Journée SYDRE: Formation "Deep Learning"

Introduction à l'apprentissage profond

Alexis Lechervy
<alexis.lechervy@unicaen.fr>

17 juin 2022

Sommaire

Sommaire

Présentation de la journée

L'orateur

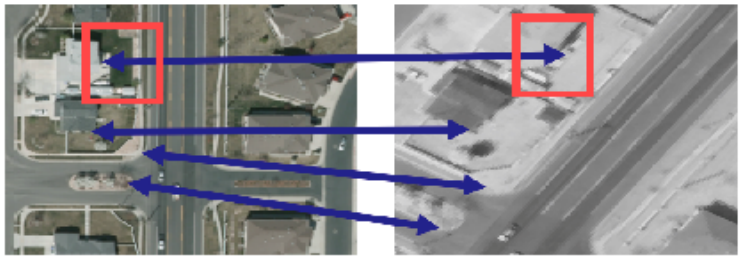
Alexis Lechervy <alexis.lechervy@unicaen.fr> : Maitre de conférence à l'université de Caen.
Thème de recherche :

- Apprentissage automatique,
- Apprentissage de fonction de similarité,
- Apprentissage avec de données multimodales,
- Apprentissage par DeepLearning, Boosting et méthode à noyau...

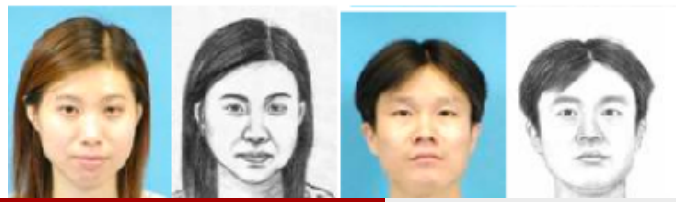
Exemples d'application :

Adaptation de domaine

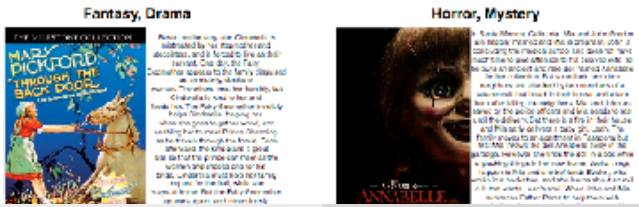
Détection de patch



Reconnaissance photos/dessin



Catégorisation de film



Présentation de la journée

Matinée (8h30-12h30) : Introduction à l'apprentissage profond

1. Introduction générale au Deep Learning,
2. Présentation de PyTorch.

Sommaire

Intelligence artificielle et apprentissage machine

Intelligence artificielle

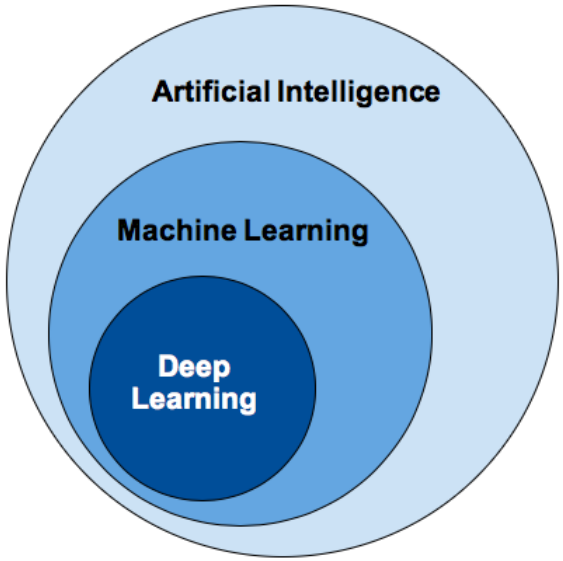
L'idée d'intelligence artificielle a été proposé par Alain Turing dans les années 50. Elle consiste à mettre en oeuvre des techniques informatiques pour simuler et reproduire l'intelligence humaine. Elle permet à des systèmes automatiques de comprendre, apprendre et d'adapter à des situations nouvelles de manière autonome.

Apprentissage automatique

L'apprentissage automatique est une famille de méthode d'intelligence artificielle consistant à donner la capacité à des programmes d'apprendre à partir de données. La connaissance n'est plus fournie directement par le concepteur du programme, mais apprise par ce dernier à partir de données.

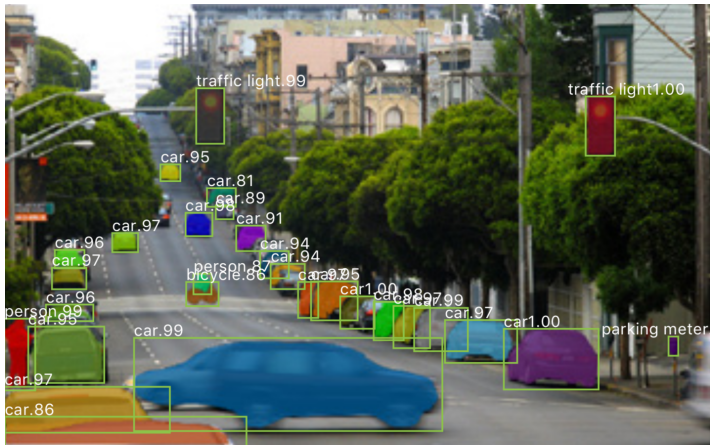
Apprentissage profond

L'apprentissage profond est une méthode particulière d'apprentissage automatique.



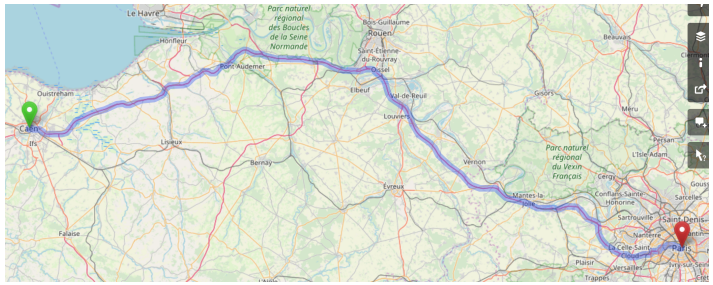
Exemples d'application : Navigation autonome

Analyse sémantique de scène



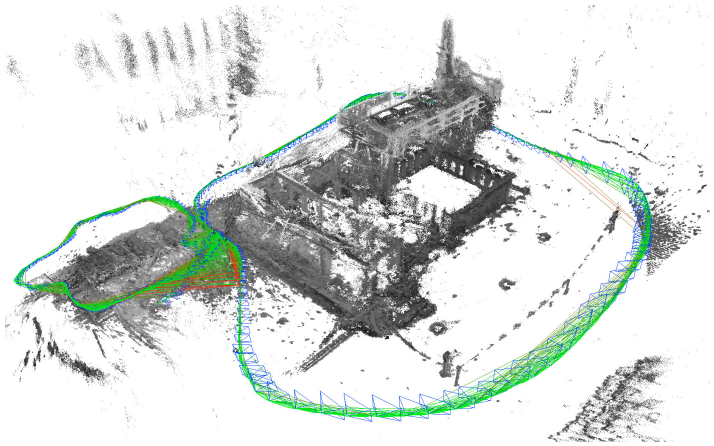
Source : Mask R-CNN

Calcul de trajectoire sous contraintes



Suivi temps réel de personne

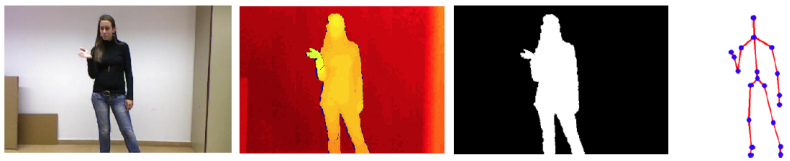
Reconstruction 3D d'environnement



Source : LSD-SLAM: Large-Scale Direct Monocular

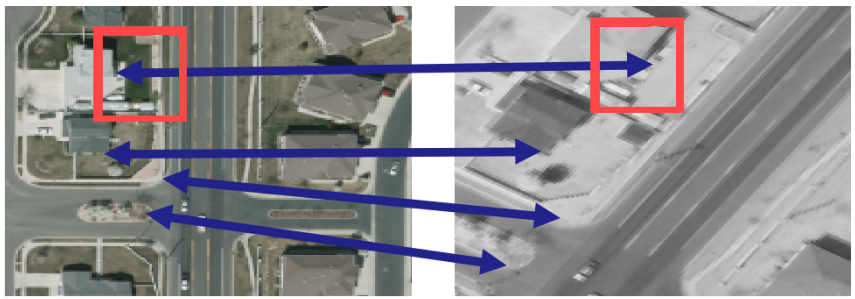
Ex d'application : Analyse et traitement multicapteurs

Reconnaissance d'actions à l'aide de plusieurs capteurs



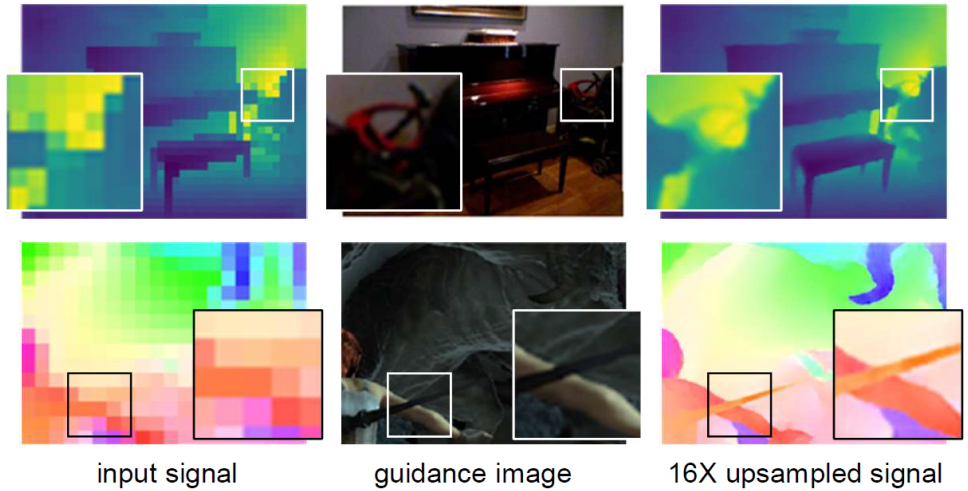
Source : CentralNet: a Multilayer Approach for Multimodal Fusion

Mise en correspondance de modalité différente



Source : Dataset Vedai

Sur-échantillonnage à l'aide d'une autre modalité



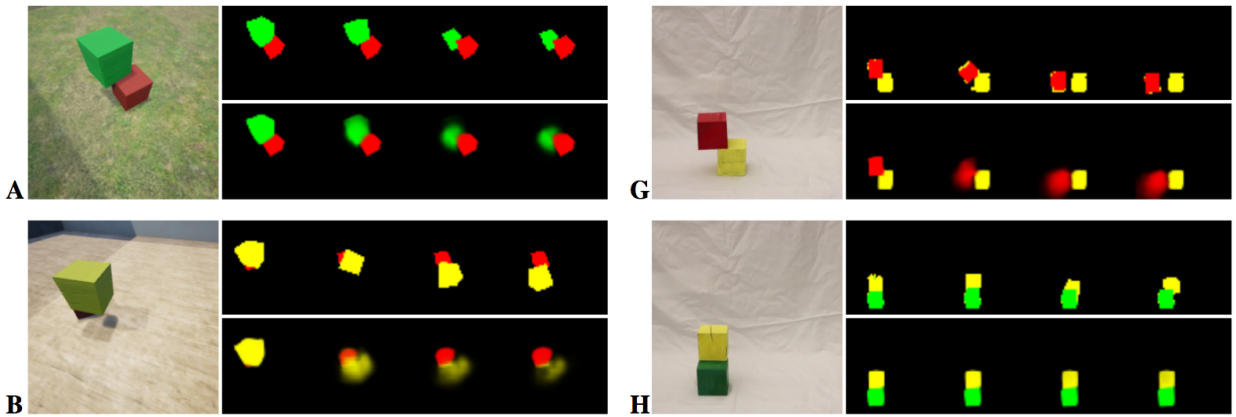
Source :Pixel-Adaptive Convolutional Neural Networks

Ex d'application : Apprentissage à partir de règles

AlphaGo Zero

- En mars 2016, le logiciel AlphaGo de Google DeepMind bat Lee Sedol un des meilleurs joueurs de Go.
- En octobre 2017, AlphaGo Zero bat AlphaGo Lee 100 parties à 0.
- AlphaGo Zero a été appris uniquement en jouant contre elle-même (4,9 millions parties).
- Article Mastering the game of Go without human knowledge dans Nature

PhysNet



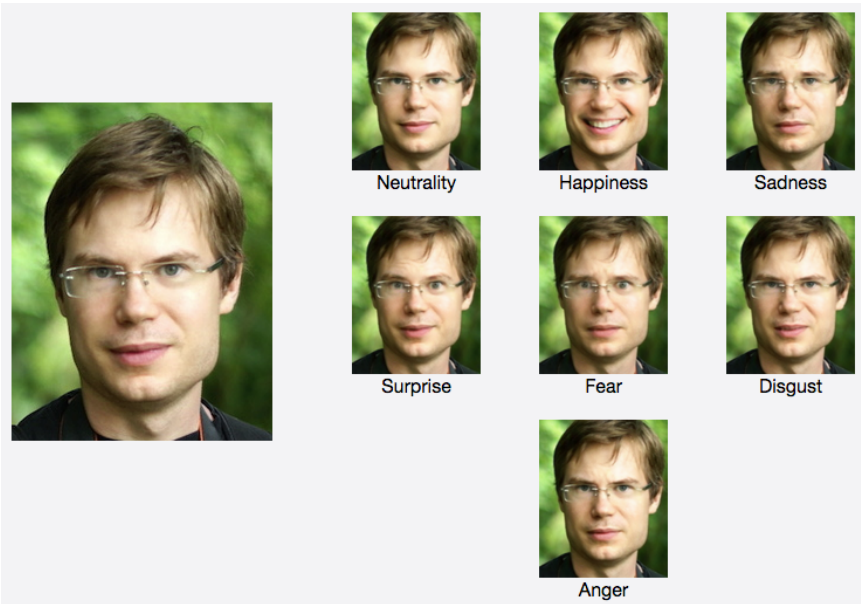
Source : Learning Physical Intuition of Block Towers by Example

Ex d'application : Application sur des visages

Similarité d'images de visages

Source Large Scale Metric Learning from Equivalence Constraints

Synthèse de visage avec émotion spécifique

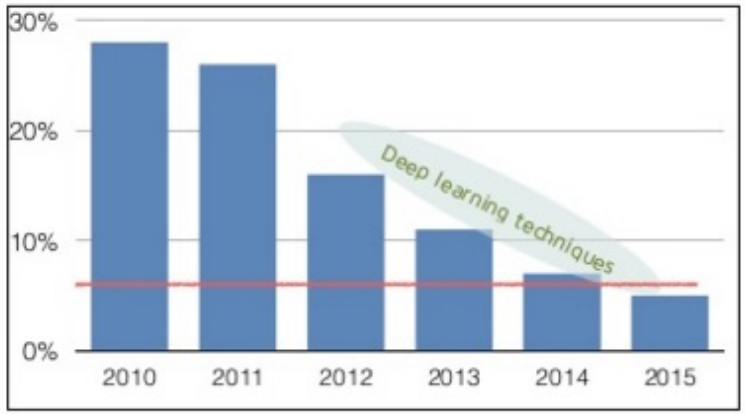


Source CAKE: Compact and Accurate K-dimensional representation of Emotion

Historique des réseaux de neurones en informatique

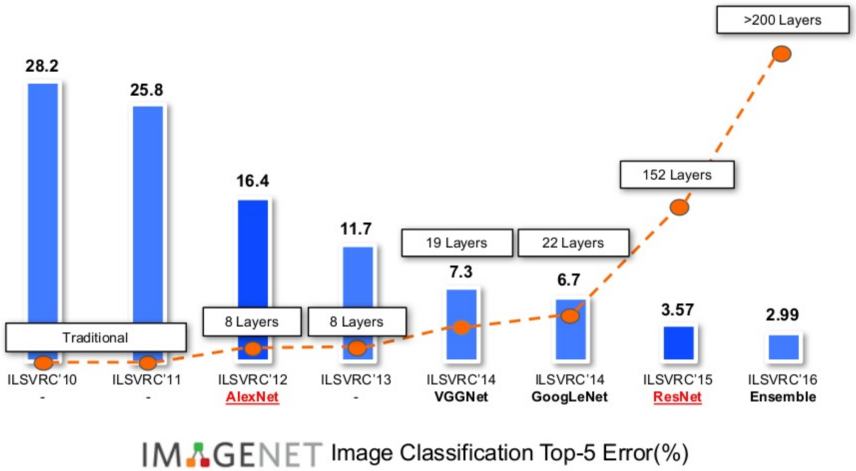
Histoire des réseaux de neurones

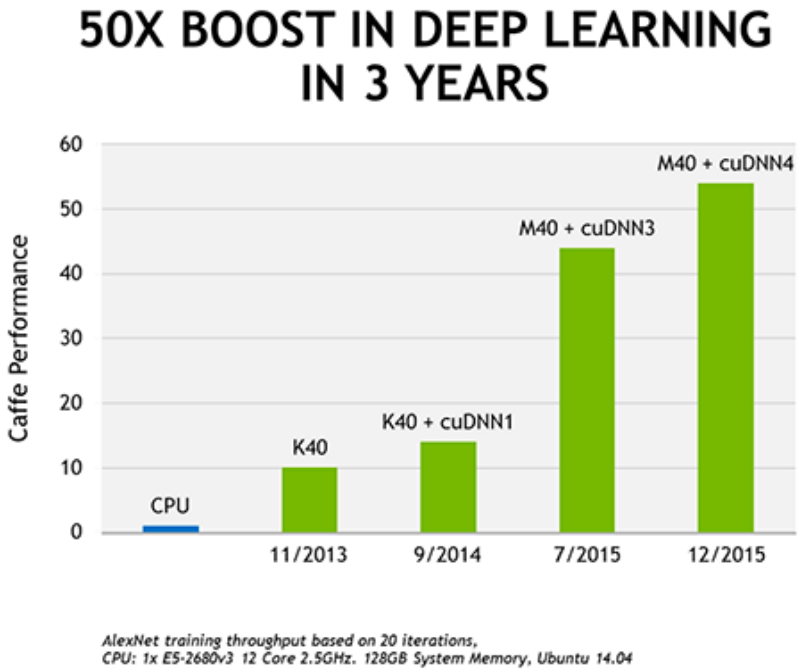
- 1943 : le neurone formel (Mc Culloch et Pitts),
- 1948 : Les réseaux d'automates (Von Neuman),
- 1949 : La règle de Hebb,
- 1958 : Le perceptron de Rosenblatt,
- 1969 : Les limites du Perceptron et des classifieurs linéaires (Minsky et Papert),
- 1986 : Les perceptrons multicouches et la rétro-propagation du gradient (Lecun, Mc Clelland, Werbos et Rumelhar),
- 1998 : Les réseaux convolutionnels et Lenet 5 (Lecun),
- 2012 : Implémentation sur GPU de réseaux profonds (Ciresan et al.).



ImageNet challenge error rates
Photo by: red line = human performance

La "révolution" du Deep Learning



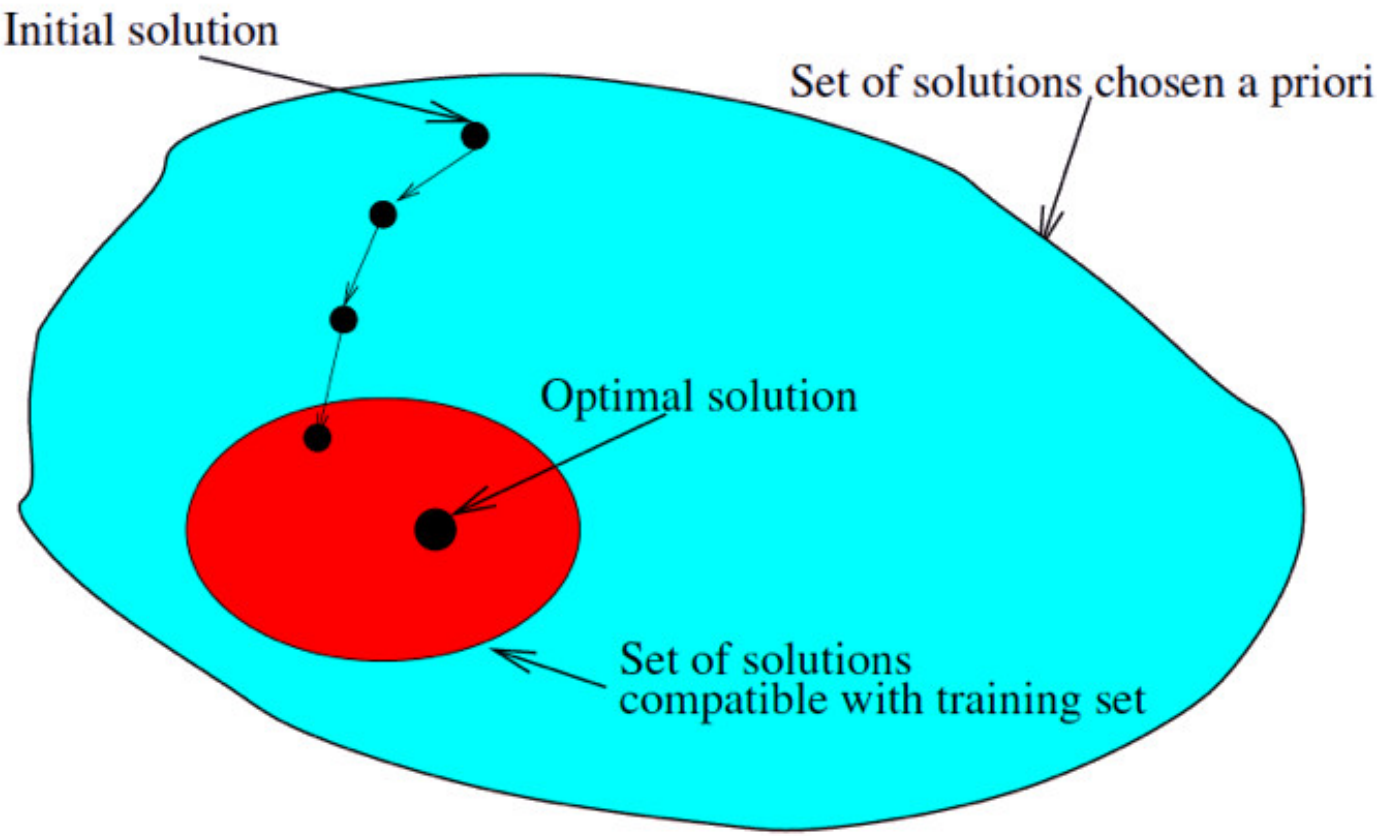


Sommaire

Sommaire

L'apprentissage : une recherche d'une solution parmi un ensemble de possibilité.

