

Apprentissage profond

Mathieu Lefort

2 & 3 septembre 2024

Planning

- Introduction aux réseaux de neurones profonds et à PyTorch
(4,5h CM + 6h TP) - Mathieu Lefort
- Apprentissage par renforcement profond
(9h CM/TP) - Laëtitia Matignon
- MCTS et application aux jeux
(3h CM + 7.5h TP) - Stéphane Bonnevay

Évaluation

- Examen final (sur les 3 parties) - 40%
- Un projet par partie - 20% chacun

Ressources

- Tout sera mis sous Moodle (cours Apprentissage Profond)

Plan

- Principes généraux de l'apprentissage profond (neurones, couches, règle de chaînage, ...)
- Perceptron (Multi Couches)
- Réseaux Convolutionnels
- Panorama général du domaine et "Trucs & Astuces"
- Limitations/Problèmes de l'apprentissage profond

Objectifs

- Compréhension des principes, propriétés et limitations des modèles présentés
- Prise en main de l'apprentissage profond (réglages, framework, ...)

Question

Qu'attendre d'un système apprenant ? Pourquoi ? Comment ?

Question

Qu'attendre d'un système apprenant ? Pourquoi ? Comment ?

Objectifs

- mémorisation : retrouver la réponse à une entrée déjà vue
- généralisation : trouver la réponse à une entrée inconnue (\neq sur-apprentissage)
- hypothèses : fonction localement continue, rasoir d'Occam
- techniques : ensemble apprentissage/validation/test, régularisation, ...

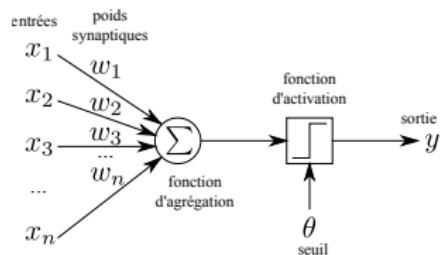
Types

- supervisé : $y = f(x)$ avec $x \in \mathcal{X}$ et $y \in \mathcal{Y}$
 - régression : \mathcal{Y} continu, typiquement \mathbb{R}^n
 - classification : \mathcal{Y} discret
 - en pratique $f(x) = \sum_{i=1}^n (a_i^\top \cdot x + b_i)\phi(x, \theta_i)$ avec ϕ une fonction non linéaire paramétrée par θ_i , $b_i \in \mathbb{R}^{card(\mathcal{Y})}$ et $a_i \in \mathbb{R}^{card(\mathcal{Y}) \times card(\mathcal{X})}$
- non supervisé : $f(x)$ avec $x \in \mathcal{X}$
 - clustering : regroupement des entrées par groupes
 - auto-supervisé : $y' = f(x)$ avec $y' = t(x)$ une tâche prétexte
- par renforcement, par imitation, ...

Neurone artificiel - McCulloch et Pitts (1943)

L'inspiration biologique :

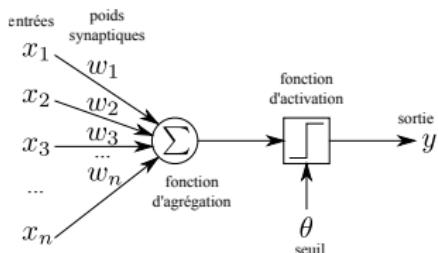
$$\text{Sortie du neurone} : y = \begin{cases} 1 & \text{si } \sum_{i=1}^n w_i x_i - \theta > 0 \\ 0 & \text{sinon} \end{cases}$$



Neurone artificiel - McCulloch et Pitts (1943)

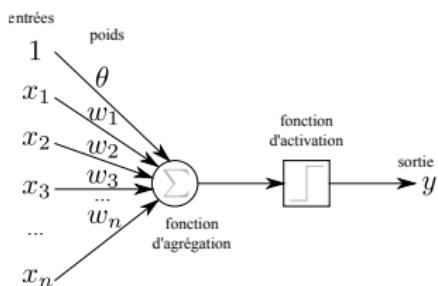
L'inspiration biologique :

$$\text{Sortie du neurone} : y = \begin{cases} 1 & \text{si } \sum_{i=1}^n w_i x_i - \theta > 0 \\ 0 & \text{sinon} \end{cases}$$



Le neurone en apprentissage automatique :

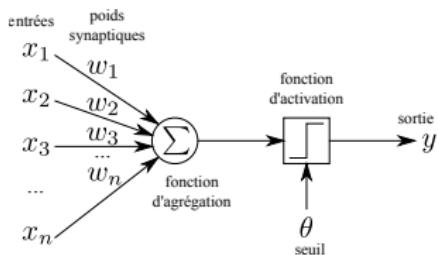
$$\text{Sortie du neurone} : y = \begin{cases} 1 & \text{si } \sum_{i=0}^n w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$$



Neurone artificiel - McCulloch et Pitts (1943)

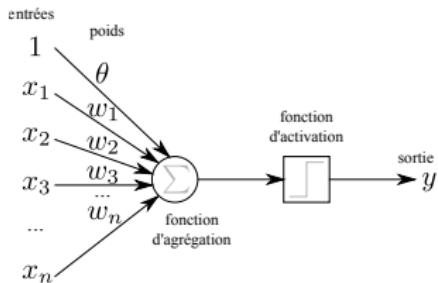
L'inspiration biologique :

$$\text{Sortie du neurone} : y = \begin{cases} 1 & \text{si } \sum_{i=1}^n w_i x_i - \theta > 0 \\ 0 & \text{sinon} \end{cases}$$



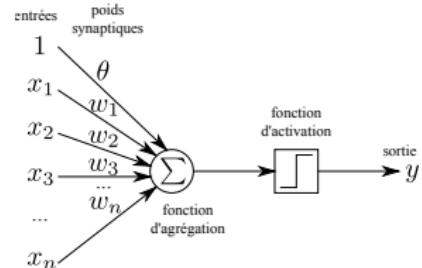
Le neurone en apprentissage automatique :

$$\text{Sortie du neurone} : y = \begin{cases} 1 & \text{si } w^T x > 0 \\ 0 & \text{sinon} \end{cases}$$



Perceptron - Rosenblatt (1957)

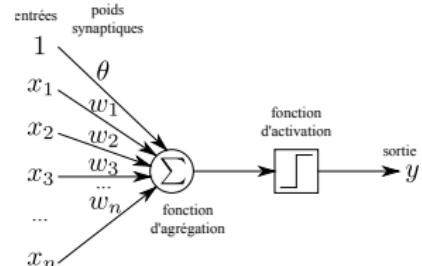
Sortie du neurone : $y = \begin{cases} 1 & \text{si } \sum_{i=0}^n w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$



Perceptron - Rosenblatt (1957)

Sortie du neurone : $y = \begin{cases} 1 & \text{si } \sum_{i=0}^n w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$

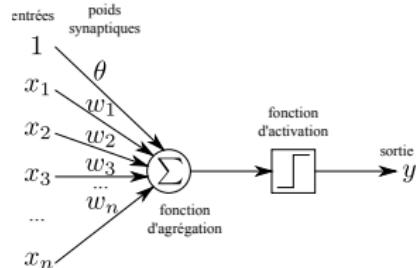
Apprentissage : $\Delta W = \eta X(t - y)$



Perceptron - Rosenblatt (1957)

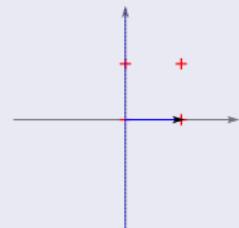
Sortie du neurone : $y = \begin{cases} 1 & \text{si } \sum_{i=0}^n w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$

Apprentissage : $\Delta W = \eta X(t - y)$



Un exemple : le OU

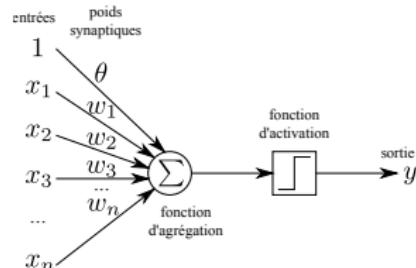
Entrée X			Poids W			Sortie		Sortie voulue	
x_0	x_1	x_2	w_0	w_1	w_2	y		t	0
1	0	0	0	1	0	0			



Perceptron - Rosenblatt (1957)

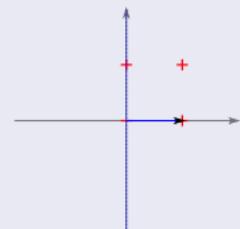
Sortie du neurone : $y = \begin{cases} 1 & \text{si } \sum_{i=0}^n w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$

Apprentissage : $\Delta W = \eta X(t - y)$



Un exemple : le OU

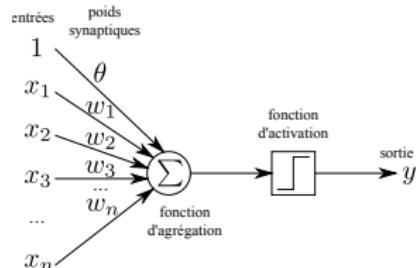
Entrée X			Poids W			Sortie	Sortie voulue	
x_0	x_1	x_2	w_0	w_1	w_2	y	t	1
1	0	1	0	1	0	0		



Perceptron - Rosenblatt (1957)

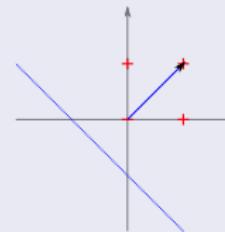
Sortie du neurone : $y = \begin{cases} 1 & \text{si } \sum_{i=0}^n w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$

Apprentissage : $\Delta W = \eta X(t - y)$



Un exemple : le OU

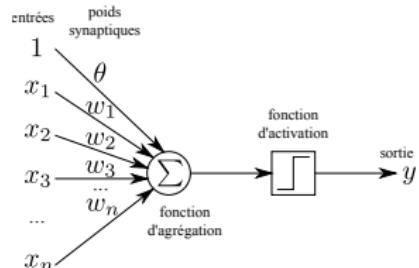
Entrée X			Poids W			Sortie		Sortie voulue	
x_0	x_1	x_2	w_0	w_1	w_2	y	t	1	
1	0	1	1	1	1	0			



Perceptron - Rosenblatt (1957)

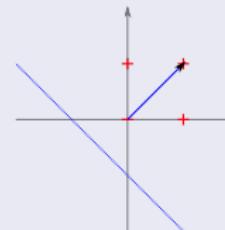
Sortie du neurone : $y = \begin{cases} 1 & \text{si } \sum_{i=0}^n w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$

Apprentissage : $\Delta W = \eta X(t - y)$



Un exemple : le OU

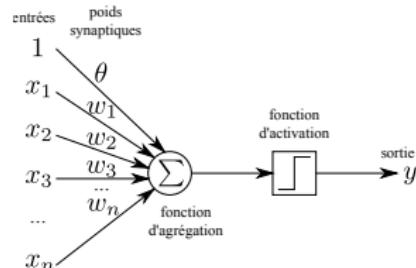
Entrée X			Poids W			Sortie		Sortie voulue	
x_0	x_1	x_2	w_0	w_1	w_2	y		t	1
1	1	0	1	1	1	1			



Perceptron - Rosenblatt (1957)

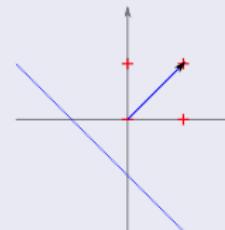
Sortie du neurone : $y = \begin{cases} 1 & \text{si } \sum_{i=0}^n w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$

Apprentissage : $\Delta W = \eta X(t - y)$



Un exemple : le OU

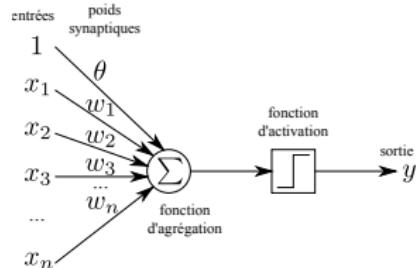
Entrée X			Poids W			Sortie		Sortie voulue	
x_0	x_1	x_2	w_0	w_1	w_2	y		t	1
1	1	1	1	1	1	1			



Perceptron - Rosenblatt (1957)

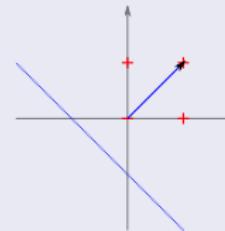
Sortie du neurone : $y = \begin{cases} 1 & \text{si } \sum_{i=0}^n w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$

Apprentissage : $\Delta W = \eta X(t - y)$



Un exemple : le OU

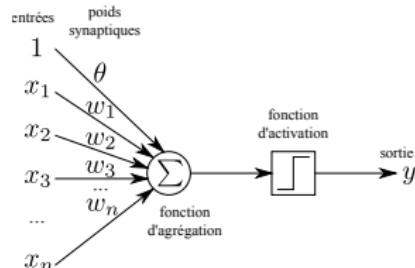
Entrée X			Poids W			Sortie		Sortie voulue	
x_0	x_1	x_2	w_0	w_1	w_2	y	t	0	
1	0	0	1	1	1	1	1	0	



Perceptron - Rosenblatt (1957)

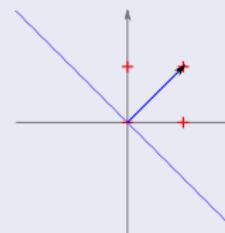
Sortie du neurone : $y = \begin{cases} 1 & \text{si } \sum_{i=0}^n w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$

