat{c} = c = 1$



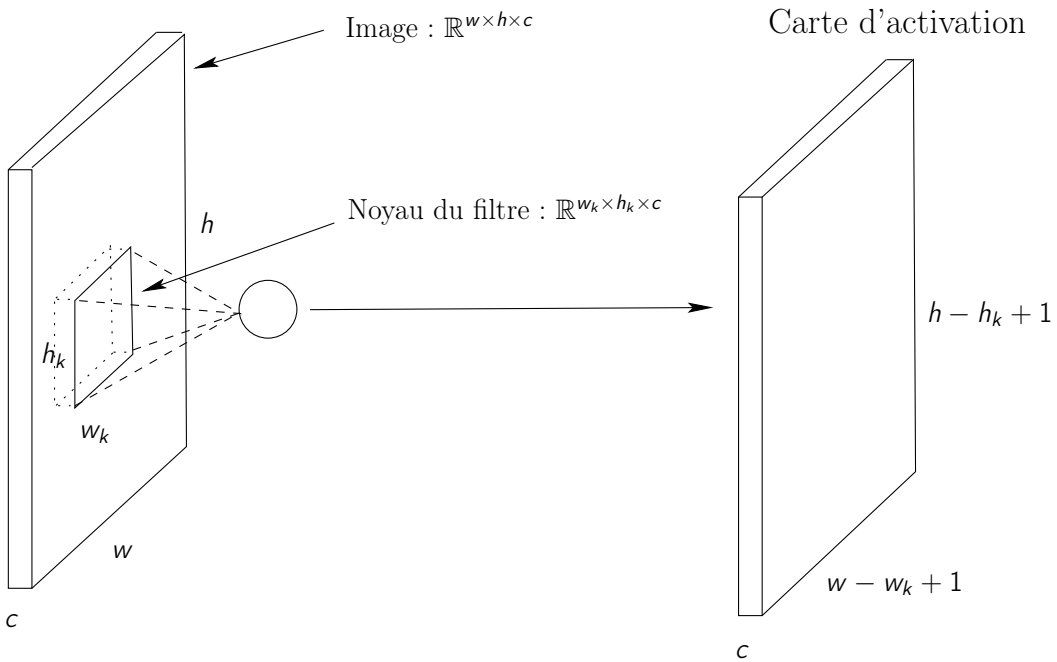
0-padding

Paramètres

Taille de la sortie identique à l'entrée : $\hat{h} = h$, $\hat{w} = w$, $\hat{c} = c = 1$



CNN 2D



Principe

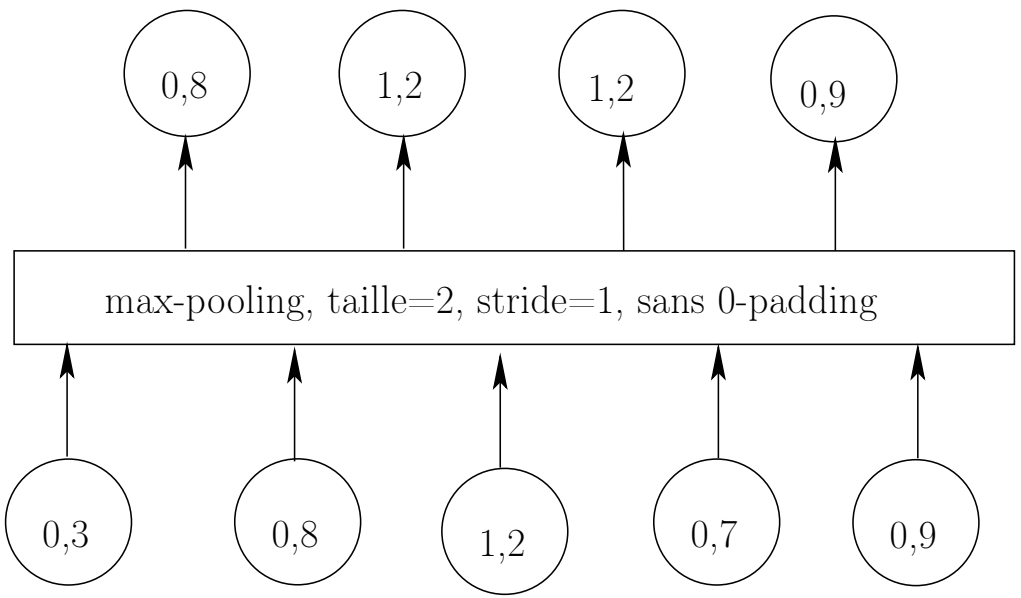
- Une carte d'activation produite par chaque filtre (nombre fixé par l'utilisateur)
- Une couche de convolution peut être composée de plusieurs filtres (neurones) suivie d'une fonction d'activation.

Pooling (mutualisation)

Principe

- Réduit la dimension de l'entrée en appliquant une statistique sur chaque partie de l'entrée
- Aide à rendre la carte d'activation invariante à des petites translations de l'entrée
- Comme pour les neurones convolutifs, on on peut définir une taille de kernel, un pas de déplacement (stride) et une interpolation (padding).