L'apprentissage consiste à choisir une solution parmi un ensemble de solution défini selon un apriori. On recherche la solution optimale. C'est à dire la solution "la plus simple" qui répond à notre problème.



Pourquoi ne pas faire une approche bio-inspirée ?

But

Le cerveau humain excelle dans l'apprentissage de nouvelle situation. Pourquoi ne pas s'en inspirer pour élaborer une méthode d'apprentissage ?

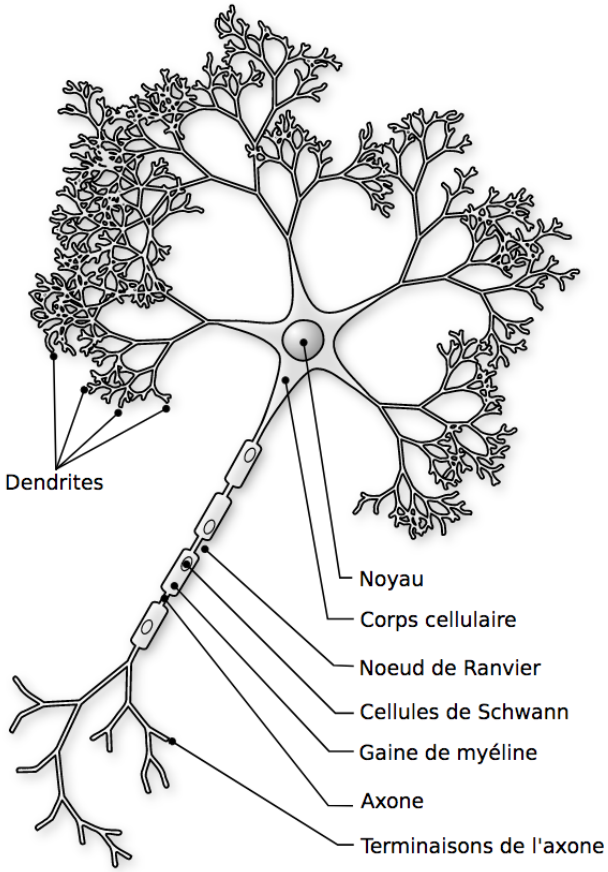
Les avantages d'un "cerveau naturel"

- Flexible, peut s'adapter à de nouvelles données .
- Robuste et tolérant aux erreurs dans la base d'apprentissage.
- Peu fonctionner avec des données incomplètes ou en partie fausse.
- A des grandes capacités d'apprentissage.
- Est rapide et utilise un système massivement parallèle.
- Repose sur des éléments dont le fonctionnement est relativement simple et facilement reproductible.

Un neurone réel

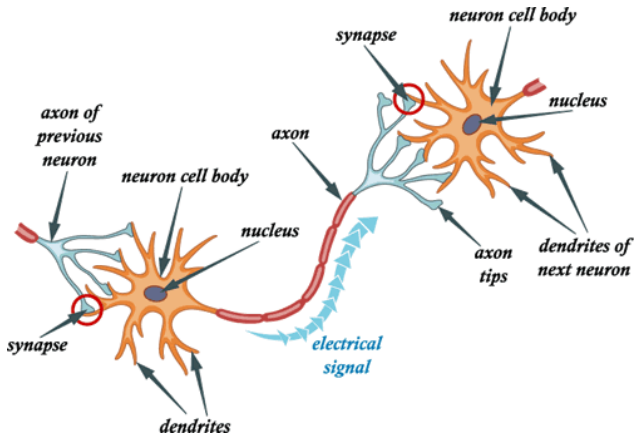
Composition simplifié d'un cerveau humain

- 100 milliards de neurones,
- en moyenne 10.000 connexion par neurones.



Composition d'un neurone

1. Des **dendrites** qui réceptionne les impulsions nerveuses.
2. Un **corps cellulaire** qui active ou non le neurone en fonction des entrées.
3. Un **axone** qui propage l'état d'activation du neurone.



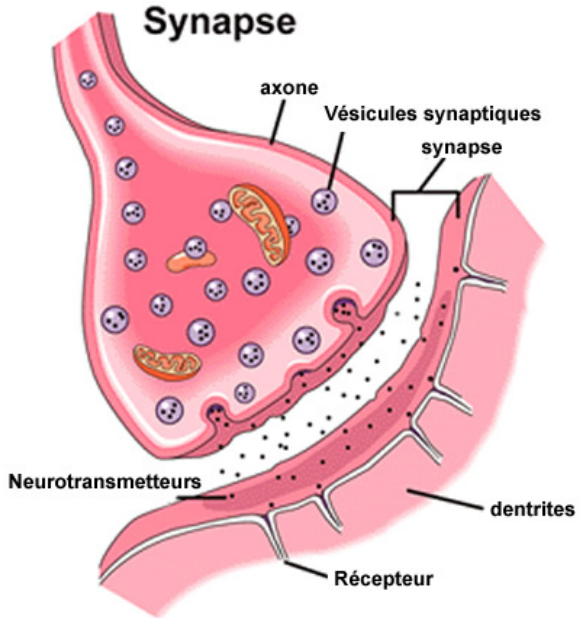
L'apprentissage et la règle de Hebb

Plasticité synaptique

Hebb propose en 1949 le modèle d'apprentissage par plasticité synaptique. Il suppose que l'efficacité synaptique augmente lorsque l'axone d'un neurone A entraîne de façon répétée une activation d'un neurone B :

Faisons l'hypothèse qu'une activité persistante et répétée d'une activité avec réverbération (ou trace) tend à induire un changement cellulaire persistant qui augmente sa stabilité. Quand un axone d'une cellule A est assez proche pour exciter une cellule B de manière répétée et persistante, une croissance ou des changements métaboliques prennent place dans l'une ou les deux cellules ce qui entraîne une augmentation de l'efficacité de A comme cellule stimulant B.

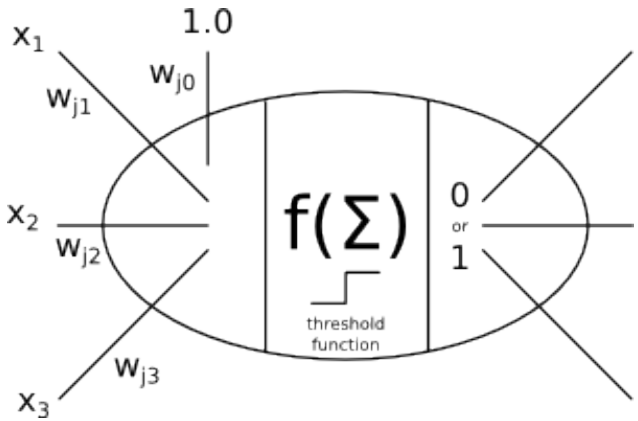
— Donald Hebb, 1942.



Du neurone réel au neurone formel

Le neurone formel

- Un neurone possèdent des entrées $x_1, \cdots, x_d$.
- Le signal de chaque entrée est pondéré par un poids $w_1, \cdots, w_d$.
- La sortie dépend des entrées et des poids.
- La sortie est un signal d'activation ou non du neurone (par exemple 0 ou 1).

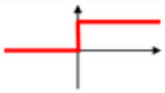
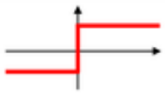
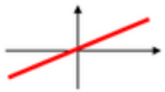
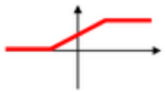
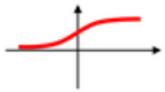
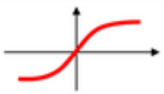


Composition d'un neurone formel

Une fonction d'activation

La fonction d'activation ϕ permet de mettre l'état du neurone entre 0 et 1 ou -1 et 1 (un courant passe ou non)

Exemple de fonction d'activation possible

Activation function	Equation	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	
Linear	$\phi(z) = z$	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1+e^{-z}}$	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	

Sortie du neurone

$$f(x) = \phi(\langle w, x \rangle + w_0)$$

Apprentissage du neurone

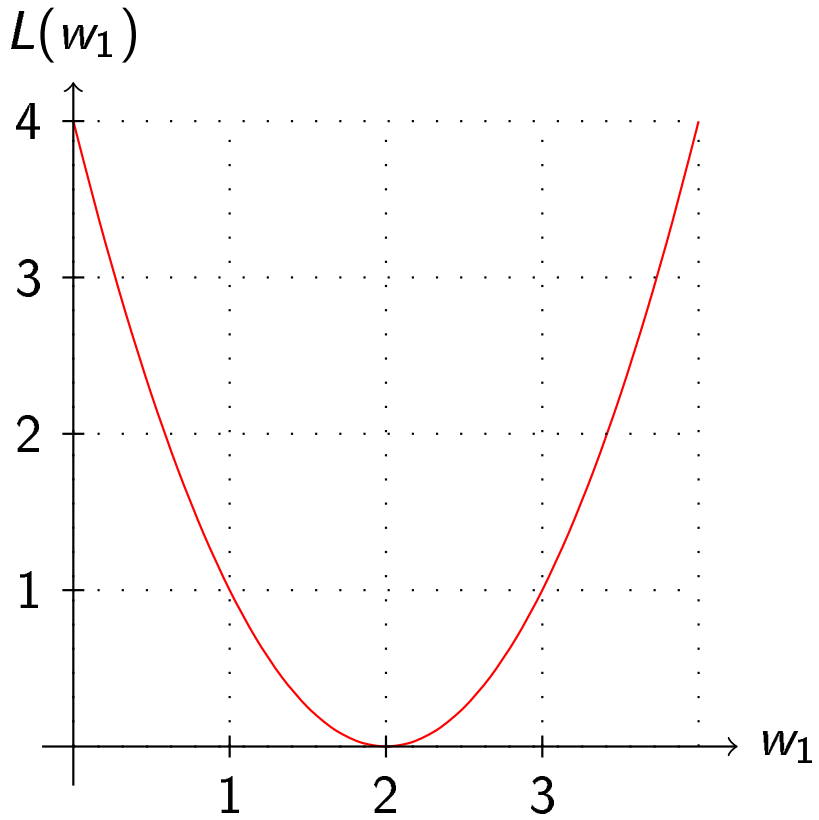
On cherche à minimiser le nombre d'erreur sur les données d'apprentissage :

$$\arg\min_{w} L(f_w(x), y)$$

Comment résoudre $\arg\min_w L(f_w(x), y)$

Idée

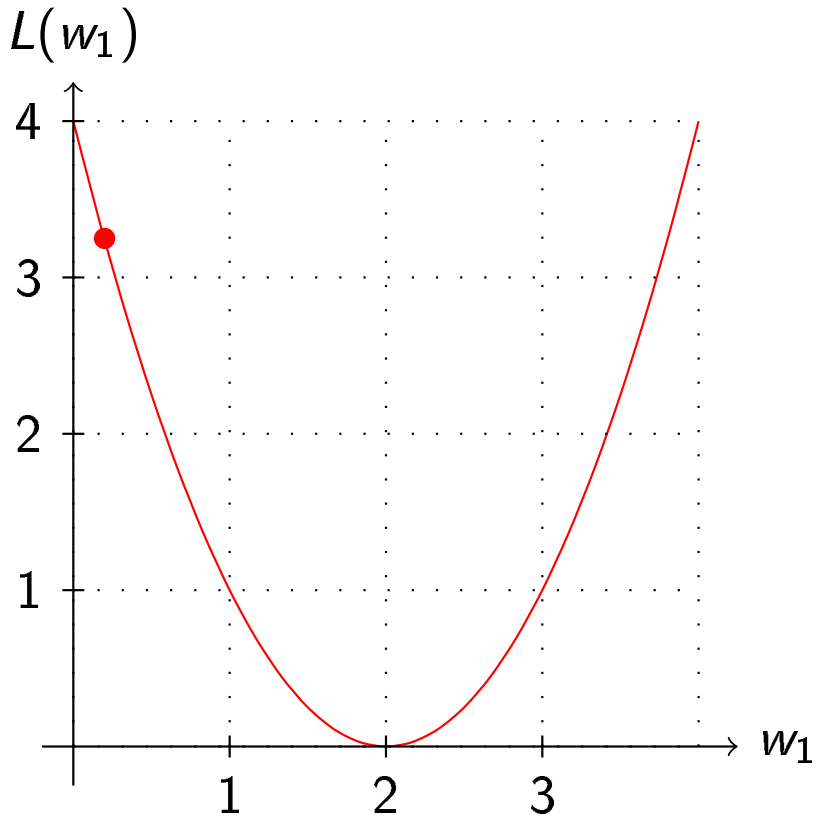
On peut atteindre le minimum en se dirigeant itérativement dans la direction **de plus forte pente**.



Comment résoudre $\arg\min_w L(f_w(x), y)$

Idée

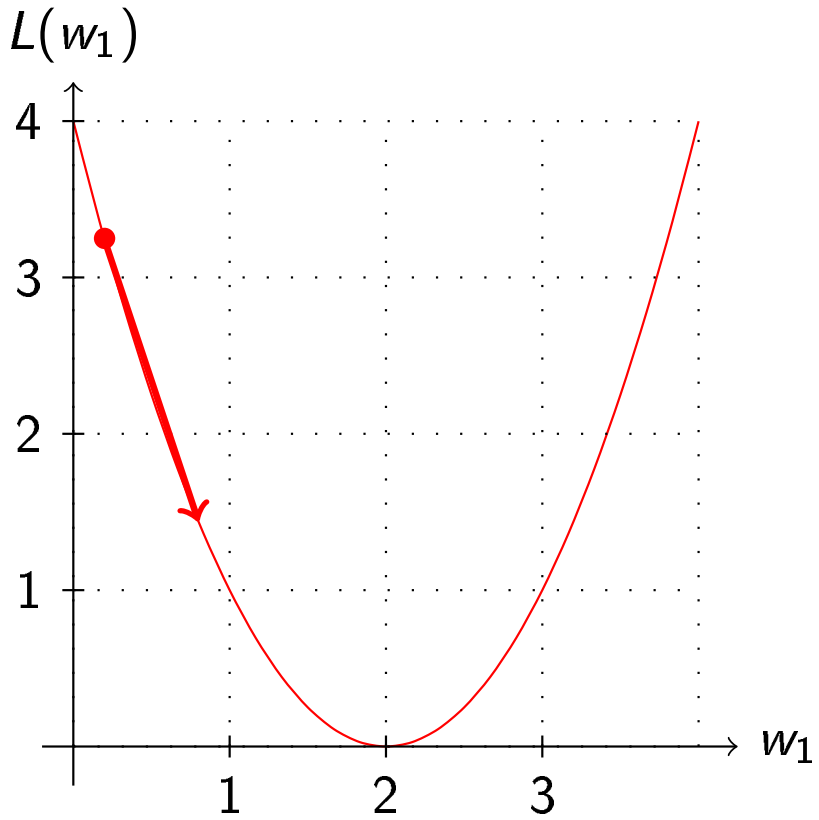
On peut atteindre le minimum en se dirigeant itérativement dans la direction **de plus forte pente**.



Comment résoudre $\arg\min_w L(f_w(x), y)$

Idée

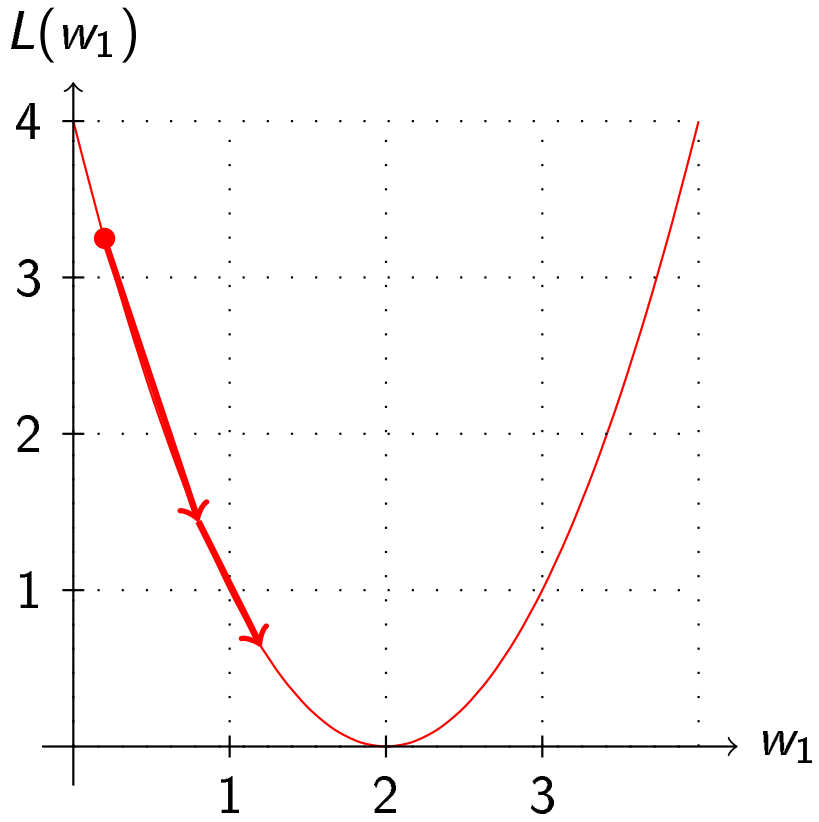
On peut atteindre le minimum en se dirigeant itérativement dans la direction **de plus forte pente**.



Comment résoudre $\arg\min_w L(f_w(x), y)$

Idée

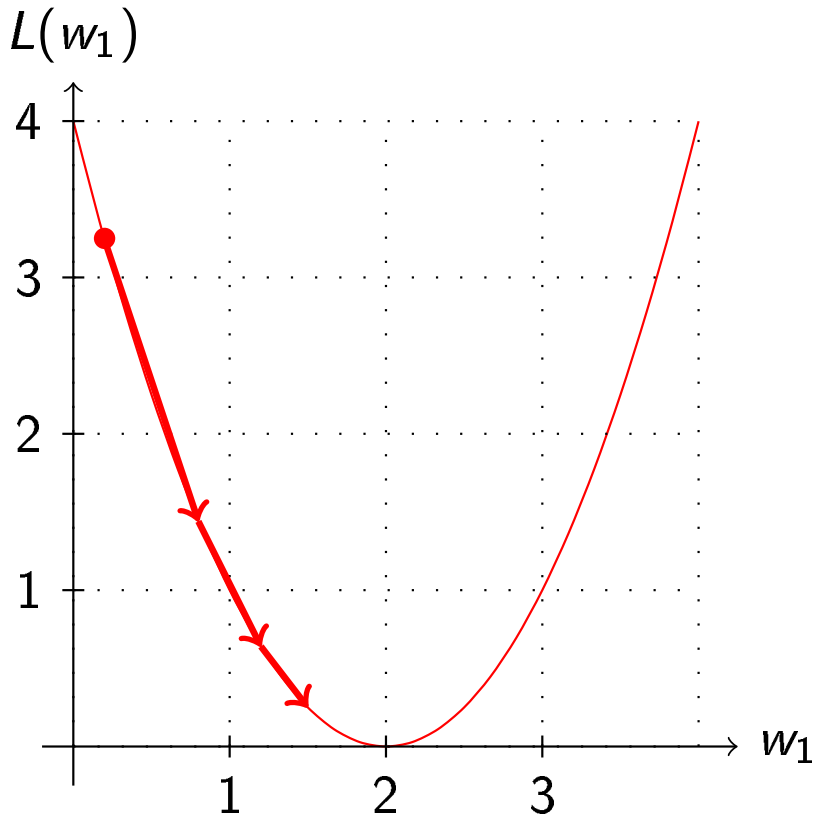
On peut atteindre le minimum en se dirigeant itérativement dans la direction **de plus forte pente**.



Comment résoudre $\arg\min_w L(f_w(x), y)$

Idée

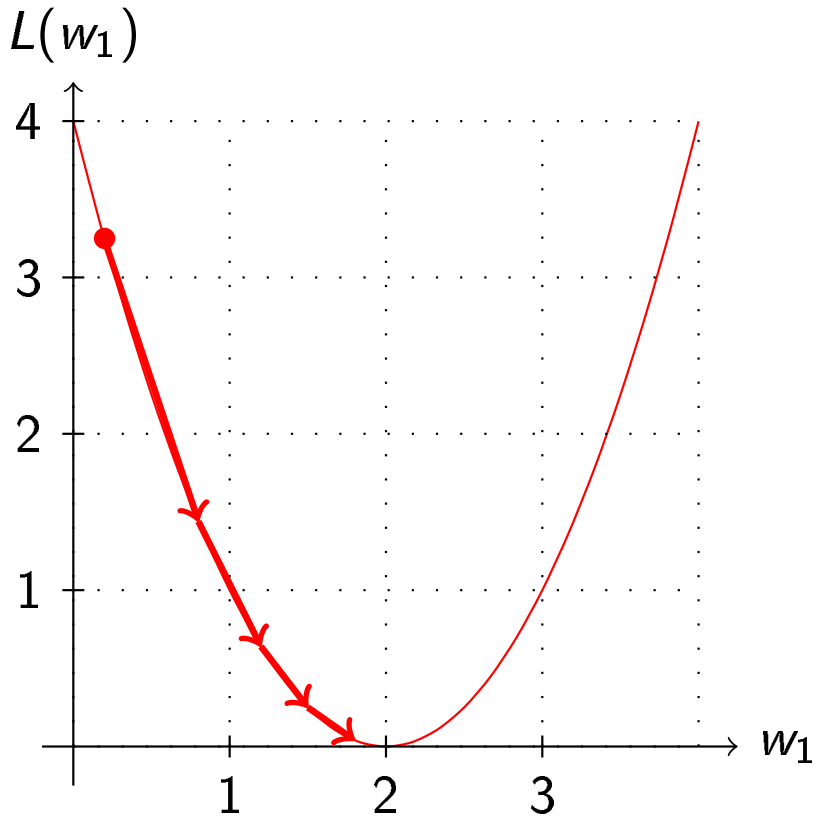
On peut atteindre le minimum en se dirigeant itérativement dans la direction **de plus forte pente**.