Apprentissage : $\Delta W = \eta X(t - y)$



Un exemple : le OU

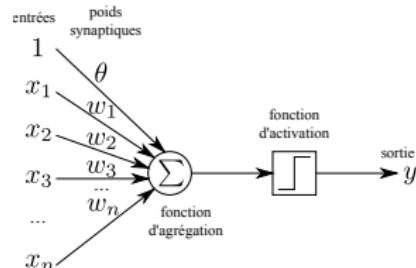
Entrée X			Poids W			Sortie		Sortie voulue	
x_0	x_1	x_2	w_0	w_1	w_2	y	t	0	
1	0	0	0	1	1	1			



Perceptron - Rosenblatt (1957)

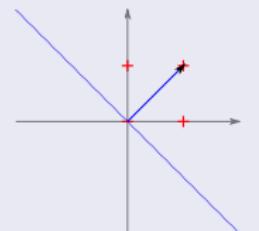
Sortie du neurone : $y = \begin{cases} 1 & \text{si } \sum_{i=0}^n w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$

Apprentissage : $\Delta W = \eta X(t - y)$



Un exemple : le OU

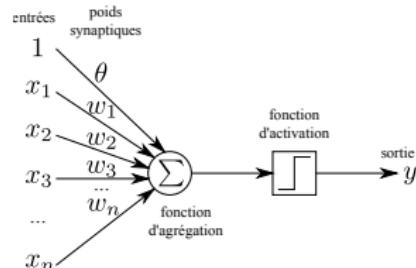
Entrée X			Poids W			Sortie		Sortie voulue	
x_0	x_1	x_2	w_0	w_1	w_2	y		t	1
1	0	1	0	1	1	1			



Perceptron - Rosenblatt (1957)

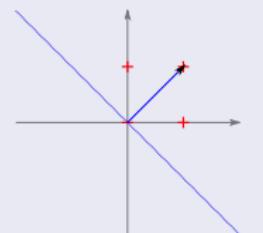
Sortie du neurone : $y = \begin{cases} 1 & \text{si } \sum_{i=0}^n w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$

Apprentissage : $\Delta W = \eta X(t - y)$



Un exemple : le OU

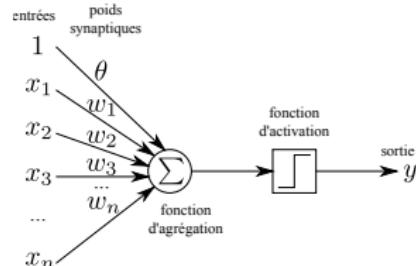
Entrée X			Poids W			Sortie		Sortie voulue	
x_0	x_1	x_2	w_0	w_1	w_2	y		t	1
1	1	0	0	1	1	1			



Perceptron - Rosenblatt (1957)

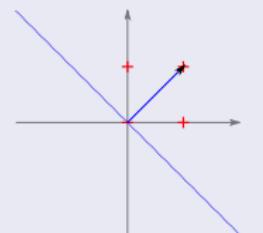
Sortie du neurone : $y = \begin{cases} 1 & \text{si } \sum_{i=0}^n w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$

Apprentissage : $\Delta W = \eta X(t - y)$



Un exemple : le OU

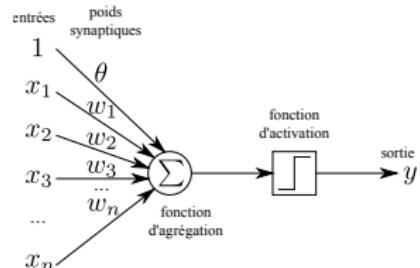
Entrée X			Poids W			Sortie		Sortie voulue	
x_0	x_1	x_2	w_0	w_1	w_2	y		t	1
1	1	1	0	1	1	1			



Perceptron - Rosenblatt (1957)

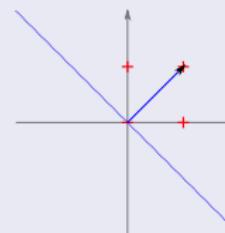
Sortie du neurone : $y = \begin{cases} 1 & \text{si } \sum_{i=0}^n w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$

Apprentissage : $\Delta W = \eta X(t - y)$



Un exemple : le OU

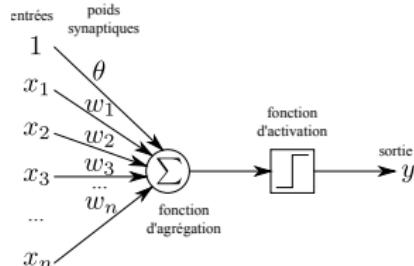
Entrée X			Poids W			Sortie		Sortie voulue	
x_0	x_1	x_2	w_0	w_1	w_2	y		t	0
1	0	0	0	1	1	0			



Perceptron - Rosenblatt (1957)

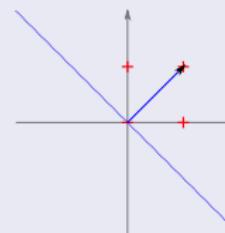
Sortie du neurone : $y = \begin{cases} 1 & \text{si } \sum_{i=0}^n w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$

Apprentissage : $\Delta W = \eta X(t - y)$



Un exemple : le OU

Entrée X			Poids W			Sortie		Sortie voulue	
x_0	x_1	x_2	w_0	w_1	w_2	y	t	0	
1	0	0	0	1	1	0	1	0	



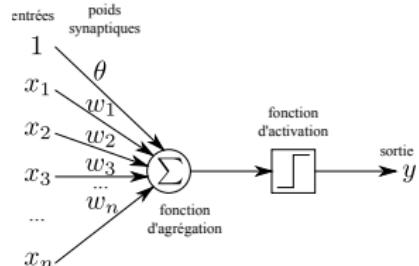
Propriétés

- Classifieur linéaire (découpage de l'espace avec un hyperplan)
- Poids = vecteur normal à l'hyperplan, Seuil (biais) = ordonnée à l'origine
- Convergence si η est (infinitésimale) petit et si les données sont linéairement séparables

Perceptron - Rosenblatt (1957)

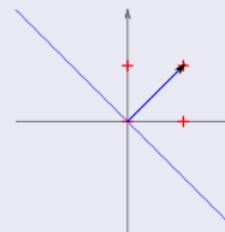
Sortie du neurone : $y = \begin{cases} 1 & \text{si } \sum_{i=0}^n w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$

Apprentissage : $\Delta W = \eta X(t - y)$



Un exemple : le OU

Entrée X			Poids W			Sortie		Sortie voulue	
x_0	x_1	x_2	w_0	w_1	w_2	y	t	0	
1	0	0	0	1	1	0	1	0	

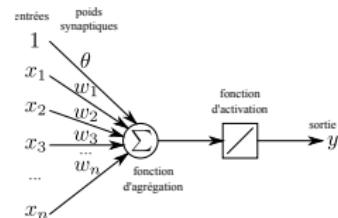


Question

Que se passe-t-il en cas de bruit / point aberrant (i.e. les données ne sont pas linéairement séparables) ?

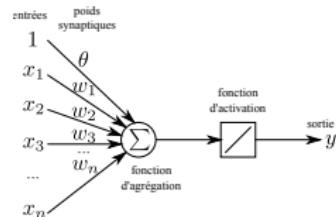
Perceptron - Descente de gradient stochastique

Sortie du neurone : $y = \sum_{i=0}^n w_i x_i$



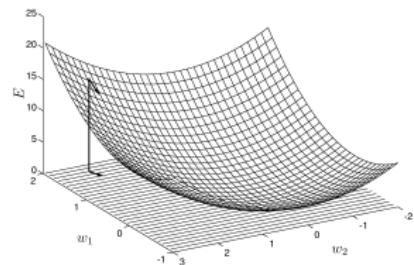
Perceptron - Descente de gradient stochastique

Sortie du neurone : $y = \sum_{i=0}^n w_i x_i$



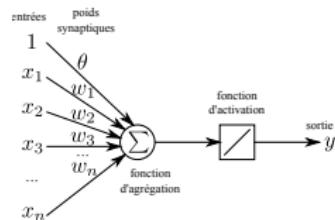
Apprentissage : On veut minimiser un coût, ici

l'erreur quadratique $E = \frac{1}{2} \sum_x (t - y)^2$



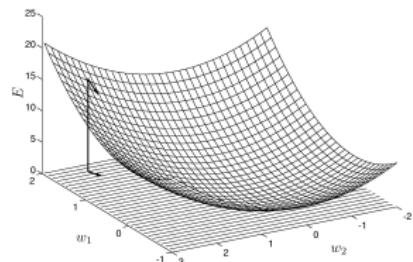
Perceptron - Descente de gradient stochastique

Sortie du neurone : $y = \sum_{i=0}^n w_i x_i$



Apprentissage : On veut minimiser un coût, ici

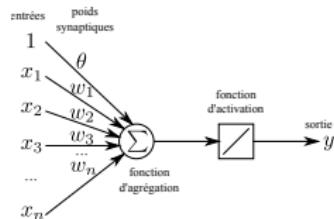
l'erreur quadratique $E = \frac{1}{2} \sum_x (t - y)^2$



$$\begin{aligned}\Delta w_i &= -\eta \frac{\partial E}{\partial w_i} \\&= -\frac{\eta}{2} \sum_x \frac{\partial (t - y)^2}{\partial w_i} \\&= -\frac{\eta}{2} \sum_x -2 \frac{\partial y}{\partial w_i} (t - y) \\&= \sum_x \eta x_i (t - y)\end{aligned}$$

Perceptron - Descente de gradient stochastique

Sortie du neurone : $y = \sum_{i=0}^n w_i x_i$

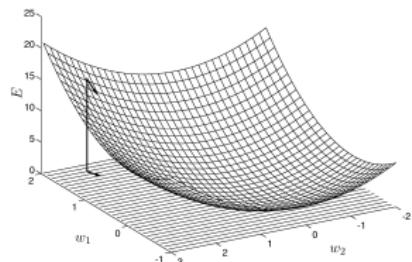


Apprentissage : On veut minimiser un coût, ici

l'erreur quadratique $E = \frac{1}{2} \sum_x (t - y)^2$ par

descente de gradient stochastique :

$$\Delta w = \sum_x \eta x(t - y)$$

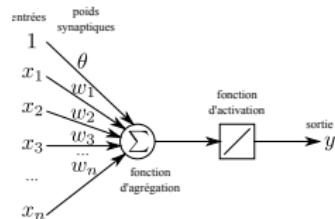


Propriétés

- Classifieur linéaire
- Convergence garantie si η est faible (même si les données ne sont pas linéairement séparables)
- Convergence efficace (car la fonction à optimiser est quadratique)
- Convergence souvent plus rapide en mode en ligne mais pas garantie comme en mode *batch*

Perceptron - Descente de gradient stochastique

Sortie du neurone : $y = \sum_{i=0}^n w_i x_i$

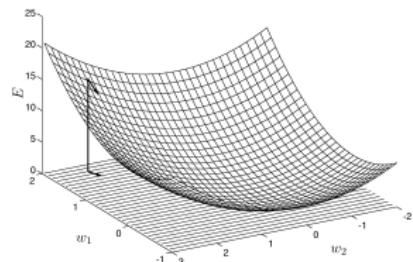


Apprentissage : On veut minimiser un coût, ici

l'erreur quadratique $E = \frac{1}{2} \sum_x (t - y)^2$ par

descente de gradient stochastique :

$$\Delta w = \sum_x \eta x(t - y)$$

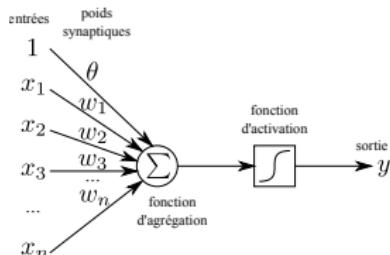


Question

Comment apprendre des fonctions non linéaires ?

Perceptron multi-couches - Werbos & Rumelhard (1984-1986)

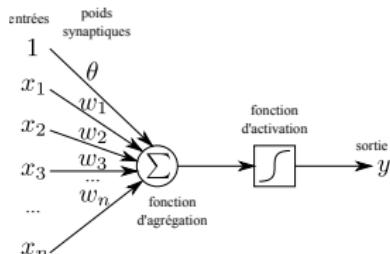
$$\begin{aligned}\text{Sortie du neurone : } s &= \sum_{i=0}^n w_i x_i \\ y &= \frac{1}{1 + e^{-s}}\end{aligned}$$



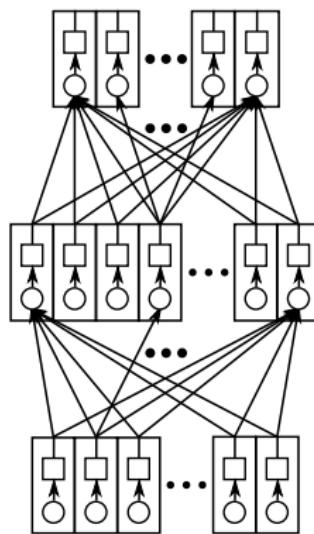
Perceptron multi-couches - Werbos & Rumelhard (1984-1986)

Sortie du neurone :

$$s = \sum_{i=0}^n w_i x_i$$
$$y = \frac{1}{1 + e^{-s}}$$



Réseau : connectivité complète entre la couche d'entrée, la(les) couche(s) cachée(s) et la couche de sortie



Perceptron multi-couches - Werbos & Rumelhard (1984-1986)

Apprentissage : On veut minimiser l'erreur

quadratique $E = \frac{1}{2} \sum_{x,i} (t - y)^2$ par descente de gradient stochastique.

