

# A2DI: Réseaux de neurones

John Klein

Université de Lille - CRIStAL UMR CNRS 9189



Où en est-on dans notre problème d'apprentissage supervisé ?

- On sait que si  $n$ , la taille des données est suffisamment grand par rapport à la taille de  $\theta$  alors  $Err_{train}$  ne dévie pas de  $Err_{esp}$ .

Où en est-on dans notre problème d'apprentissage supervisé ?

- On sait que si  $n$ , la taille des données est suffisamment grand par rapport à la taille de  $\theta$  alors  $Err_{train}$  ne dévie pas de  $Err_{esp}$ .
- On sait que des décisions optimales se prennent si on a accès à  $p_{Y|X}$ .

Où en est-on dans notre problème d'apprentissage supervisé ?

- On sait que si  $n$ , la taille des données est suffisamment grand par rapport à la taille de  $\theta$  alors  $Err_{train}$  ne dévie pas de  $Err_{esp}$ .
- On sait que des décisions optimales se prennent si on a accès à  $p_{Y|X}$ .
- On a vu que  $Err_{train}$  est liée à la NLL de modèles probabilistes :
  - génératif (ex : classifieur naïf Bayésien),
  - discriminatif (ex : régression logistique).

Où en est-on dans notre problème d'apprentissage supervisé ?

- On sait que si  $n$ , la taille des données est suffisamment grand par rapport à la taille de  $\theta$  alors  $Err_{train}$  ne dévie pas de  $Err_{esp}$ .
- On sait que des décisions optimales se prennent si on a accès à  $p_{Y|X}$ .
- On a vu que  $Err_{train}$  est liée à la NLL de modèles probabilistes :
  - génératif (ex : classifieur naïf Bayésien),
  - discriminatif (ex : régression logistique).

L'approche générative requiert un peu plus de données en général et une intuition sur les familles de lois  $p_{X|Y}$ .

Où en est-on dans notre problème d'apprentissage supervisé ?

- On sait que si  $n$ , la taille des données est suffisamment grand par rapport à la taille de  $\theta$  alors  $Err_{train}$  ne dévie pas de  $Err_{esp}$ .
- On sait que des décisions optimales se prennent si on a accès à  $p_{Y|X}$ .
- On a vu que  $Err_{train}$  est liée à la NLL de modèles probabilistes :
  - génératif (ex : classifieur naïf Bayésien),
  - discriminatif (ex : régression logistique).

L'approche générative requiert un peu plus de données en général et une intuition sur les familles de lois  $p_{X|Y}$ .

La régression logistique reste un modèle linéaire, c'est à dire que la frontière séparatrice est une droite.

Où en est-on dans notre problème d'apprentissage supervisé ?

- On sait que si  $n$ , la taille des données est suffisamment grand par rapport à la taille de  $\theta$  alors  $Err_{train}$  ne dévie pas de  $Err_{esp}$ .
- On sait que des décisions optimales se prennent si on a accès à  $p_{Y|X}$ .
- On a vu que  $Err_{train}$  est liée à la NLL de modèles probabilistes :
  - génératif (ex : classifieur naïf Bayésien),
  - discriminatif (ex : régression logistique).

L'approche générative requiert un peu plus de données en général et une intuition sur les familles de lois  $p_{X|Y}$ .

La régression logistique reste un modèle linéaire, c'est à dire que la frontière séparatrice est une droite.

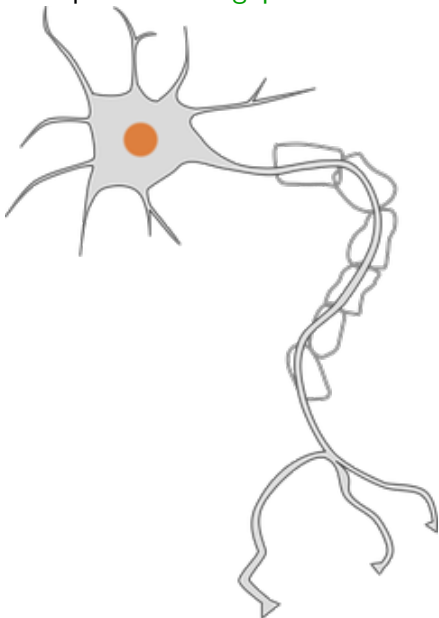
Les réseaux de neurones permettent de converger vers d'autres types de frontière séparatrice, avec comme brique de base la régression logistique.

# Plan du chapitre

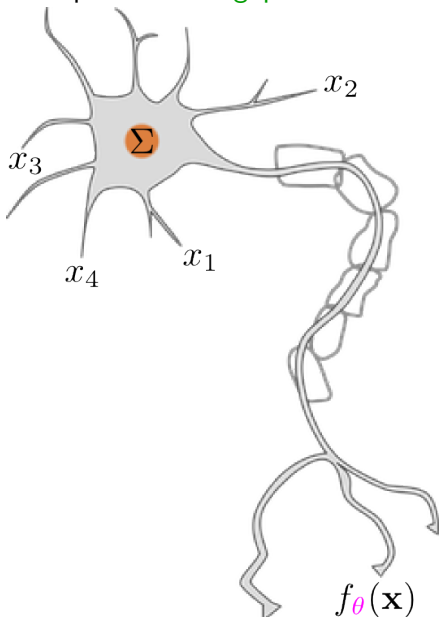
- 1 Généralités
- 2 Rétropropagation
- 3 Réseaux profonds
- 4 Conclusions



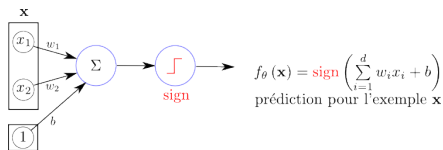
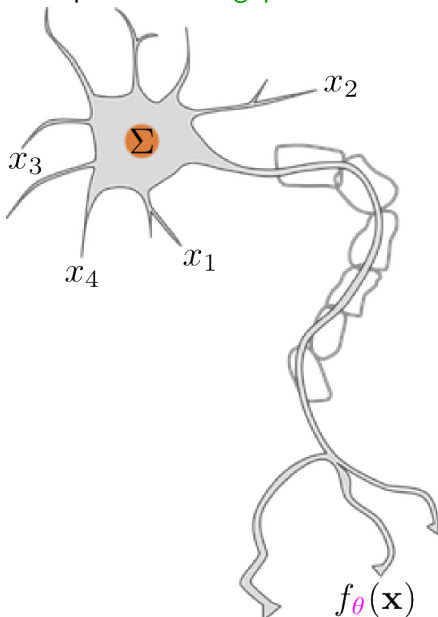
## L'inspiration **biologique** :



L'inspiration **biologique** :



# L'inspiration **biologique** :



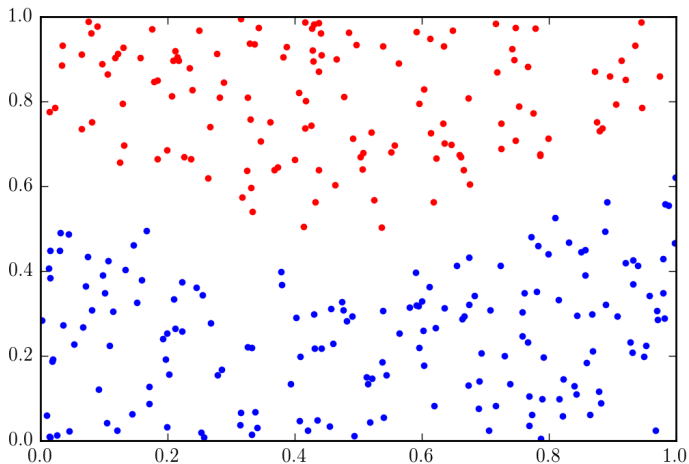
Juste une inspiration !



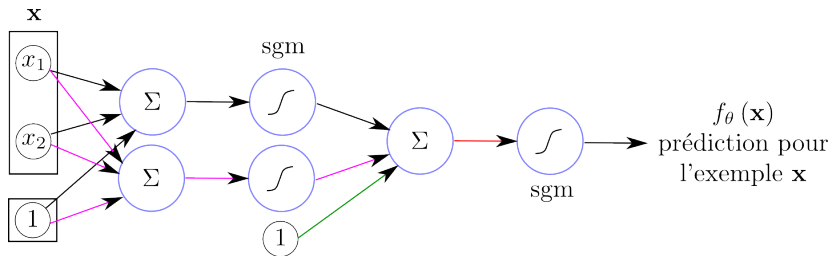
Juste une inspiration !



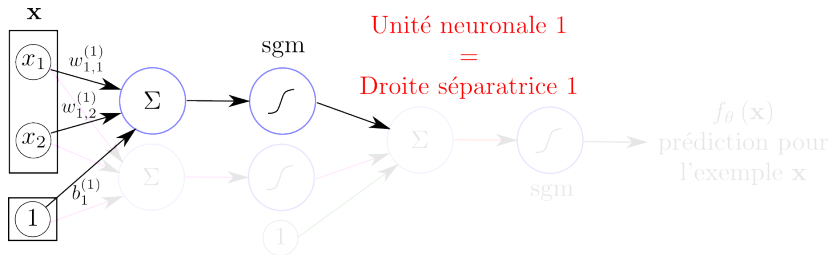
Imaginons apprendre à partir des données suivantes :



Pour résoudre le problème, il faut 2 droites séparatrices combinées en AND.

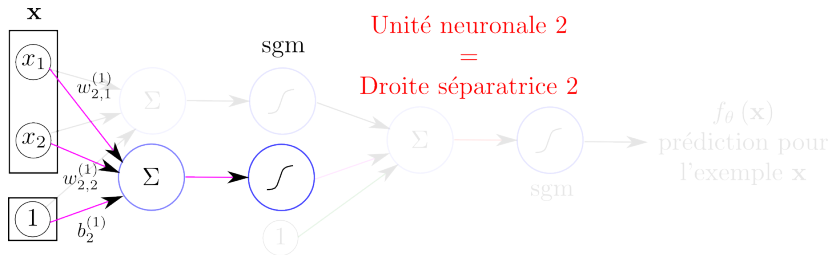


Pour résoudre le problème, il faut 2 droites séparatrices combinées en AND.