Comment résoudre $\arg\min_w L(f_w(x), y)$

Idée

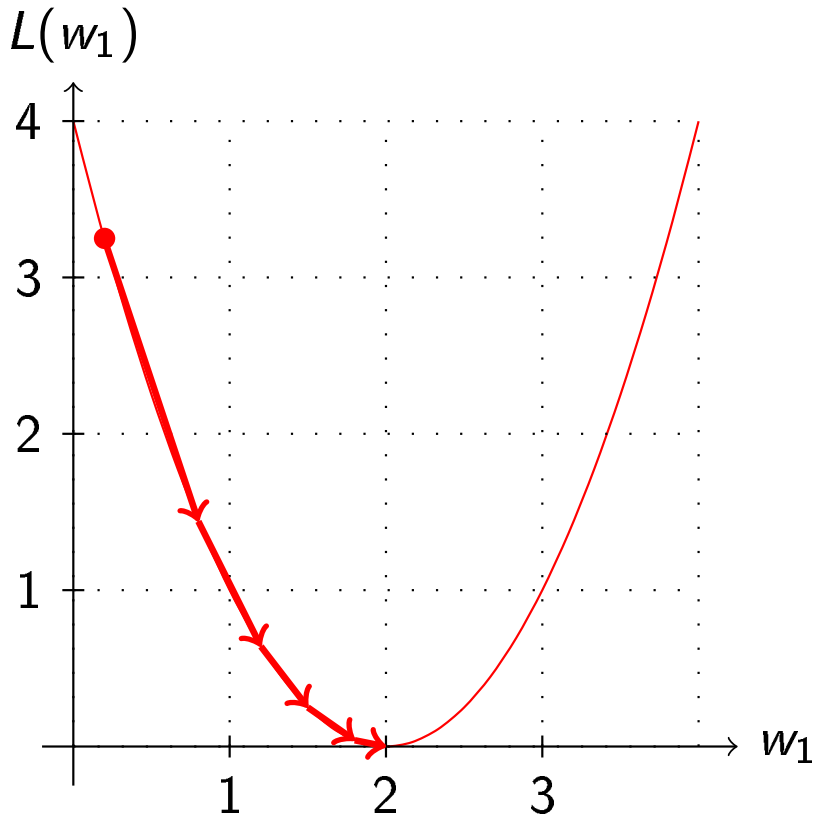
On peut atteindre le minimum en se dirigeant itérativement dans la direction **de plus forte pente**.



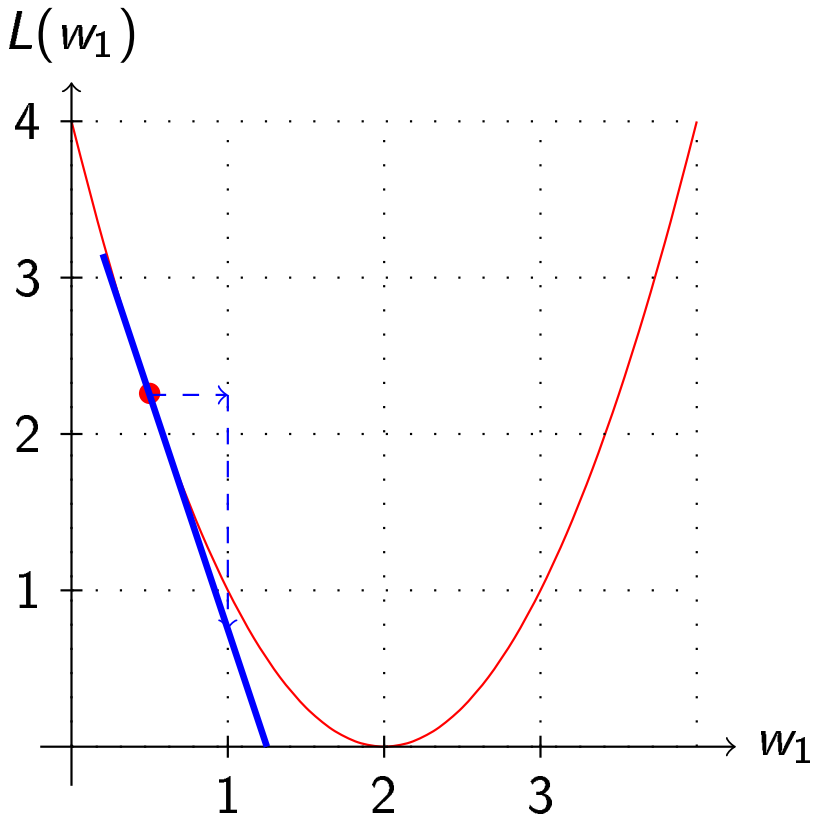
Comment résoudre $\arg\min_w L(f_w(x), y)$

Idée

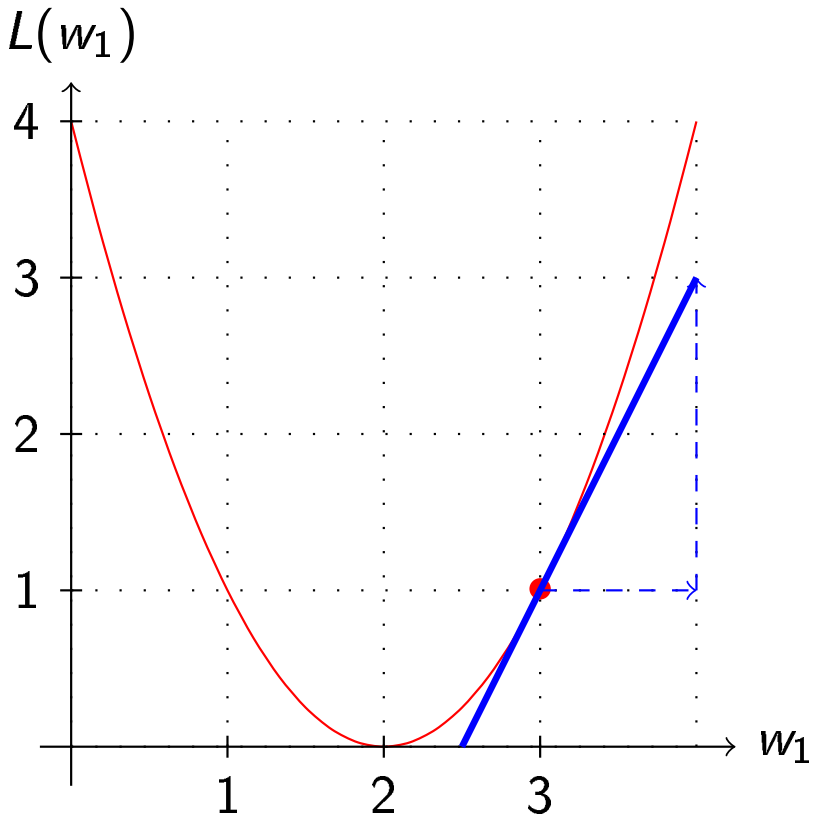
On peut atteindre le minimum en se dirigeant itérativement dans la direction **de plus forte pente**.



Direction de la pente.

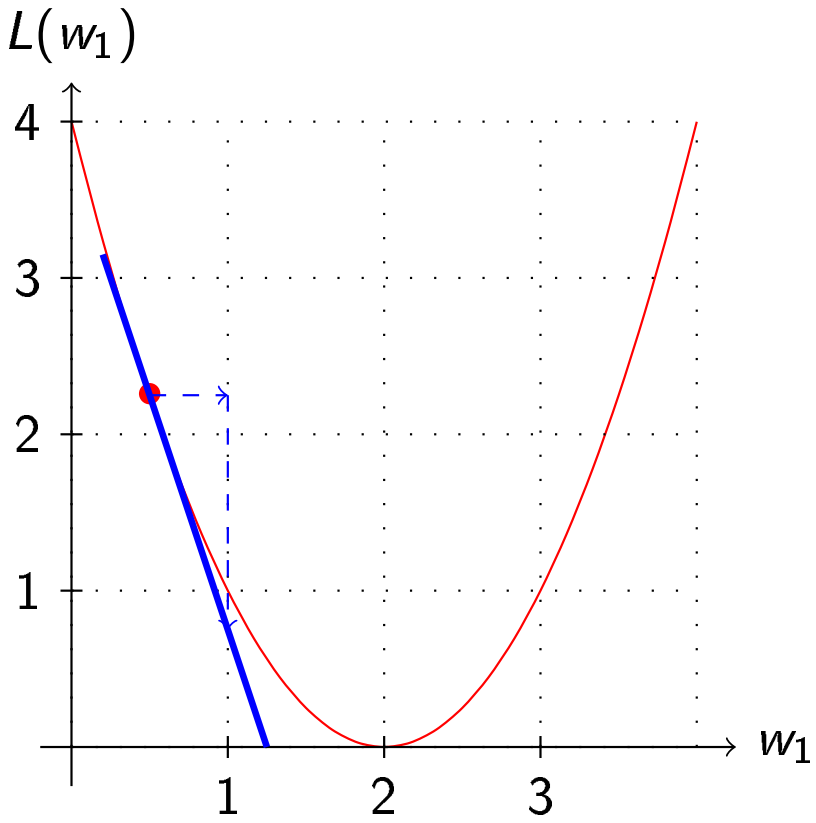


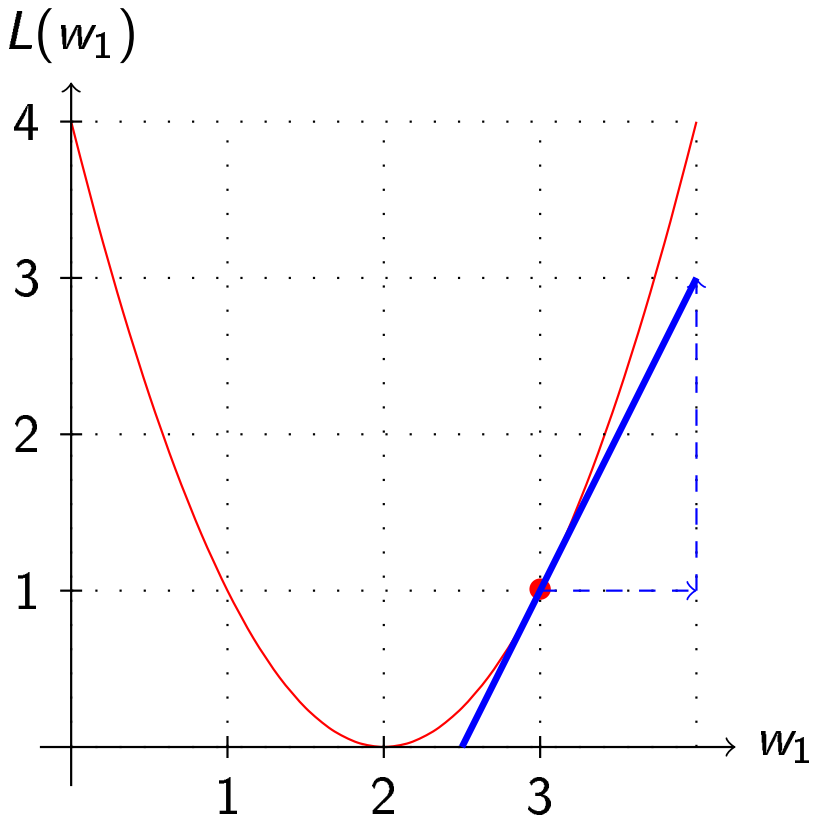
Valeur de la dérivée en 0.5 : -3
Direction de la pente : positive.



Valeur de la dérivée en 3 : 2
Direction de la pente : négative.

Direction de la pente.





Comment trouver la direction de la pente ?

Le signe opposé de la dérivée nous donnes la direction de la pente.

Résolution par descente de gradient

Problème d'optimisation à résoudre

$$\arg\min_{w} L(f_w(x), y).$$

Solution par descente de gradient

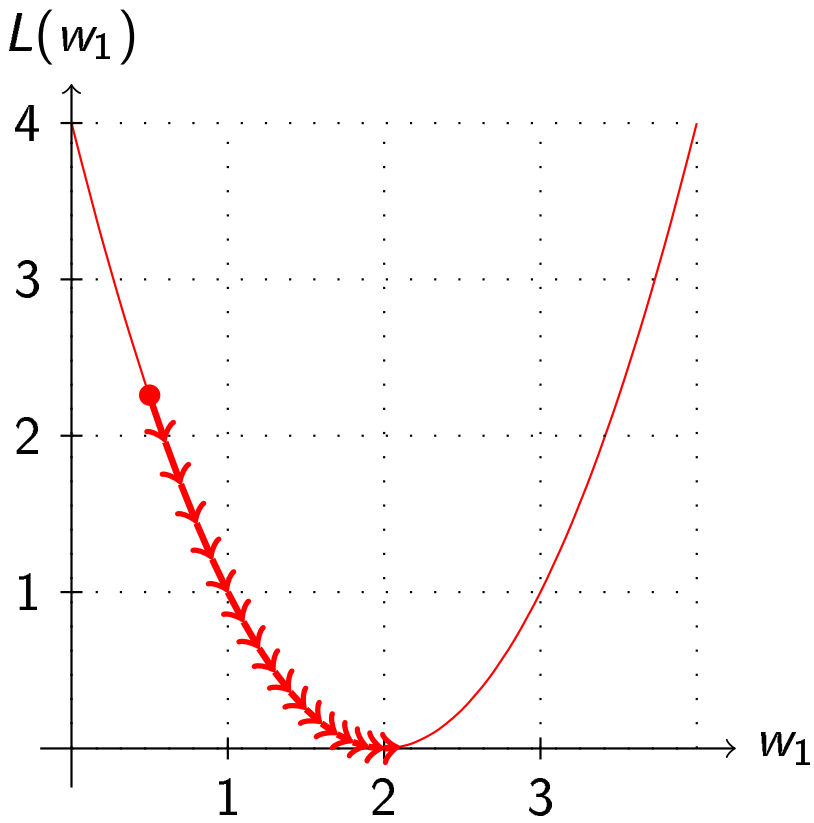
Initialiser avec un w choisi aléatoirement.
Répéter jusqu'à convergence :

$$w_i := w_i - \eta \frac{\partial}{\partial w_i} L(f_w(x), y) \quad \forall i.$$

η est une constante correspondant au taux d'apprentissage.
Lorsque l'on s'approche du minimum la dérivée tend vers 0, en conséquence les pas sont de plus en plus petit. Il n'est pas nécessaire d'avoir une valeur de η qui décroit.

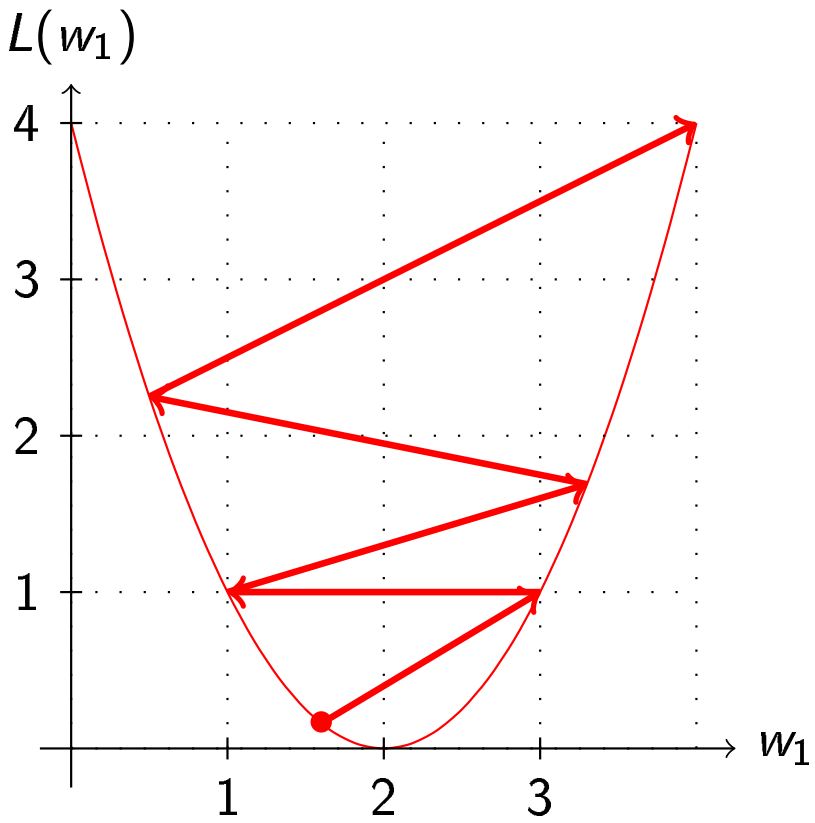
Bien choisir la valeur de η

Valeur de η trop petite



La convergence est très lente.

Valeur de η trop grande



L'algorithme diverge.

Formulation du problème d'optimisation du perceptron

Exemple de choix possible pour définir un neurone formel

- Fonction d'activation : la fonction linéaire
- Fonction de coût de l'apprentissage : $L(f(x), y) = \sum_i \max(0, -y_i f(x_i))$.

Apprentissage du neurone

- On cherche à minimiser la fonction de coût pour trouver les w et b optimaux.
- Cela peut être fait itérativement par une descente de gradient.

On a donc :

1. $f(x) = \langle w, x \rangle + w_0$,
2. $L(f(x), y) = \sum_i \max(0, -y_i \ (\langle w, x_i \rangle + w_0))$,
3. $$\frac{\partial L(f(x_i), y_i)}{\partial w} = \sum_i \begin{cases} 0 & \text{si} \quad y_i f(x_i) > 0 \\ -y_i x_i & \text{sinon} \end{cases}.$$
4. Mise à jour par descente de gradient :
$$w_{t+1} = w_t + \eta \sum_i \begin{cases} 0 & \text{si} \quad y_i f(x_i) > 0 \\ y_i x_i & \text{sinon} \end{cases}.$$

Algorithme du Perceptron (dans le primal)

La formule d'apprentissage que l'on vient de trouver correspond exactement au Perceptron de Rosenblatt.

Algorithme

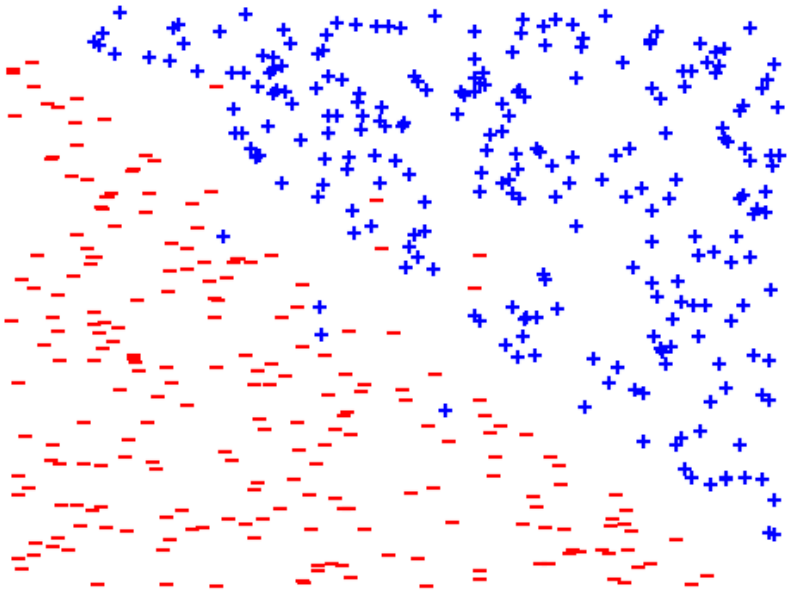
Data: Un ensemble $\mathcal{S}$ d'apprentissage linéaire séparable et un taux $\eta \in \mathbb{R}^+$
Initialisation : $w_0 \longleftarrow 0$; $b_0 \longleftarrow 0$; $k \longleftarrow 0$;
$R \longleftarrow \max_{1 \leq i \leq n} \|x_i\|$;
repeat
  for $i = 1, \cdots, n$ **do**
    if $y_i(\langle w_k, x_i \rangle + b_k) \leq 0$ **then**
      $w_{k+1} \longleftarrow w_k + \eta y_i x_i$;
      $b_{k+1} \longleftarrow b_k + \eta y_i R^2$;
      $k \longleftarrow k + 1$;
    end
  end
until *aucune erreur pour une itération de la boucle*;
Result: l'hyperplan (w_k, b_k)

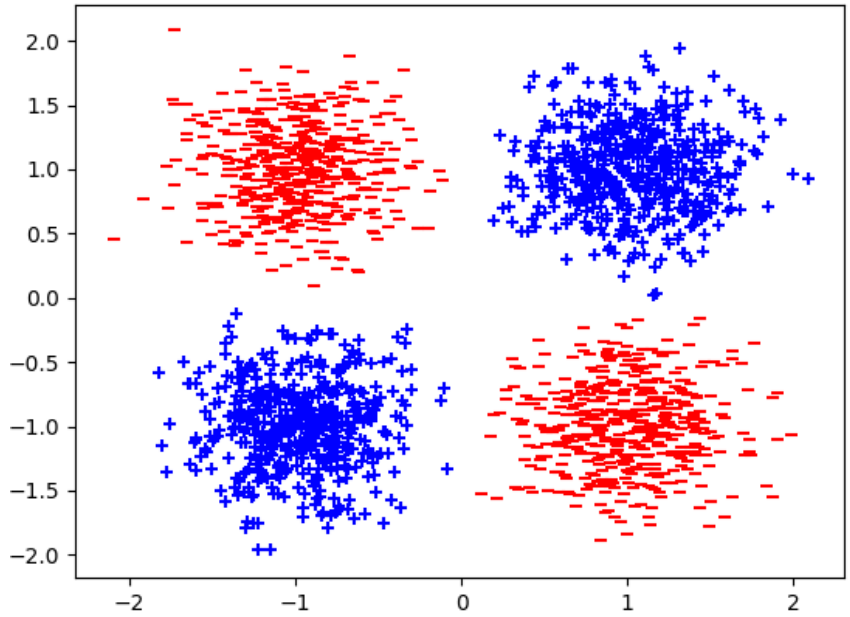
Sommaire

Les limites d'un seul neurone formel

Limites

- L'algorithme ne converge pas si les données ne sont pas linéairement séparables.
- Ne peut donc par résoudre un problème quasi-linéaire.
- Pas de garantie de bonne généralisation et de maximisation de marge.
- Ne peut par résoudre de problème de type XOR.