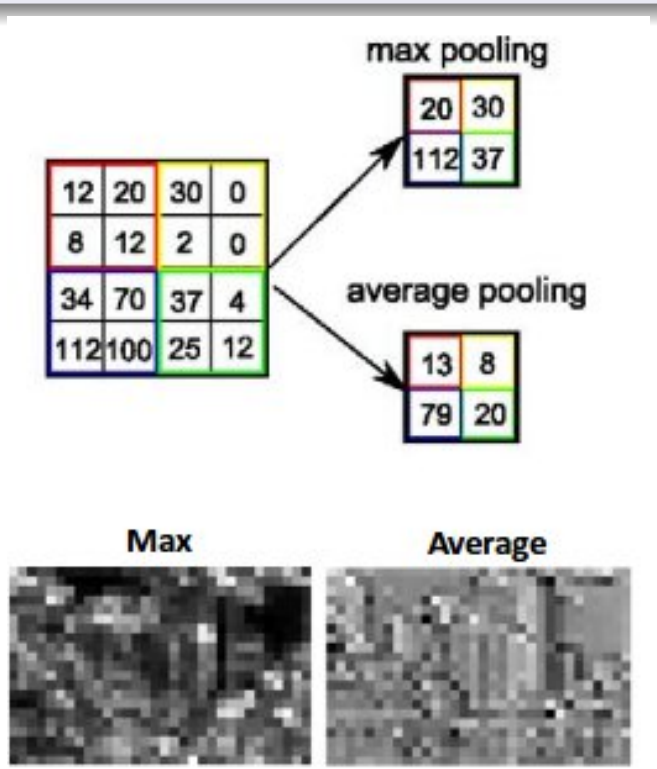
Exemple (max-pooling) :



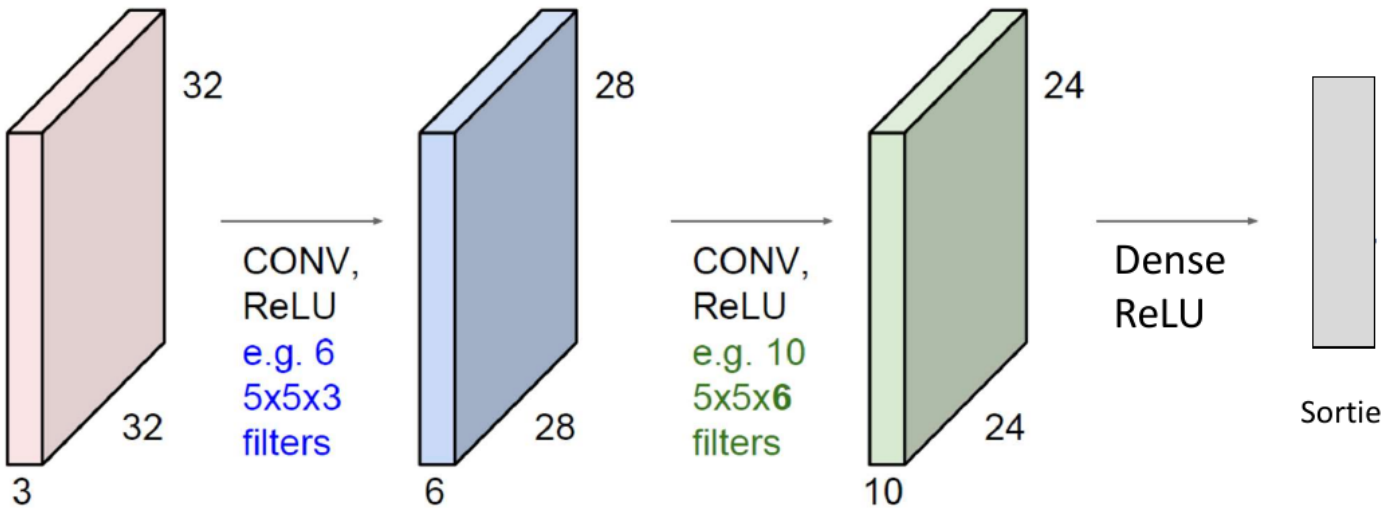
Pooling (mutualisation)

Exemples de fonctions de pooling

- max-pooling [Zhou, Chellappa 1898]
- average-pooling
- max-min pooling [Blot, Cord, Thome, 2016]
- Zeckendorf-pooling [Vigneron et al., 2021]
- etc.



Classification d'images par CNN



Principe

- Succession de plusieurs couches de convolution et de fonctions d'activation pour obtenir une représentation de l'entrée (*features maps* ou *deep features*)
- Une ou plusieurs couches denses (*fully connected*) connectées sur la sortie de la dernière couche de convolution pour la classification

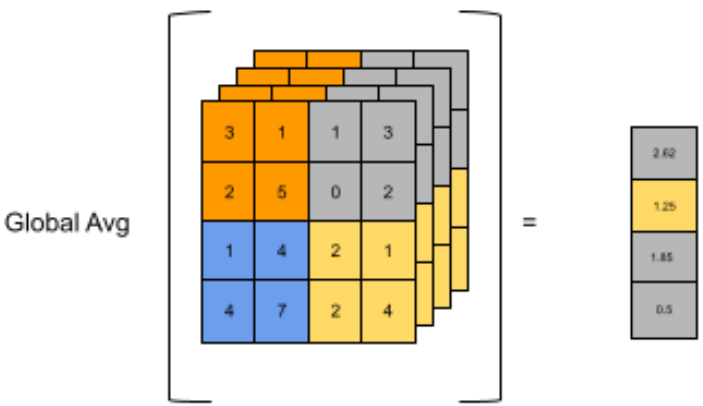
Global Pooling

[Lin, Chen, Yan, 2013] Network In Network [8]