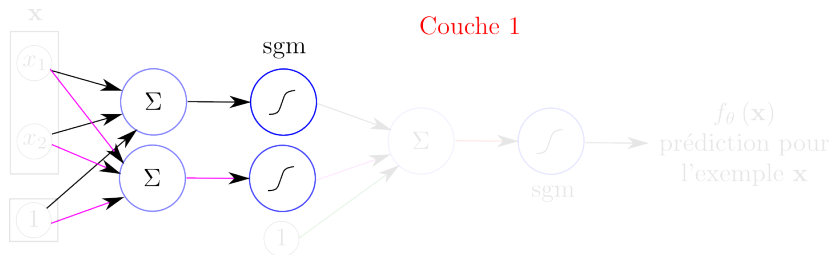




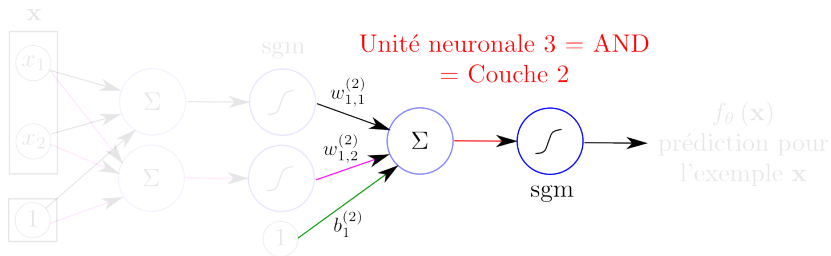
Pour résoudre le problème, il faut 2 droites séparatrices combinées en AND.



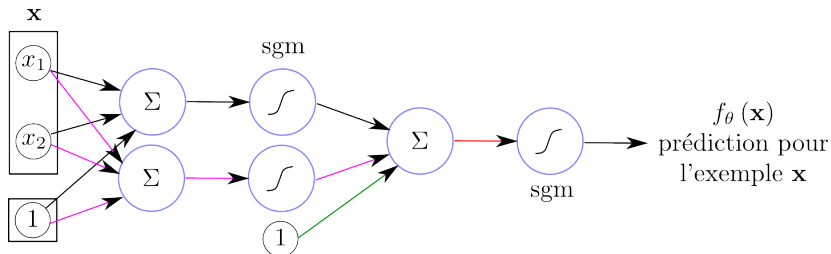
Pour résoudre le problème, il faut 2 droites séparatrices combinées en AND.



Pour résoudre le problème, il faut 2 droites séparatrices combinées en AND.



Cette structure est appelée **MLP** (*multi-layer perceptron*)

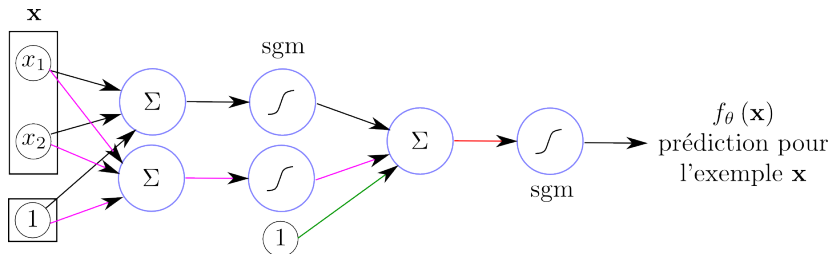


Les paramètres sont indexés comme suit :

$$w_{\mathbf{v},\mathbf{u}}^{(k)}$$

avec :

Cette structure est appelée **MLP** (*multi-layer perceptron*)



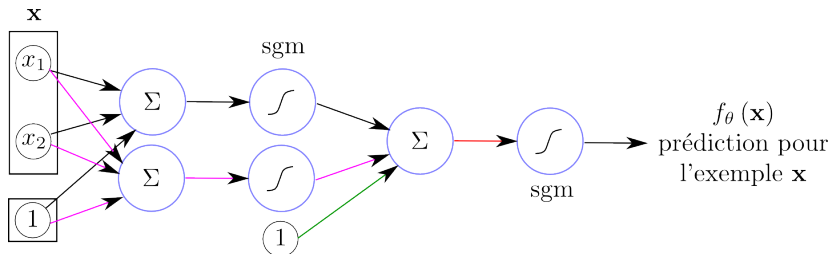
Les paramètres sont indexés comme suit :

$$w_{\mathbf{v},\mathbf{u}}^{(k)}$$

avec :

- $k$  l'indice de la couche variant de 1 à  $K$ .

Cette structure est appelée **MLP** (*multi-layer perceptron*)



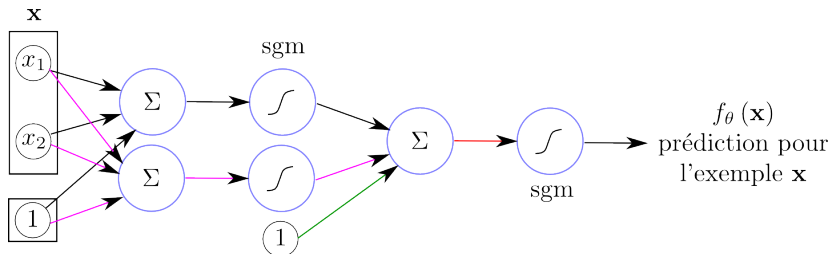
Les paramètres sont indexés comme suit :

$$w_{\mathbf{v},\mathbf{u}}^{(k)}$$

avec :

- $k$  l'indice de la couche variant de 1 à  $K$ .
- $\mathbf{v}$  l'indice des dimensions de l'entrée de la couche  $K$ .

Cette structure est appelée **MLP** (*multi-layer perceptron*)



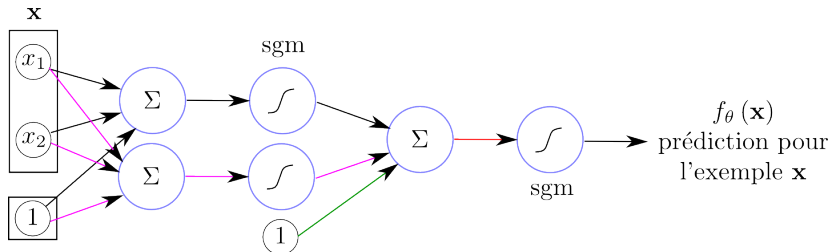
Les paramètres sont indexés comme suit :

$$w_{\mathbf{v},\mathbf{u}}^{(k)}$$

avec :

- $k$  l'indice de la couche variant de 1 à  $K$ .
- $\mathbf{v}$  l'indice des dimensions de l'entrée de la couche  $K$ . Il varie de 1 à  $d^{(k)} + 1$  où  $d^{(k)}$  est la taille du vecteur d'entrée dans la couche  $k$ .

Cette structure est appelée **MLP** (*multi-layer perceptron*)



Les paramètres sont indexés comme suit :

$$w_{\mathbf{v}, \mathbf{u}}^{(k)}$$

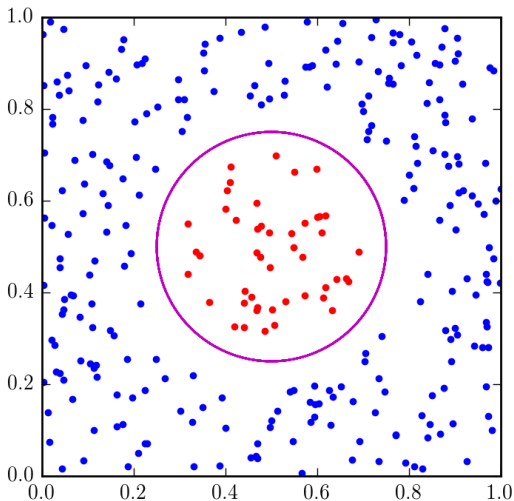
avec :

- $k$  l'indice de la couche variant de 1 à  $K$ .
- $\mathbf{v}$  l'indice des dimensions de l'entrée de la couche  $K$ . Il varie de 1 à  $d^{(k)} + 1$  où  $d^{(k)}$  est la taille du vecteur d'entrée dans la couche  $k$ .
- $\mathbf{u}$  l'indice de l'unité neuronale variant de 1 à  $U^{(k)}$ .



Capacité des réseaux de neurones  $\gg$  capacité de la reglog.

Est elle si grande qu'on puisse espérer traiter le dataset suivant ?



Capacité des réseaux de neurones >> capacité de la reglog.

- Cela semble faisable en rajoutant beaucoup d'unités à la 1<sup>ère</sup> couche.
- Deux dangers :
  - Un peu plus de paramètres  $\Rightarrow$  nettement plus de données nécessaires,
  - Optimisation pour trouver les  $w_{v,u}^{(k)}$  plus difficile.

Capacité des réseaux de neurones >> capacité de la reglog.

- Cela semble faisable en rajoutant beaucoup d'unités à la 1<sup>ère</sup> couche.
- Deux dangers :
  - Un peu plus de paramètres  $\Rightarrow$  nettement plus de données nécessaires,
  - Optimisation pour trouver les  $w_{v,u}^{(k)}$  plus difficile.

Voyons justement comment se passe l'apprentissage des  $w_{v,u}^{(k)}$ .