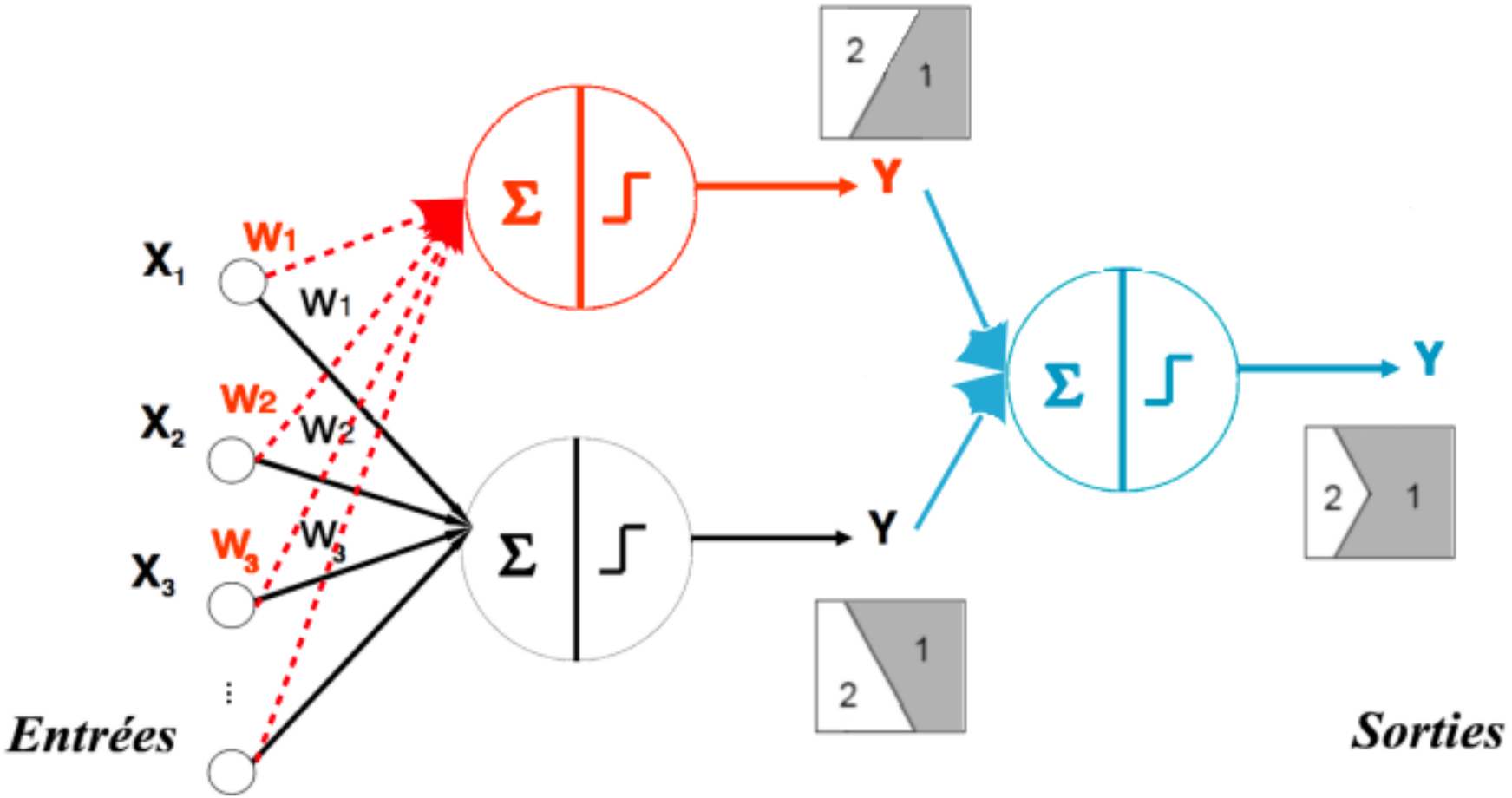




Du neurone au réseau de neurones

Une idée pour résoudre des séparations non linéaire

Un réseau à deux couches :



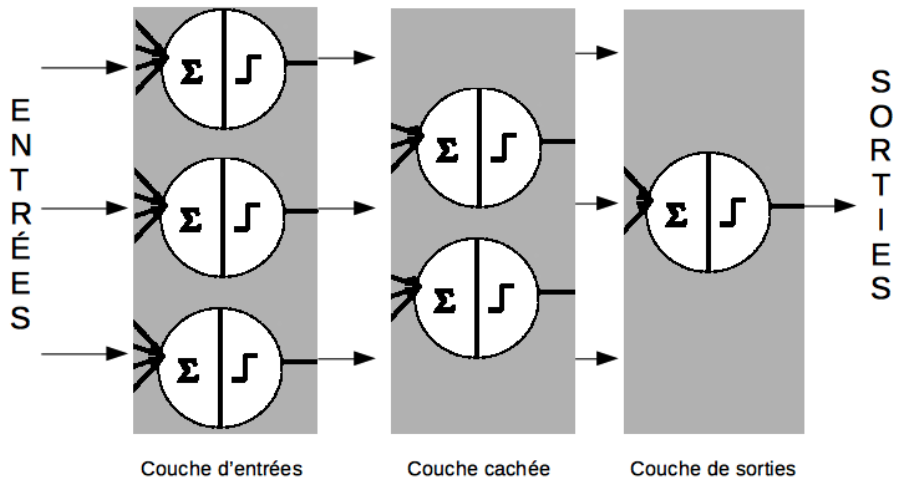
Les réseaux de neurones non bouclé

Définition

Un réseau de neurones non bouclé est un enchainement de neurone consistant à appliquer une somme pondérée de ses entrées suivis d'une fonction non linéaire Il est représenté graphiquement par un ensemble de neurones connectés entre eux, l'information circulant des entrées vers les sorties sans retour en arrière possible.
On distinguera trois types de couches neuronales :

1. la couche d'entrée,
2. les couches cachées,
3. la couche de sortie.

Exemple de réseau de neurones non bouclé



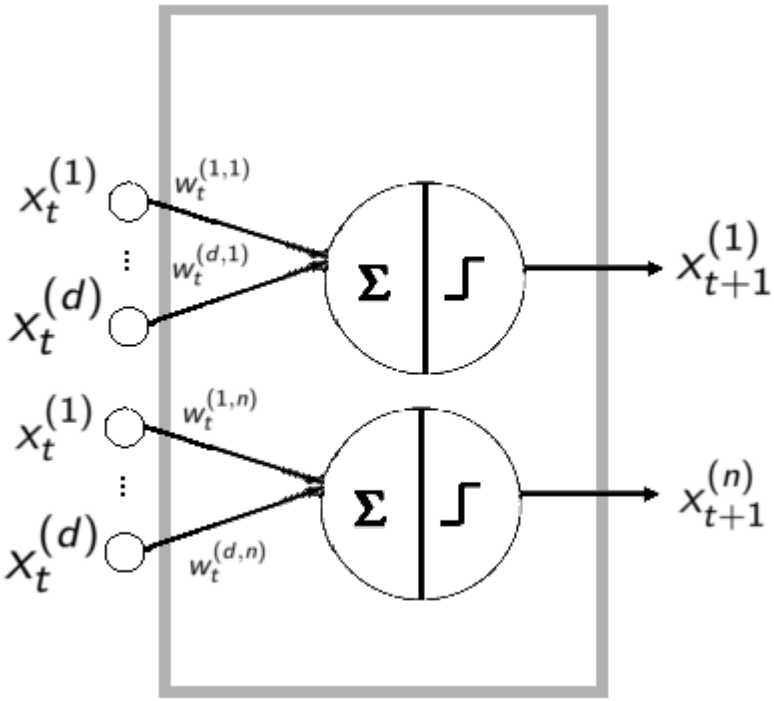
Le perceptron multi-couches

Le perceptron multi-couches

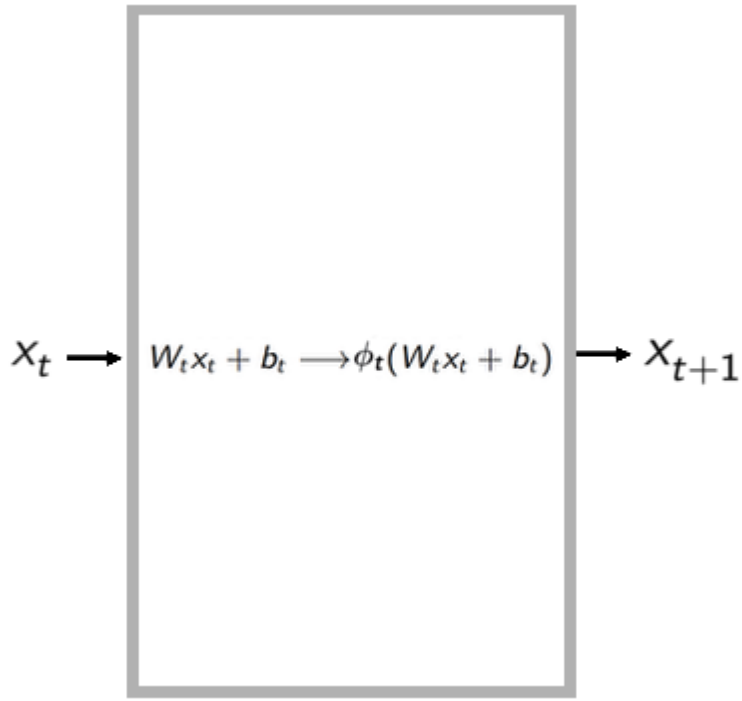
- Le perceptron multicouches (multilayer perceptron MLP) est un réseau de neurones non bouclé organisé en couches.
- Chaque neurone d'une couche est relié à tous les neurones de la couche suivante.
- L'information circule de l'entrée vers les sorties en passant successivement dans chaque couche.
- Tous les neurones d'une même couche partage la même fonction d'activation.

Une couche d'un réseau de neurone

Représentation avec des neurones



Représentation par couche



Calcul des sorties du réseau

Propagation

On calcule les sorties du réseau en propageant les valeurs des entrées x_0 de couche en couche :

1. Entrées : x_0
2. Couche 1 : $x_1 := \phi_1(W_1 x_0 + b_1)$
3. Couche 2 : $x_2 := \phi_2(W_2 x_1 + b_2)$
4. $\vdots$
5. Sortie : $F(x) := \phi_t(W_T x_{T-1} + b_T)$

Apprentissage des paramètres du réseau

Apprentissage

- Le but de l'apprentissage est de trouver les poids des matrices W_t et b_t de chacune des couches en fonction des données d'apprentissages.
- On cherche à minimiser une fonction d'erreur mesurant la différence entre les valeurs prédites et les valeurs attendues.
- L'apprentissage est donc un problème d'optimisation qui cherchent à trouver les valeurs des W en minimisant une fonction de coût sous contrainte de la topologie du réseau.

Remarques sur l'apprentissage du réseau

Apprendre un réseau comme un perceptron ?

- Pour l'apprentissage du perceptron, on mettait à jours le neurone en comparant la valeur attendue et la valeur obtenue.
- En introduisant une couche supplémentaire, comment mettre à jours les poids en fonction uniquement de l'impact de la couche sur la sortie ?
- On ne connaît a priori pas la valeur "attendue" en sortie des couches cachés.
- On ne peut donc pas utiliser la même technique d'apprentissage que le perceptron.

Idée de la solution

- Partir de la sortie que l'on compare avec la sortie désirée.
- Mettre à jours la dernière couche puis rétropropager l'erreur en remontant couche par couche.

Apprentissage par rétro-propagation du gradient

La rétro-propagation du gradient

- On effectue une descente de gradient sur la fonction de coût en fonction des poids de chacune des couches.
- On prend en compte les poids de toutes les couches (aussi bien la couche de sortie que les couches cachées).
- On commence par mettre à jour la couche de sortie en fonction de ces entrées puis on rétropropage l'erreur aux couches d'avant.

Mise à jours des poids du réseau

Mise à jours par descente de gradient :

$$W_t \longleftarrow W_t - \lambda \frac{\partial L(F(x), y)}{\partial W_t}, b_t \longleftarrow b_t - \lambda \frac{\partial L(F(x), y)}{\partial b_t}$$

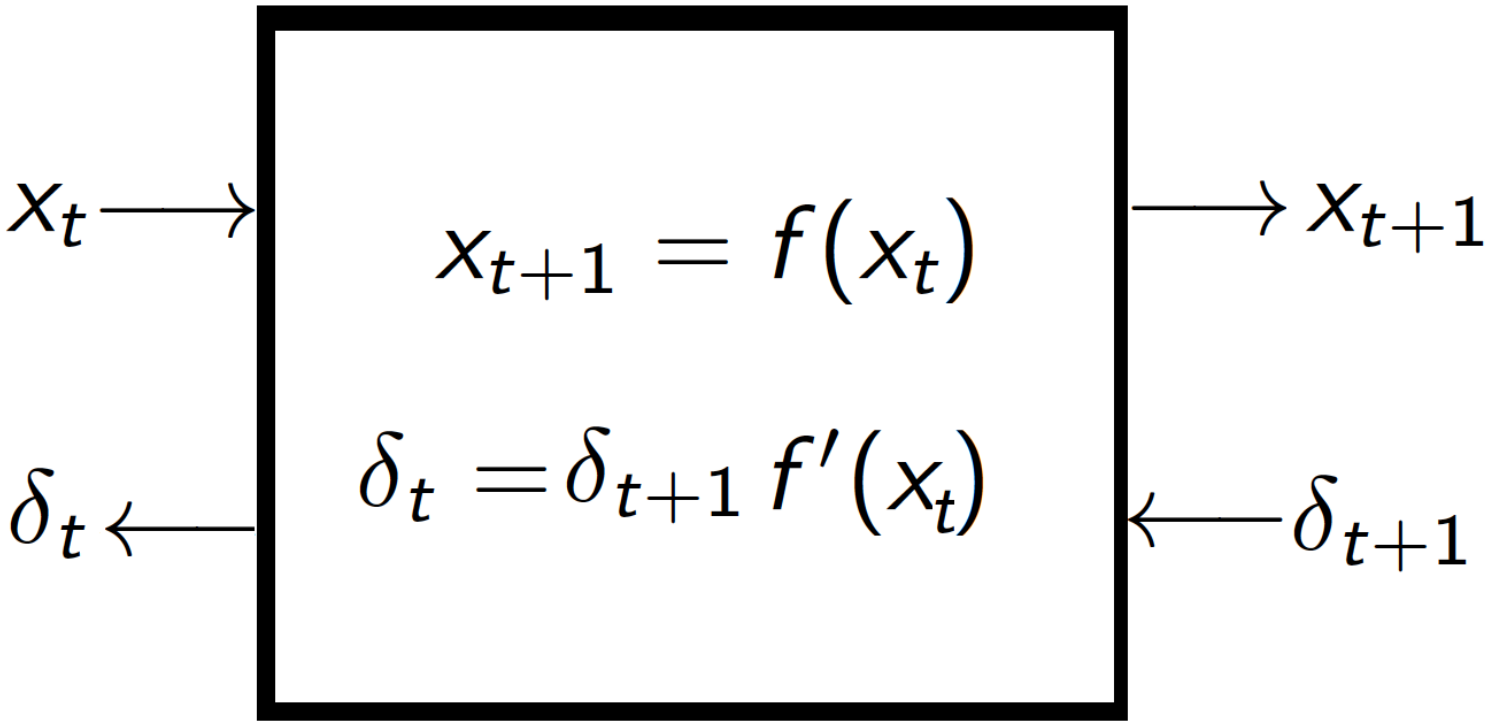
λ est un taux d'apprentissage fixé comme paramètre d'apprentissage.

$\Longrightarrow$ Comment calculer $\frac{\partial L(F(x), y)}{\partial W_t}$ et $\frac{\partial L(F(x), y)}{\partial b_t}$?

La vision en bloc d'un réseau de neurone

Conséquence

On peut voir un réseau de neurone comme un enchaînement de bloc définis par :



Algorithme d'apprentissage d'un réseau de neurone

1. Choisir un taux d'apprentissage η.
2. On initialise au hasard les paramètres du réseau (W_t, b_t).
3. Pour chaque itération faire :
   1. Propager les exemples d'apprentissages dans le réseau (calculer les x_t),
   2. Rétro-propager les gradients dans le réseau (calculer les δ_t).
   3. Mettre à jours les paramètres du réseau

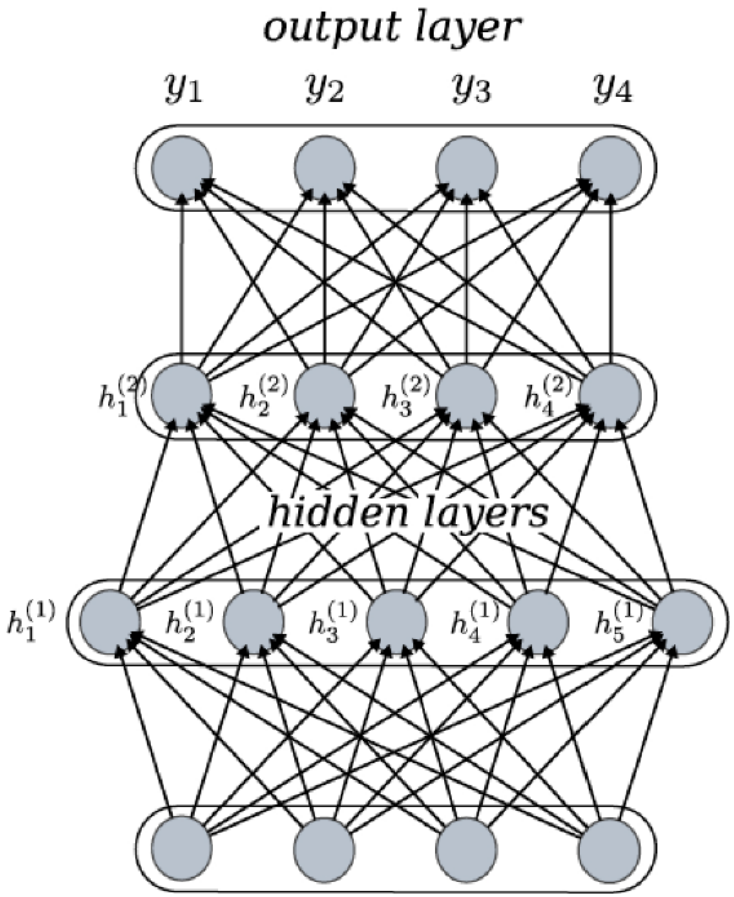
$$W_t \longleftarrow W_t - \eta \delta_t x_t^\top,$$

$$b_t \longleftarrow b_t - \eta \sum \delta_t$$

Les réseaux de neurones : des approximateurs universels

Propriétés d'approximateurs universels

George Cybenko montre en 1989 que l'on peut, avec un réseau à une couche cachée avec un nombre fini de neurones, approximer toutes fonctions continues sur des sous-ensembles compacts de R^n



Inconvénients des MLP

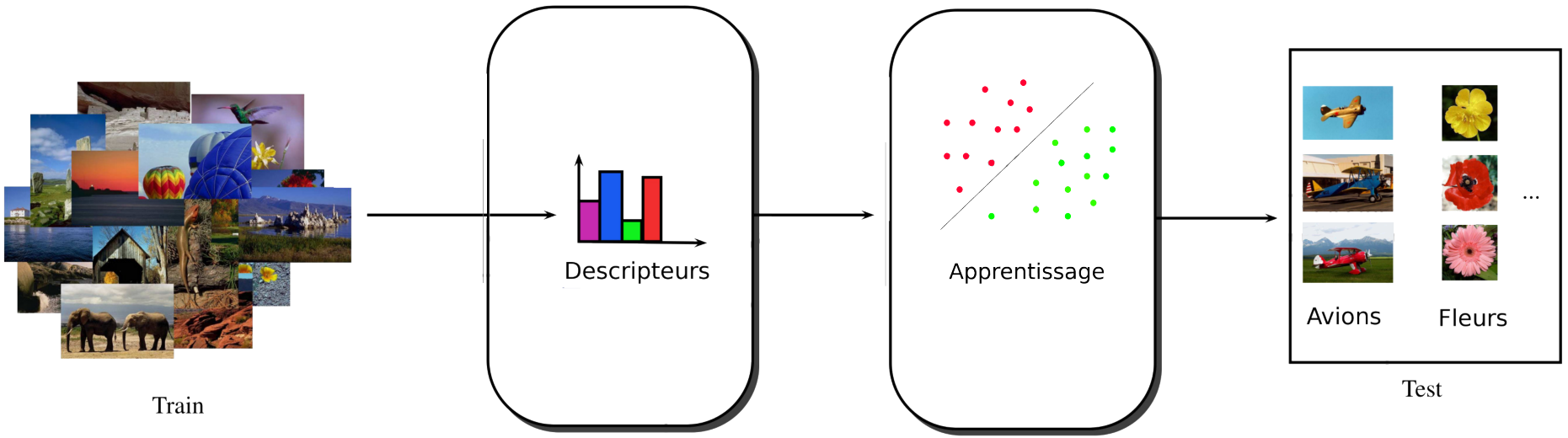
Inconvénients des MLP

- Nécessite un nombre important de neurones,
- Temps de calcul + Ressource mémoire,
- Il faut beaucoup d'exemples d'apprentissages,
- Parfois on a plus de paramètres que d'exemple d'apprentissage,
- Pas de prise en compte de contraintes spatiales,
- Optimisation non convexe => problème d'initialisation.