Principe

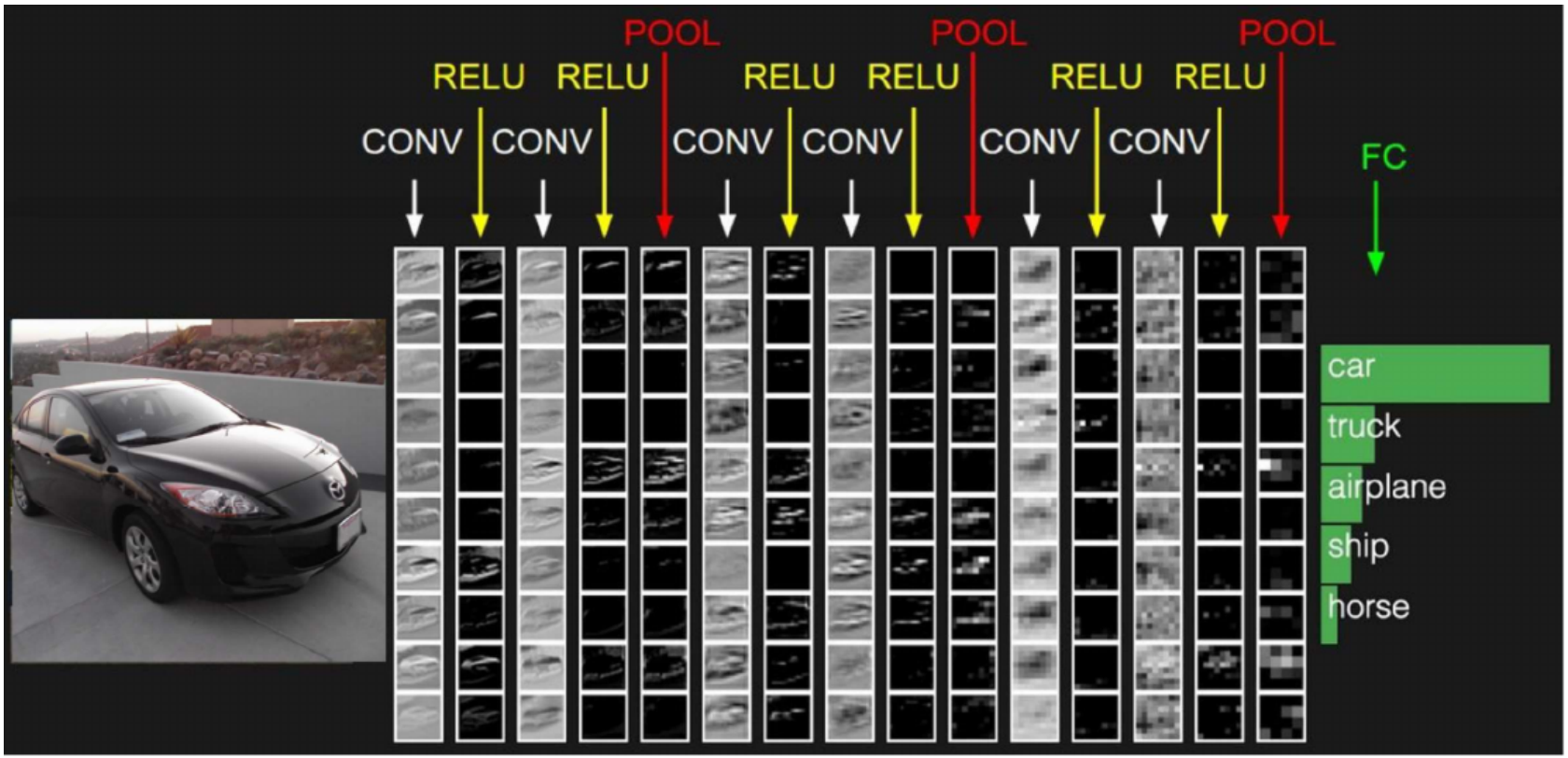
- Remplace la transformation en couches denses par un *pooling* appliqué sur la dernière couche de convolution
- Calcule un résumé statistique de la dernière couche permettant de traiter une tâche de classification
- Semble mieux fonctionner que l'ajout de couches denses *fully connected* avant la sortie du réseau

exemple (Global-Average-Pooling) :