# Plan du chapitre

- 1 Généralités
- 2 Rétropropagation**
- 3 Réseaux profonds
- 4 Conclusions

## MLP : apprentissage des poids

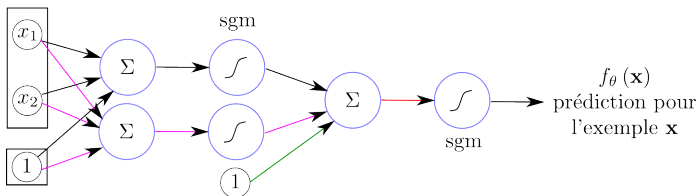
- Comme pour la régression logistique, les paramètres d'un MLP vont être estimés par descente de gradient.

## MLP : apprentissage des poids

- Comme pour la régression logistique, les paramètres d'un MLP vont être estimés par descente de gradient.
- Pour la dernière couche, on retombe sur la reglog et on sait qu'une bonne fonction de coût est la NLL correspondante.

## MLP : apprentissage des poids

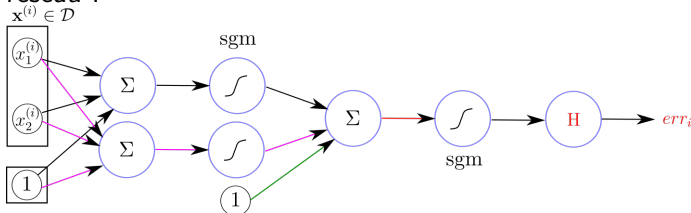
- Comme pour la **régression logistique**, les paramètres d'un **MLP** vont être estimés par **descente de gradient**.
- Pour la dernière couche, on retombe sur la **reglog** et on sait qu'une bonne fonction de coût est la **NLL** correspondante.
- Intégrons la perte **H** (entropie croisée) comme couche finale du réseau :



## MLP : apprentissage des poids

- Comme pour la **régression logistique**, les paramètres d'un **MLP** vont être estimés par **descente de gradient**.
- Pour la dernière couche, on retombe sur la **reglog** et on sait qu'une bonne fonction de coût est la **NLL** correspondante.
- Intégrons la perte **H** (entropie croisée) comme couche finale du

réseau :

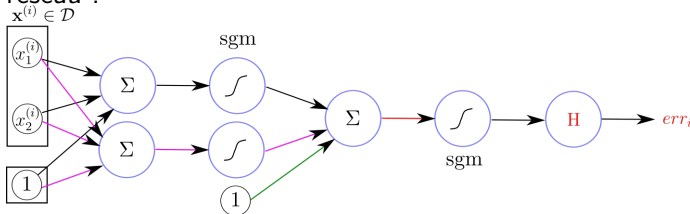




## MLP : apprentissage des poids

- Comme pour la **régression logistique**, les paramètres d'un **MLP** vont être estimés par **descente de gradient**.
- Pour la dernière couche, on retombe sur la **reglog** et on sait qu'une bonne fonction de coût est la **NLL** correspondante.
- Intégrons la perte **H** (entropie croisée) comme couche finale du

réseau :



- La fonction à **optimiser** pour un **MLP** est donc :

$$J(\theta) = \sum_i \text{entropie croisée} \left( y^{(i)}, \text{proba prédite} \right) \quad (1)$$

MLP : apprentissage des poids

- Simplification :

MLP : apprentissage des poids

- Simplification :
  - dérivée d'une somme = somme des dérivées.

## MLP : apprentissage des poids

- Simplification :
  - dérivée d'une somme = somme des dérivées.
  - Pour la descente de gradient, autant calculer le gradient exemple par exemple !

## MLP : apprentissage des poids

- Simplification :

- dérivée d'une somme = somme des dérivées.
- Pour la descente de gradient, autant calculer le gradient exemple par exemple !
- Considérons donc temporairement que :

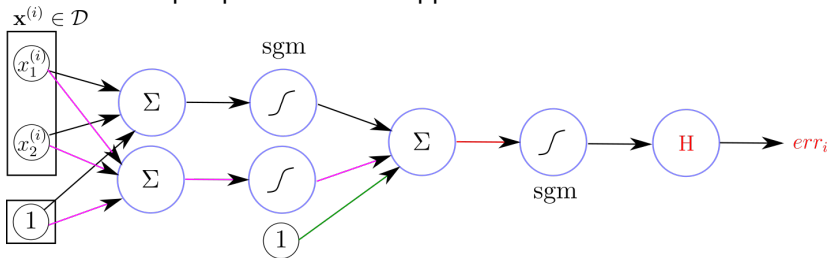
$$J(\theta) = \text{entropie croisée} \left( \text{proba prédite}, y^{(i)} \right) \quad (2)$$

## MLP : apprentissage des poids

- Introduisons quelques variables supplémentaires :

# MLP : apprentissage des poids

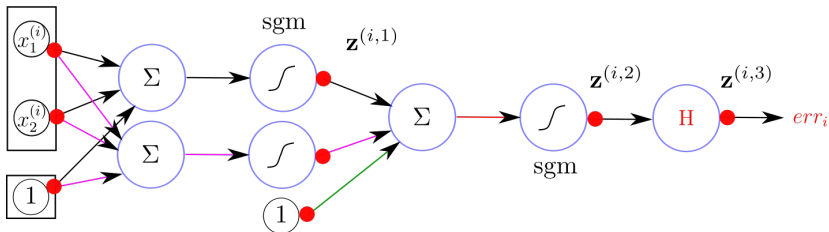
- Introduisons quelques variables supplémentaires :



# MLP : apprentissage des poids

- Introduisons quelques variables supplémentaires :

$$\mathbf{x}^{(i)} = \mathbf{z}^{(i,0)}$$

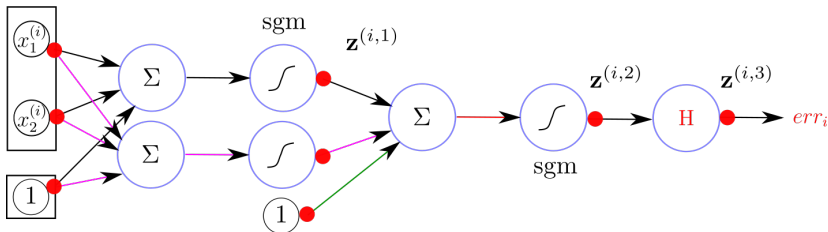




# MLP : apprentissage des poids

- Introduisons quelques variables supplémentaires :

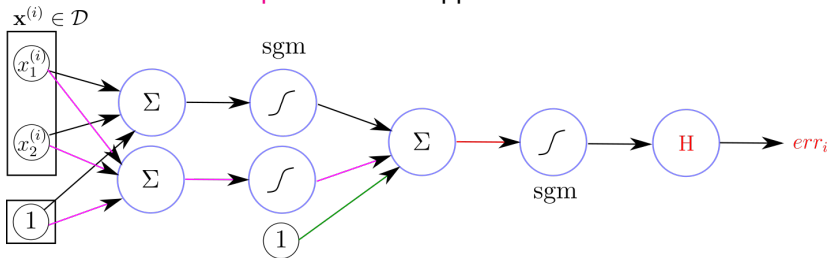
$$\mathbf{x}^{(i)} = \mathbf{z}^{(i,0)}$$



- $\mathbf{z}^{(i,k)}$  : sortie de la couche  $k$ .

## MLP : apprentissage des poids

- Faisons un bilan des paramètres à apprendre :



MLP : apprentissage des poids : descente de gradient

- Simplifications encore :

## MLP : apprentissage des poids : descente de gradient

- Simplifions encore :
  - Calculons les dérivées de l'erreur  $err_i$  par rapport à chaque paramètre  $\theta_j^{(k)}$  en remontant à "contre-courant"

## MLP : apprentissage des poids : descente de gradient

- Simplifications encore :
  - Calculons les dérivées de l'erreur  $err_i$  par rapport à chaque paramètre  $\theta_j^{(k)}$  en remontant à "contre-courant"
  - Pour la dernière couche, c'est pas trop dur : elle n'a aucun paramètre !

## MLP : apprentissage des poids : descente de gradient

- Simplifions encore :

- Calculons les dérivées de l'erreur  $err_i$  par rapport à chaque paramètre  $\theta_j^{(k)}$  en remontant à "contre-courant"
- Pour la dernière couche, c'est pas trop dur : elle n'a aucun paramètre !
- En revanche, on peut calculer la dérivée de sa sortie par rapport à son entrée :

$$\frac{d}{dz^{(i,2)}} err_i(\theta) = \frac{d}{dz^{(i,2)}} z^{(i,3)} =$$

MLP : apprentissage des poids : descente de gradient

- Simplifions encore :
  - Passons à la couche précédente.

MLP : apprentissage des poids : descente de gradient

- Simplifions encore :

- Passons à la couche précédente.
- Elle contient 3 paramètres :  $w_{1,1}^{(2)}$ ,  $w_{2,1}^{(2)}$  et  $b_1^{(2)}$ .



## MLP : apprentissage des poids : descente de gradient

- Simplifications encore :

- Passons à la couche précédente.
- Elle contient 3 paramètres :  $w_{1,1}^{(2)}$ ,  $w_{2,1}^{(2)}$  et  $b_1^{(2)}$ .
- Calculons par exemple :

$$\frac{\partial}{\partial w_{1,1}^{(2)}} err_i(\theta) = \frac{\partial}{\partial w_{1,1}^{(2)}} z^{(i,3)} =$$

## MLP : apprentissage des poids : descente de gradient

- Simplifions encore :

- Passons à la couche précédente.
- On peut aussi calculer la dérivée de sa sortie par rapport à ses entrées, par exemple :

$$\frac{\partial}{\partial z_1^{(i,1)}} z^{(i,2)} =$$

MLP : apprentissage des poids : descente de gradient

- Simplifions encore :
  - Passons encore à la couche précédente (la 1<sup>ère</sup>).