Sommaire

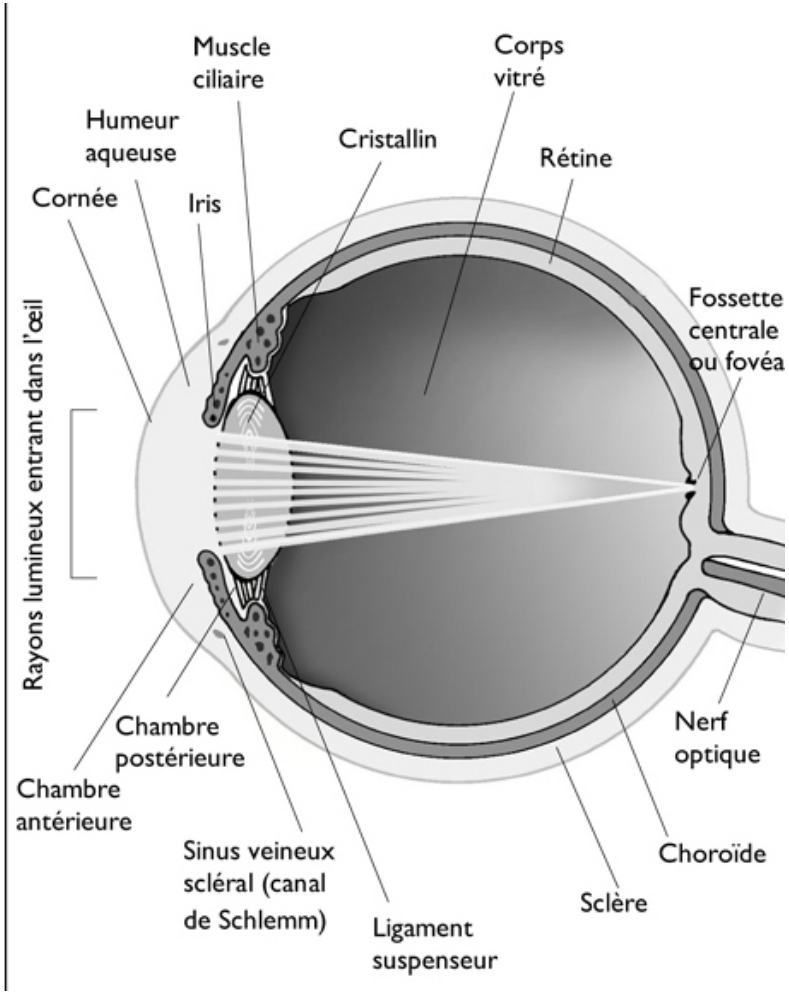
Sommaire

Les étapes de l'apprentissage automatique

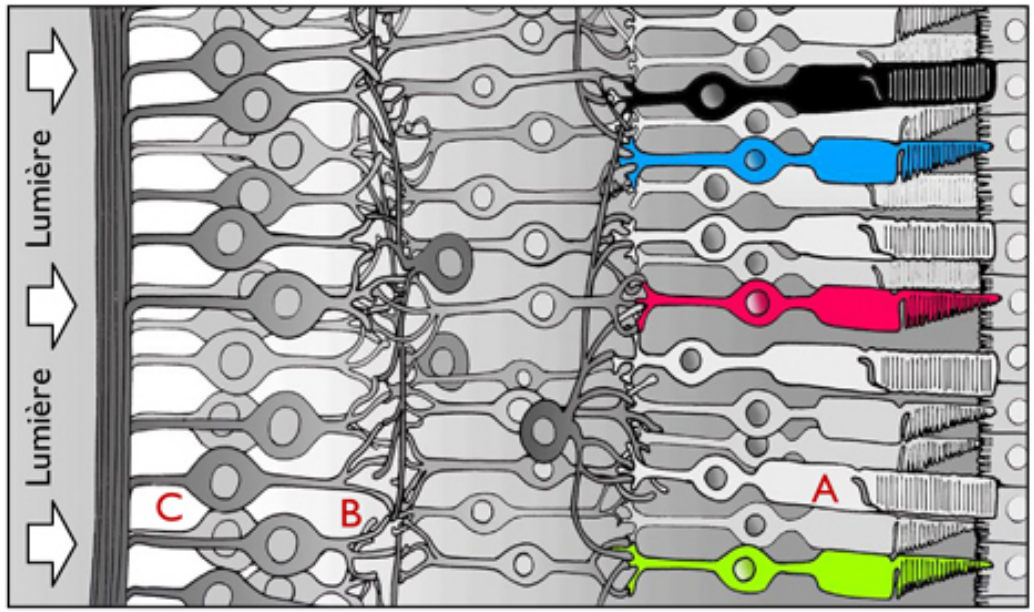


La vue : Système sensoriel de l'oeil

Anatomie de l'oeil



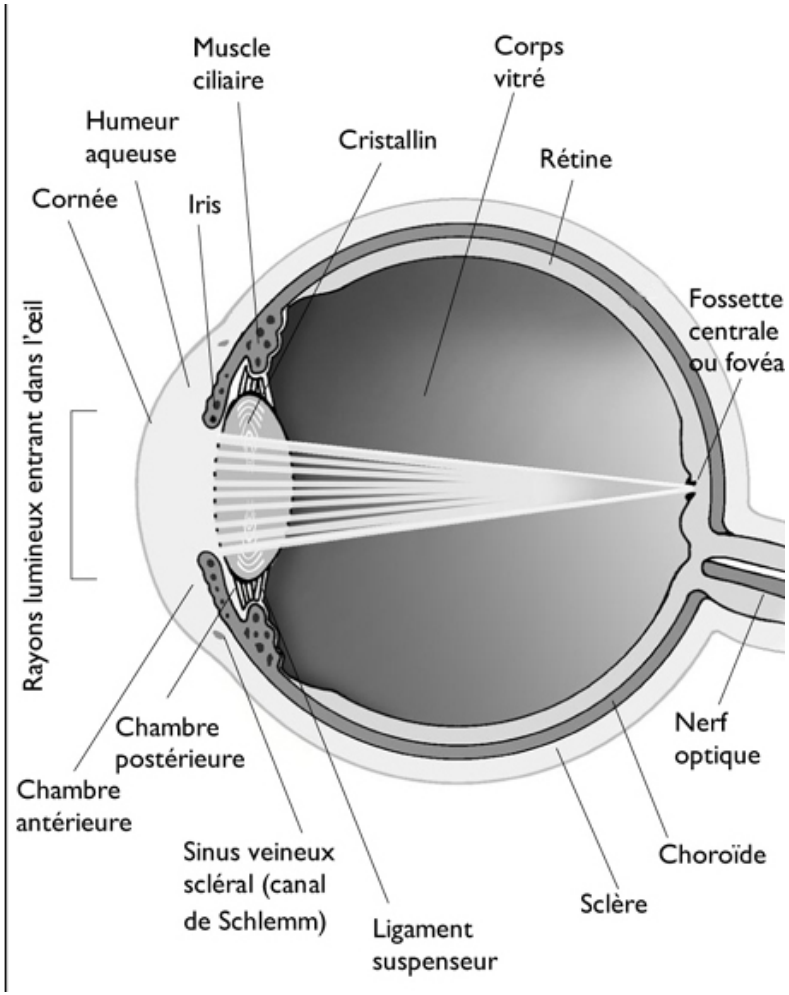
Récepteur sensoriel de l'oeil



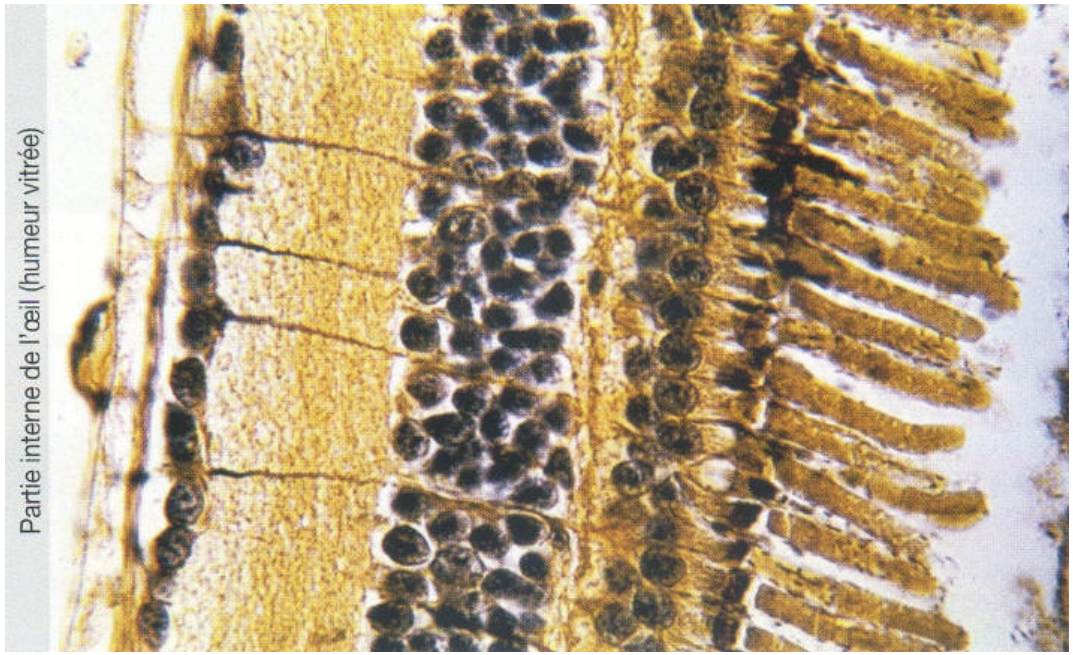
La lumière traverse la rétine jusqu'au récepteur photo-sensible de l'oeil (A) situé au plus profond de l'oeil. 4 types de récepteur (les cônes et les bâtonnets) convertissent la lumière en signal sensoriel qui traverse ensuite les neurones horizontaux (B) puis les neurones ganglionnaires (C) avant d'être transmis au cerveau au travers du nerf optique.

La vue : Système sensoriel de l'oeil

Anatomie de l'oeil



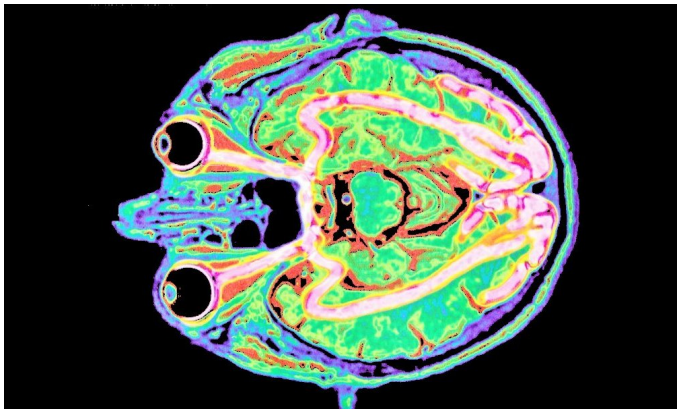
Récepteur sensoriel de l'oeil



La lumière traverse la rétine jusqu'au récepteur photo-sensible de l'oeil (A) situé au plus profond de l'oeil. 4 types de récepteur (les cônes et les bâtonnets) convertissent la lumière en signal sensoriel qui traverse ensuite les neurones horizontaux (B) puis les neurones ganglionnaires (C) avant d'être transmis au cerveau au travers du nerf optique.

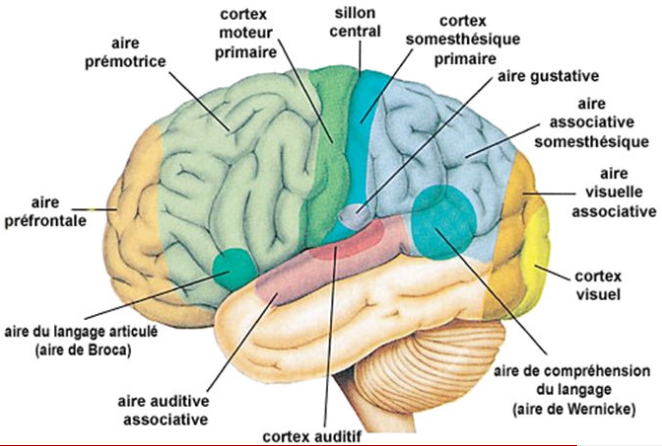
La vue (suite) : Cortex visuel

Trajet de l'information visuel



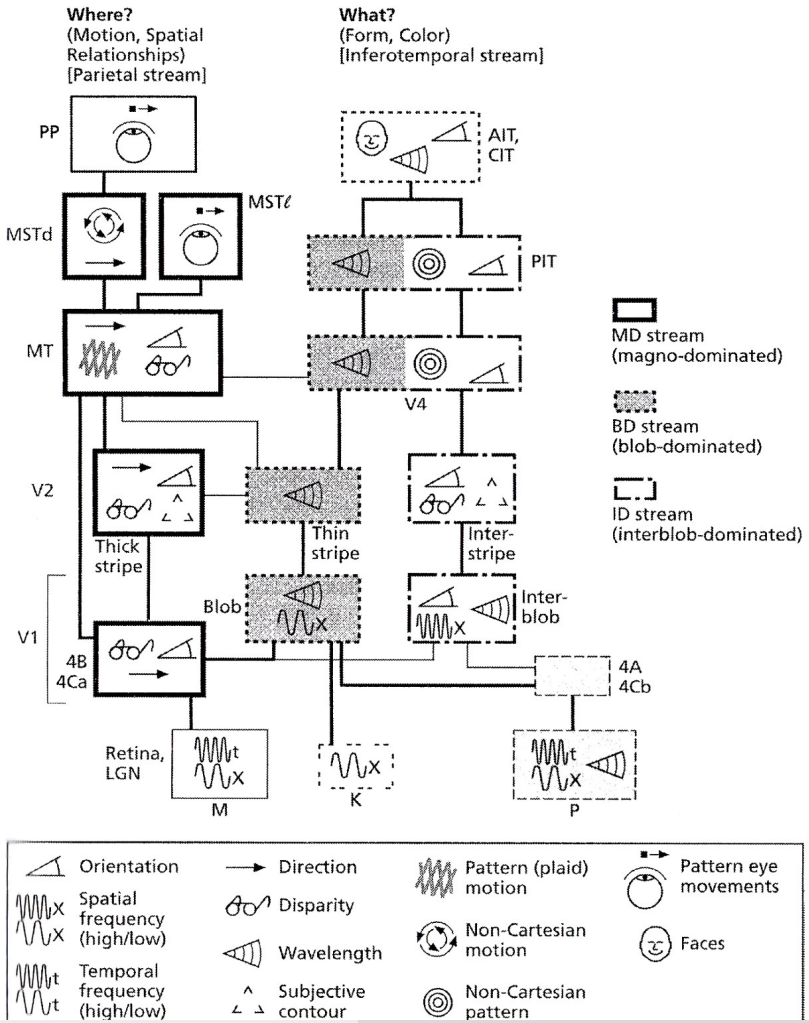
IRM des voies visuelles.

Le cortex cérébral

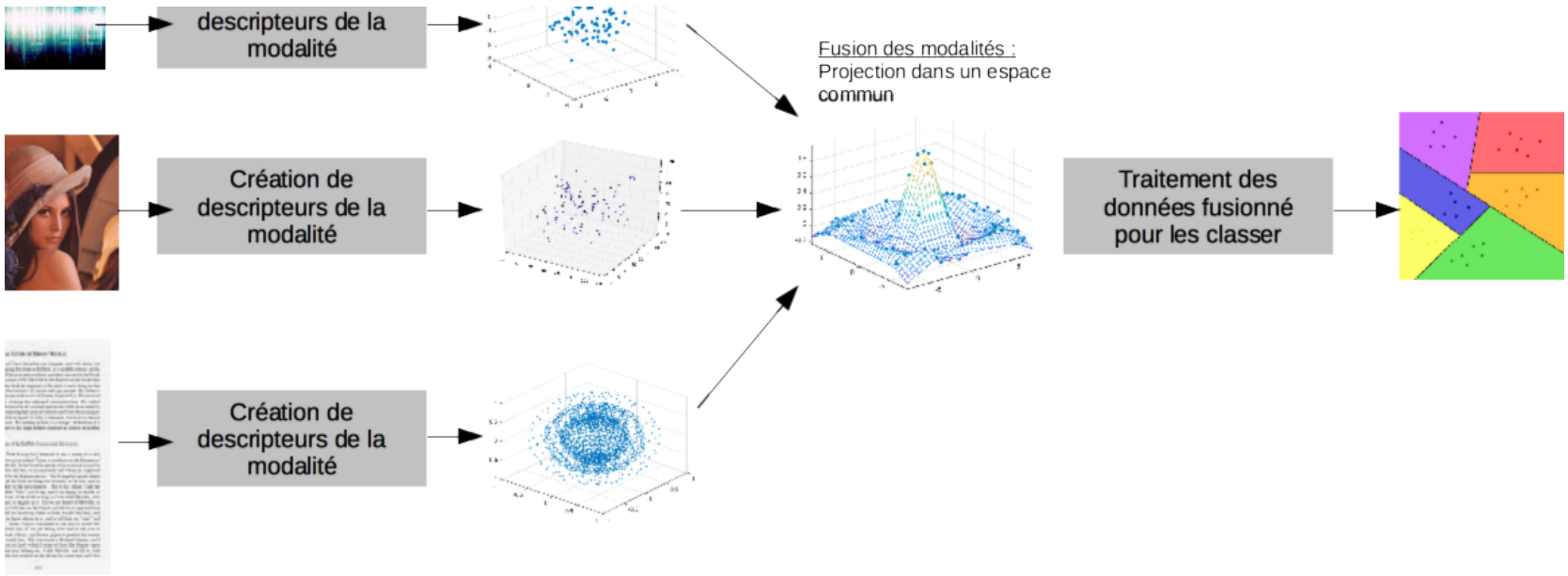


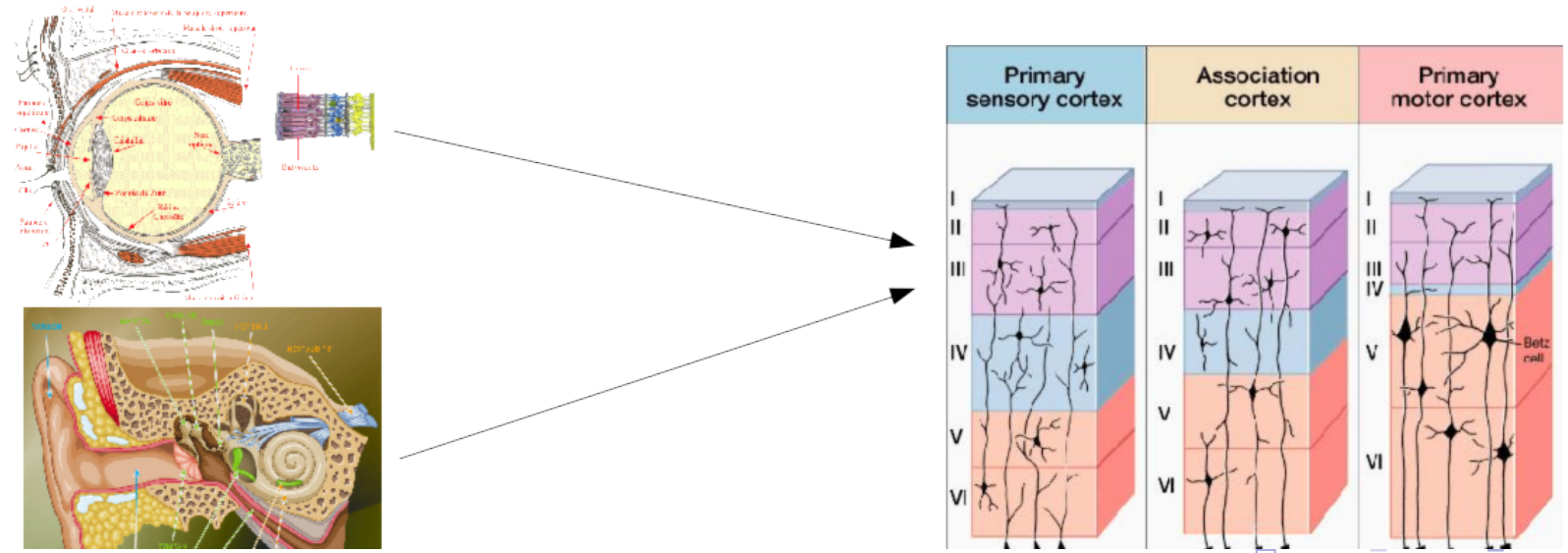
Les différents niveaux de compréhension

Le modèle de Van Essen et Gallan :



Comparaison apprentissage machine multimodale et fonctionnement biologique



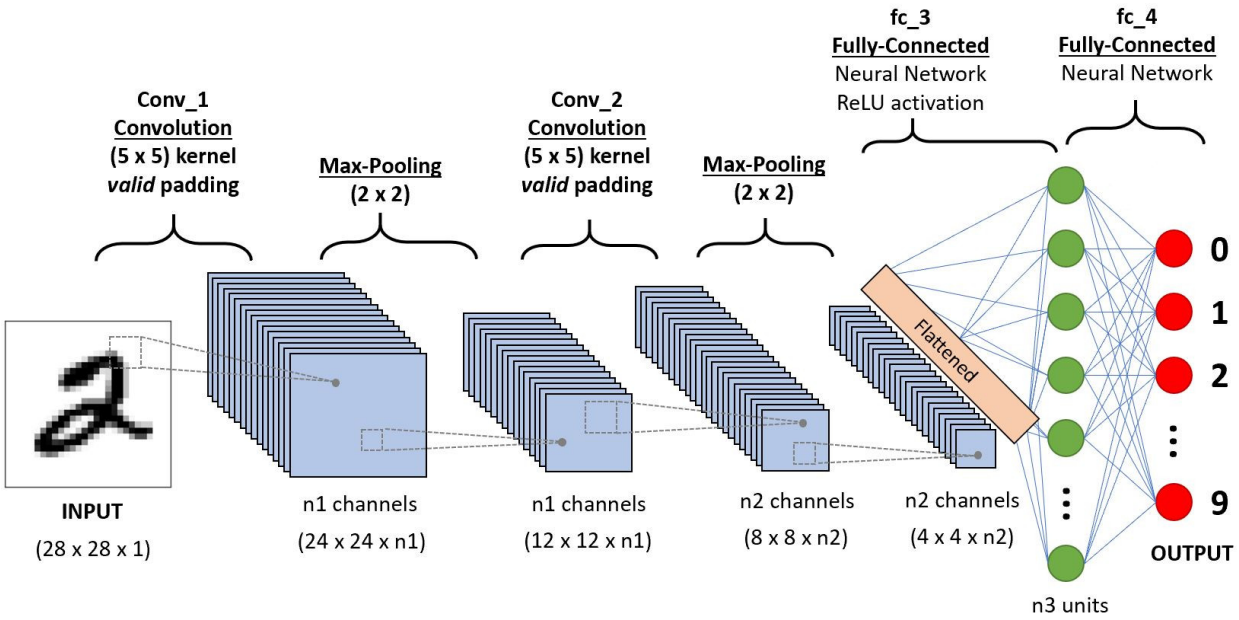


L'apprentissage de bout en bout

Problématique

- Peut-on directement à partir du signal d'entrée ?
- Peut-on introduire de l'apprentissage dès l'étape d'extraction de caractéristiques ?
- Comment faire des traitements locaux de l'information ?

Architecture d'un réseau de bout en bout (LeNet5)



Sommaire

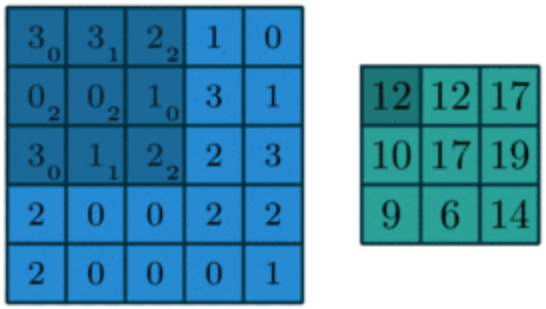
L'opérateur de convolution

Paramètres des opérateurs de convolutions

- in_channels (int) – Nombre de canaux des images d'entrées.
- out_channels (int) – Nombre de canaux des images de sorties.
- kernel_size (int or tuple) – Taille du noyau de convolution

Attention :Lors de l'exécution les tenseurs doivent être de dimension (nb d'image, nb de canaux, dim image).

Source des images : A guide to convolution arithmetic for deep learning de Vincent Dumoulin et Francesco Visin



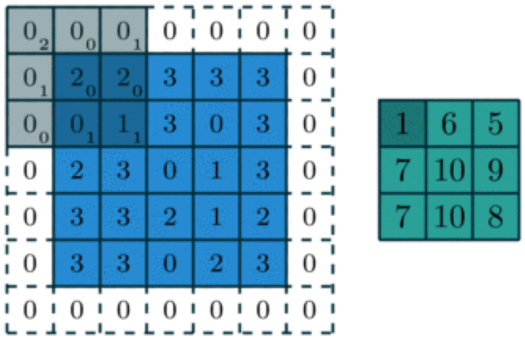
L'opérateur de convolution

Paramètres des opérateurs de convolutions

- in_channels (int) – Nombre de canaux des images d'entrées.
- out_channels (int) – Nombre de canaux des images de sorties.
- kernel_size (int or tuple) – Taille du noyau de convolution
- stride (int or tuple, optional) – Décalage de la convolution, par défaut : 1

Attention : Lors de l'exécution les tenseurs doivent être de dimension (nb d'image, nb de canaux, dim image).