Pour chaque neurone :

$$s = \sum_{i=0}^n w_i x_i$$

$$y = \frac{1}{1 + e^{-s}}$$

Perceptron multi-couches - Werbos & Rumelhard (1984-1986)

Apprentissage : On veut minimiser l'erreur

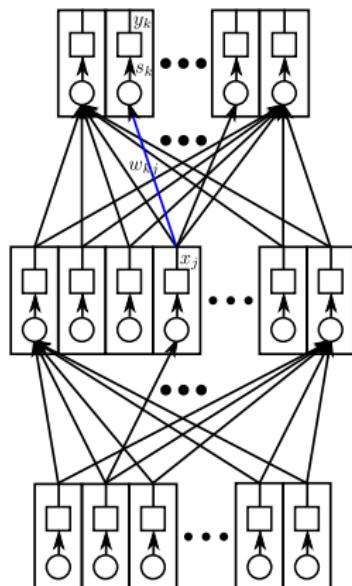
quadratique $E = \frac{1}{2} \sum_{x,i} (t - y)^2$ par descente de gradient stochastique.

Pour chaque neurone :

$$s = \sum_{i=0}^n w_i x_i$$

$$y = \frac{1}{1 + e^{-s}}$$

$$\begin{aligned}\Delta w_{kj} &= -\eta \frac{\partial E}{\partial w_{kj}} \\ &= -\eta \frac{\partial E}{\partial s_k} \frac{\partial s_k}{\partial w_{kj}} \\ &= \eta \delta_k x_j\end{aligned}$$



Apprentissage : On veut minimiser l'erreur

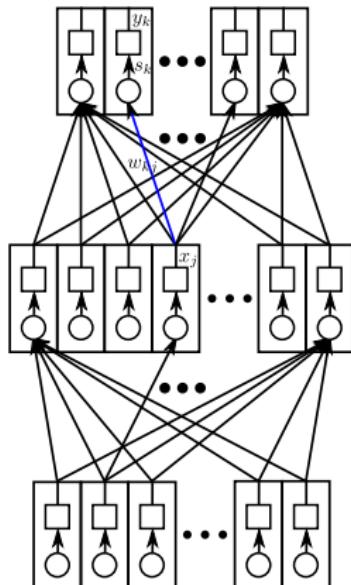
quadratique $E = \frac{1}{2} \sum_{x,i} (t - y)^2$ par descente de gradient stochastique.

$$\begin{aligned}
 \Delta w_{kj} &= -\eta \frac{\partial E}{\partial w_{kj}} \\
 &= -\eta \frac{\partial E}{\partial s_k} \frac{\partial s_k}{\partial w_{kj}} \\
 &= \eta \delta_k x_j \\
 \delta_k &= -\frac{\partial E}{\partial s_k} \\
 &= \sum_x \frac{\partial y_k}{\partial s_k} (t_k - y_k) \\
 &= \sum_x y_k (1 - y_k) (t_k - y_k) \\
 &\quad (\text{ou plus généralement}) \\
 &= \sum_x (t_k - y_k)
 \end{aligned}$$

Pour chaque neurone :

$$s = \sum_{i=0}^n w_i x_i$$

$$y = \frac{1}{1 + e^{-s}}$$



Perceptron multi-couches - Werbos & Rumelhard (1984-1986)

Apprentissage : On veut minimiser l'erreur

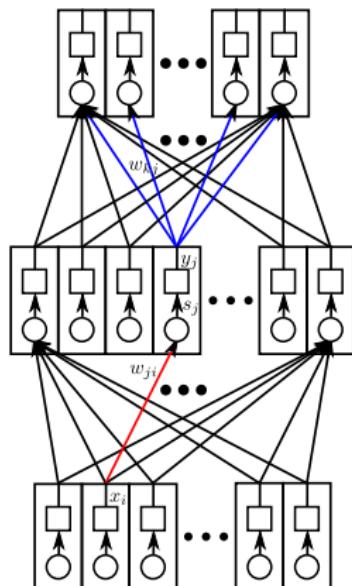
quadratique $E = \frac{1}{2} \sum_{x,i} (t - y)^2$ par descente de gradient stochastique.

Pour chaque neurone :

$$s = \sum_{i=0}^n w_i x_i$$

$$y = \frac{1}{1 + e^{-s}}$$

$$\begin{aligned}\Delta w_{ji} &= -\eta \frac{\partial E}{\partial w_{ji}} \\ &= -\eta \frac{\partial E}{\partial s_j} \frac{\partial s_j}{\partial w_{ji}} \\ &= \eta \delta_j x_i\end{aligned}$$



Apprentissage : On veut minimiser l'erreur

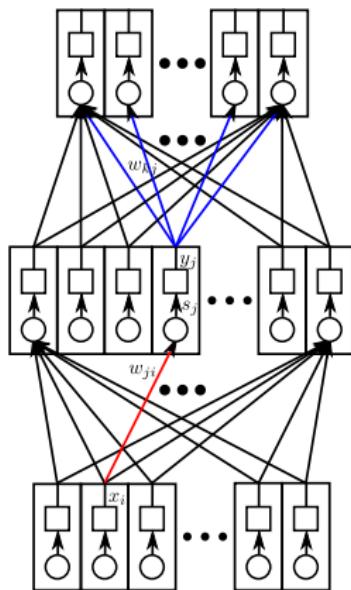
quadratique $E = \frac{1}{2} \sum_{x,i} (t - y)^2$ par descente de gradient stochastique.

$$\begin{aligned}
\Delta \textcolor{red}{w}_{ji} &= -\eta \frac{\partial E}{\partial w_{ji}} \\
&= -\eta \frac{\partial E}{\partial s_j} \frac{\partial s_j}{\partial w_{ji}} \\
&= \eta \delta_j x_i \\
\delta_j &= -\frac{\partial E}{\partial s_j} \\
&= -\sum_k \frac{\partial E}{\partial s_k} \frac{\partial s_k}{\partial y_j} \frac{\partial y_j}{\partial s_j} \\
&= \sum_k \delta_k \textcolor{blue}{w}_{kj} y_j (1 - y_j) \\
&= y_j (1 - y_j) \sum_k \delta_k w_{jk}
\end{aligned}$$

Pour chaque neurone :

$$s = \sum_{i=0}^n w_i x_i$$

$$y = \frac{1}{1 + e^{-s}}$$



Apprentissage : On veut minimiser l'erreur

quadratique $E = \frac{1}{2} \sum_{x,i} (t - y)^2$ par descente de gradient stochastique.

Pour chaque entrée reçue :

- ➊ Calculer la sortie y du réseau par propagation (couche par couche) de l'activité

- ➋ Calculer l'erreur de la couche de sortie :

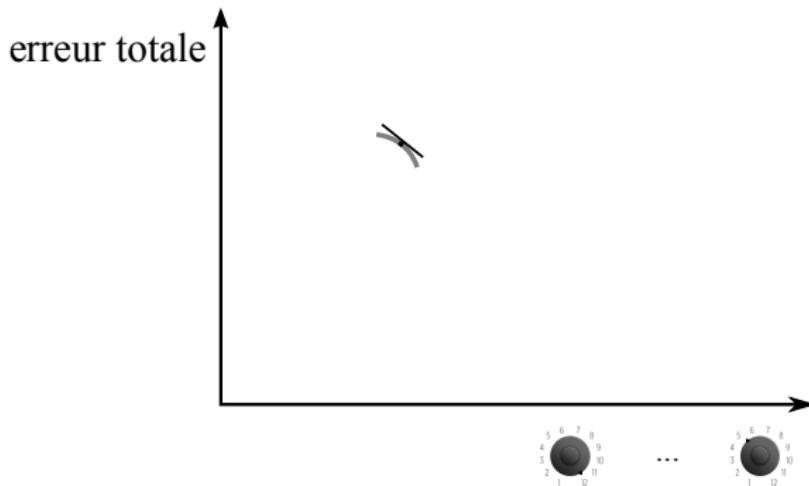
$$\delta_k = y_k(1 - y_k)(t_k - y_k)$$

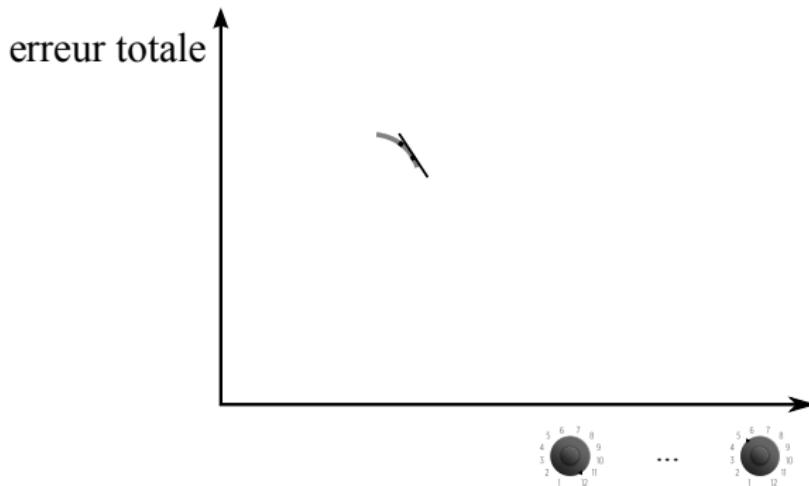
- ➌ Rétropager l'erreur à travers chaque couche

$$j \text{ du réseau } \delta_j = y_j(1 - y_j) \sum_k \delta_k w_{jk}$$

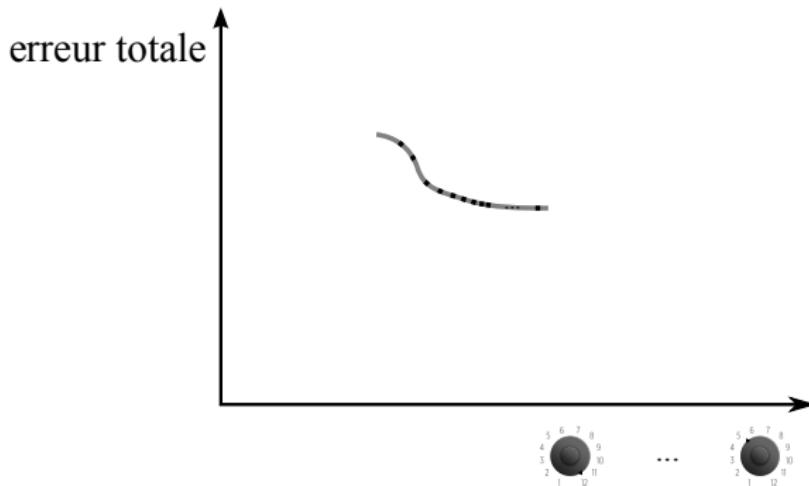
- ➍ Modifier chaque poids $\Delta w_{ij} = \sum_{x_i} \eta \delta_j x_i$

Perceptron multi-couches - Werbos & Rumelhard (1984-1986)

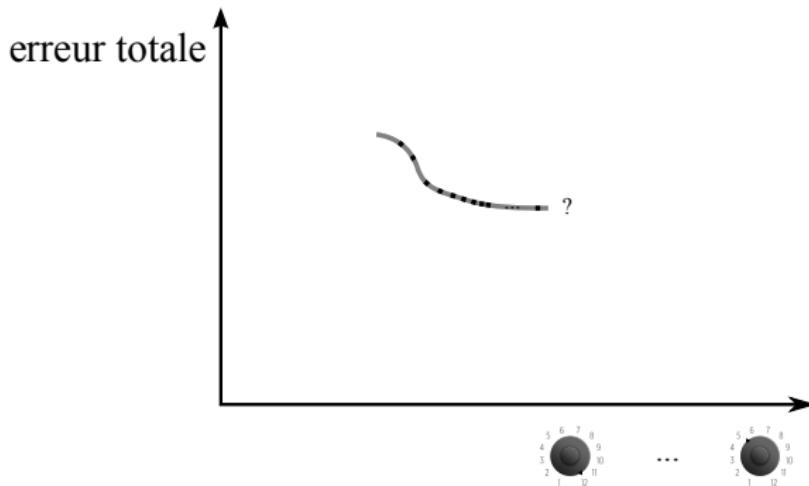




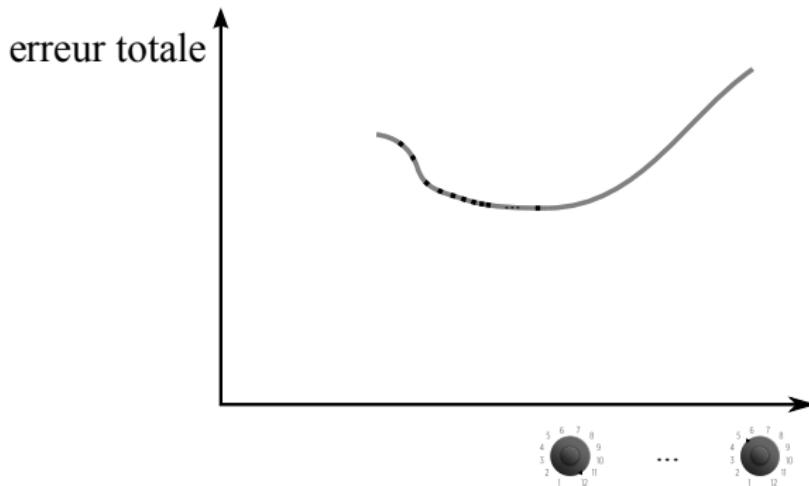
Perceptron multi-couches - Werbos & Rumelhard (1984-1986)