## MLP : apprentissage des poids : descente de gradient

- Simplifions encore :
  - Passons encore à la couche précédente (la 1<sup>ère</sup>).
  - Elle contient 6 paramètres :
    - $w_{1,1}^{(1)}$ ,  $w_{2,1}^{(1)}$  et  $b_1^{(1)}$  pour la 1<sup>ère</sup> unité,

## MLP : apprentissage des poids : descente de gradient

- Simplifions encore :
  - Passons encore à la couche précédente (la 1<sup>ère</sup>).
  - Elle contient 6 paramètres :
    - $w_{1,1}^{(1)}$ ,  $w_{2,1}^{(1)}$  et  $b_1^{(1)}$  pour la 1<sup>ère</sup> unité,
    - $w_{1,2}^{(1)}$ ,  $w_{2,2}^{(1)}$  et  $b_2^{(1)}$  pour la 2<sup>ème</sup> unité.

## MLP : apprentissage des poids : descente de gradient

- Simplifions encore :

- Passons encore à la couche précédente (la 1<sup>ère</sup>).
- Elle contient 6 paramètres :
  - $w_{1,1}^{(1)}$ ,  $w_{2,1}^{(1)}$  et  $b_1^{(1)}$  pour la 1<sup>ère</sup> unité,
  - $w_{1,2}^{(1)}$ ,  $w_{2,2}^{(1)}$  et  $b_2^{(1)}$  pour la 2<sup>ème</sup> unité.
- Calculons par exemple :

$$\frac{\partial}{\partial w_{1,1}^{(1)}} err_i(\theta) = \frac{\partial}{\partial w_{1,1}^{(1)}} z^{(i,3)} =$$

MLP : apprentissage des poids : descente de gradient

- Simplifions encore :
  - Passons encore à la couche précédente (la 1<sup>ère</sup>).
  - Et c'est fini !!

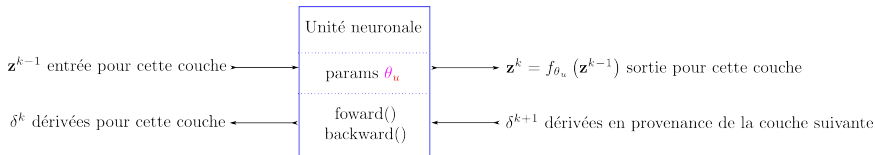
MLP : apprentissage des poids : descente de gradient

- On va donc calculer les gradients **itérativement** de la dernière couche en remontant à la 1<sup>ère</sup>, d'où le terme **retropropagation** !



## MLP : apprentissage des poids : descente de gradient

- On va donc calculer les gradients **itérativement** de la dernière couche en remontant à la 1<sup>ère</sup>, d'où le terme **retropropagation** !
- Le **MLP** est bien adapté à la philosophie POO :
  - On a envie de créer une **classe** pour une unité neuronale

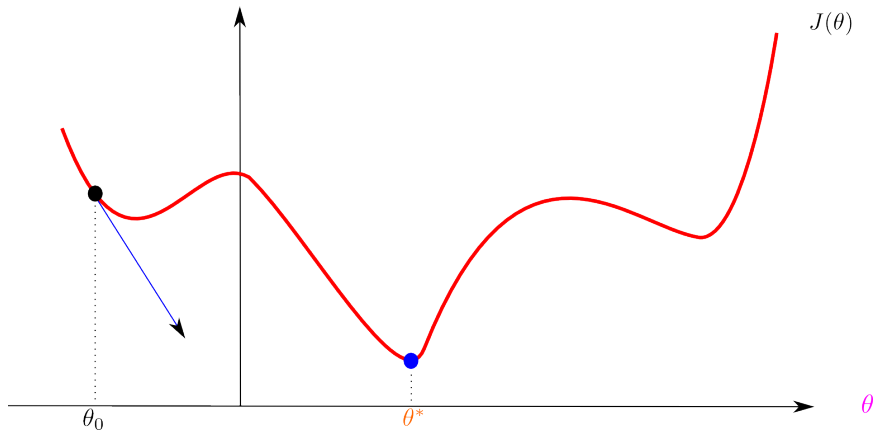


MLP : apprentissage des poids : descente de gradient

- Est ce que la descente de gradient va bien se passer ?

## MLP : apprentissage des poids : descente de gradient

- Est ce que la descente de gradient va bien se passer ?



MLP : apprentissage des poids : descente de gradient

- La descente de gradient converge vers un minimum local.

MLP : apprentissage des poids : descente de gradient

- La descente de gradient converge vers un minimum local.
- Elle dépend de l'initialisation  $\theta_0$ .

## MLP : apprentissage des poids : descente de gradient

- La descente de gradient converge vers un minimum local.
- Elle dépend de l'initialisation  $\theta_0$ .
- En revanche, le réglage du *learning rate*  $\eta$  n'est pas spécialement plus difficile que pour la *reglog*.

# Plan du chapitre

- 1 Généralités
- 2 Rétropropagation
- 3 Réseaux profonds**
- 4 Conclusions

# Réseaux profonds : idée de base



## Réseaux profonds : idée de base

- Quand la distribution des classes est visiblement complexe et que les performances sont décevantes, le réflexe est d'augmenter la **capacité** du modèle en rajoutant des **couches** et des **unités neuronales**. Voyons ce que ça donne avec une petite [visualisation](#).

## Réseaux profonds : idée de base

- Quand la distribution des classes est visiblement complexe et que les performances sont décevantes, le réflexe est d'augmenter la **capacité** du modèle en rajoutant des **couches** et des **unités neuronales**. Voyons ce que ça donne avec une petite [visualisation](#).
- Comme vu dans la partie précédente, faire ce choix fait prendre un gros risque d'**overfitting**.

## Réseaux profonds : idée de base

- Quand la distribution des classes est visiblement complexe et que les performances sont décevantes, le réflexe est d'augmenter la **capacité** du modèle en rajoutant des **couches** et des **unités neuronales**. Voyons ce que ça donne avec une petite [visualisation](#).
- Comme vu dans la partie précédente, faire ce choix fait prendre un gros risque d'**overfitting**.
- Les **réseaux profonds** introduits vers les années 90 proposent une architecture multi-couches sans pour autant faire exploser le nombre de paramètres du modèle.

## CNN (*convolutional neural network*) :

- Le premier algo *deep* ayant rencontré un réel succès est dû à Yann Lecun et alterne deux types de couches :

## CNN (*convolutional neural network*) :

- Le premier algo *deep* ayant rencontré un réel succès est dû à Yann Lecun et alterne deux types de couches :
  - 1 une couche *convolutionnelle* où les paramètres sont *partagés* entre unités neuronales,

## CNN (*convolutional neural network*) :

- Le premier algo *deep* ayant rencontré un réel succès est dû à **Yann Lecun** et **alterne** deux types de couches :
  - 1 une couche **convolutionnelle** où les paramètres sont **partagés** entre unités neuronales,
  - 2 un passage par une **non-linéarité** ,

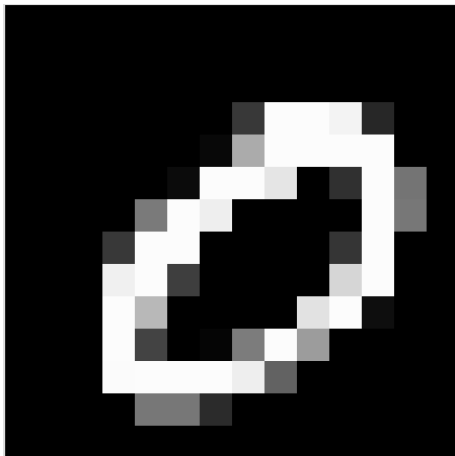
## CNN (*convolutional neural network*) :

- Le premier algo *deep* ayant rencontré un réel succès est dû à **Yann Lecun** et **alterne** deux types de couches :
  - 1 une couche **convolutionnelle** où les paramètres sont **partagés** entre unités neuronales,
  - 2 un passage par une **non-linéarité** ,
  - 3 une couche de **pooling** pour réduire la dimension des sorties de la couche convolutionnelle.

Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov 1998

## CNN (*convolutional neural network*) : convolution

- Supposons que les exemples  $\mathbf{x}^{(i)}$  sont des images :





# CNN (*convolutional neural network*) : convolution

- Prenons un exemple où les exemples  $\mathbf{x}^{(i)}$  sont des images :

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	56	252	252	243	39	0	0
0	0	0	0	0	0	8	171	252	252	252	252	0	0
0	0	0	0	0	11	252	252	229	0	48	252	116	0
0	0	0	0	122	252	238	0	0	0	0	252	119	0
0	0	0	56	252	252	0	0	0	0	53	252	0	0
0	0	0	240	252	62	0	0	0	0	214	252	0	0
0	0	0	252	184	0	0	0	0	226	252	14	0	0
0	0	0	252	67	0	5	124	252	156	0	0	0	0
0	0	0	251	252	252	252	238	98	0	0	0	0	0
0	0	0	0	119	119	43	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

## CNN (*convolutional neural network*) : convolution

- L'opération de convolution entre deux vecteurs  $\mathbf{x}$  et  $\mathbf{y}$  se note  $\star$ .

## CNN (*convolutional neural network*) : convolution

- L'opération de **convolution** entre deux vecteurs  $\mathbf{x}$  et  $\mathbf{y}$  se note  $\star$ .
- Le résultat de cette opération est aussi un vecteur  $\mathbf{z} = \mathbf{x} \star \mathbf{y}$  et on a :

$$z_i = \sum_{k=1}^{\text{len}(y)} x_{i-k} y_k \quad (3)$$

## CNN (*convolutional neural network*) : convolution

- L'opération de **convolution** entre deux vecteurs  $\mathbf{x}$  et  $\mathbf{y}$  se note  $\star$ .
- Le résultat de cette opération est aussi un vecteur  $\mathbf{z} = \mathbf{x} \star \mathbf{y}$  et on a :

$$z_i = \sum_{k=1}^{\text{len}(y)} x_{i-k} y_k \quad (3)$$

La **convolution** ressemble à un **produit scalaire** où l'un des 2 vecteurs a subi une symétrie "miroir" !