Source des images : A guide to convolution arithmetic for deep learning de Vincent Dumoulin et Francesco Visin



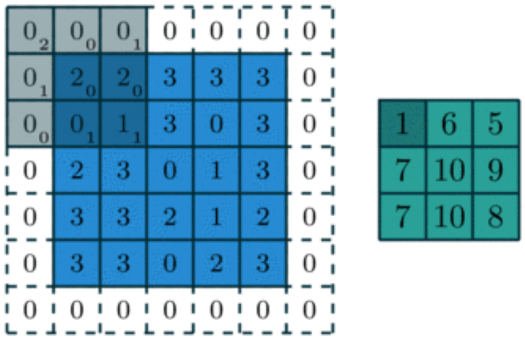
L'opérateur de convolution

Paramètres des opérateurs de convolutions

- in_channels (int) – Nombre de canaux des images d'entrées.
- out_channels (int) – Nombre de canaux des images de sorties.
- kernel_size (int or tuple) – Taille du noyau de convolution
- stride (int or tuple, optional) – Décalage de la convolution, par défaut : 1
- padding (int or tuple, optional) – Nombre de zéro ajouté au bord. , par défaut : 0

Attention : **Lors de l'exécution les tenseurs doivent être de dimension (nb d'image, nb de canaux, dim image).**

Source des images : A guide to convolution arithmetic for deep learning de Vincent Dumoulin et Francesco Visin



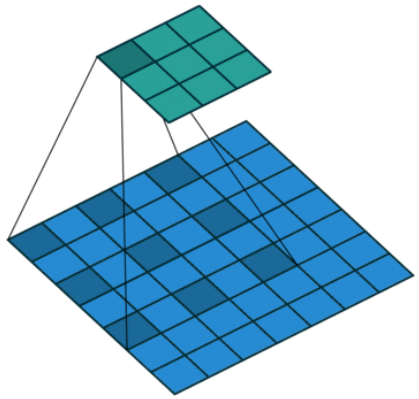
L'opérateur de convolution

Paramètres des opérateurs de convolutions

- in_channels (int) – Nombre de canaux des images d'entrées.
- out_channels (int) – Nombre de canaux des images de sorties.
- kernel_size (int or tuple) – Taille du noyau de convolution
- stride (int or tuple, optional) – Décalage de la convolution, par défaut : 1
- padding (int or tuple, optional) – Nombre de zéro ajouté au bord. , par défaut : 0
- dilation (int or tuple, optional) – Espace entre chaque noyau. , par défaut : 1

Attention :Lors de l'exécution les tenseurs doivent être de dimension (nb d'image, nb de canaux, dim image).

Source des images : A guide to convolution arithmetic for deep learning de Vincent Dumoulin et Francesco Visin



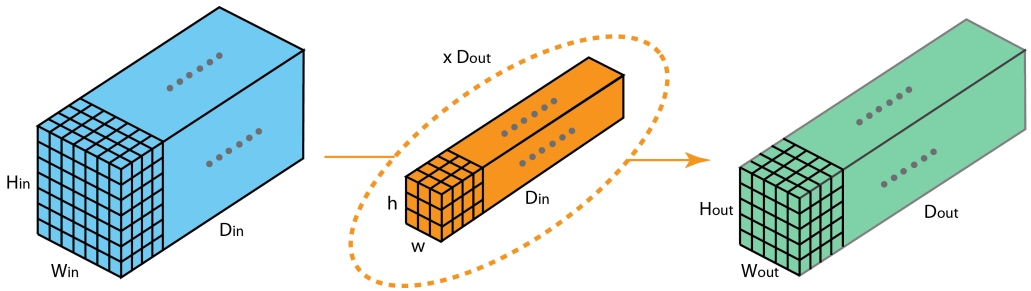
L'opérateur de convolution

Paramètres des opérateurs de convolutions

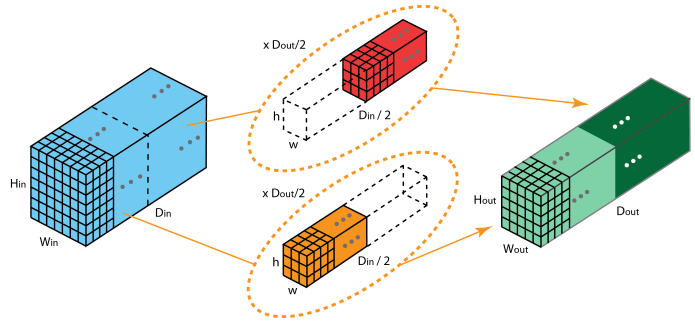
- in_channels (int) – Nombre de canaux des images d'entrées.
- out_channels (int) – Nombre de canaux des images de sorties.
- kernel_size (int or tuple) – Taille du noyau de convolution
- stride (int or tuple, optional) – Décalage de la convolution, par défaut : 1
- padding (int or tuple, optional) – Nombre de zéro ajouté au bord. , par défaut : 0
- dilation (int or tuple, optional) – Espace entre chaque noyau. , par défaut : 1
- groups (int, optional) – Nombre de connexions bloquées des canaux d'entrée vers les canaux de sortie, par défaut : 1

Attention :Lors de l'exécution les tenseurs doivent être de dimension (nb d'image, nb de canaux, dim image).

Source des images : Towards Data Science



Convolution 2D standard



Convolution 2D avec un groupe de 2

L'opérateur de convolution

Paramètres des opérateurs de convolutions

- in_channels (int) – Nombre de canaux des images d'entrées.
- out_channels (int) – Nombre de canaux des images de sorties.
- kernel_size (int or tuple) – Taille du noyau de convolution
- stride (int or tuple, optional) – Décalage de la convolution, par défaut : 1
- padding (int or tuple, optional) – Nombre de zéro ajouté au bord. , par défaut : 0
- dilation (int or tuple, optional) – Espace entre chaque noyau. , par défaut : 1
- groups (int, optional) – Nombre de connexions bloquées des canaux d'entrée vers les canaux de sortie, par défaut : 1
- bias (bool, optional) – Ajout d'un terme de biais pour l'apprentissage , par défaut : True

Attention : **Lors de l'exécution les tenseurs doivent être de dimension (nb d'image, nb de canaux, dim image).**

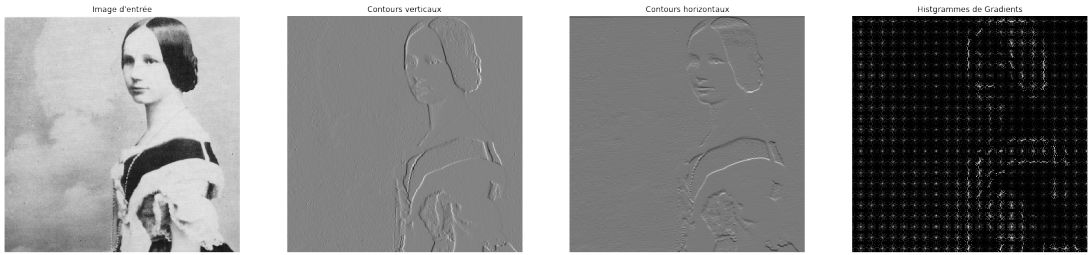
Lien entre la convolution et l'extraction de caractéristique

Filtre de Sobel

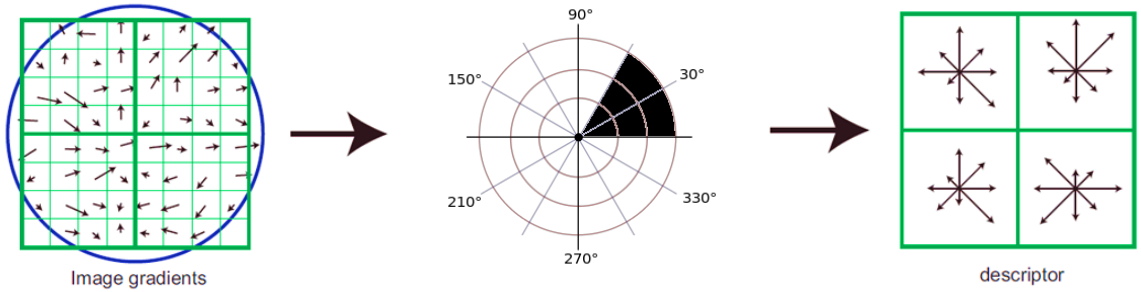
L'opérateur de convolution avec les opérateurs

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \text{Im} \text{ et } G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \text{Im}$$

permet d'extraire les contours verticaux et horizontaux dans une image Im.



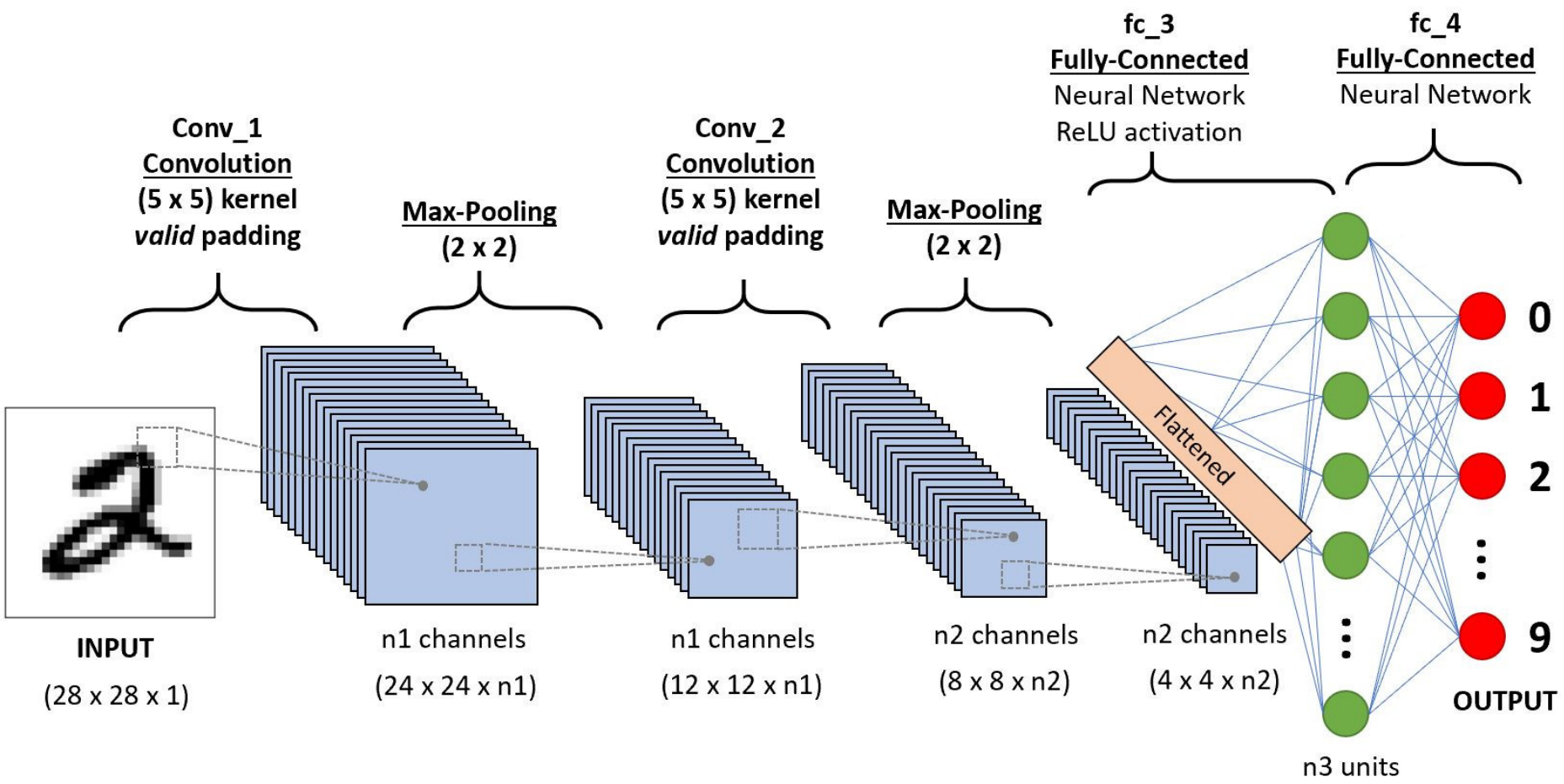
Rappel sur les Hog



Sommaire

Lenet 5

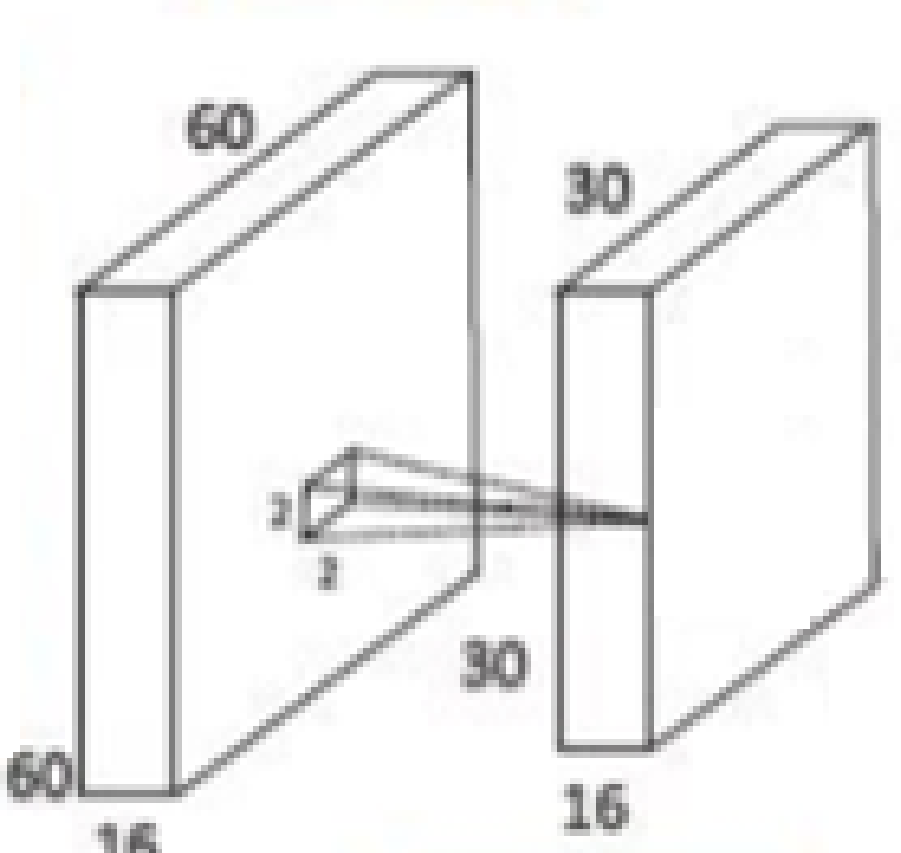
Architecture d'un réseau de bout en bout (LeNet5)



Couches de pooling

La couche de pooling

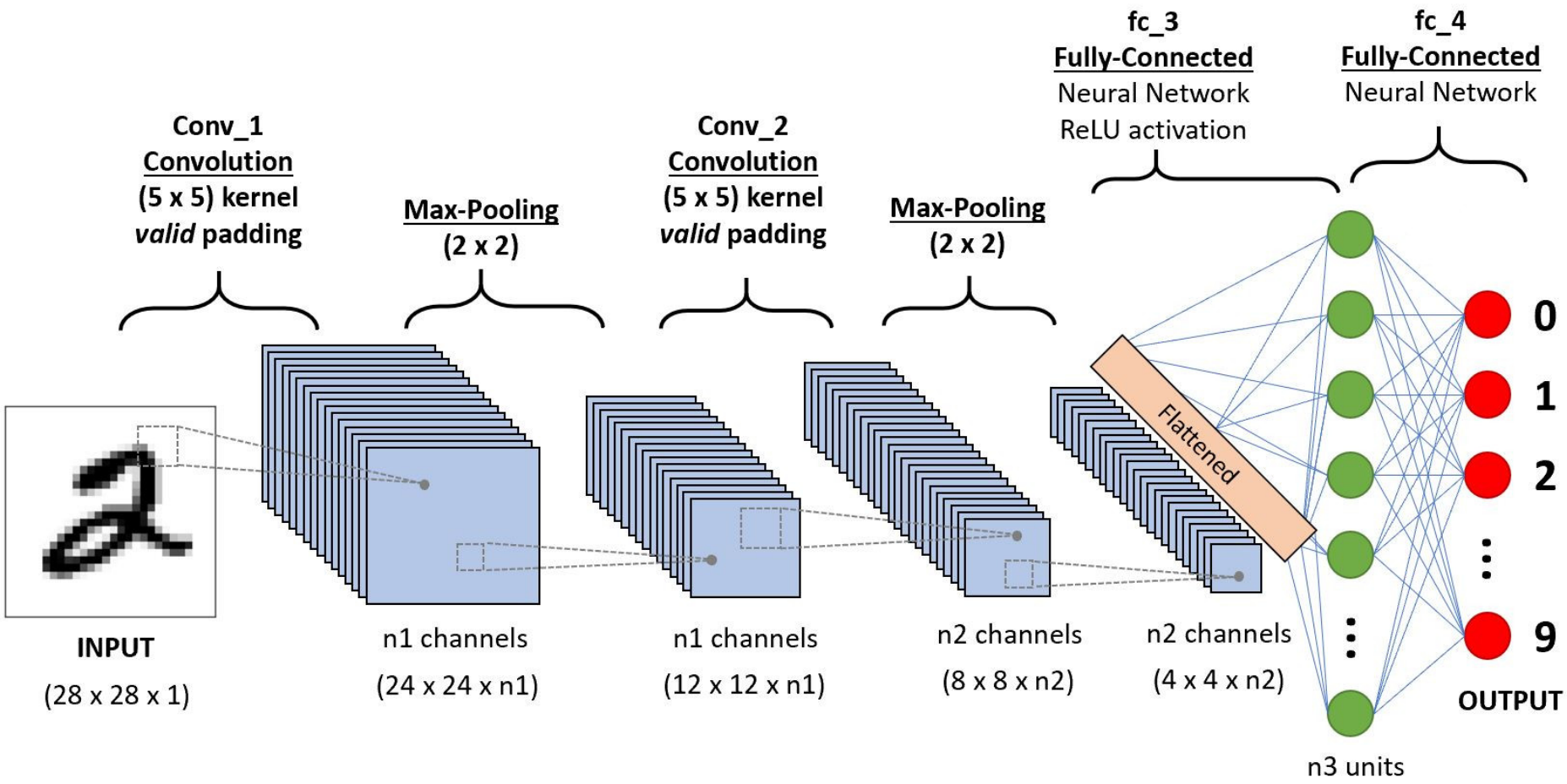
On trouve fréquemment dans les réseau des couches de réduction de dimension par exemple à l'adie de maximum local (maxpooling) ou de moyenne local (mean pooling).



Sommaire

Lenet 5

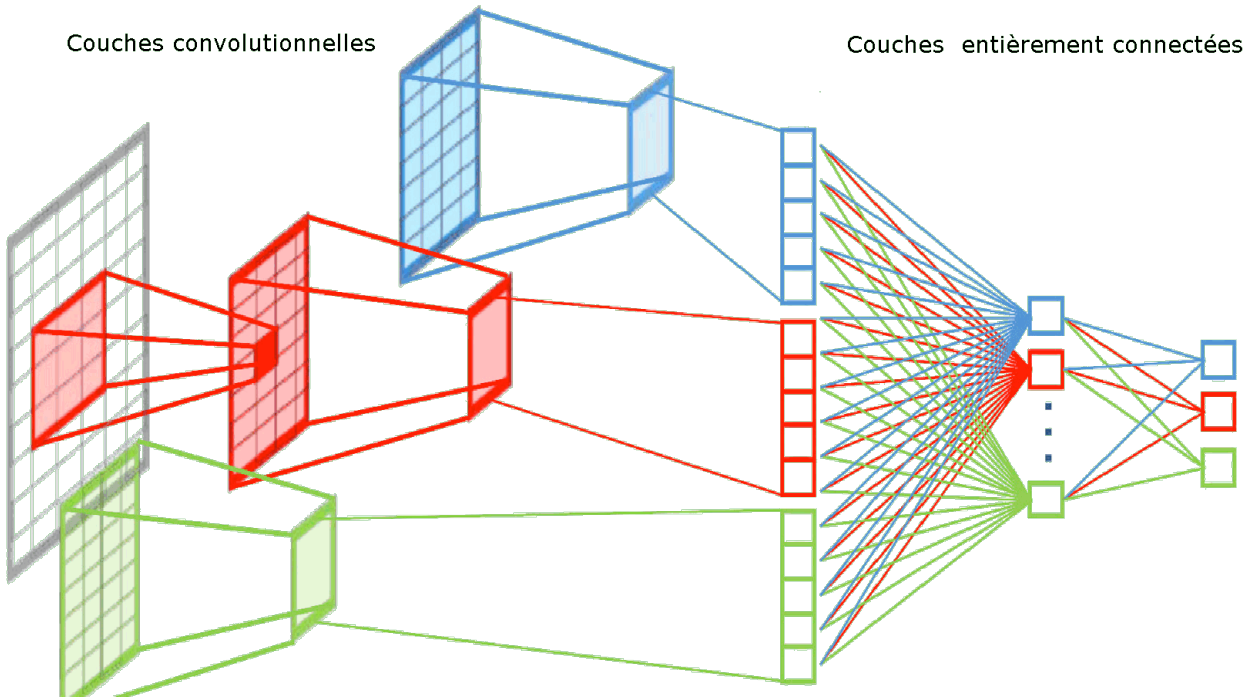
Architecture d'un réseau de bout en bout (LeNet5)



Des couches de convolution aux couches FC

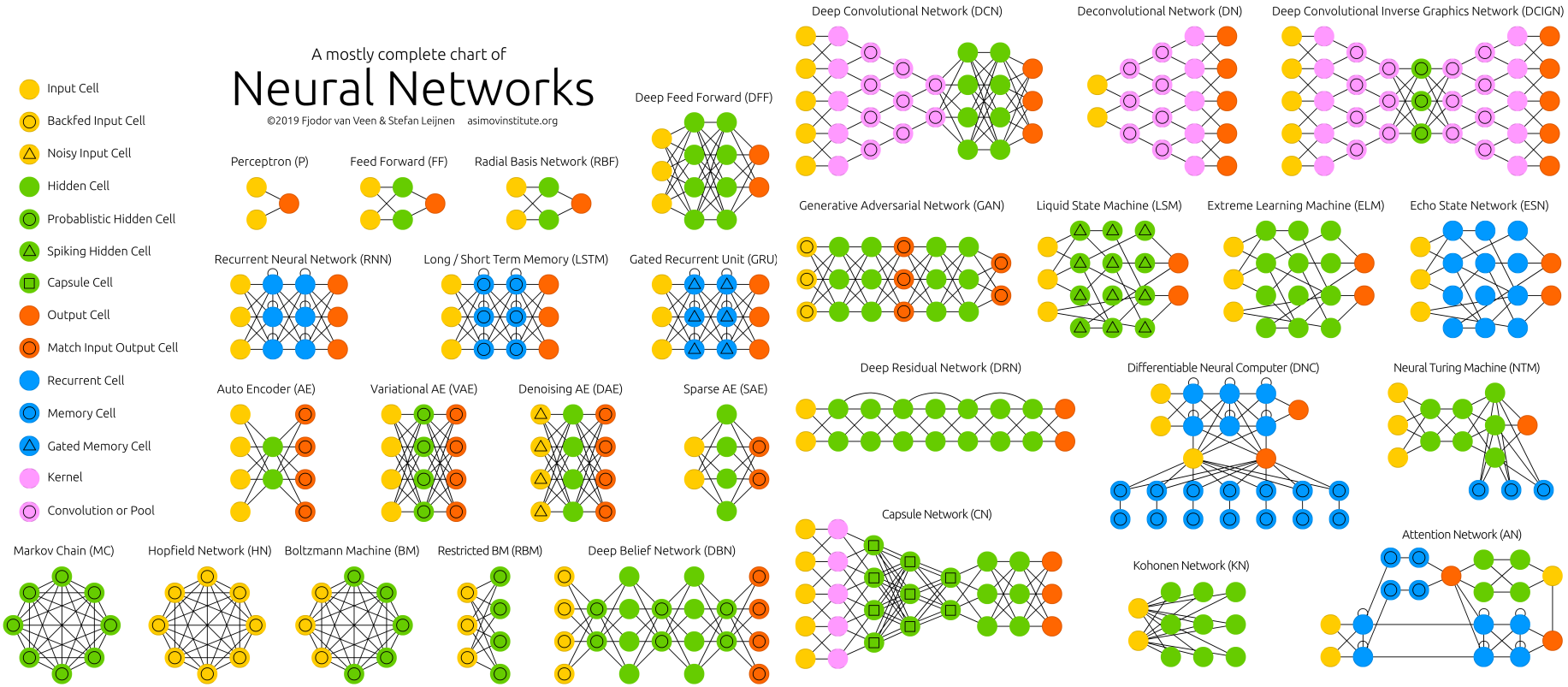
La couche entre les convolutions et les couches entièrement connectés

Lors du passage entre la partie convolutionnelle du réseau et les parties entièrement connectées, il est nécessaire de redimensionner le tenseur.