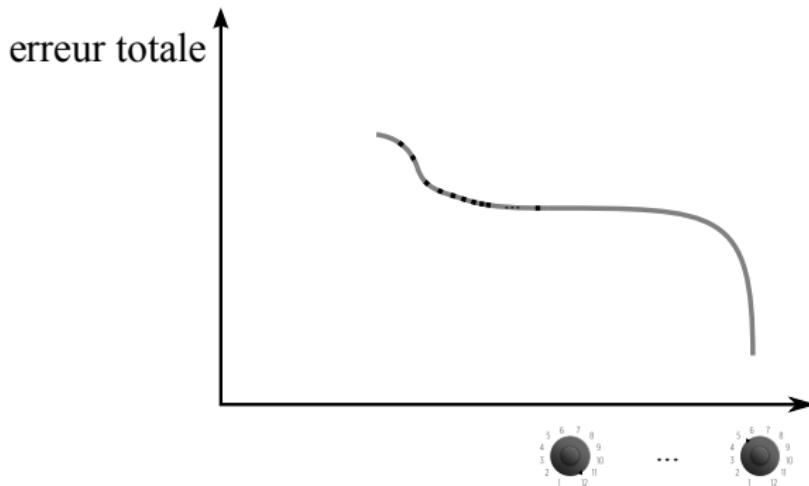
Perceptron multi-couches - Werbos & Rumelhard (1984-1986)



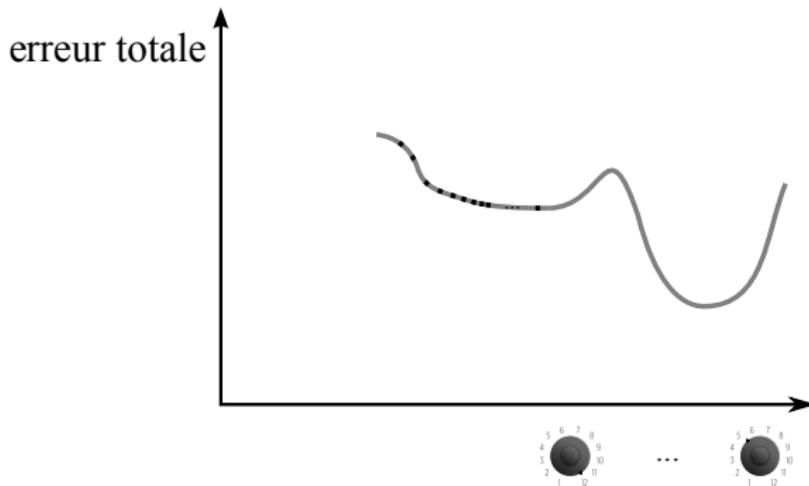
Perceptron multi-couches - Werbos & Rumelhard (1984-1986)



Perceptron multi-couches - Werbos & Rumelhard (1984-1986)



Perceptron multi-couches - Werbos & Rumelhard (1984-1986)



Propriétés

- Approximateur universel (avec fonction d'activation sigmoïde ou gaussienne entre autres)
- Convergence potentiellement peu efficace (non convexe, espace de haute dimension, gradient évanescents ou non borné, ...)

Propriétés

- Approximateur universel (avec fonction d'activation sigmoïde ou gaussienne entre autres)
- Convergence potentiellement peu efficace (non convexe, espace de haute dimension, gradient évanescents ou non borné, ...)

Démo

<http://playground.tensorflow.org/>

Question

Comment on apprend avec des images en entrée ?

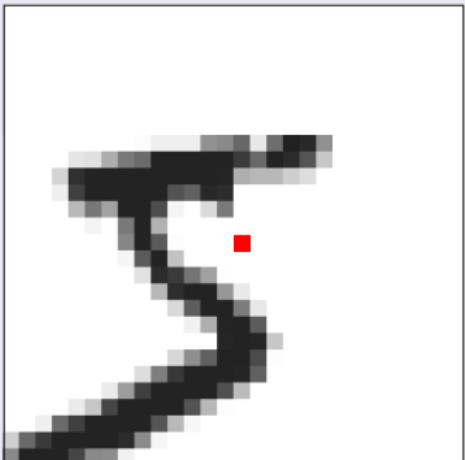
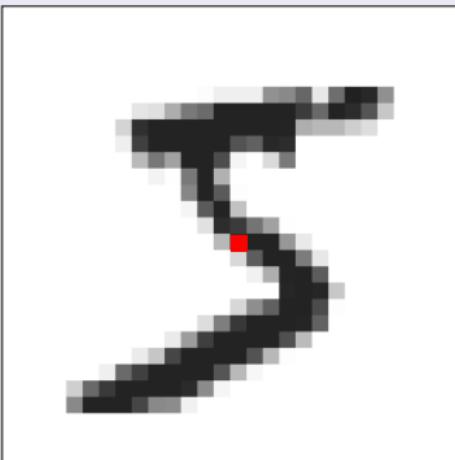
Question

Comment on apprend avec des images en entrée ?

(Mauvaise) solution

On peut utiliser un Perceptron multi-couche mais

- grand nombre de paramètres (une image fait minimum 1000 dimensions)
- problème de la translation

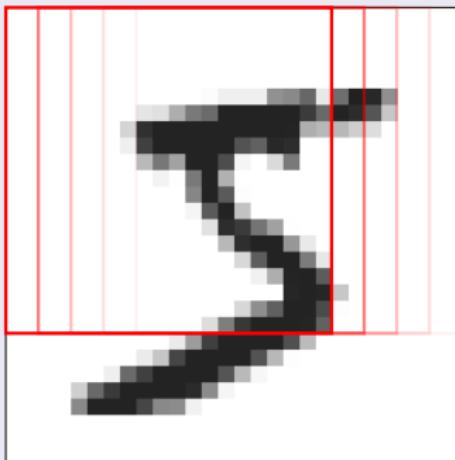


Question

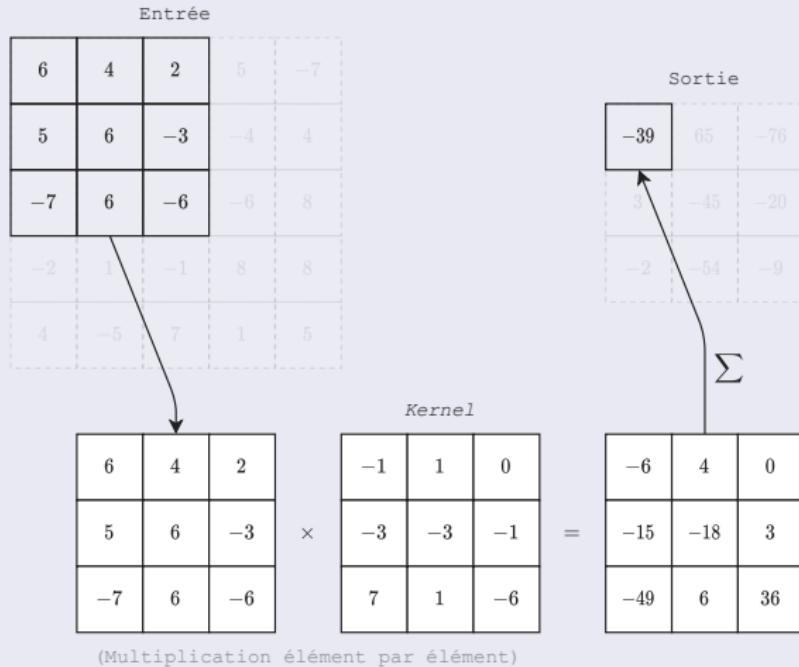
Comment on apprend avec des images en entrée ?

Bonne solution

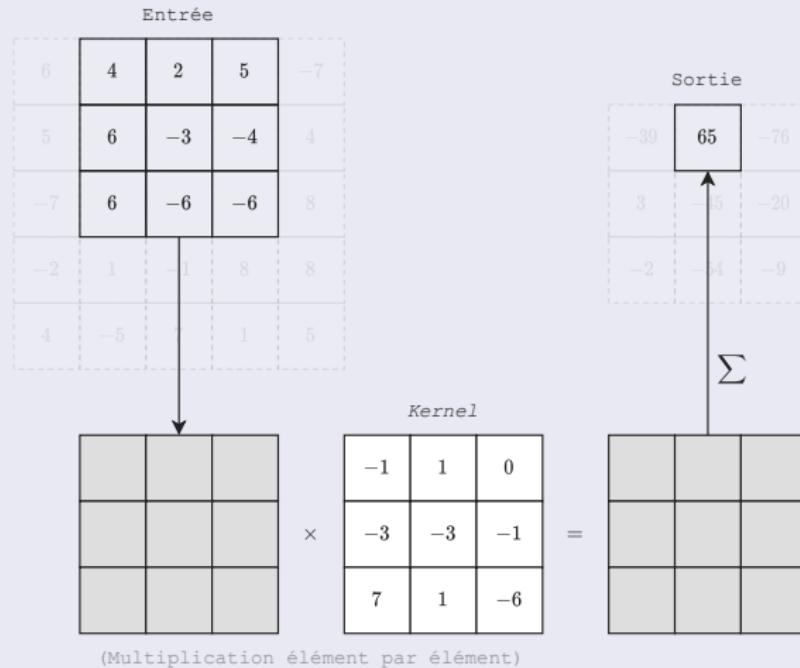
On modifie nos neurones et notre architecture pour faire de la convolution.