## CNN (*convolutional neural network*) : convolution

- L'opération de **convolution** entre deux vecteurs  $\mathbf{x}$  et  $\mathbf{y}$  se note  $\star$ .
- Le résultat de cette opération est aussi un vecteur  $\mathbf{z} = \mathbf{x} \star \mathbf{y}$  et on a :

$$z_i = \sum_{k=1}^{\text{len}(y)} x_{i-k} y_k \quad (3)$$

La **convolution** est **commutative** :  $\mathbf{x} \star \mathbf{y} = \mathbf{y} \star \mathbf{x}$ .

## CNN (*convolutional neural network*) : convolution

- L'opération de **convolution** entre deux vecteurs  $\mathbf{x}$  et  $\mathbf{y}$  se note  $\star$ .
- Le résultat de cette opération est aussi un vecteur  $\mathbf{z} = \mathbf{x} \star \mathbf{y}$  et on a :

$$z_i = \sum_{k=1}^{\text{len}(y)} x_{i-k} y_k \quad (3)$$

La **convolution** est **associative** :  $\mathbf{x} \star (\mathbf{y} \star \mathbf{u}) = (\mathbf{x} \star \mathbf{y}) \star \mathbf{u}$ .

## CNN (*convolutional neural network*) : convolution

- L'opération de **convolution** entre deux vecteurs  $\mathbf{x}$  et  $\mathbf{y}$  se note  $\star$ .
- Le résultat de cette opération est aussi un vecteur  $\mathbf{z} = \mathbf{x} \star \mathbf{y}$  et on a :

$$z_i = \sum_{k=1}^{\text{len}(\mathbf{y})} x_{i-k} y_k \quad (3)$$

La **convolution** est **distributive** par rapport à la somme :

$$\mathbf{x} \star (\mathbf{y} + \mathbf{u}) = \mathbf{x} \star \mathbf{y} + \mathbf{x} \star \mathbf{u}.$$

CNN (*convolutional neural network*) : **convolution**

- Appliquons cette définition à une image binaire :



## CNN (*convolutional neural network*) : **convolution**

- Appliquons cette définition à une image binaire :

1	0	1	0	1
1	1	0	0	0
0	0	0	0	1
1	0	0	1	1
1	1	1	0	0

image  $x$   
(après miroir)

## CNN (*convolutional neural network*) : convolution

- Appliquons cette définition à une image binaire :

1	0	1	0	1
1	1	0	0	0
0	0	0	0	1
1	0	0	1	1
1	1	1	0	0

image **x**  
(après miroir)

-1	0	+1
-1	0	+1
-1	0	+1

filtre **y**

# CNN (*convolutional neural network*) : **convolution**

- Appliquons cette définition à une image binaire :

1	0	1	0	1
1	1	0	0	0
0	0	0	0	1
1	0	0	1	1
1	1	1	0	0

image  $x$ 

-1	0	+1
-1	0	+1
-1	0	+1

filtre  $y$ 

-1		

convolved image  $x \star y$

# CNN (*convolutional neural network*) : **convolution**

- Appliquons cette définition à une image binaire :

1	0	1	0	1
1	1	0	0	0
0	0	0	0	1
1	0	0	1	1
1	1	1	0	0

image  $x$ 

-1	0	+1
-1	0	+1
-1	0	+1

filtre  $y$ 

-1	-1	

image convoluée  $x \star y$

# CNN (*convolutional neural network*) : **convolution**

- Appliquons cette définition à une image binaire :

1	0	1	0	1
1	1	0	0	0
0	0	0	0	1
1	0	0	1	1
1	1	1	0	0

image  $x$ 

-1	0	+1
-1	0	+1
-1	0	+1

filtre  $y$ 

-1	-1	+1

image convoluée  $x \star y$

# CNN (*convolutional neural network*) : **convolution**

- Appliquons cette définition à une image binaire :

1	0	1	0	1
1	1	0	0	0
0	0	0	0	1
1	0	0	1	1
1	1	1	0	0

image  $x$ 

-1	0	+1
-1	0	+1
-1	0	+1

filtre  $y$ 

-1	-1	+1
-2		

image convoluée  $x \star y$

# CNN (*convolutional neural network*) : **convolution**

- Appliquons cette définition à une image binaire :

1	0	1	0	1
1	1	0	0	0
0	0	0	0	1
1	0	0	1	1
1	1	1	0	0

image  $x$ 

-1	0	+1
-1	0	+1
-1	0	+1

filtre  $y$ 

-1	-1	+1
-2	0	+2
-1	0	+1

... etc. image convoluée  $x \star y$

## CNN (*convolutional neural network*) : **convolution**

- Observez que la formule devrait être :

$$z_i = \sum_{k=1}^{\text{len}(y)} x_{i-k} y_k, \quad (4)$$








pour  $i$  de  $\boxed{\text{floor}\left(\frac{\text{len}(y)}{2}\right)}$  à  $\boxed{\text{len}(x) - \text{floor}\left(\frac{\text{len}(y)}{2}\right)}$ .

On ne peut pas commencer la convolution au **bord** de l'image



## Exemples de filtres d'images

(source : Wikipedia - images kernels)

Operation	Filter	Convolved Image
<b>Identity</b>	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
<b>Edge detection</b>	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
<b>Sharpen</b>	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
<b>Box blur</b> (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
<b>Gaussian blur</b> (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

## CNN (*convolutional neural network*) : Paramètres d'une couche de convolution

- **depth** : c'est le nombre de filtre(s) qu'on utilise dans la couche.
  - Dans l'exemple précédent, nous avons utilisé un seul filtre, mais il est fréquent d'en utiliser des dizaines en parallèle.
  - Le filtre numéro  $m$  a ses propres paramètres  $\mathbf{w}^{(k,m)}$ .
  - Chaque filtre produit une image convoluée  $\mathbf{z}^{(k,m)}$  appelée *feature map*.
  - Ces images sont convoluées globalement et  $\mathbf{w}^{(k,m)}$  est un tenseur 3D :

$$\underbrace{\mathbf{z}^{(k,1)}}_{\text{tenseur 2D}} \quad \text{shape} = (\bullet, \bullet) = f_{\text{act}} \left( \underbrace{\mathbf{w}^{(k,1)}}_{\text{tenseur 3D}} \star \underbrace{\mathbf{z}^{(k-1)}}_{\text{tenseur 3D}} \right).$$

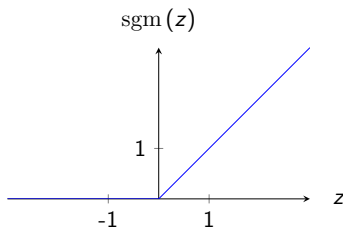
$\text{shape} = (\text{Hauteur}_k, \text{Largeur}_k, \text{Depth}_{k-1}) \quad \text{shape} = (\bullet, \bullet, \text{Depth}_{k-1})$

- **stride** : on décide de "sauter" de *stride* pixels entre chaque itération de la convolution.

CNN (*convolutional neural network*) : Paramètres d'une couche de convolution

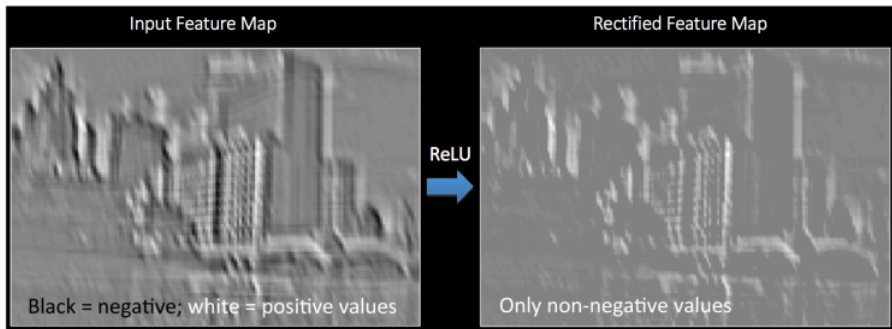
- padding : gestion des bords de l'image
- fonction d'activation  $f_{\text{act}}$  : souvent ReLU (*rectified linear unit*). C'est une fonction continue de  $\mathbb{R}$  dans  $[0; 1]$ , telle que

$$\text{ReLU}(z) = \begin{cases} z & \text{si } z \geq 0 \\ 0 & \text{sinon} \end{cases}. \quad (5)$$



## CNN (*convolutional neural network*) : fonction d'activation

- Son action consiste à conserver la partie positive d'une *feature map*.

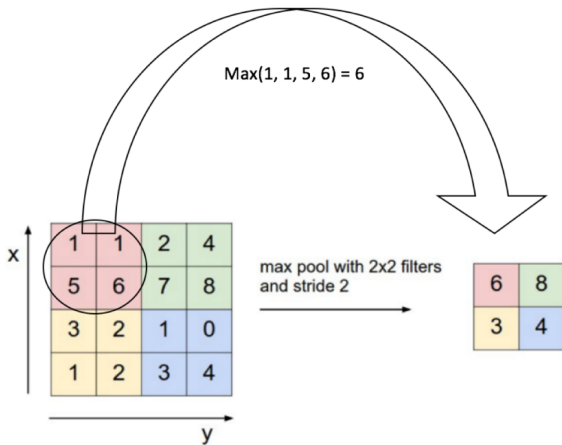


(source : Rob Fergus - Facebook AI research)

CNN (*convolutional neural network*) : pooling

Il s'agit pour un voisinage donné de la feature map (après passage par ReLU) de prendre :

- le max (*max pooling*),
- ou la moyenne (*mean pooling*).