Sommaire

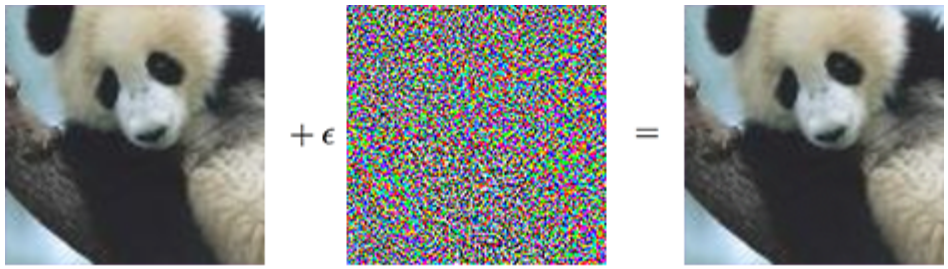
Des architectures de réseaux variées



Source : `http://www.asimovinstitute.org/neural-network-zoo/`

Limites et enjeux de recherche en Deep Learning

Attaques adverses et robustesse des réseaux



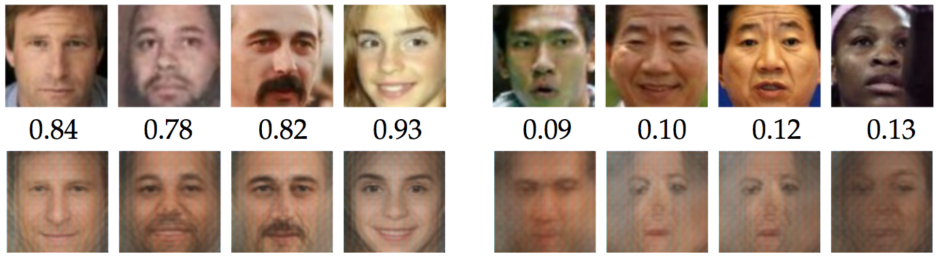
Source : Explaining and Harnessing Adversarial Examples

Le Deep learning vert

- Large scale GAN Training for High fidelity natural image synthesis a nécessité 512 TPU pendant 24h à 48h chaque TPU consomme 200 Wh, soit un total compris entre 2.457 et 4.915 kWh.
- Jer Thorp écrit sur Twitter: "The good news is that AI can now give you a more believable image of a plate of spaghetti, the bad news is that it used roughly enough energy to power Cleveland for the afternoon."

Limites et enjeux de recherche en Deep Learning (suite)

Protection de la vie privée



Source : On the Reconstruction of Face Images fromDeep Face Templates

Éthique et moral

L'apprentissage automatique et les réseaux de neurones posent de nombreux problème éthique et moraux. On peut notamment citer les biais dans la construction des bases de données (par exemple : Identifying and Correcting Label Bias in Machine Learning)

Interprétabilité

Il est souvent reproché l'aspect "boite noire" des réseaux de neurones. Quelques travaux sur le sujet : Visual Interpretability for Deep Learning: a Survey

Le Deep Learning en Normandie

Recherche en informatique en Normandie : Greyc et Litis

- NormaSTIC : fédération CNRS `http://www.normastic.fr/`
- Une recherche en algorithme d'apprentissage (et notamment en deep learning) et en analyse et traitement d'image au travers des équipes :
  - Image du Greyc `https://www.greyc.fr/?page_id=445`,
  - Apprentissage du Litis `http://www.litislab.fr/equipe/app/`.
- L'accès au supercalculateur Myria du criann `https://www.criann.fr/calcul-intensif/`.
- De nombreux projets industrielles et plusieurs projets ANR.

Pour aller plus loin

Livre

I. Goodfellow, Y. Bengio et A. Courville, Deep Learning, MIT Press book, 2016
`http://www.deeplearningbook.org/`, existe en version traduite en français.

Conférences

NIPS, ICLR, xCML, AIStats, CVPR, ICCV,ECCV,ACCV,BMVC,ICIP,ESANN ...

Journaux

JMLR, PAMI, IJCV, Machine Learning...

Cours en ligne

- Deep Learning : Cours de Yann LeCun au Collège de France en 2016
  `https://www.college-de-france.fr/site/yann-lecun/course-2015-2016.htm`,
- Cours de Andrew Ng de Standford `https://fr.coursera.org/learn/machine-learning`,
- Introduction à Tensorflow
  `https://eu.udacity.com/course/intro-to-tensorflow-for-deep-learning--ud187`,
- Introduction à Pytorch `https://pytorch.org/tutorials/`.

Sommaire

Sommaire

Les librairies de Deep Learning

Un grand nombre de librairies de développement

Les questions à se poser

- Le langage de programmation du projet,
- Les types de réseau de neurones.
- Les services de calculs notamment en cas d'utilisation de solution de cloud.
- Les besoins en déploiement.
- Les besoins en support et en documentation.

PyTorch et Tensorflow dans la recherche

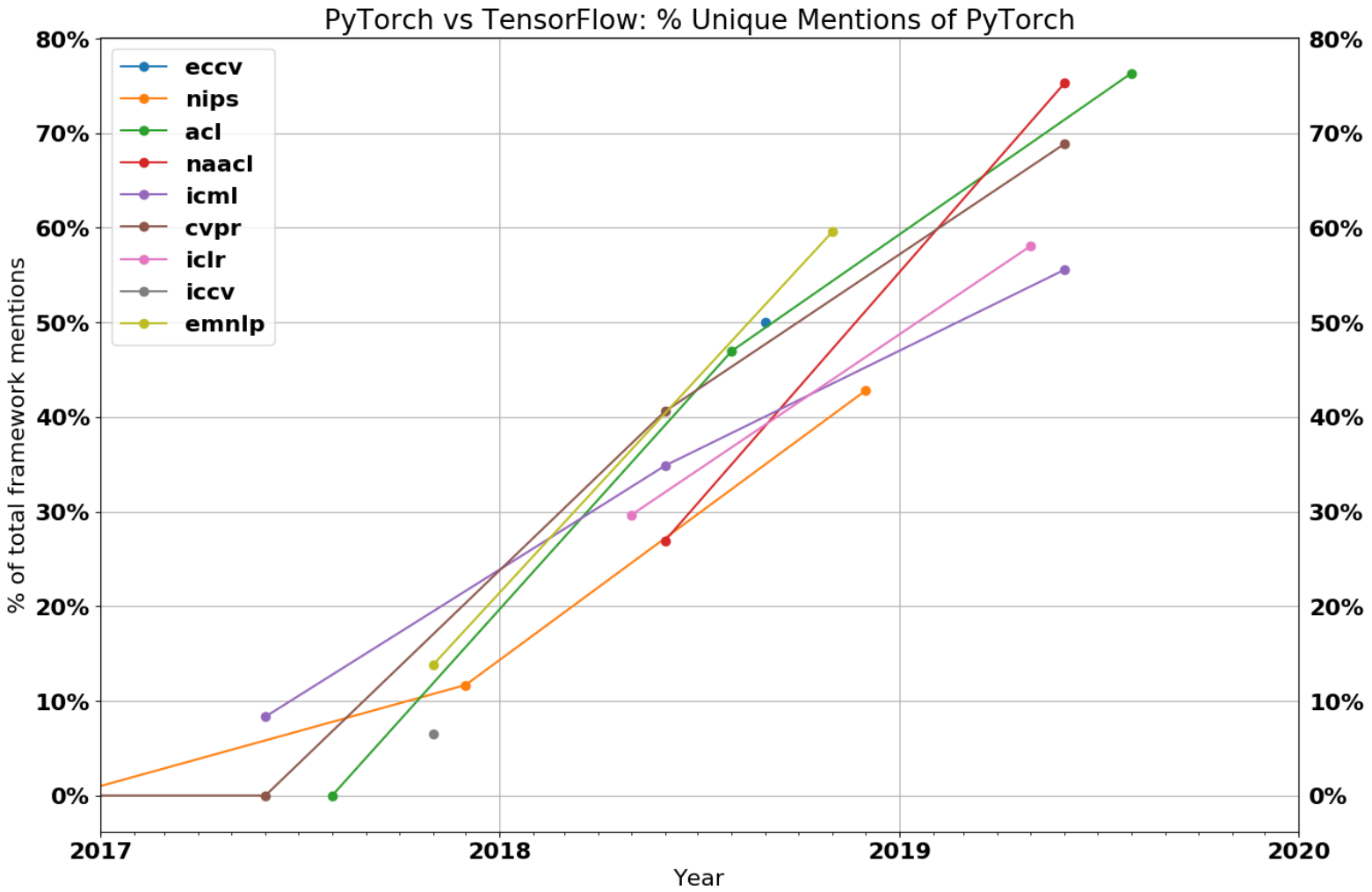


Figure – Source : https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominate

Évolution des frameworks

Vers une convergence des frameworks ?

- Tensorflow 2.0 (septembre 2019) : Eager Execution,
- PyTorch 1.0 (décembre 2018) : TorchScript,
- ONNX (Open Neural Exchange Format),
- Outils de visualisation commun : Tensorboard,
- Librairies haut-niveau multi-framework : Keras/Pytorch lightning, Horovod...

Exemples de codes

Code Tensorflow :

```python
import tensorflow as tf
class Model_tf(object):
    def __init__(self):
        super(Model_tf,self).__init__()
        self.W = tf.Variable(5.0)
        self.b = tf.Variable(0.0)
    def __call__(self, x):
        return self.W * x + self
```

Code PyTorch :

```python
import torch
class Model_torch(toch.nn.Module):
    def __init__(self):
        super(Model_torch, self).__init__()
        self.W = torch.Tensor([5.0])
        self.b = torch.Tensor([0.0])
    def forward(self, x):
        return self.W * x + self.b
```

Sommaire

PyTorch

Qu'est-ce que PyTorch ?

- C'est un framework de DeepLearning open source développé par Facebook depuis 2016,
- PyTorch est dérivé du framework Torch qui s'utilise en Lua.
- PyTorch est écrit en C++, C et Python et s'interface en Python.
- Supporte les calculs CPU et GPU,
- S'interface facilement avec la librairie de manipulation de tableau Numpy,
- Bénéficie d'une communauté très active proche de la recherche.

L'écosystème PyTorch

PyTorch : Librairie bas niveau

- Librairie pour la création de réseau de neurones.
- Très proche de Numpy dans son fonctionnement.
- Portage sur GPU.
- Adapté à des experts pour le développement de fonctionnalité non existante.

PyTorch Lightning : Librairie de niveau intermédiaire

- L'objectif de la librairie est de diminuer le travail d'ingénierie.
- Prise en compte rapide du hardware, de l'apprentissage distribué, du logging, de la visualisation et des mécanismes standards.
- Adapté aux chercheurs et aux ingénieurs en machine learning.
- Lightning est construit au-dessus de PyTorch et peut être facilement enrichie.

Lightning Flash : Librairie haut niveau

- Pour le prototypage rapide, la reproduction de baseline ou la résolution de taches d'apprentissage standard.
- Simple et facile à prendre en main.
- Adapté aux débutants.
- Flash est construit au-dessus de Lightning et peut être facilement enrichie.

L'écosystème PyTorch

Torchvision

Boite à outils pour les applications de vision. La librairie contient

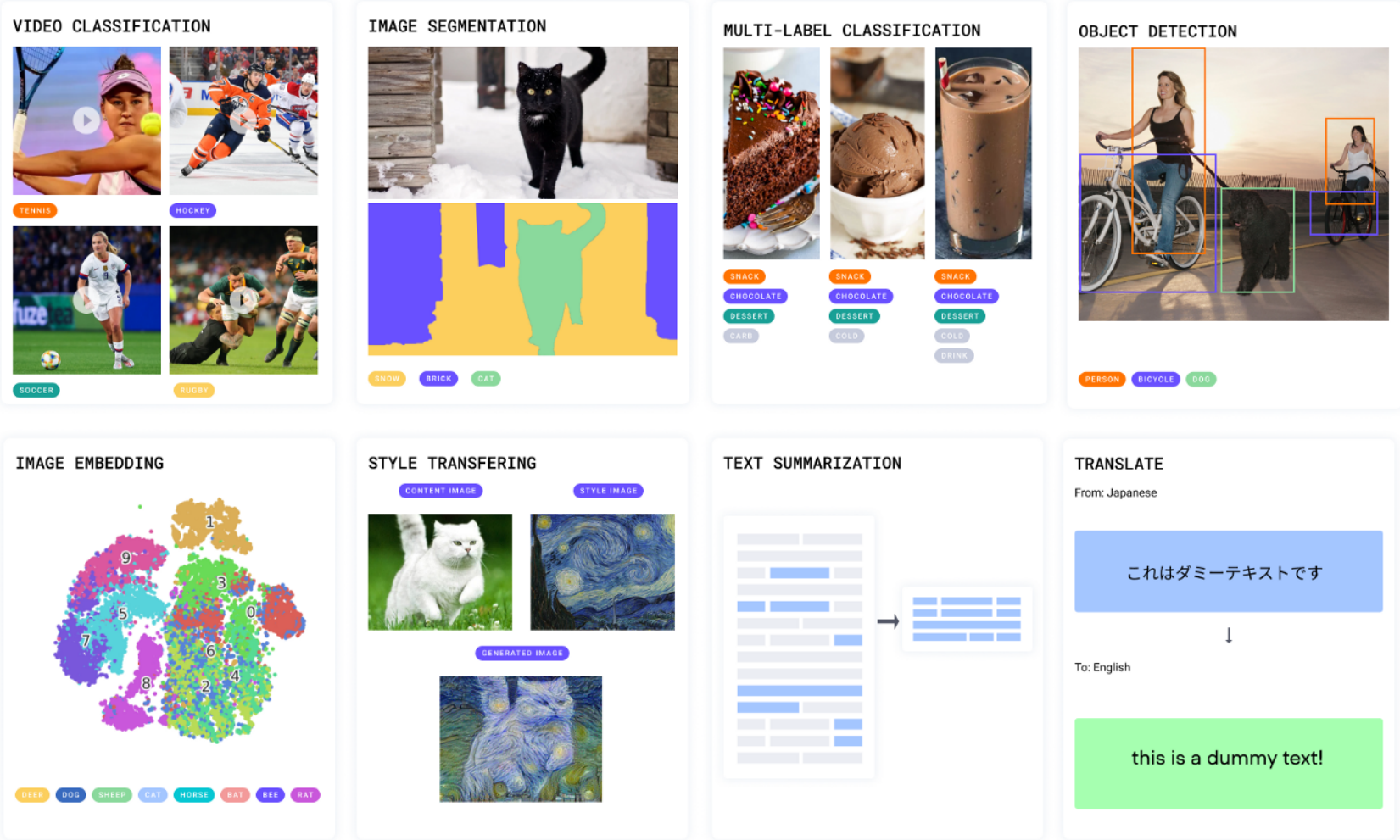
- Les transformations classiques sur les images (noir et blanc, redimensionnement, miroir, translation, rotations...)
- Les modèles classiques de la littérature,
- Les bases de données classiques de la littérature.

TorchMetrics

- À l'origine inclus dans Pytorch Lightning.
- Contient l'implémentation des principales métriques (collection de +80 métriques standards).

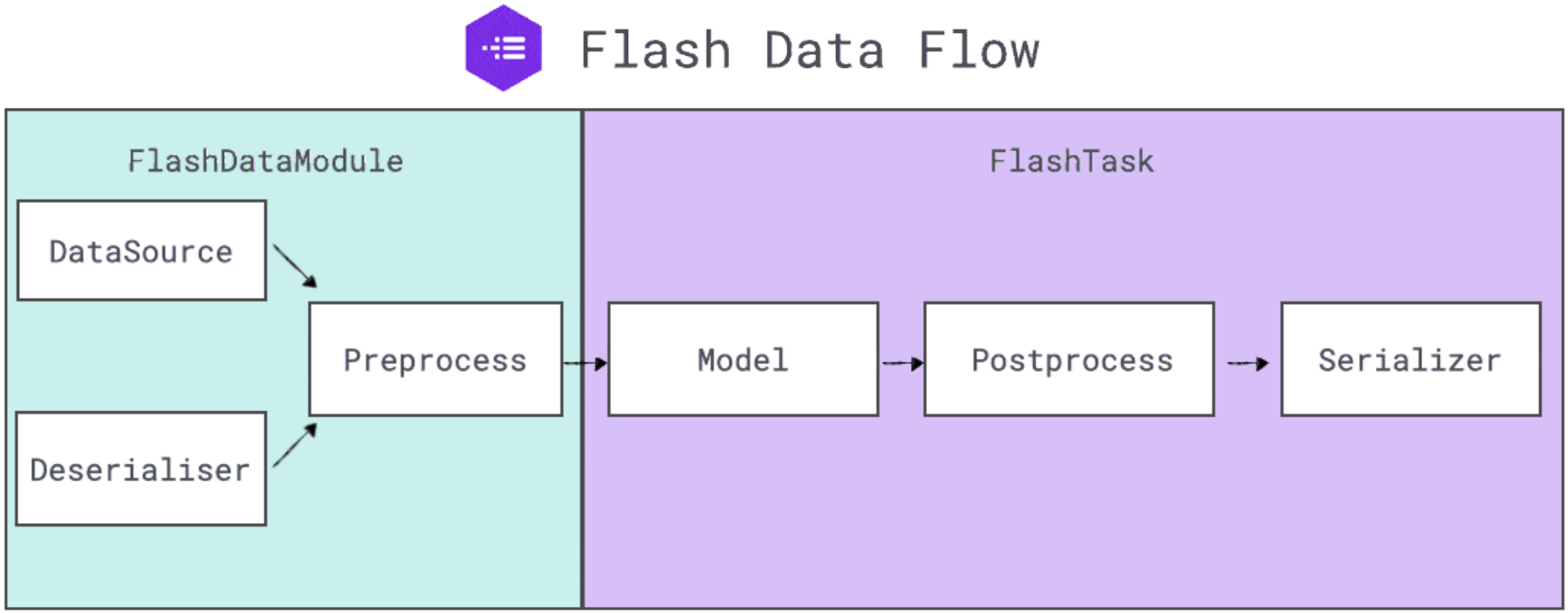
Sommaire

Les tâches dans Lightning Flash



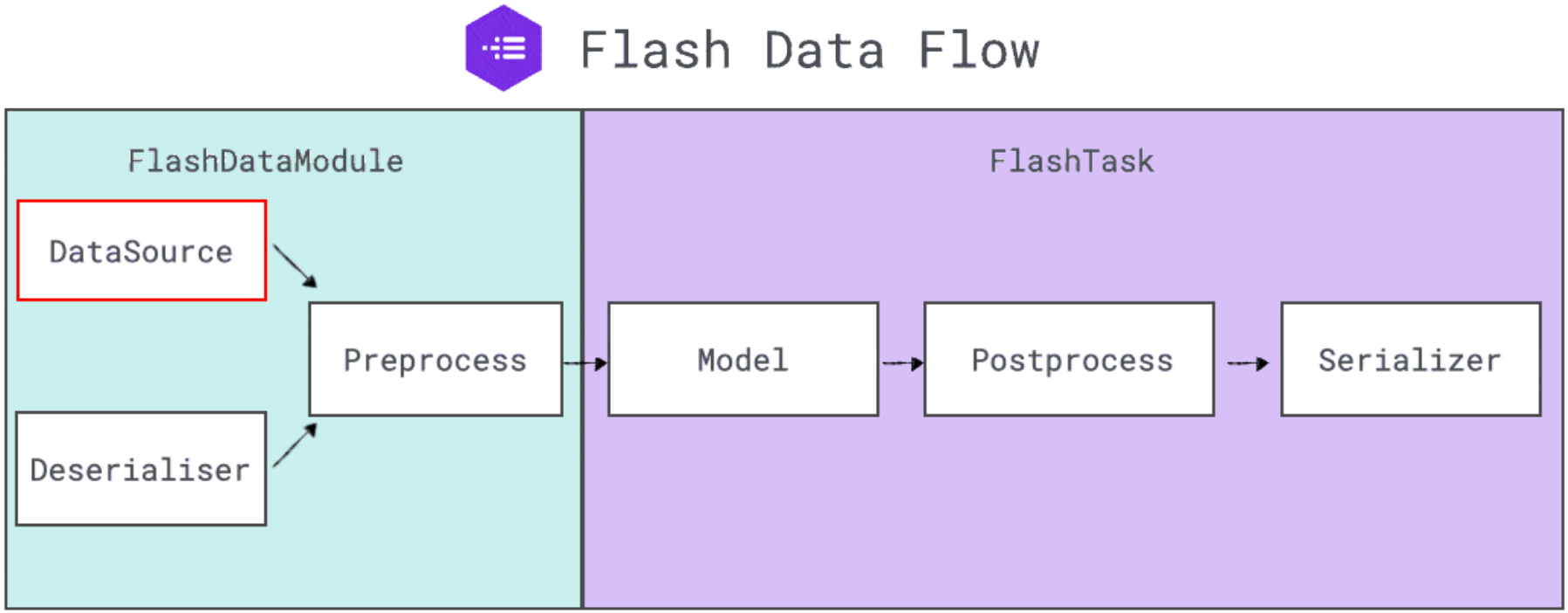
Nous illustrons dans la suite sur la tâche de classification d'images.