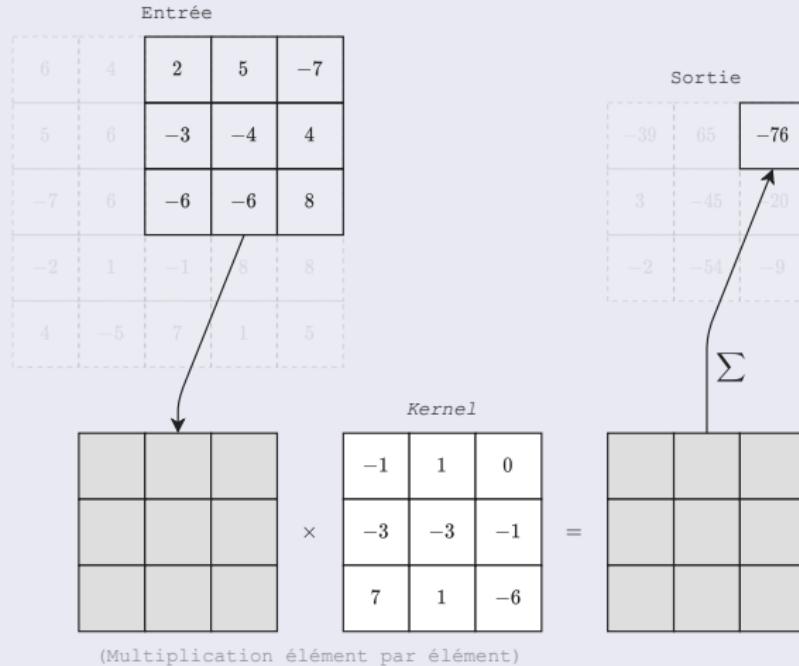
Convolution



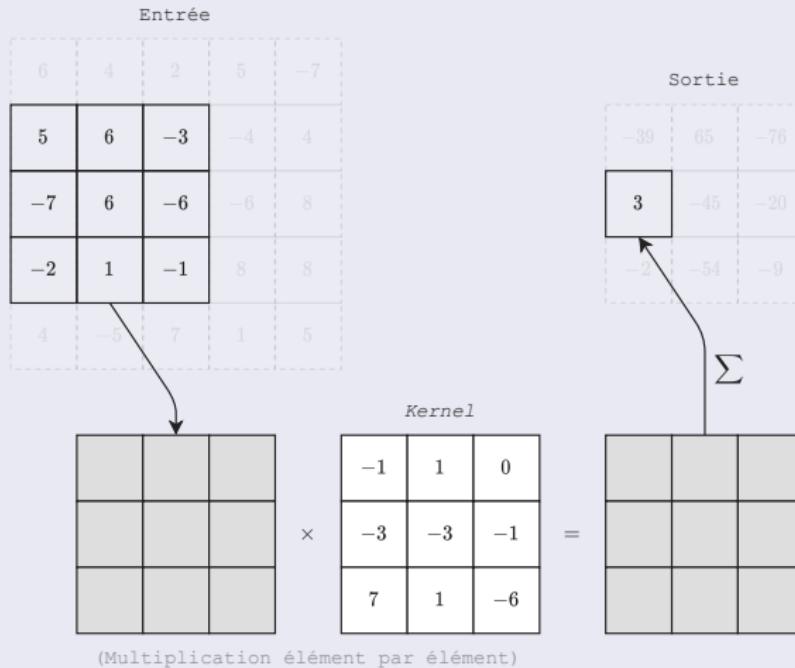
Convolution



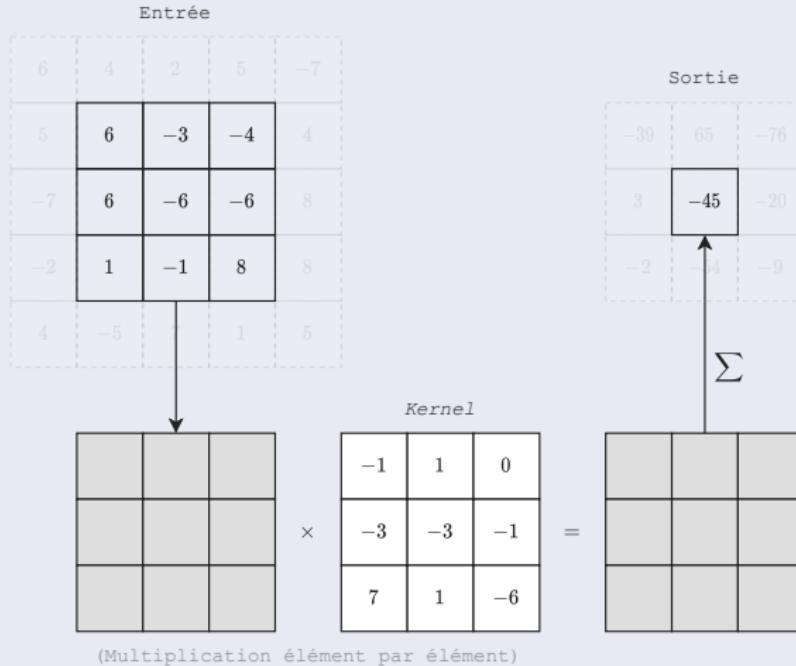
Convolution



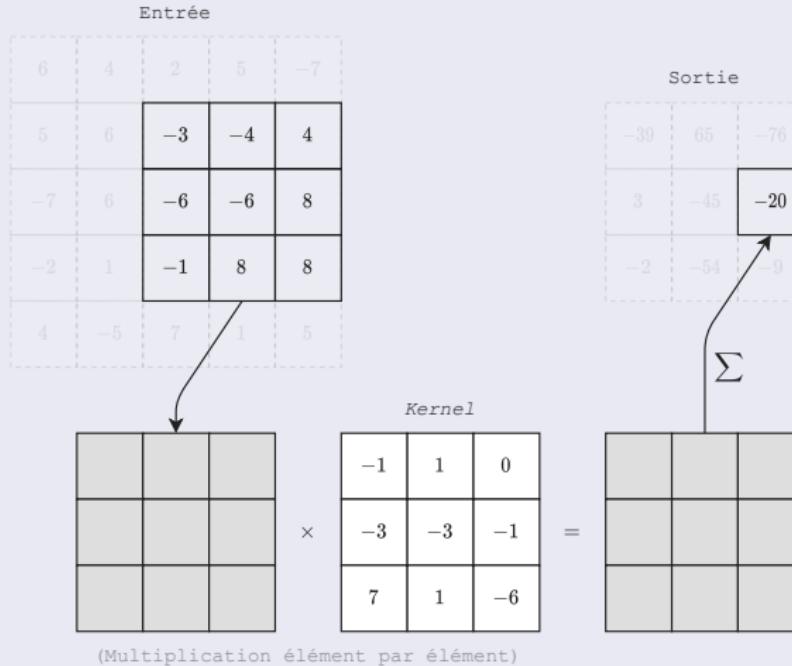
Convolution



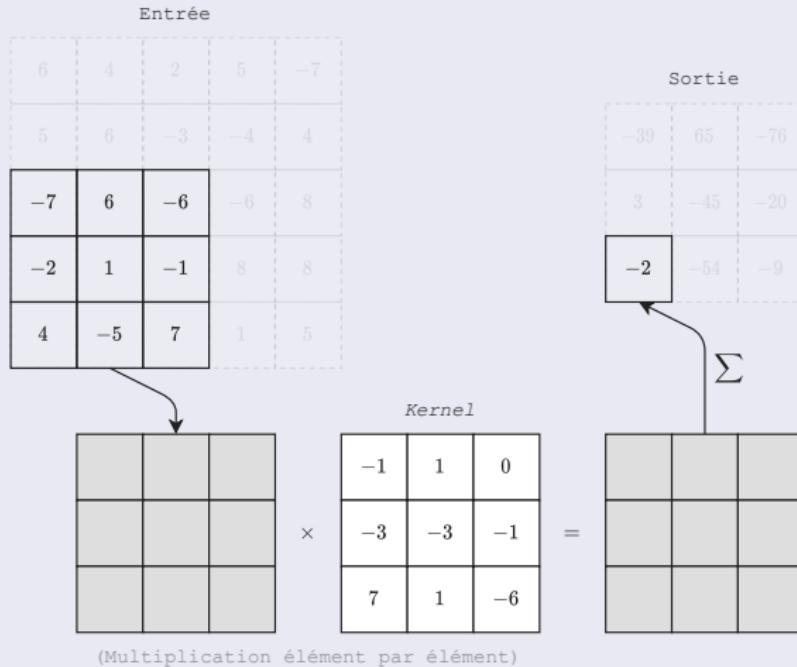
Convolution



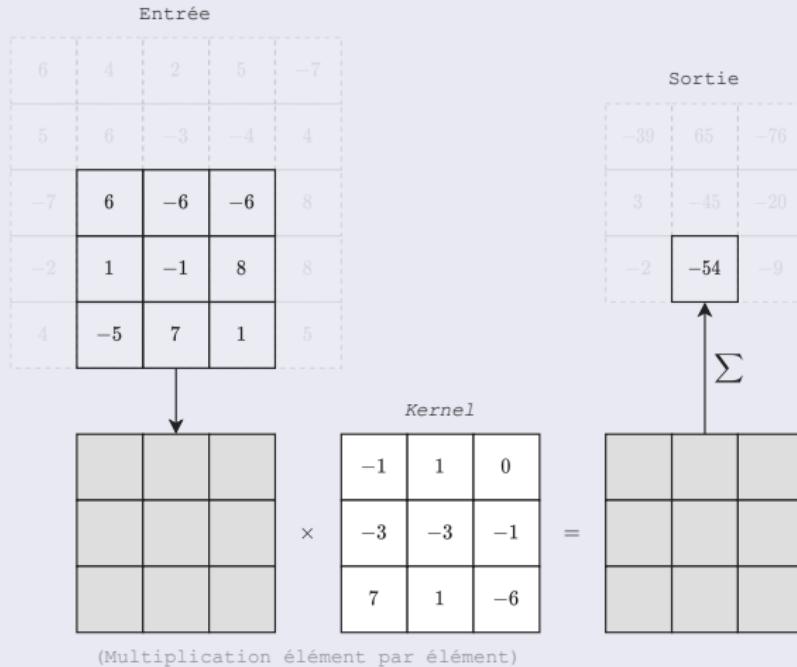
Convolution



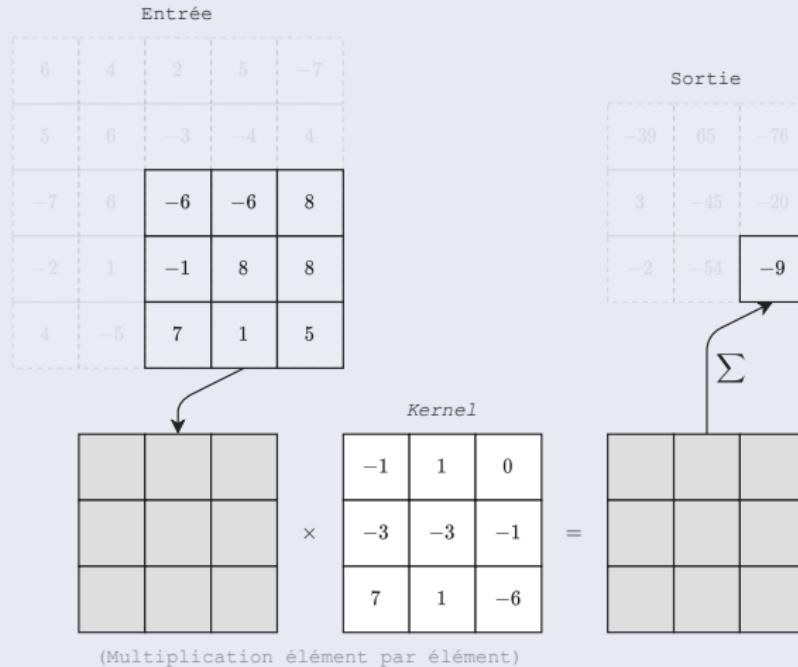
Convolution



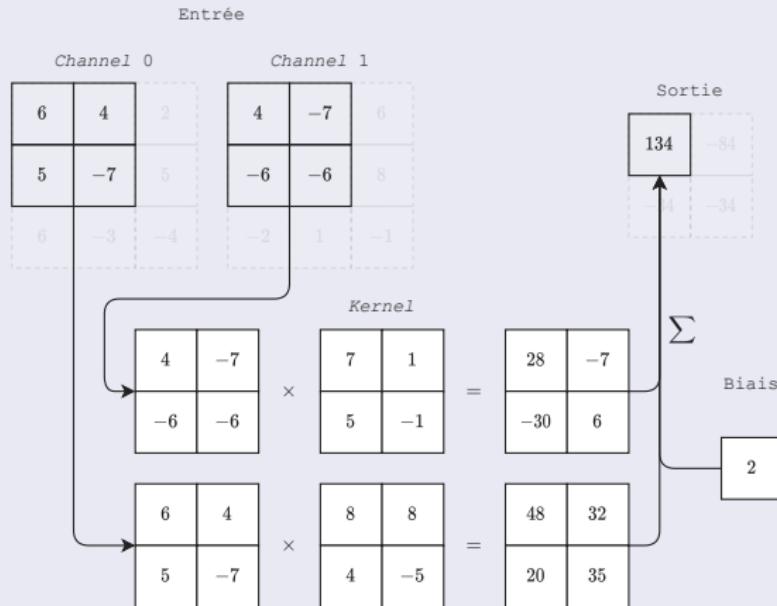
Convolution



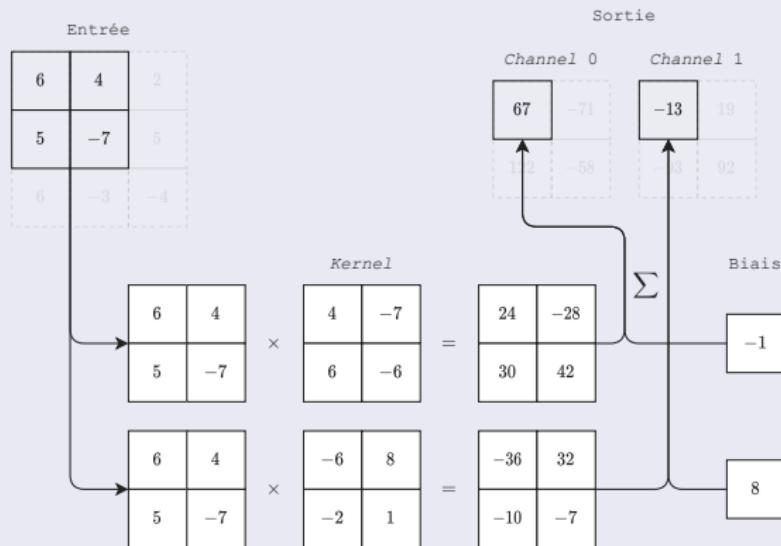
Convolution



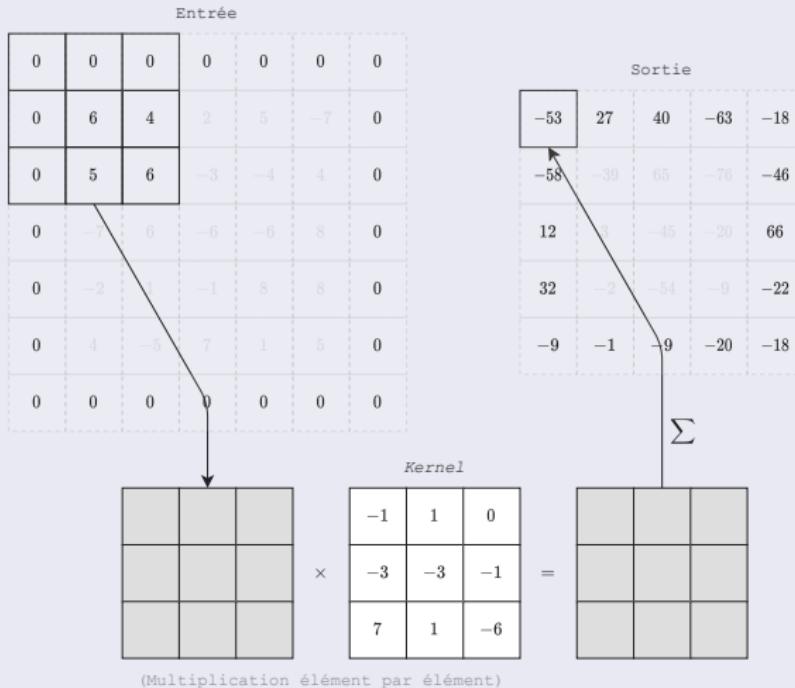
Neurone de convolution



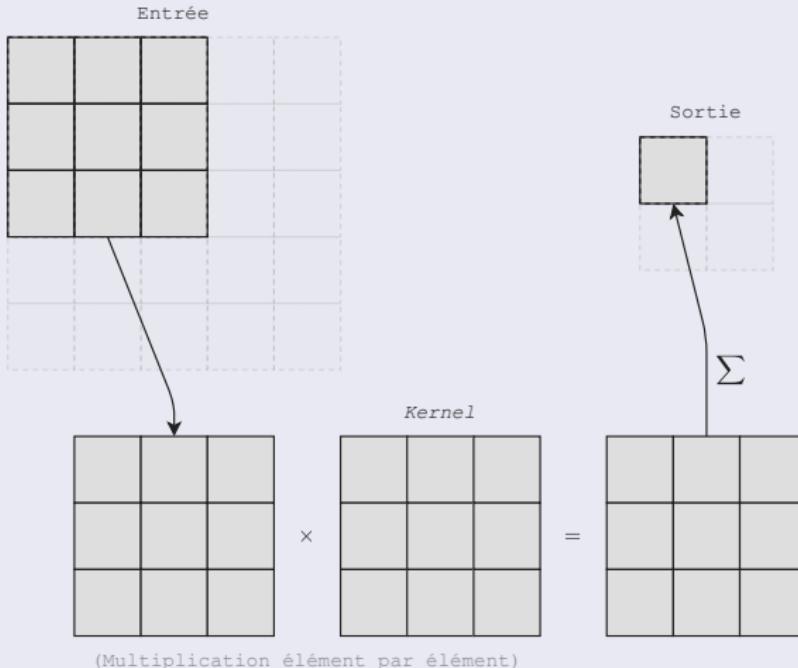
Couche de convolution



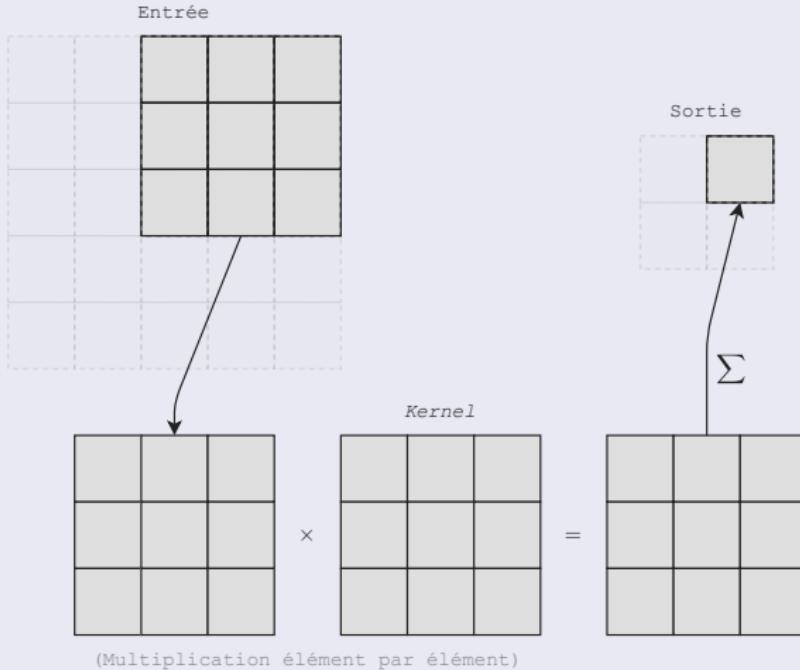
Couche de convolution - *Padding*



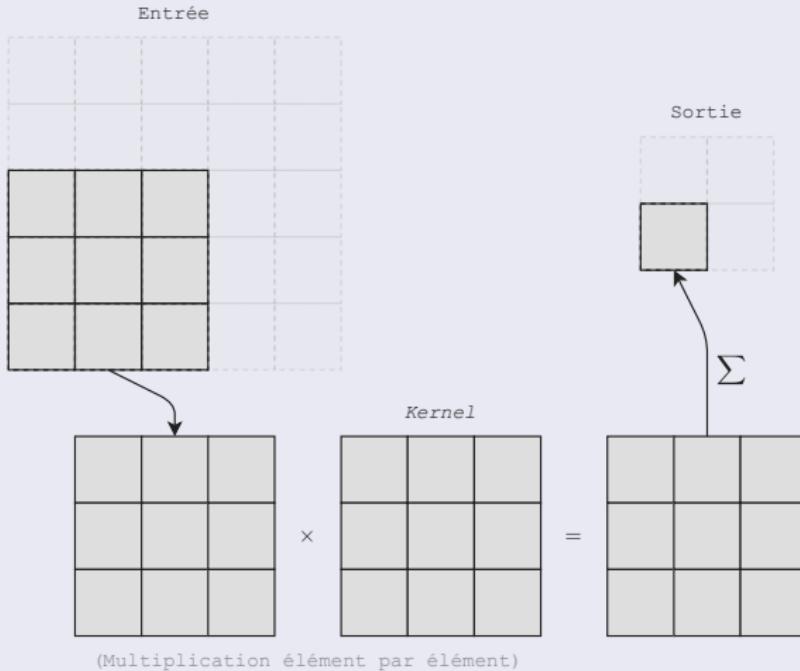
Couche de convolution - *Stride*



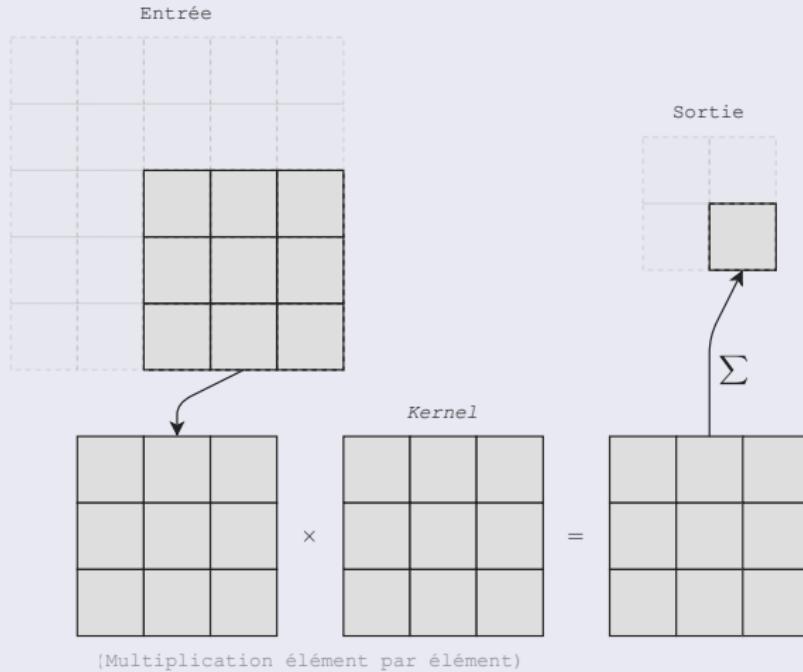
Couche de convolution - *Stride*



Couche de convolution - *Stride*



Couche de convolution - *Stride*



Couche de convolution - Bilan

- Entrée (de la couche et de chaque neurone) : Channel \times Largeur \times Hauteur
$$dim_{in}$$