CNN (*convolutional neural network*) : pooling

Rectified Feature Map

Exemples de *max pooling*(source : [cs231n.github.io](https://cs231n.github.io))

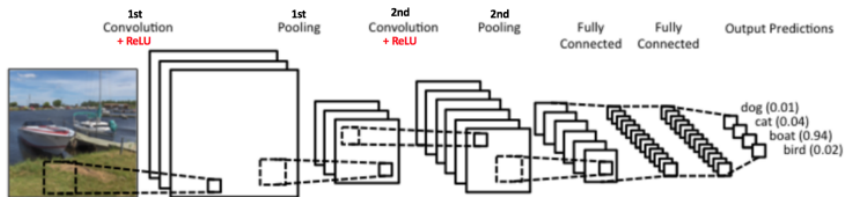
## CNN (*convolutional neural network*) : pooling

La couche de pooling nous aide à :

- réduire la dimension des représentations intermédiaires,
- se rendre invariant à de petites distortions/translations locales.

# CNN (*convolutional neural network*) : architecture globale

Quand on assemble tout, cela donne par exemple :



(source : WILDML - understanding CNNs for nlp)



## CNN (*convolutional neural network*) : **Rétropropagation**

Le principe d'une **descente de gradient** itérative (couche par couche) reste valable !

## CNN (*convolutional neural network*) : **Rétropropagation**

Le principe d'une **descente de gradient** itérative (couche par couche) reste valable !

- Les **dérivées** pour la couche de **max pooling** sont transférées aux pixels ayant été sélectionnés par le max,

## CNN (*convolutional neural network*) : **Rétropropagation**

Le principe d'une **descente de gradient** itérative (couche par couche) reste valable !

- Les **dérivées** pour la couche de **max pooling** sont transférées aux pixels ayant été sélectionnés par le max,
- Les **dérivées** pour **ReLU** sont ultra simples,

## CNN (*convolutional neural network*) : Rétropropagation

Le principe d'une **descente de gradient** itérative (couche par couche) reste valable !

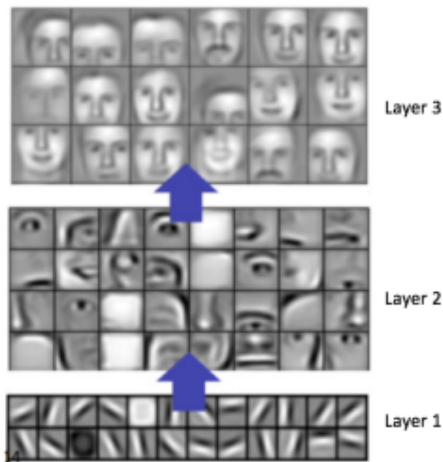
- Les **dérivées** pour la couche de **max pooling** sont transférées aux pixels ayant été sélectionnés par le max,
- Les **dérivées** pour **ReLU** sont ultra simples,
- Les **dérivées** par rapport aux poids  $\mathbf{w}^{(k,m)}$  qui interviennent dans une **convolution** sont également simples à calculer.

## CNN (*convolutional neural network*) : **Rétropropagation**

Le principe d'une **descente de gradient** itérative (couche par couche) reste valable !

- Les **dérivées** pour la couche de **max pooling** sont transférées aux pixels ayant été sélectionnés par le max,
- Les **dérivées** pour **ReLU** sont ultra simples,
- Les **dérivées** par rapport aux poids  $\mathbf{w}^{(k,m)}$  qui interviennent dans une **convolution** sont également simples à calculer.
- Les  $\mathbf{w}^{(k,m)}$  subissent plusieurs m à j en provenance de la **feature map** par itération.

## CNN (*convolutional neural network*) : Filter Visualisation



(Fig. tirée de "Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations", Lee et al., ICML 2009.)

# CNN (*convolutional neural network*) : Feature maps Visualisation



(image générée par <http://scs.ryerson.ca/~aharley/vis/conv/flat.html>.)

## CNN (*convolutional neural network*) / adds-on : Batch normalization

- Une modification conséquente des inputs au niveau de la première couche impacte fortement le modèle -> (Centrage + réduction).
- Mais qu'en est il dans les couches intermédiaires ? Peut on centrer et réduire les  $\mathbf{z}^{(i,k)}$  ?



## CNN (*convolutional neural network*) / adds-on : Batch normalization

- Une modification conséquente des inputs au niveau de la première couche impacte fortement le modèle -> (Centrage + réduction).
- Mais qu'en est il dans les couches intermédiaires ? Peut on centrer et réduire les  $\mathbf{z}^{(i,k)}$  ?
- Dans cet objectif, on peut appliquer une transformation BN en sortie de la convolution (juste avant le passage par la fonction d'activation).

## CNN (*convolutional neural network*) / adds-on : Batch normalization

- Une modification conséquente des inputs au niveau de la **première couche** impacte fortement le modèle -> (Centrage + réduction).
- Mais qu'en est il dans les couches intermédiaires ? Peut on centrer et réduire les  $\mathbf{z}^{(i,k)}$  ?
- Dans cet objectif, on peut appliquer une **transformation BN** en sortie de la convolution (juste avant le passage par la fonction d'activation).
- Cette transformation n'est valable que pour **1 itération** de **SGD** en version **mini-batch**.

S. Ioffe, C. Szegedy, Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift, ICML 2015.

## CNN (*convolutional neural network*) / adds-on : Batch normalization

- Soit  $\mathbf{v}^{(i)}$  la sortie d'une couche de convolution pour l'exemple  $\mathbf{x}^{(i)}$  appartenant au mini-batch courant :

$$\mathbf{v}^{(i)} = \mathbf{x}^{(i)} \star \text{filtre}.$$

## CNN (*convolutional neural network*) / adds-on : Batch normalization

- Soit  $\mathbf{v}^{(i)}$  la sortie d'une couche de convolution pour l'exemple  $\mathbf{x}^{(i)}$  appartenant au mini-batch courant :

$$\mathbf{v}^{(i)} = \mathbf{x}^{(i)} \star \text{filtre}.$$

- On calcule  $\mu_j$  et  $\sigma_j$  la moyenne et l'écart type de de la  $j^{\text{ème}}$  dimension des  $\mathbf{v}^{(i)}$ .

## CNN (*convolutional neural network*) / adds-on : Batch normalization

- Soit  $\mathbf{v}^{(i)}$  la sortie d'une couche de convolution pour l'exemple  $\mathbf{x}^{(i)}$  appartenant au mini-batch courant :

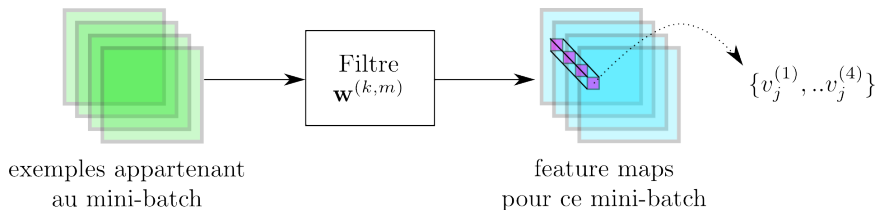
$$\mathbf{v}^{(i)} = \mathbf{x}^{(i)} \star \text{filtre}.$$

- On calcule  $\mu_j$  et  $\sigma_j$  la moyenne et l'écart type de de la  $j^{\text{ème}}$  dimension des  $\mathbf{v}^{(i)}$ .
- On remplace chaque  $\mathbf{v}^{(i)}$  par

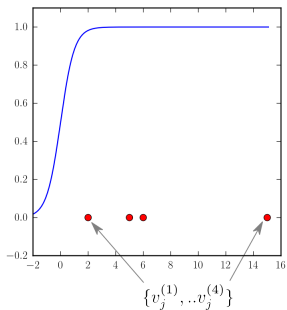
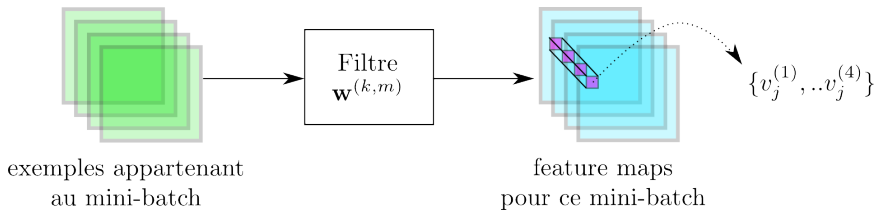
$$\text{BN}(\mathbf{v}^{(i)}) = \begin{pmatrix} \vdots \\ \gamma_j * \frac{v_j^{(i)} - \mu_j}{\sigma_j^2 + \epsilon} + \beta_j \\ \vdots \end{pmatrix} \quad (6)$$

- Les  $\gamma_j$  et  $\beta_j$  seront appris tandis que  $\epsilon$  est fixe et permet d'éviter une division par zéro.

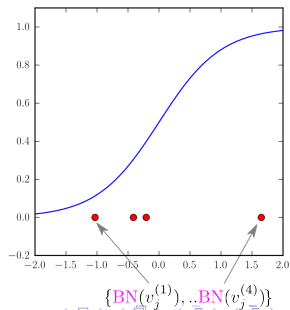
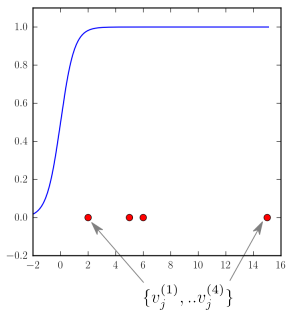
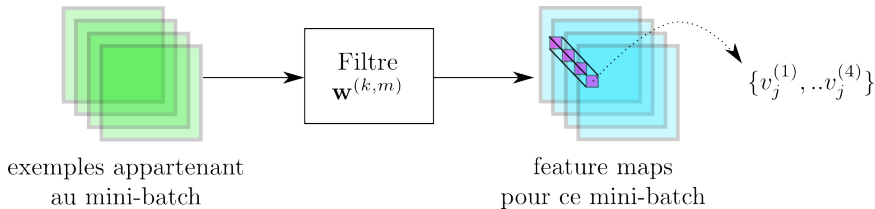
# CNN (*convolutional neural network*) / add-on : Batch normalization



# CNN (*convolutional neural network*) / add-on : Batch normalization



# CNN (*convolutional neural network*) / add-on : Batch normalization





## CNN (*convolutional neural network*) / adds-on : Dropout

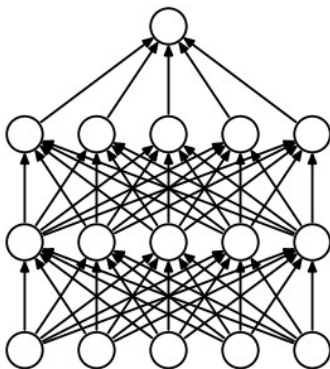
- Pour rendre l'apprentissage plus robuste on peut lui compliquer la vie.