8. https://paperswithcode.com/method/global-average-pooling

Classification d'images par CNN

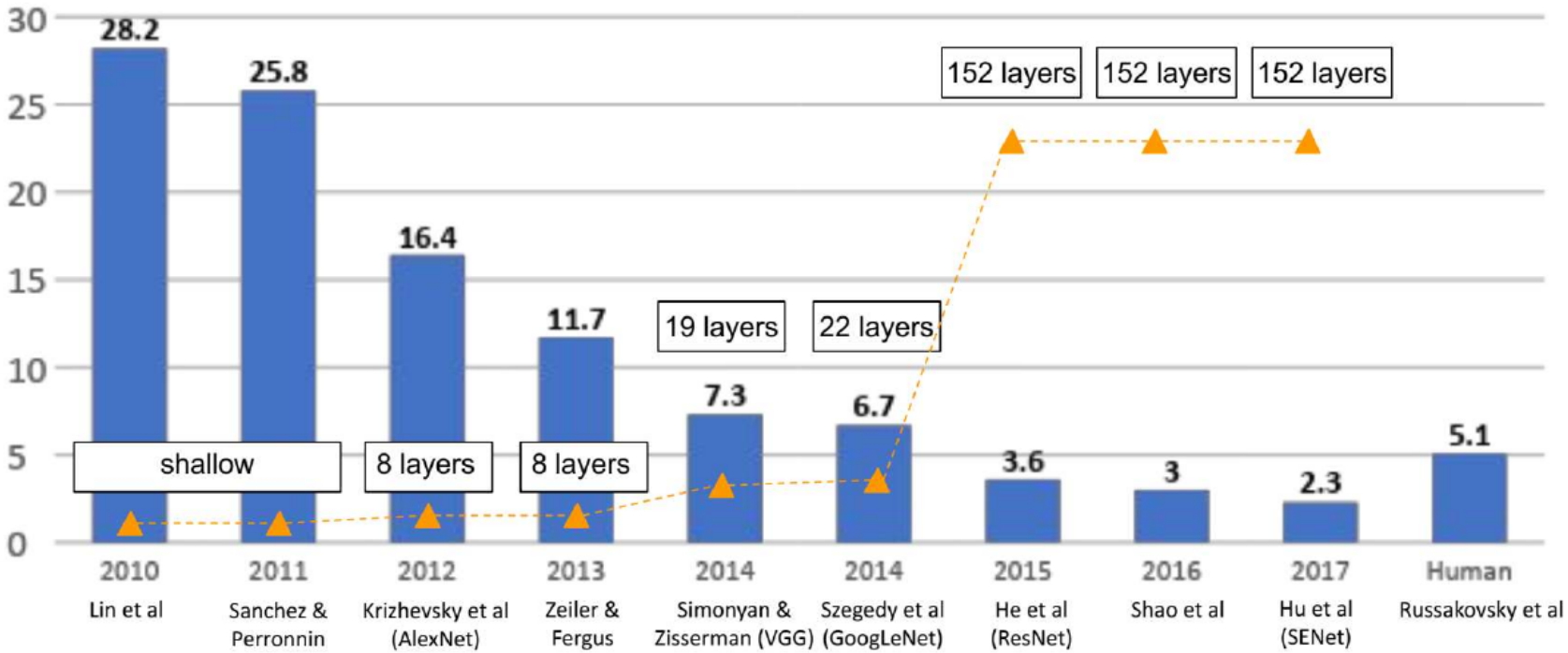


Recommandations

- Nombre de filtres : 32, 64, 128, 256
- Dimension des filtres $w_k = h_k = 3$ ou 5, stride $s = 1$
- Pooling : $w_k = h_k = 2$, stride $s = 2$

Taux d'erreur (%) sur ImageNet à ILSVRC

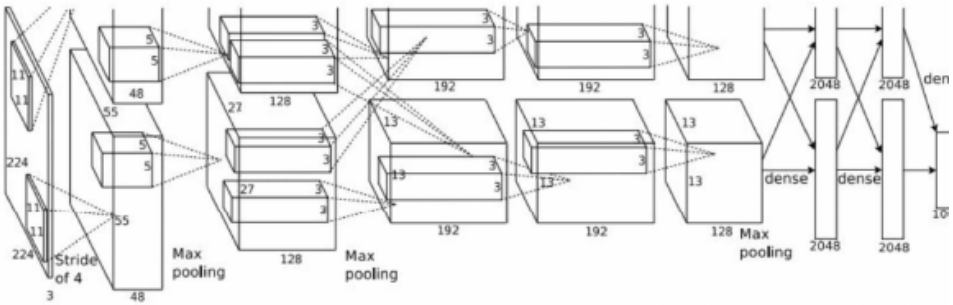
ILSVRC : ImageNet Large Scale Visual Recognition Challenge



Alexnet

Case Study: AlexNet

[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Implementation (Pytorch)

`torchvision.models.alexnet.AlexNet`

VGG 1

Case Study: VGGNet

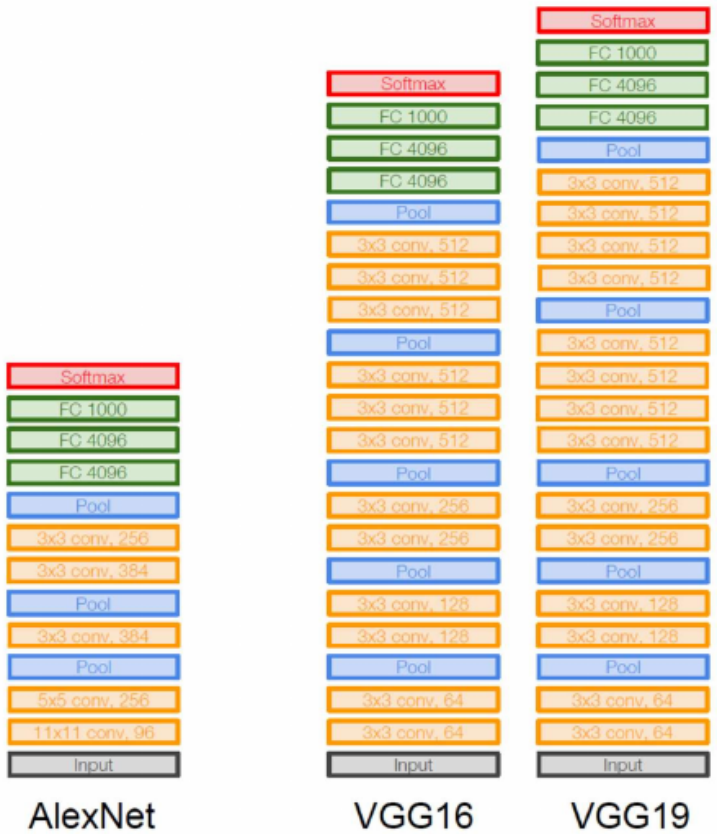
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14

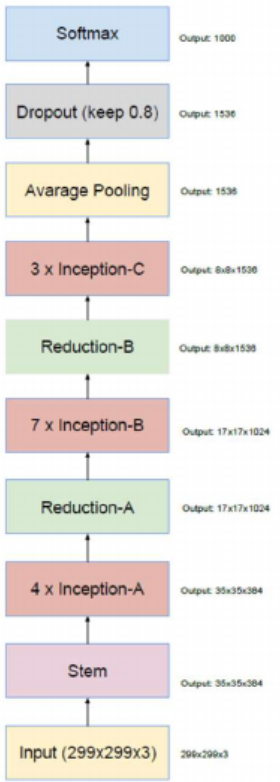


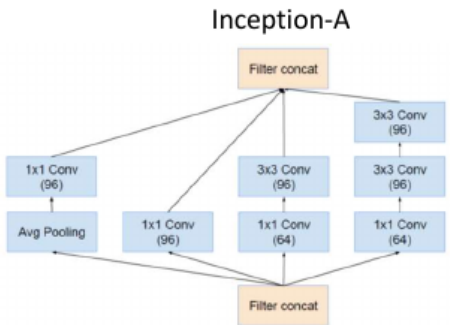
Implementation (Keras ou Pytorch)