- Taille de la sortie (d'un neurone) :

Taille effective de l'entrée

$$dim_{out} = \left\lfloor \frac{dim_{in} + 2 \times padding - taille_noyau}{stride} \right\rfloor + 1$$

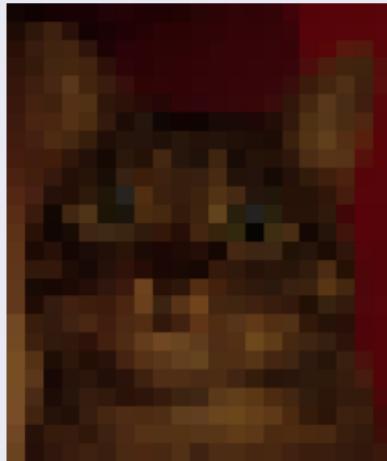
Nombre de déplacements possibles du noyau

Position initiale

- Taille de la sortie (d'une couche) : Nombre neurones $\times dim_{out}$

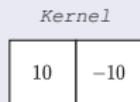
Couche de *Pooling*

Pour la reconnaissance pas besoin de trop grande résolution / dimensionnalité ...

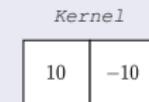


Couche de *Pooling*

Pour la reconnaissance pas besoin de trop grande résolution / dimensionnalité ...
... mais augmenter le *stride* est risqué ...



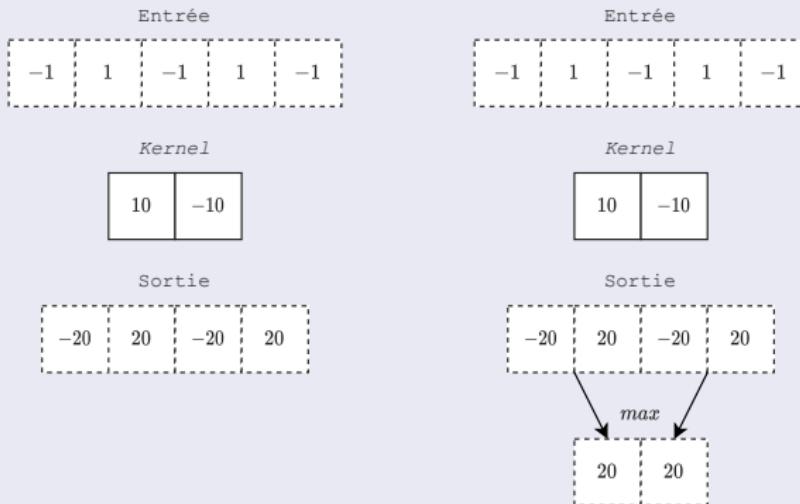
(Stride de 1)



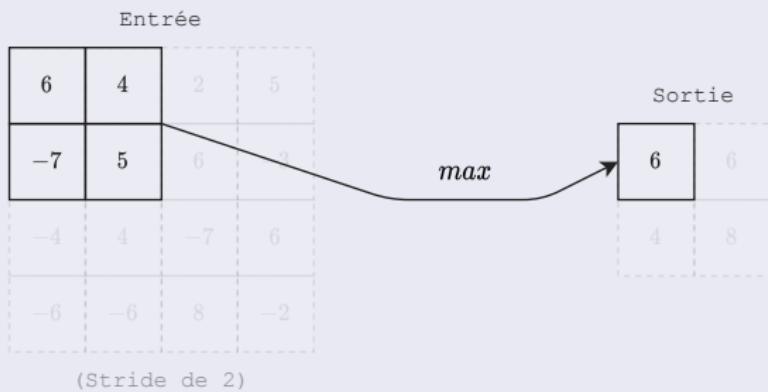
(Stride de 2)

Couche de *Pooling*

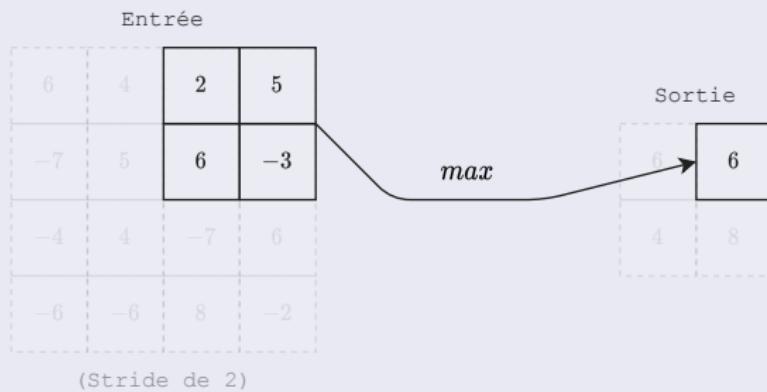
Pour la reconnaissance pas besoin de trop grande résolution / dimensionnalité ...
... mais augmenter le *stride* est risqué ...
... on préfère faire un *pooling* généralement avec un max (parfois une moyenne)