## CNN (*convolutional neural network*) / adds-on : Dropout

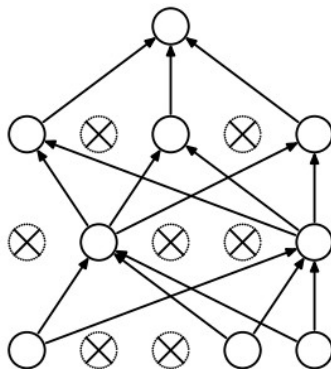
- Pour rendre l'apprentissage plus robuste on peut lui compliquer la vie.
- Dropout propose de faire cela en "éteignant" aléatoirement certaines unités neuronales.

## CNN (*convolutional neural network*) / adds-on : Dropout

- Pour rendre l'apprentissage plus **robuste** on peut lui compliquer la vie.
- **Dropout** propose de faire cela en "éteignant" aléatoirement certaines unités neuronales.



(a) Standard Neural Net



(b) After applying dropout.

Fig. tirée de N. Srivastava et al., Dropout : A Simple Way to Prevent Neural Networks from Overfitting, JMLR 2014.

## CNN (*convolutional neural network*) / adds-on : Dropout

- Pendant la phase d'apprentissage, chaque unité est éteinte avec la probabilité  $p$ .

## CNN (*convolutional neural network*) / add-on : Dropout

- Pendant la phase d'apprentissage, chaque unité est éteinte avec la probabilité  $p$ .
- Pendant la phase de test, les poids connectés en sortie à l'unité sont multipliés par  $p$ .

## CNN (convolutional neural network) / adds-on : Dropout

- Pendant la phase d'**apprentissage**, chaque unité est éteinte avec la probabilité  $p$ .
- Pendant la phase de **test**, les poids connectés en sortie à l'unité sont multipliés par  $p$ .

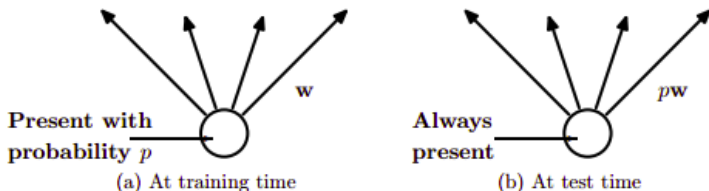


Fig. tirée de N. Srivastava et al., Dropout : A Simple Way to Prevent Neural Networks from Overfitting, JMLR 2014.

## CNN (*convolutional neural network*) : Initialisation

- Les poids des filtres d'un CNN ne doivent pas être initialisés à zéro.
- L'heuristique est :

$$\mathbf{w}^{(k,m)} \leftarrow \sqrt{\frac{2}{\dim(\mathbf{w}^{(k,m)})}} \times \mathbf{u}, \quad (7)$$

avec  $\mathbf{u}$  un vecteur choisi au hasard et dont chaque composante suit la loi normale centrée réduite  $u_j \sim \mathcal{N}(0, 1)$ .

## CNN (*convolutional neural network*) : Célébrités

- LeNet : 90's, 2 séries de (conv + max pooling) + MLP,



## CNN (*convolutional neural network*) : Célébrités

- **LeNet** : 90's, 2 séries de (conv + max pooling) + MLP,
- **AlexNet** : 2012, 5 couches de conv (max pooling pas systématique) + MLP, 60.000.000 de paramètres,

## CNN (*convolutional neural network*) : Célébrités

- **LeNet** : 90's, 2 séries de (conv + max pooling) + MLP,
- **AlexNet** : 2012, 5 couches de conv (max pooling pas systématique) + MLP, 60.000.000 de paramètres,
- **GoogleLeNet** (inception-v4) : 2014, jusqu'à 12 couches de conv, mais certaines en parallèle puis concaténation régulières,

## CNN (*convolutional neural network*) : Célébrités

- **LeNet** : 90's, 2 séries de (conv + max pooling) + MLP,
- **AlexNet** : 2012, 5 couches de conv (max pooling pas systématique) + MLP, 60.000.000 de paramètres,
- **GoogleLeNet** (inception-v4) : 2014, jusqu'à 12 couches de conv, mais certaines en parallèle puis concaténation régulières,
- **VGGNet** : 2014, jusqu'à 19 couches, 140.000.000 de paramètres,

## CNN (*convolutional neural network*) : Célébrités

- **LeNet** : 90's, 2 séries de (conv + max pooling) + MLP,
- **AlexNet** : 2012, 5 couches de conv (max pooling pas systématique) + MLP, 60.000.000 de paramètres,
- **GoogleLeNet** (inception-v4) : 2014, jusqu'à 12 couches de conv, mais certaines en parallèle puis concaténation régulières,
- **VGGNet** : 2014, jusqu'à 19 couches, 140.000.000 de paramètres,
- **ResNet** : 2015, certaines inputs peuvent "sauter" des couches,

## CNN (*convolutional neural network*) : Célébrités

- **LeNet** : 90's, 2 séries de (conv + max pooling) + MLP,
- **AlexNet** : 2012, 5 couches de conv (max pooling pas systématique) + MLP, 60.000.000 de paramètres,
- **GoogleLeNet** (inception-v4) : 2014, jusqu'à 12 couches de conv, mais certaines en parallèle puis concaténation régulières,
- **VGGNet** : 2014, jusqu'à 19 couches, 140.000.000 de paramètres,
- **ResNet** : 2015, certaines inputs peuvent "sauter" des couches,
- **DenseNet** : 2016, toutes les inputs atteignent toutes les couches suivantes.

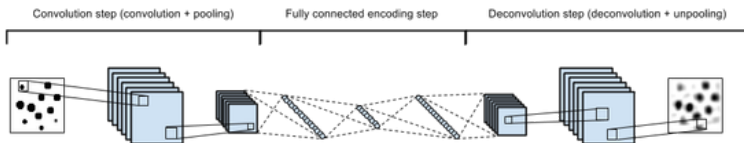
CNN (*convolutional neural network*) : une **démo** sympa

[CNN sur dataset cifar10.](#)

## Autres réseaux profonds orientés **pre-training** :

### ● **Auto-encoders** :

- Un réseau de type CNN est construit selon une architecture “papillon”.
- La première partie transforme le vecteur d'entrée en une représentation **compressée**.
- La 2ème partie reconstruit l'entrée à partir de cette représentation (phase de **pre-training** non supervisée).
- Dans second temps, la partie reconstruction du réseau est abandonnée et on plug un MLP en sortie de la partie restante.
- Ce nouveau réseau est ensuite entraîné de manière supervisée (**fine-tuning**) comme un CNN normal dont certains paramètres sont judicieusement initialisés

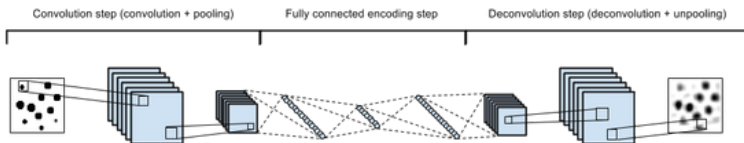


source : Blog de Mike Swarbrick Jones.

## Autres réseaux profonds orientés **pre-training** :

- **Auto-encoders** :

- Un réseau de type CNN est construit selon une architecture “papillon”.
- La première partie transforme le vecteur d'entrée en une représentation **compressée**.
- La 2ème partie reconstruit l'entrée à partir de cette représentation (phase de **pre-training** non supervisée).
- Dans second temps, la partie reconstruction du réseau est abandonnée et on plug un MLP en sortie de la partie restante.
- Ce nouveau réseau est ensuite entraîné de manière supervisée (**fine-tuning**) comme un CNN normal dont certains paramètres sont judicieusement initialisés



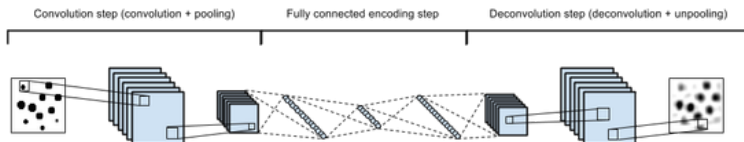
source : Blog de Mike Swarbrick Jones.



## Autres réseaux profonds orientés **pre-training** :

### • **Auto-encoders** :

- Un réseau de type CNN est construit selon une architecture “papillon”.
- La première partie transforme le vecteur d'entrée en une représentation **compressée**.
- La 2ème partie reconstruit l'entrée à partir de cette représentation (phase de **pre-training** non supervisée).
- Dans second temps, la partie reconstruction du réseau est abandonnée et on plug un MLP en sortie de la partie restante.
- Ce nouveau réseau est ensuite entraîné de manière supervisée (**fine-tuning**) comme un CNN normal dont certains paramètres sont judicieusement initialisés

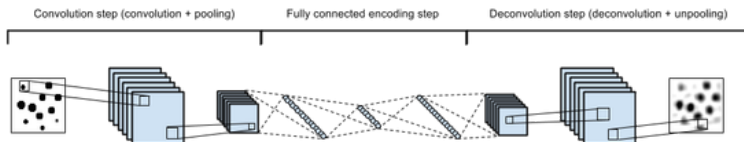


source : Blog de Mike Swarbrick Jones.