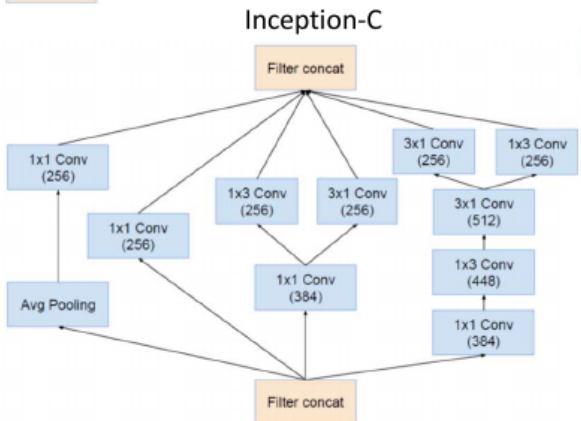
`tf.keras.applications.VGG16`
`torchvision.models.vgg.VGG`

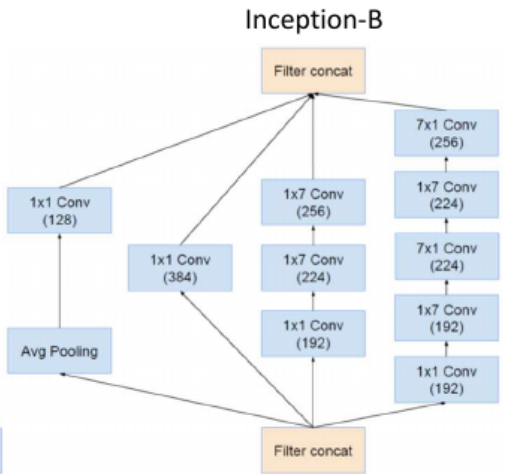
Inception

Inception V4





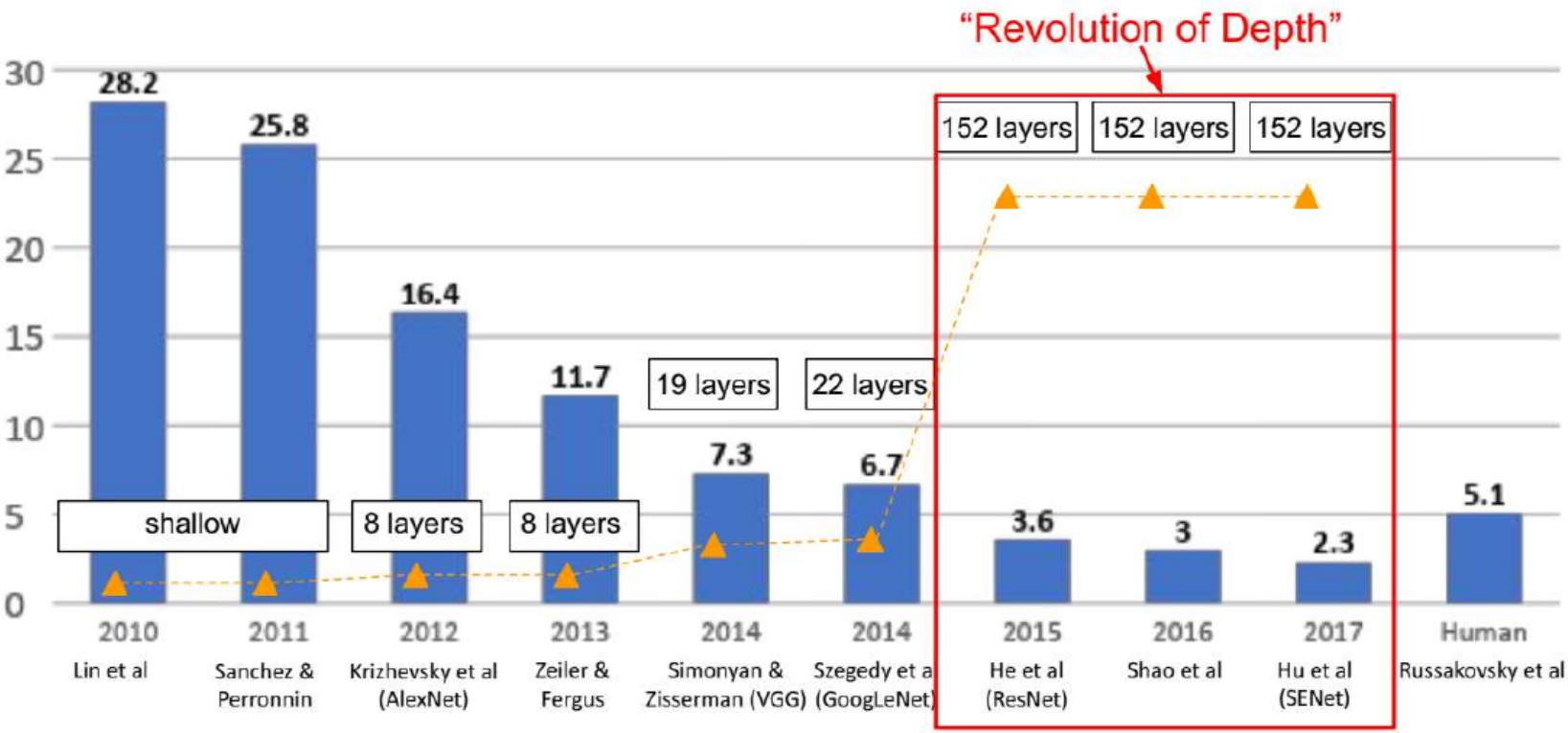




Implementation (Keras / Pytorch)

```
tf.keras.applications.InceptionV3
torchvision.models.inception.Inception3
```