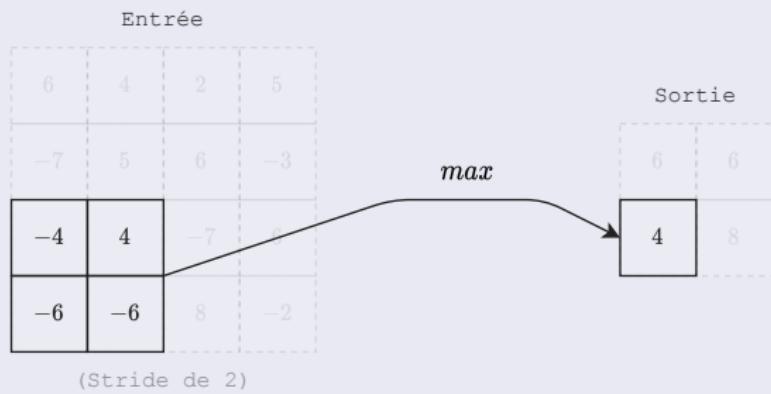
Couche de *Pooling*



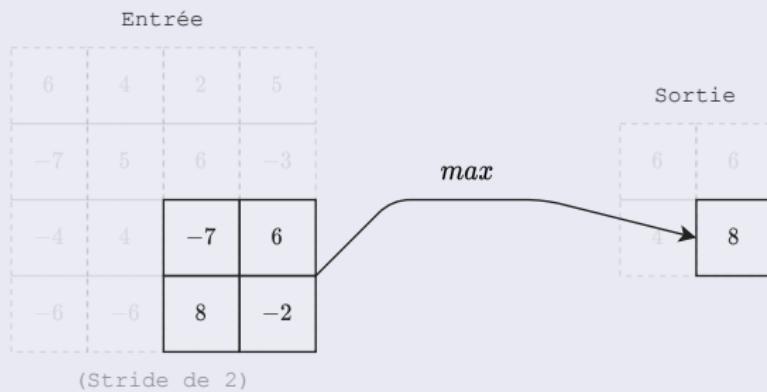
Couche de *Pooling*



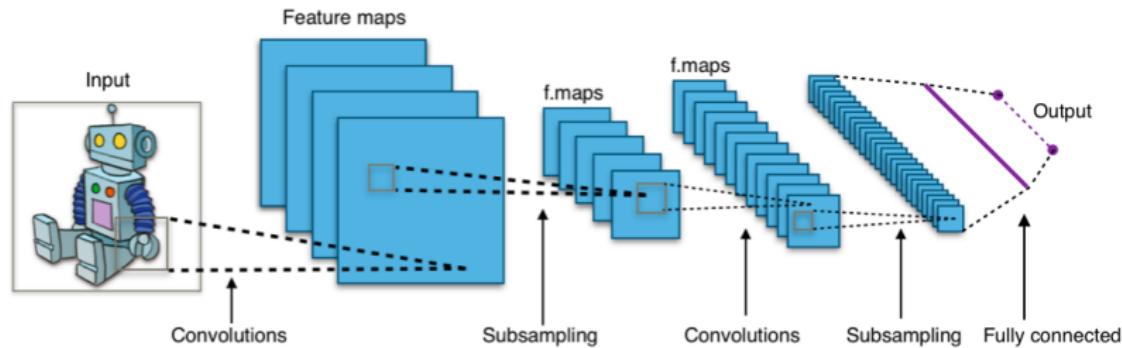
Couche de *Pooling*



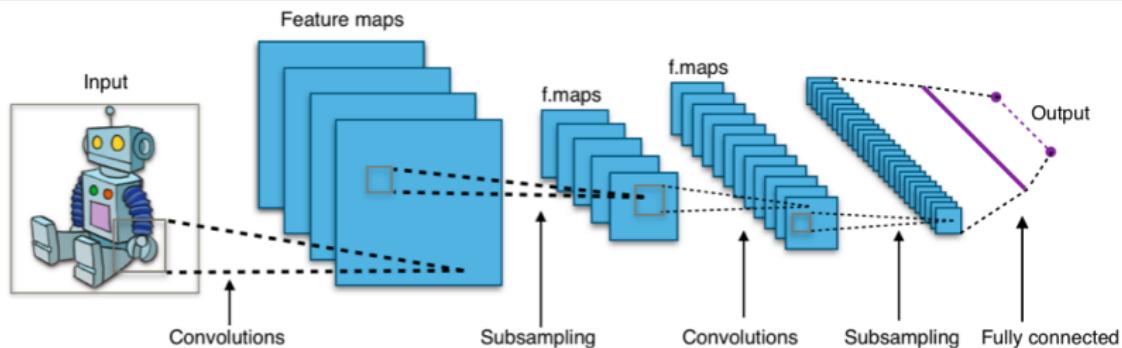
Couche de *Pooling*



Architecture



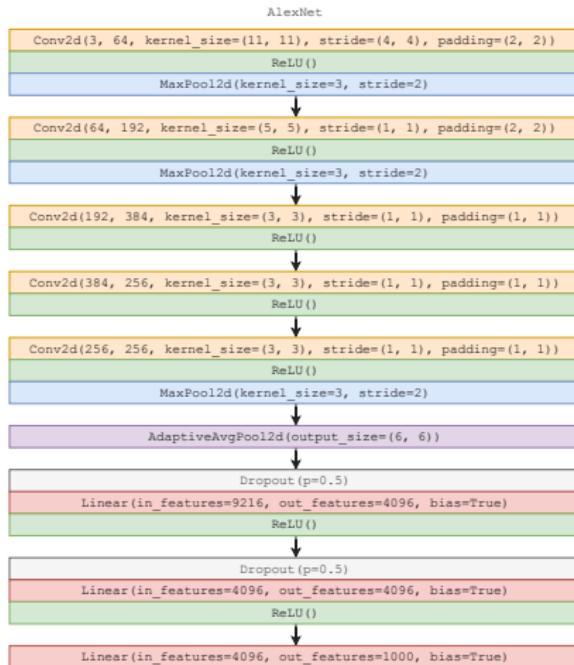
Architecture



Apprentissage

- Chaque neurone fait toujours une somme pondérée, on fait donc comme avant i.e. une descente de gradient sur une fonction de coût.
- Les différences :
 - le (noyau) de poids est partagé entre les neurones d'une même couche de convolution donc le gradient est aggrégé (somme ou moyenne)
 - pour le pooling, dans le cas du *max* on rétropage le gradient d'où vient le max, dans le cas de l'*average* c'est une somme pondérée on fait donc comme pour les neurones

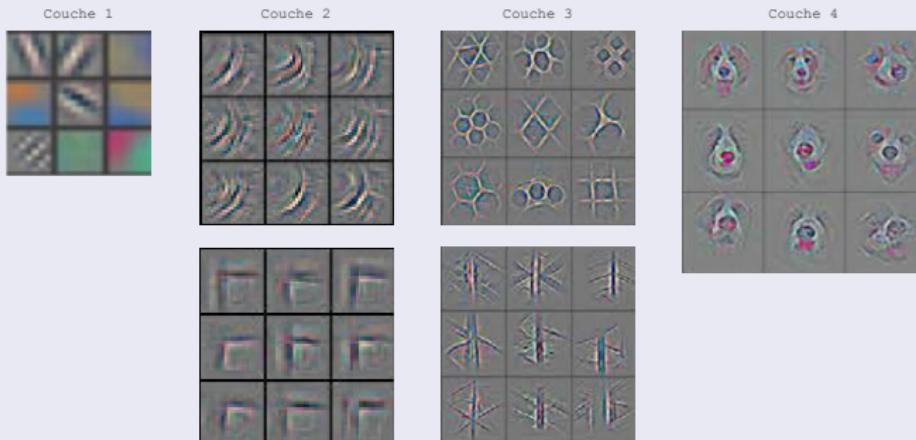
Réseaux convolutifs - AlexNet (2012)



Principe

- De la convolution pour son invariance à la translation et partage des poids
- Du *pooling* pour réduire les dimensionnalités
- Un perceptron à la fin car plus de structure spatiale en haut niveau
- Architecture transposable à la 1D (audio), à la 3D (vidéo), aux graphs, ...

Filtres appris



Si vous voulez creuser : <https://youtu.be/AgkfIQ4IGaM>

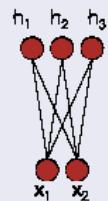
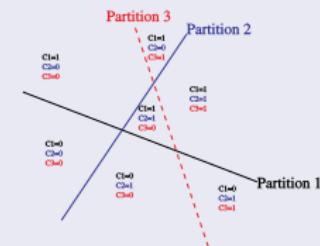
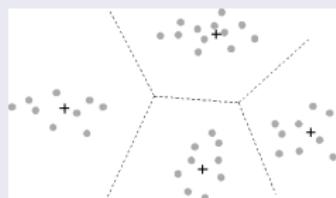
Pourquoi ça marche ?

- Approximateur universel + beaucoup de données et de GPUs ... oui mais pas que
- *Features* (vs prototypes) et leur apprentissage
- Pouvoir (exponentiel) d'abstraction
- Appröhende mieux les données/tâches

Bilan apprentissage profond

Pourquoi ça marche ?

- Approximateur universel + beaucoup de données et de GPUs ... oui mais pas que
- Features (vs prototypes) et leur apprentissage



- Pouvoir (exponentiel) d'abstraction
- Apprécie mieux les données/tâches

Pourquoi ça marche ?

- Approximateur universel + beaucoup de données et de GPUs ... oui mais pas que
- *Features* (vs prototypes) et leur apprentissage
- Pouvoir (exponentiel) d'abstraction
- Appröhende mieux les données/tâches

Bilan apprentissage profond

Pourquoi ça marche ?

- Approximateur universel + beaucoup de données et de GPUs ... oui mais pas que
- *Features* (vs prototypes) et leur apprentissage
- Pouvoir (exponentiel) d'abstraction
- Appréhende mieux les données/tâches

	Architecture	# Param.	# Hidden units	Err.
DNN	2000-2000 + dropout	~10M	4k	57.8%
SNN-30k	128c-p-1200L-30k + dropout input&hidden	~70M	~190k	21.8%
single-layer feature extraction	4000c-p followed by SVM	~125M	~3.7B	18.4%
CNN[11] (no augmentation)	64c-p-64c-p-64c-p-16c + dropout on lc	~10k	~110k	15.6%
CNN[21] (no augmentation)	64c-p-64c-p-128c-p-fc + dropout on fc and stochastic pooling	~56k	~120k	15.13%
teacher CNN (no augmentation)	128c-p-128c-p-128c-p-1kf + dropout on fc and stochastic pooling	~35k	~210k	12.0%
ECNN (no augmentation)	ensemble of 4 CNNs	~140k	~840k	11.0%
SNN-CNN-MIMIC-30k trained on a single CNN	64c-p-1200L-30k with no regularization	~54M	~110k	15.4%
SNN-CNN-MIMIC-30k trained on a single CNN	128c-p-1200L-30k with no regularization	~70M	~190k	15.1%
SNN-ECNN-MIMIC-30k trained on ensemble	128c-p-1200L-30k with no regularization	~70M	~190k	14.2%

Table 2: Comparison of shallow and deep models: classification error rate on CIFAR-10. Key: c, convolution layer; p, pooling layer; lc, locally connected layer; fc, fully connected layer

Ba, J., & Caruana, R. (2014). Do deep nets really need to be deep ?. In NIPS.

- Input Cell
- Backfed Input Cell
- △ Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- △ Spiking Hidden Cell
- Capsule Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- △ Gated Memory Cell
- Kernel
- Convolution or Pool

A mostly complete chart of Neural Networks

©2019 Fjodor van Veen & Stefan Leijnen asimovinstitute.org

Perceptron (P)



Feed Forward (FF)



Radial Basis Network (RBF)



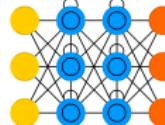
Deep Feed Forward (DFF)



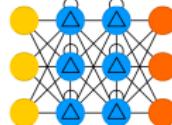
Recurrent Neural Network (RNN)



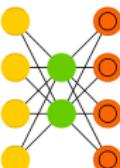
Long / Short Term Memory (LSTM)



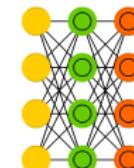
Gated Recurrent Unit (GRU)



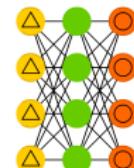
Auto Encoder (AE)



Variational AE (VAE)



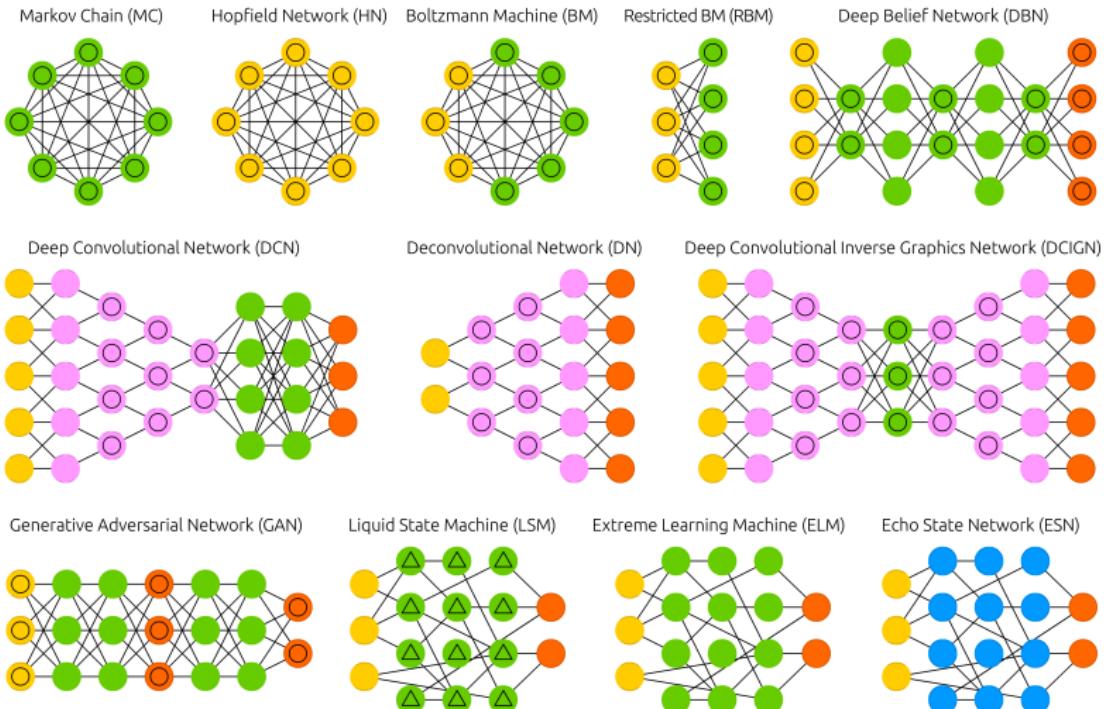
Denoising AE (DAE)



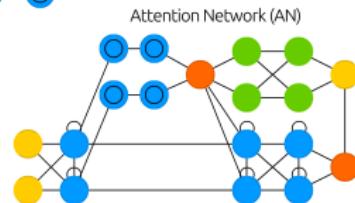
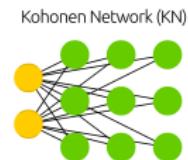
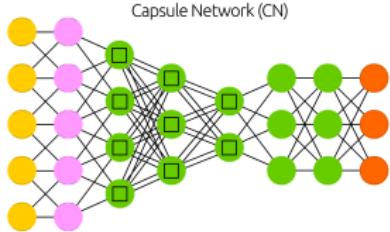
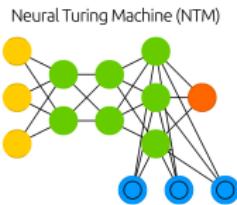
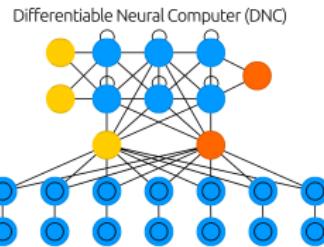
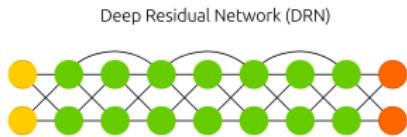
Sparse AE (SAE)



Le bestiaire de l'apprentissage profond



Le bestiaire de l'apprentissage profond



Et en pratique ?