## Autres réseaux profonds orientés **pre-training** :

### • **Auto-encoders** :

- Un réseau de type CNN est construit selon une architecture “papillon”.
- La première partie transforme le vecteur d'entrée en une représentation **compressée**.
- La 2ème partie reconstruit l'entrée à partir de cette représentation (phase de **pre-training** non supervisée).
- Dans second temps, la partie reconstruction du réseau est abandonnée et on plug un MLP en sortie de la partie restante.
- Ce nouveau réseau est ensuite entraîné de manière supervisée (**fine-tuning**) comme un CNN normal dont certains paramètres sont judicieusement initialisés

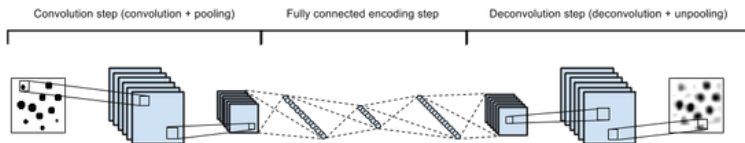


source : Blog de Mike Swarbrick Jones.

## Autres réseaux profonds orientés **pre-training** :

### • **Auto-encoders** :

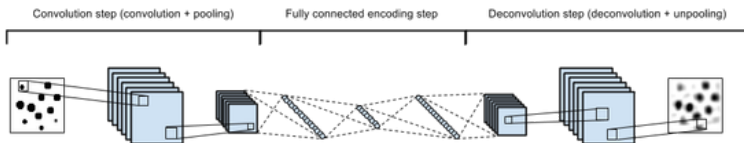
- Un réseau de type CNN est construit selon une architecture “papillon”.
- La première partie transforme le vecteur d'entrée en une représentation **compressée**.
- La 2ème partie reconstruit l'entrée à partir de cette représentation (phase de **pre-training** non supervisée).
- Dans second temps, la partie reconstruction du réseau est abandonnée et on plug un MLP en sortie de la partie restante.
- Ce nouveau réseau est ensuite entraîné de manière supervisée (**fine-tuning**) comme un CNN normal dont certains paramètres sont judicieusement initialisés



source : Blog de Mike Swarbrick Jones.

Autres réseaux profonds orientés **pre-training** :● **Auto-encoders** :

- Un réseau de type CNN est construit selon une architecture “papillon”.
- La première partie transforme le vecteur d'entrée en une représentation **compressée**.
- La 2ème partie reconstruit l'entrée à partir de cette représentation (phase de **pre-training** non supervisée).
- Dans second temps, la partie reconstruction du réseau est abandonnée et on plug un MLP en sortie de la partie restante.
- Ce nouveau réseau est ensuite entraîné de manière supervisée (**fine-tuning**) comme un CNN normal dont certains paramètres sont judicieusement initialisés



source : Blog de Mike Swarbrick Jones.

## Autres réseaux profonds orientés pre-training :

- **Deep Belief Networks** : les couches reposent sur un modèle probabiliste appelé **Machine de Boltzmann restreinte**.
  - Ce modèle représente la distribution jointe d'un exemple  $\mathbf{x} = \mathbf{z}^{(0)}$  et des sorties de chaque couche d'un MLP :  $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(n_c)}$  où  $n_c$  est le nombre de couches pré-entraînées.
  - Dans ce modèle, la loi conditionnelle  $p\left(\mathbf{z}_j^{(i+1)} | \mathbf{z}^{(i)}\right)$  est de la forme  $\text{sgm}\left(\boldsymbol{\theta} \cdot \mathbf{z}_+^{(i)}\right)$ .
  - L'algorithme *contrastive divergence* entraîne ce modèle à reconstruire l'entrée  $\mathbf{x}$  en mode non supervisé.
  - Après convergence, on plug un MLP et on entraîne le tout en mode supervisé.
  - Comme pour les auto-encoders, les paramètres des premières couches sont judicieusement initialisés.

Autres réseaux profonds orientés **pre-training** :

- **Deep Belief Networks** : les couches reposent sur un modèle probabiliste appelé **Machine de Boltzmann restreinte**.
  - Ce modèle représente la distribution jointe d'un exemple  $\mathbf{x} = \mathbf{z}^{(0)}$  et des sorties de chaque couche d'un MLP :  $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(n_c)}$  où  $n_c$  est le nombre de couches pré-entraînées.
  - Dans ce modèle, la loi conditionnelle  $p\left(\mathbf{z}_j^{(i+1)} | \mathbf{z}^{(i)}\right)$  est de la forme  $\text{sgm}\left(\boldsymbol{\theta} \cdot \mathbf{z}_+^{(i)}\right)$ .
  - L'algorithme *contrastive divergence* entraîne ce modèle à reconstruire l'entrée  $\mathbf{x}$  en mode non supervisé.
  - Après convergence, on plug un MLP et on entraîne le tout en mode supervisé.
  - Comme pour les auto-encoders, les paramètres des premières couches sont judicieusement initialisés.

Autres réseaux profonds orientés pre-training :

- **Deep Belief Networks** : les couches reposent sur un modèle probabiliste appelé **Machine de Boltzmann restreinte**.
  - Ce modèle représente la distribution jointe d'un exemple  $\mathbf{x} = \mathbf{z}^{(0)}$  et des sorties de chaque couche d'un MLP :  $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(n_c)}$  où  $n_c$  est le nombre de couches pré-entraînées.
  - Dans ce modèle, la loi conditionnelle  $p\left(\mathbf{z}_j^{(i+1)} | \mathbf{z}^{(i)}\right)$  est de la forme  $\text{sgm}\left(\boldsymbol{\theta} \cdot \mathbf{z}_+^{(i)}\right)$ .
  - L'algorithme *contrastive divergence* entraîne ce modèle à reconstruire l'entrée  $\mathbf{x}$  en mode non supervisé.
  - Après convergence, on plug un MLP et on entraîne le tout en mode supervisé.
  - Comme pour les auto-encoders, les paramètres des premières couches sont judicieusement initialisés.

Autres réseaux profonds orientés pre-training :

- **Deep Belief Networks** : les couches reposent sur un modèle probabiliste appelé **Machine de Boltzmann restreinte**.
  - Ce modèle représente la distribution jointe d'un exemple  $\mathbf{x} = \mathbf{z}^{(0)}$  et des sorties de chaque couche d'un MLP :  $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(n_c)}$  où  $n_c$  est le nombre de couches pré-entraînées.
  - Dans ce modèle, la loi conditionnelle  $p\left(\mathbf{z}_j^{(i+1)} | \mathbf{z}^{(i)}\right)$  est de la forme  $\text{sgm}\left(\boldsymbol{\theta} \cdot \mathbf{z}_+^{(i)}\right)$ .
  - L'algorithme *contrastive divergence* entraîne ce modèle à reconstruire l'entrée  $\mathbf{x}$  en mode non supervisé.
  - Après convergence, on plug un MLP et on entraîne le tout en mode supervisé.
  - Comme pour les auto-encoders, les paramètres des premières couches sont judicieusement initialisés.



Autres réseaux profonds orientés **pre-training** :

- **Deep Belief Networks** : les couches reposent sur un modèle probabiliste appelé **Machine de Boltzmann restreinte**.
  - Ce modèle représente la distribution jointe d'un exemple  $\mathbf{x} = \mathbf{z}^{(0)}$  et des sorties de chaque couche d'un MLP :  $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(n_c)}$  où  $n_c$  est le nombre de couches pré-entraînées.
  - Dans ce modèle, la loi conditionnelle  $p\left(\mathbf{z}_j^{(i+1)} | \mathbf{z}^{(i)}\right)$  est de la forme  $\text{sgm}\left(\boldsymbol{\theta} \cdot \mathbf{z}_+^{(i)}\right)$ .
  - L'algorithme *contrastive divergence* entraîne ce modèle à reconstruire l'entrée  $\mathbf{x}$  en mode non supervisé.
  - Après convergence, on plug un MLP et on entraîne le tout en mode supervisé.
  - Comme pour les auto-encoders, les paramètres des premières couches sont judicieusement initialisés.

Autres réseaux profonds orientés pre-training :

- **Deep Belief Networks** : les couches reposent sur un modèle probabiliste appelé **Machine de Boltzmann restreinte**.
  - Ce modèle représente la distribution jointe d'un exemple  $\mathbf{x} = \mathbf{z}^{(0)}$  et des sorties de chaque couche d'un MLP :  $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(n_c)}$  où  $n_c$  est le nombre de couches pré-entraînées.
  - Dans ce modèle, la loi conditionnelle  $p\left(\mathbf{z}_j^{(i+1)} | \mathbf{z}^{(i)}\right)$  est de la forme  $\text{sgm}\left(\boldsymbol{\theta} \cdot \mathbf{z}_+^{(i)}\right)$ .
  - L'algorithme *contrastive divergence* entraîne ce modèle à reconstruire l'entrée  $\mathbf{x}$  en mode non supervisé.
  - Après convergence, on plug un MLP et on entraîne le tout en mode supervisé.
  - Comme pour les auto-encoders, les paramètres des premières couches sont judicieusement initialisés.

Autres réseaux profonds orientés pre-training :

- Enorme avantage :

Ces deux approches peuvent être utilisées pour un apprentissage semi-supervisé ! (pre-training avec données non labélisées, fine-tuning avec données labélisées).

## Messages importants du chapitre :

- Les **réseaux de neurones** offrent la possibilité de traiter des données dont la frontière séparatrice n'est à l'évidence pas linéaire.
- Grâce à l'algorithme de **rétropropagation**, l'apprentissage de ces modèles n'a pas un coût exorbitant.
- Les architectures modernes des **réseaux de neurones** produisent des résultats d'une qualité remarquable, notamment en vision par ordinateur.