Réseaux très profonds



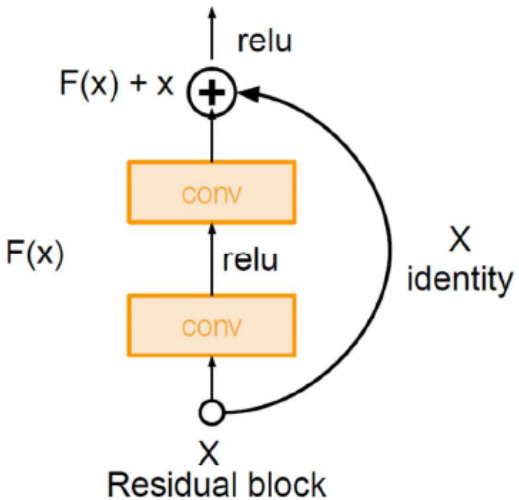
Resnet 1/2

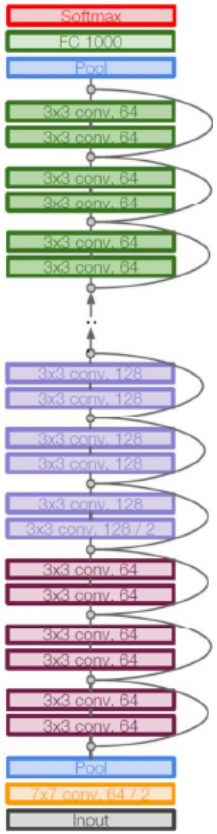
Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!





Implementation (Keras / Pytorch)

`tf.keras.applications.ResNet152V2`
`torchvision.models.resnet.ResNet`

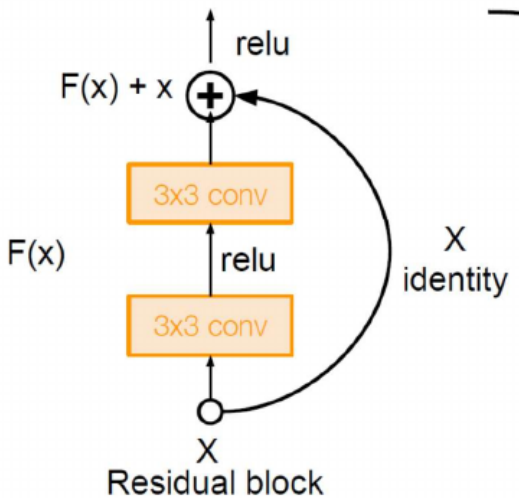
Resnet 2/2

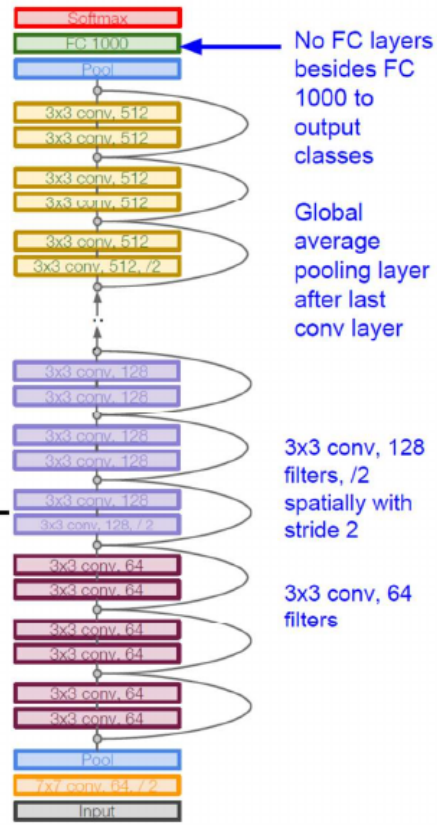
Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)



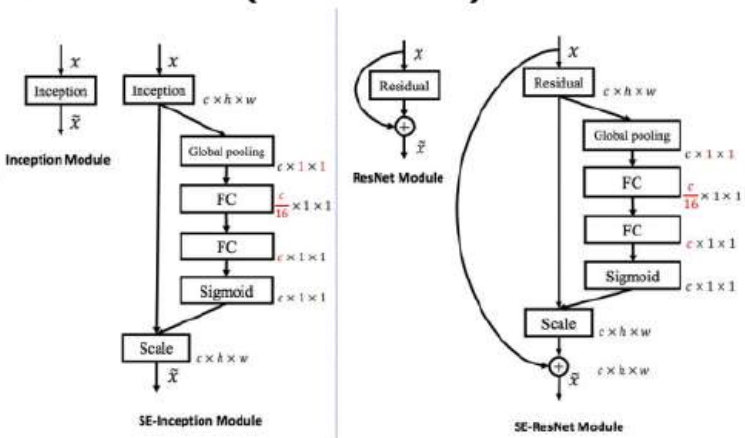


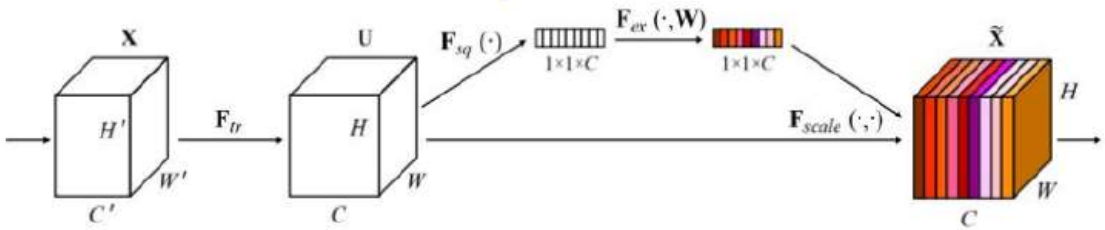
SEnet

Squeeze-and-Excitation Networks (SENet)

[Hu et al. 2017]