- Traitement des données
- Choix du modèle
- Apprentissage
- Obtenir les meilleures performances

Et en pratique ?

- Traitement des données
 - Équilibrage/représentativité des données
 - Augmentation de données : pour les images changement de couleur, de zoom, d'orientation, ...
 - Données centrées normées : $\frac{x - \bar{x}}{\sigma(x)}$
 - Utilisation de mini batchs
- Choix du modèle
- Apprentissage
- Obtenir les meilleures performances

Et en pratique ?

- Traitement des données
 - Équilibrage/représentativité des données
 - Augmentation de données : pour les images changement de couleur, de zoom, d'orientation, ...
 - Données centrées normées : $\frac{x - \bar{x}}{\sigma(x)}$
 - Utilisation de mini batchs
- Choix du modèle
- Apprentissage
- Obtenir les meilleures performances

Et en pratique ?

- Traitement des données
 - Équilibrage/représentativité des données
 - Augmentation de données : pour les images changement de couleur, de zoom, d'orientation, ...
 - Données centrées normées : $\frac{x - \bar{x}}{\sigma(x)}$
 - Utilisation de mini batchs
- Choix du modèle
- Apprentissage
- Obtenir les meilleures performances

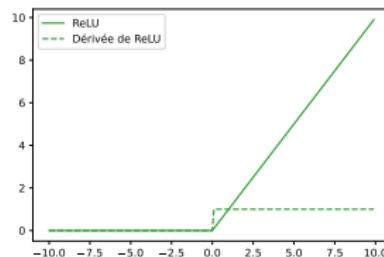
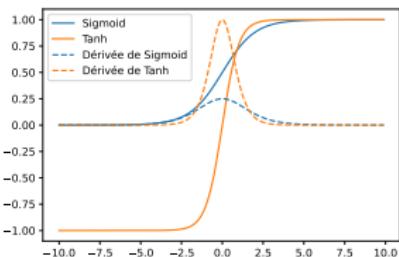
- Traitement des données
 - Équilibrage/représentativité des données
 - Augmentation de données : pour les images changement de couleur, de zoom, d'orientation, ...
 - Données centrées normées : $\frac{x - \bar{x}}{\sigma(x)}$
 - Utilisation de mini batchs
- Choix du modèle
- Apprentissage
- Obtenir les meilleures performances

Et en pratique ?

- Traitement des données
- Choix du modèle
 - Type d'architecture suivant les données / le problème
 - Choix de la fonction d'activation
- Apprentissage
- Obtenir les meilleures performances

Et en pratique ?

- Traitement des données
- Choix du modèle
 - Type d'architecture suivant les données / le problème
 - Choix de la fonction d'activation



- Apprentissage
- Obtenir les meilleures performances

Et en pratique ?

- Traitement des données

- Choix du modèle

- Apprentissage

- Fonction de coût :

- régression : erreur quadratique : $\frac{1}{2} \sum_{x,i} (t_i - y_i)^2$

- classification : entropie croisée : $\sum_x -\log \frac{e^{y_t}}{\sum_i e^{y_i}}$

- Régularisation : $+||w||$ dans la fonction de coût ou *drop out* (mise à 0 de certaines valeurs)

- Choix de l'optimiseur

- SGD : la base

- momentum : rajout d'inertie dans le gradient

- Adagrad : adaptation du taux d'apprentissage (en fonction du gradient)

- Adam : combinaison des 2 points précédents

- Obtenir les meilleures performances

Et en pratique ?

- Traitement des données

- Choix du modèle

- Apprentissage

- Fonction de coût :

- régression : erreur quadratique : $\frac{1}{2} \sum_{x,i} (t_i - y_i)^2$

- classification : entropie croisée : $\sum_x -\log \frac{e^{y_t}}{\sum_i e^{y_i}}$

- Régularisation : $+||w||$ dans la fonction de coût ou *drop out* (mise à 0 de certaines valeurs)

- Choix de l'optimiseur

- SGD : la base

- momentum : rajout d'inertie dans le gradient

- Adagrad : adaptation du taux d'apprentissage (en fonction du gradient)

- Adam : combinaison des 2 points précédents

- Obtenir les meilleures performances

Et en pratique ?

- Traitement des données
- Choix du modèle
- Apprentissage
 - Fonction de coût :
 - régression : erreur quadratique : $\frac{1}{2} \sum_{x,i} (t_i - y_i)^2$
 - classification : entropie croisée : $\sum_x -\log \frac{e^{y_t}}{\sum_i e^{y_i}}$
 - Régularisation : $+||w||$ dans la fonction de coût ou *drop out* (mise à 0 de certaines valeurs)
 - Choix de l'optimiseur
 - SGD : la base
 - momentum : rajout d'inertie dans le gradient
 - Adagrad : adaptation du taux d'apprentissage (en fonction du gradient)
 - Adam : combinaison des 2 points précédents
- Obtenir les meilleures performances

Et en pratique ?

- Traitement des données
- Choix du modèle
- Apprentissage
- Obtenir les meilleures performances
 - Hyperparamétrage : on part d'un réglage "de base" ...
 - pour le MLP, classiquement une grande 1^{ère} couche, puis taille décroissante
 - pour le CNN, classiquement augmentation du nombre de canaux à chaque couche (qui "compense" la réduction de la taille de la carte de *features*)
 - ... et ensuite on cherche empiriquement (typiquement *grid search*) ce qui marche sur le jeu de validation (attention au sur-apprentissage, considérer la sur-paramétrisation)
 - *Fine tuning* : on part d'un modèle ("générique") pré appris et on l'adapte ensuite à nos données

Et en pratique ?

- Traitement des données
- Choix du modèle
- Apprentissage
- Obtenir les meilleures performances
 - Hyperparamétrage : on part d'un réglage "de base" ...
 - pour le MLP, classiquement une grande 1^{ère} couche, puis taille décroissante
 - pour le CNN, classiquement augmentation du nombre de canaux à chaque couche (qui "compense" la réduction de la taille de la carte de *features*)
 - ... et ensuite on cherche empiriquement (typiquement *grid search*) ce qui marche sur le jeu de validation (attention au sur-apprentissage, considérer la sur-paramétrisation)
 - *Fine tuning* : on part d'un modèle ("générique") pré appris et on l'adapte ensuite à nos données

Et en pratique ?

- Traitement des données
- Choix du modèle
- Apprentissage
- Obtenir les meilleures performances
 - Hyperparamétrage : on part d'un réglage "de base" ...
 - pour le MLP, classiquement une grande 1^{ère} couche, puis taille décroissante
 - pour le CNN, classiquement augmentation du nombre de canaux à chaque couche (qui "compense" la réduction de la taille de la carte de *features*)
 - ... et ensuite on cherche empiriquement (typiquement *grid search*) ce qui marche sur le jeu de validation (attention au sur-apprentissage, considérer la sur-paramétrisation)
 - *Fine tuning* : on part d'un modèle ("générique") pré appris et on l'adapte ensuite à nos données

Limitations - Surapprentissage ?

data aug	dropout	weight decay	top-1 train	top-5 train	top-1 test	top-5 test
ImageNet 1000 classes with the original labels						
yes	yes	yes	92.18	99.21	77.84	93.92
yes	no	no	92.33	99.17	72.95	90.43
no	no	yes	90.60	100.0	67.18 (72.57)	86.44 (91.31)
no	no	no	99.53	100.0	59.80 (63.16)	80.38 (84.49)
Alexnet (Krizhevsky et al., 2012)			-	-	-	83.6
ImageNet 1000 classes with random labels						
no	yes	yes	91.18	97.95	0.09	0.49
no	no	yes	87.81	96.15	0.12	0.50
no	no	no	95.20	99.14	0.11	0.56

Zhang, C., Bengio, S., Hardt, M., Recht, B., & Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. arXiv preprint arXiv :1611.03530.

Limitations - Bruit

Limitations - Biais

Classifier	Metric	All	F	M	Darker	Lighter	DF	DM	LF	LM
MSFT	PPV(%)	93.7	89.3	97.4	87.1	99.3	79.2	94.0	98.3	100
	Error Rate(%)	6.3	10.7	2.6	12.9	0.7	20.8	6.0	1.7	0.0
	TPR (%)	93.7	96.5	91.7	87.1	99.3	92.1	83.7	100	98.7
	FPR (%)	6.3	8.3	3.5	12.9	0.7	16.3	7.9	1.3	0.0
Face++	PPV(%)	90.0	78.7	99.3	83.5	95.3	65.5	99.3	94.0	99.2
	Error Rate(%)	10.0	21.3	0.7	16.5	4.7	34.5	0.7	6.0	0.8
	TPR (%)	90.0	98.9	85.1	83.5	95.3	98.8	76.6	98.9	92.9
	FPR (%)	10.0	14.9	1.1	16.5	4.7	23.4	1.2	7.1	1.1
IBM	PPV(%)	87.9	79.7	94.4	77.6	96.8	65.3	88.0	92.9	99.7
	Error Rate(%)	12.1	20.3	5.6	22.4	3.2	34.7	12.0	7.1	0.3
	TPR (%)	87.9	92.1	85.2	77.6	96.8	82.3	74.8	99.6	94.8
	FPR (%)	12.1	14.8	7.9	22.4	3.2	25.2	17.7	5.20	0.4

Buolamwini, Joy, and Timnit Gebru. "Gender shades : Intersectional accuracy disparities in commercial gender classification." Conference on Fairness, Accountability and Transparency. 2018

Limitations - Impact environnementaux

- Impact des technologies du numériques estimé entre 1.5 et 4% des émissions globales de gaz à effet de serres (~37 milliards de tonnes eqCO₂ en 2023)
- Part de l'IA (très) difficile à quantifier, mais qui croît.
- Par exemple (estimations), l'entraînement de GPT3 a nécessité 1,287 MWh (~500 tonnes eqCO₂) ...
- ... mais l'inférence de ChatGPT requiert 564 MWh (~220 tonnes eqCO₂) par jour (soit ~80000 tonnes eqCO₂ par an).

Freitag, C., Berners-Lee, M., Widdicks, K., Knowles, B., Blair, G., & Friday, A. (2021). The climate impact of ICT : A review of estimates, trends and regulations. arXiv preprint arXiv:2102.02622.

Limitations - Impact environnementaux

- Impact des technologies du numériques estimé entre 1.5 et 4% des émissions globales de gaz à effet de serres (~37 milliards de tonnes eqCO₂ en 2023)
- Part de l'IA (très) difficile à quantifier, mais qui croît.
- Par exemple (estimations), l'entraînement de GPT3 a nécessité 1,287 MWh (~500 tonnes eqCO₂) ...
- ... mais l'inférence de ChatGPT requiert 564 MWh (~220 tonnes eqCO₂) par jour (soit ~80000 tonnes eqCO₂ par an).

Freitag, C., Berners-Lee, M., Widdicks, K., Knowles, B., Blair, G., & Friday, A. (2021). The climate impact of ICT : A review of estimates, trends and regulations. arXiv preprint arXiv :2102.02622.

Limitations - Impact environnementaux

- Impact des technologies du numériques estimé entre 1.5 et 4% des émissions globales de gaz à effet de serres (~37 milliards de tonnes eqCO₂ en 2023)
- Part de l'IA (très) difficile à quantifier, mais qui croît.
- Par exemple (estimations), l'entraînement de GPT3 a nécessité 1,287 MWh (~500 tonnes eqCO₂) ...
- ... mais l'inférence de ChatGPT requiert 564 MWh (~220 tonnes eqCO₂) par jour (soit ~80000 tonnes eqCO₂ par an).

Freitag, C., Berners-Lee, M., Widdicks, K., Knowles, B., Blair, G., & Friday, A. (2021). The climate impact of ICT : A review of estimates, trends and regulations. arXiv preprint arXiv :2102.02622.

Patel, D., & Ahmad, A. (2023). The Inference Cost Of Search Disruption–Large Language Model Cost Analysis. Verfügbar unter <https://www.semianalysis.com/p/theinference-cost-of-search-disruption>.

- Impact des technologies du numériques estimé entre 1.5 et 4% des émissions globales de gaz à effet de serres (~37 milliards de tonnes eqCO₂ en 2023)
- Part de l'IA (très) difficile à quantifier, mais qui croît.
- Par exemple (estimations), l'entraînement de GPT3 a nécessité 1,287 MWh (~500 tonnes eqCO₂) ...
- ... mais l'inférence de ChatGPT requiert 564 MWh (~220 tonnes eqCO₂) par jour (soit ~80000 tonnes eqCO₂ par an).

Freitag, C., Berners-Lee, M., Widdicks, K., Knowles, B., Blair, G., & Friday, A. (2021). The climate impact of ICT : A review of estimates, trends and regulations. arXiv preprint arXiv :2102.02622.

Patel, D., & Ahmad, A. (2023). The Inference Cost Of Search Disruption–Large Language Model Cost Analysis. Verfügbar unter <https://www.semianalysis.com/p/theinference-cost-of-search-disruption>.