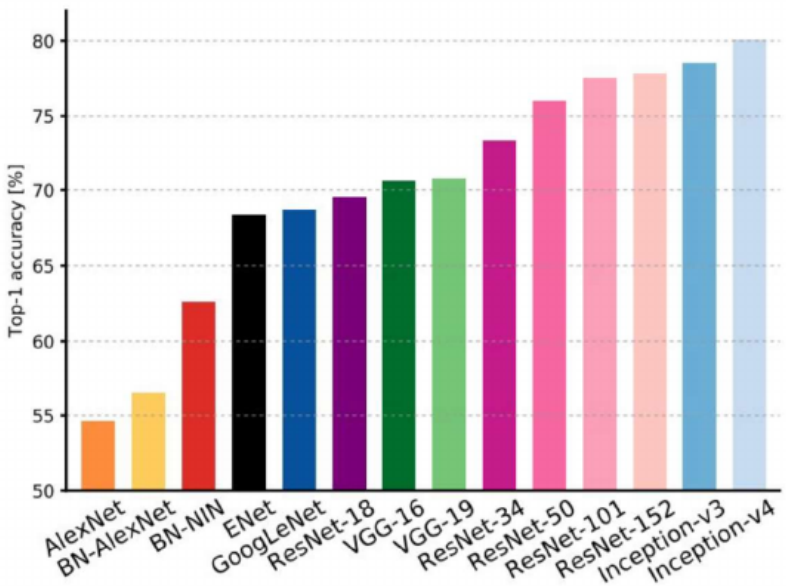
- Add a "feature recalibration" module that learns to adaptively reweight feature maps
- Global information (global avg. pooling layer) + 2 FC layers used to determine feature map weights
- ILSVRC'17 classification winner (using ResNeXt-152 as a base architecture)

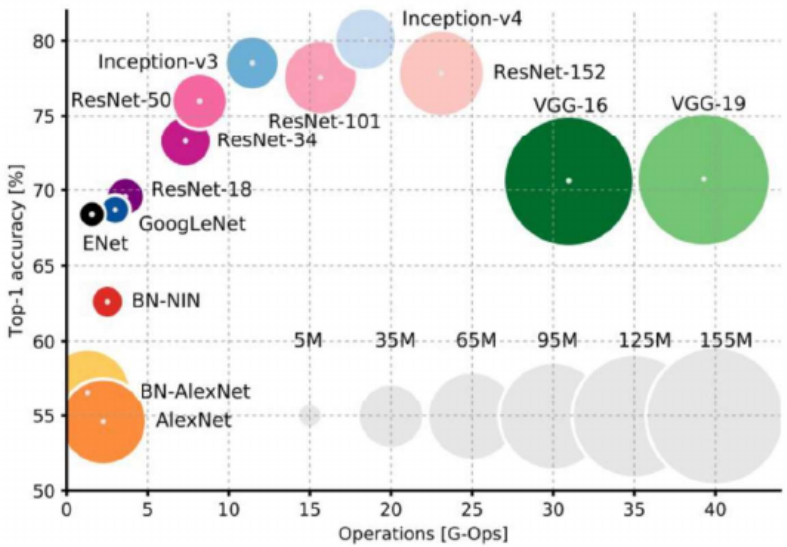




Comparatif des complexités

Comparing complexity...





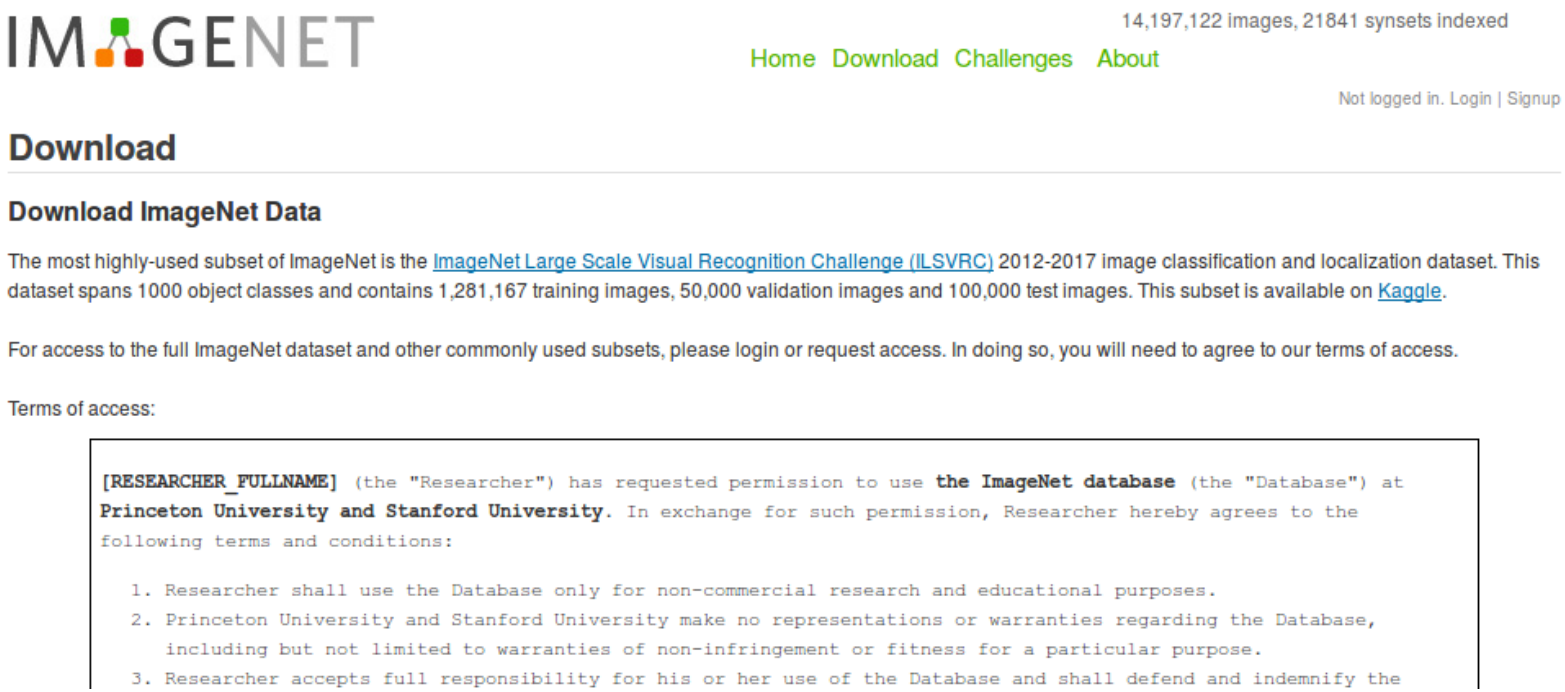
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Passage à la pratique

Imagenet

`https://www.image-net.org/`

- 1000 classes
- 14 197 122 images (sept. 22)
- $\approx 10^6$ images avec annotation de localisation
- $\approx 4,2 \times 10^6$ images avec annotation de description



Bilan et perspectives

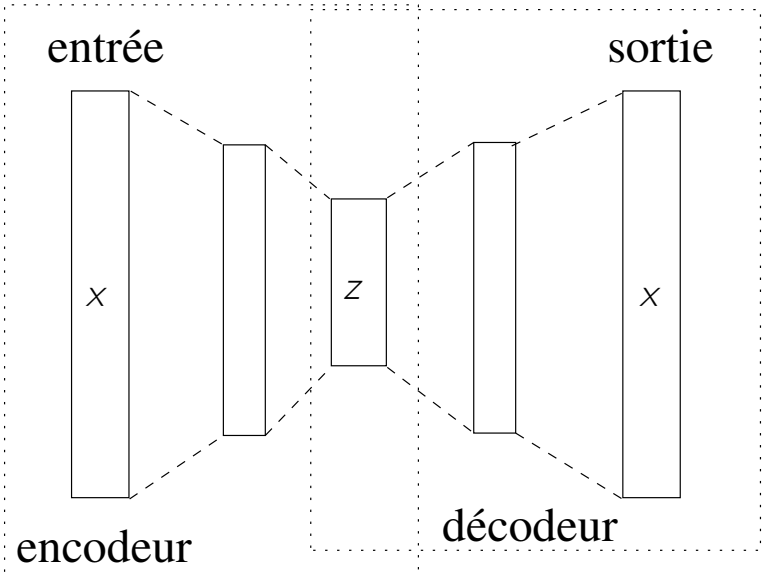
Dans cette présentation

- Introduction aux réseaux de neurones et à l'apprentissage profonds
- Fondements théoriques et présentation des principaux algorithmes d'apprentissage
- Quelques pistes bibliographiques pour comprendre et utiliser les algorithmes existants

Pour aller plus loin

- Modèles génératifs (GAN, AE, VAE)
- Modèles d'attention et modèles auto-attentifs (transformeurs)
- Interprétabilité des modèles (layer-wise Relevance Propagation, etc.)
- Apprentissage auto-supervisé et apprentissage de représentation
- Apprentissage par renforcement
- *Pruning* de modèles

Autoencodeur (déterministe)



Idée

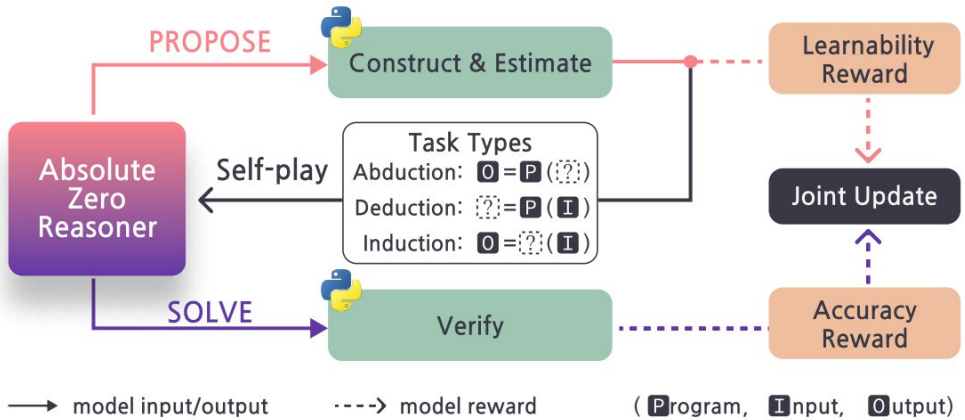
Contraint le réseau à apprendre une représentation z plus compacte des données (codage).

- Apprentissage non supervisé de représentations latentes
- Structure : encodeur $f(x)$, décodeur $g(z)$, reconstruction $\hat{x} = g(f(x))$
- Fonction de coût : $\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2$

Bengio, Y. et al. (2013). Representation Learning : A Review.
Kingma, D. P., & Welling, M. (2014). Auto-Encoding Variational Bayes.

Un nouveau paradigme pour l'apprentissage par renforcement

Absolute Zero (may 2025) [9]



- Apprentissage par renforcement sans données d'entraînement
- Génère seul et automatiquement des tâches d'apprentissage et les résoud pour renforcer un modèle
- Nécessite une graine d'entraînement très simple pour initialiser l'algorithme.

`https://github.com/LeapLabTHU/Absolute-Zero-Reasoner`

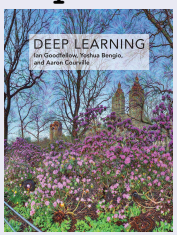
9. Zhao, A., Wu, Y., Yue, Y., Wu, T., Xu, Q., Lin, M., ... & Huang, G. (2025). Absolute zero : Reinforced self-play reasoning with zero data. arXiv preprint arXiv :2505.03335.

Bibliographie

Merci pour votre attention.

Deep learning

Bengio, Y., Goodfellow, I., & Courville, A. (2017). Deep learning (Vol. 1). Cambridge, MA, USA : MIT press. `https://www.deeplearningbook.org/`



Deep learning

Géron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : Concepts, tools, and techniques to build intelligent systems. O'Reilly Media, Inc.