Déroulement d'un programme en Lightning Flash



1. Data Loading
2. Data Transforms
3. Model Forwarding
4. Predictions Transforms
5. Predictions Serialisation

Déroulement d'un programme en Lightning Flash



1. Data Loading
2. Data Transforms
3. Model Forwarding
4. Predictions Transforms
5. Predictions Serialisation

Chargement des données

À partir de dossiers

```python
from flash.core.data.utils import download_data
from flash.image import ImageClassificationData

# 1. Create the DataModule
download_data("https://..  .zip", "./data")

datamodule = ImageClassificationData.from_folders(
    train_folder = "data/train/",
    val_folder = "data/val/",
    test_folder = "data/test/",
    batch_size = 32,
    transform_kwargs = {"image_size": (196, 196),
                "mean": (0.485, 0.456, 0.406),
                "std": (0.229, 0.224, 0.225)
            },
)
```

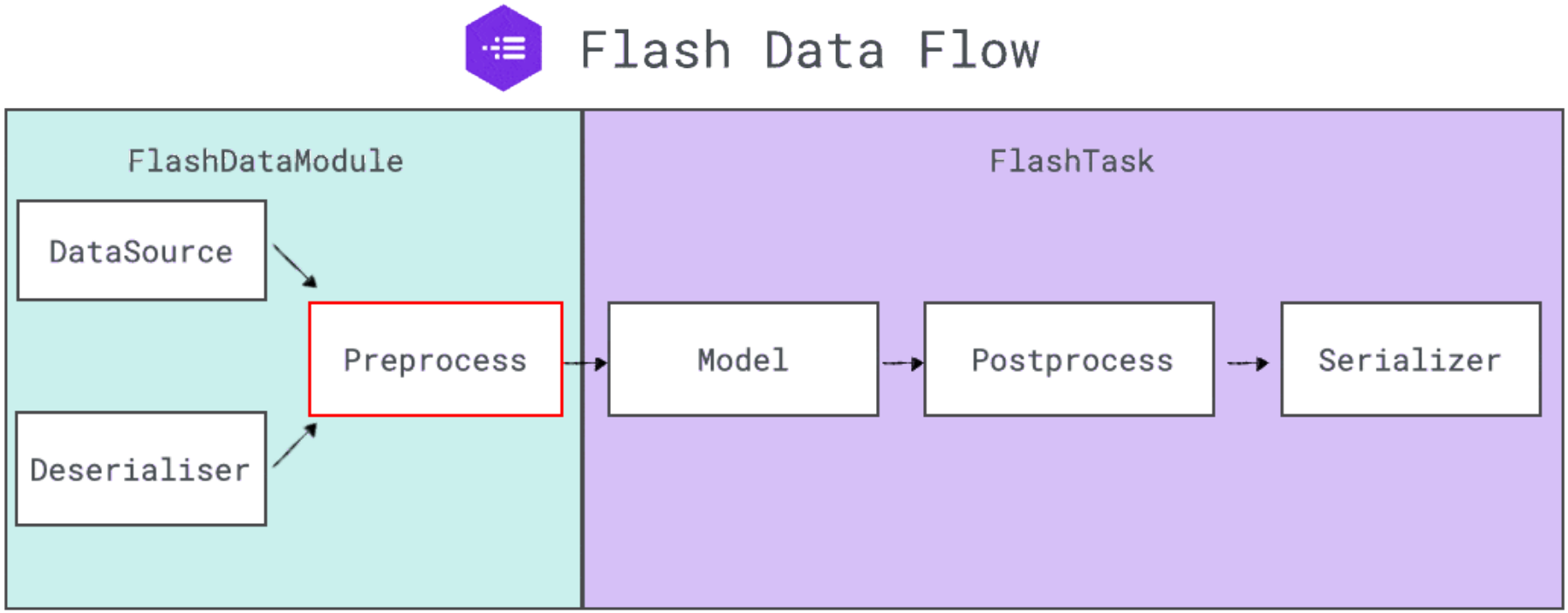
On considère que les images sont réparties dans un dossier par classe.

Chargement des données

À partir d'une source de données : torchvision

```python
from flash.image import ImageClassificationData
import torchvision
from torchvision import transforms as T

train_dataset = torchvision.datasets.MNIST('data',
    train = True,
    download = True,
    transform = T.Grayscale(num_output_channels=3) )

test_dataset = torchvision.datasets.MNIST('data',
    train = False,
    download = True,
    transform = T.Grayscale(num_output_channels=3) )

datamodule = ImageClassificationData.from_datasets(
  train_dataset = train_dataset,
  test_dataset = test_dataset,
  val_split = 0.2,
  batch_size = 32,
  transform_kwargs = {'image_size':(28, 28)})
```

Déroulement d'un programme en Lightning Flash



1. Data Loading
2. Data Transforms
3. Model Forwarding
4. Predictions Transforms
5. Predictions Serialisation

Chargement des données

Personnalisation des transformations de preprocessing des données (1/2)

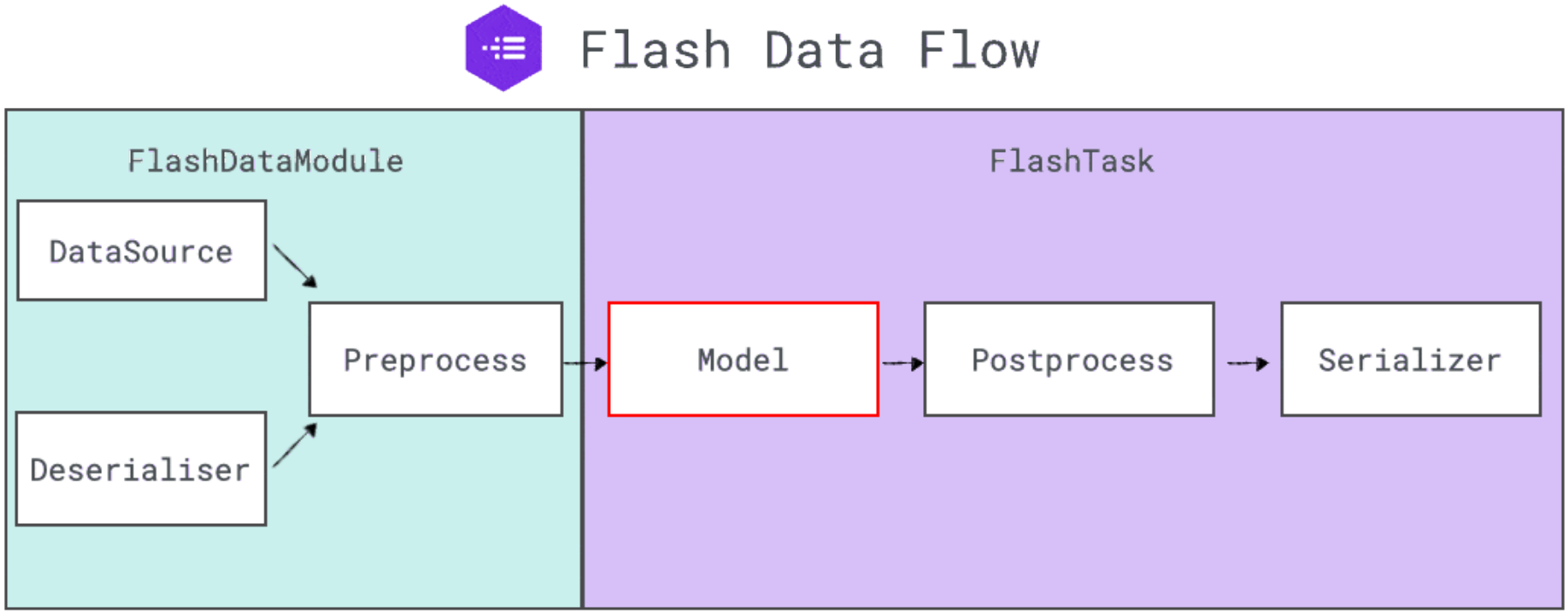
```python
from dataclasses import dataclass
from flash.image.classification.input_transform import
    ImageClassificationInputTransform
from torchvision import transforms as T

@dataclass
class MyTransform(ImageClassificationInputTransform):
  def input_per_sample_transform(self):
    return T.Compose([
      T.Grayscale(num_output_channels=3),
      super().input_per_sample_transform() ])

  def train_input_per_sample_transform(self):
    return T.Compose([
      T.Grayscale(num_output_channels=3),
      T.ToTensor(),
      T.Resize(self.image_size),
      T.Normalize(self.mean, self.std),
      T.RandomHorizontalFlip(),
      T.ColorJitter()])
```

Chargement des données

Personnalisation des transformations de preprocessing des données (2/2)

```
datamodule = ImageClassificationData.from_datasets(
  train_dataset = torchvision.datasets.MNIST('data',train = True),
  test_dataset = torchvision.datasets.MNIST('data',train = False),
  val_split=0.2,
  batch_size=32,
  train_transform = MyTransform,
  val_transform = MyTransform,
  test_transform = MyTransform,
  transform_kwargs = {'image_size':(28, 28)}
)

```

Déroulement d'un programme en Lightning Flash



1. Data Loading
2. Data Transforms
3. Model Forwarding
4. Predictions Transforms
5. Predictions Serialisation

Apprentissage d'une tâche

Déclaration et apprentissage d'une tâche

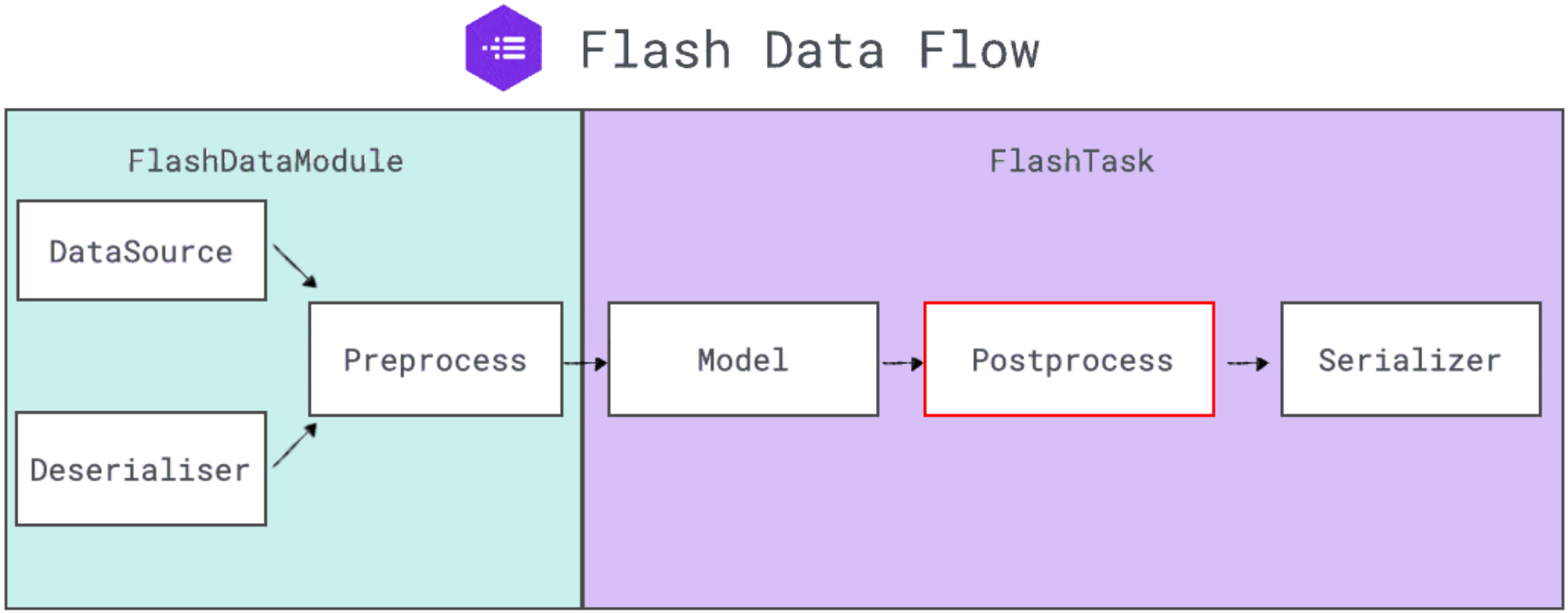
```python
from flash.image import ImageClassifier

# 2. Build the task
model = ImageClassifier(backbone="resnet18",
            num_classes = 10,
            )

# 3. Create the trainer and finetune the model
trainer = flash.Trainer(max_epochs=3,
            gpus=torch.cuda.device_count()
            )

trainer.finetune(model,
          datamodule=datamodule,
          strategy="freeze"
          )
# or
trainer.fit(model,
        datamodule=datamodule
        )
```

Personnalisation de l'apprentissage

Réglage des paramètres de l'apprentissage

```
model = ImageClassifier(
  backbone = 'mobilenet_v2',
  head='linear',
  num_classes = 10,
  optimizer = partial(torch.optim.SGD,
            momentum=0.9,
            weight_decay=5e-4
  ),
  learning_rate = 1e-3,
  pretrained = False,
  metrics = {'acc':torchmetrics.Accuracy()},
)
```

Déroulement d'un programme en Lightning Flash



1. Data Loading
2. Data Transforms
3. Model Forwarding
4. Predictions Transforms
5. Predictions Serialisation

Évaluation d'un modèle

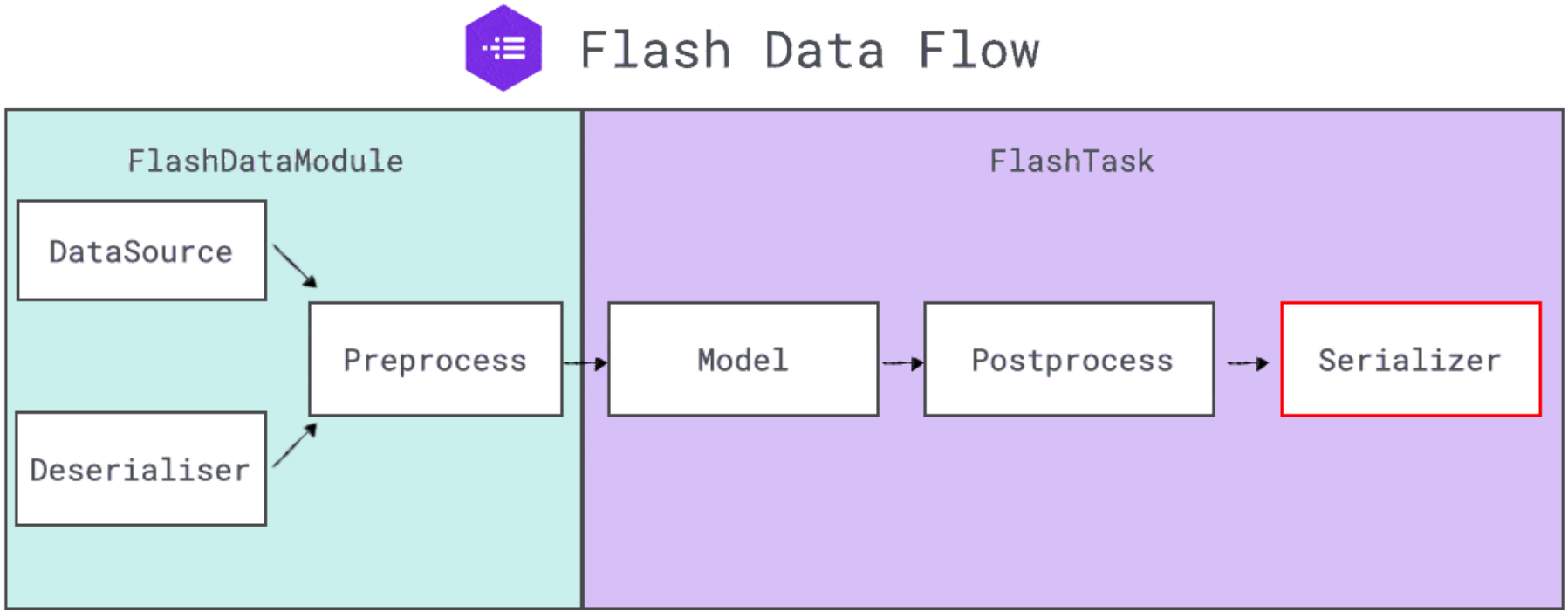
Mesure des performances sur la base de test

```
trainer.test(model,datamodule=datamodule)
```

Utilisation du modèle appris

```
# 4. Predict what's on a few images! class1 or class2 ?
datamodule = ImageClassificationData.from_files(
    predict_files=[
    "data/test/class1/..  .jpg",
    "data/test/class2/..  .jpg",
    "data/test/class2/..  .jpg",
    ],
    batch_size=3,
)

predictions = trainer.predict(model,
                  datamodule=datamodule,
                  output="labels")

print(predictions)
```

Déroulement d'un programme en Lightning Flash



1. Data Loading
2. Data Transforms
3. Model Forwarding
4. Predictions Transforms
5. Predictions Serialisation

Sauvegarde et Chargement

Sauvegarde d'un modèle appris

```
1 trainer.save_checkpoint("image_classification_model.pt")
```

Chargement d'un modèle appris

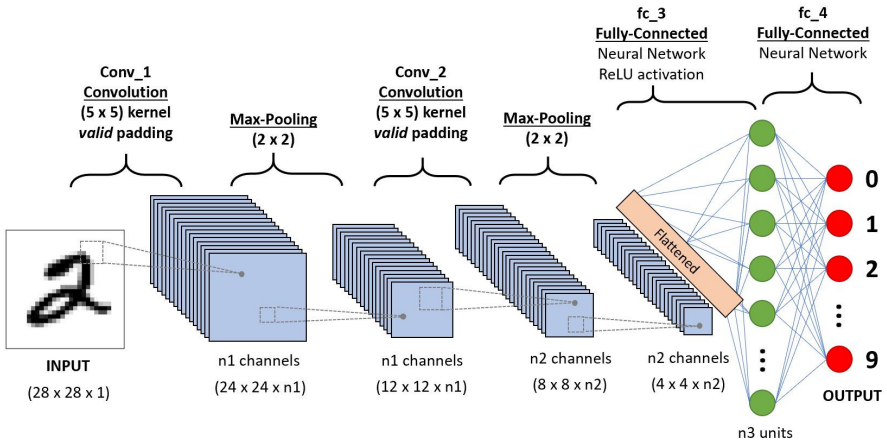
```
1 model = ImageClassifier.load_from_checkpoint(
2   "image_classification_model.pt"
3 )
```

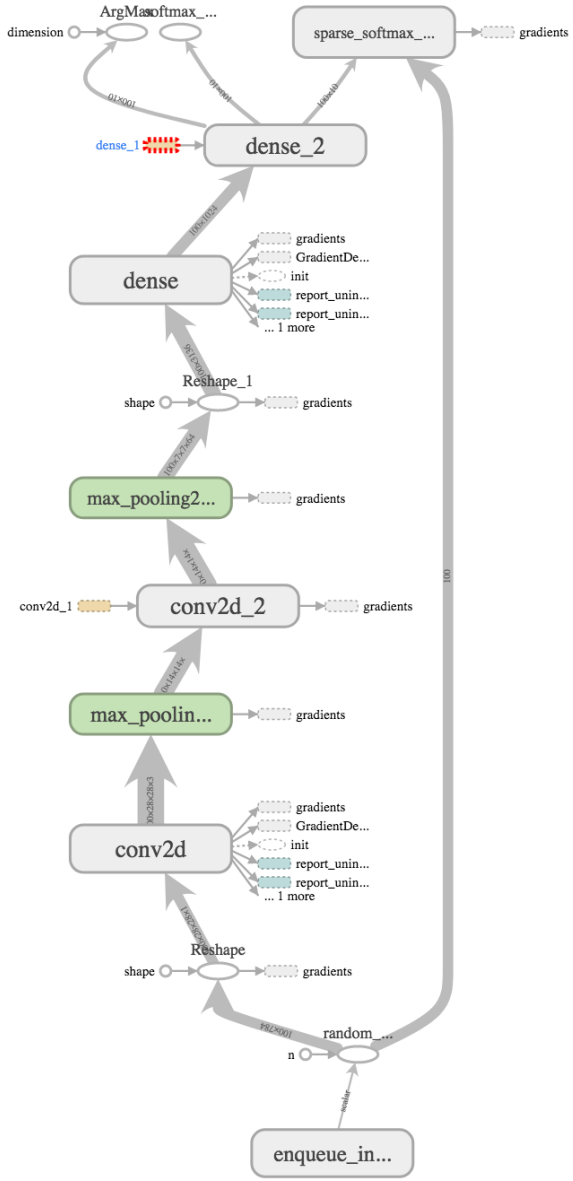
Sommaire

La notion de graphe de calcul

Un calcul par graphe

- Il est possible de définir un réseau de neurone par un graphe composé par un enchainement de blocs de calculs simples.
- Les calculs sont représentés par un graphe d'exécution.
- Chaque nœud correspond à une opération à réaliser.
- Les liens représentent les *Tensors* qui sont transmis d'une opération à l'autre.





La notion de graphe dynamique (1/2)

A graph is created on the fly


```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))
```

Source : pytorch.org

La notion de graphe dynamique (2/2)

Un graphe dynamique

- Contrairement au graphe statique (de type *Définition* puis *Exécution*), les graphes dynamiques suivent le paradigme : *Définition* par l'*Exécution*.
- Langage interprété à deux niveaux : interpréteur pour le langage de programmation (Python) et un second pour le graphe de calcul.

Avantages des graphes statiques

- Une fois que le graphe est construit, peut être utilisé de manière optimal sur un grand ensemble de données sans besoin d'être redéfini.
- Transfert sur d'autres architectures (par exemple pour mobile) plus simple.
- Opérateur plus simple qui nécessite par d'être adapté pour une gestion dynamique.
- Certaines erreurs peuvent être facilement trouvé lors de la définition du graphe sans attendre son exécution.

Avantages des graphes dynamiques

- Permet de gérer des données de tailles différentes (par exemple pour l'analyse de texte).
- Permet d'avoir des architectures dynamiques qui peuvent changer en fonction des données. Un graphe de calcul différent peut être directement construit à partir des données d'apprentissage avec des blocs partagés.
- Programmation plus proche de la programmation objet avec une définition modulaire des réseaux.
- Débuggage dynamique possible.

Sommaire

torch.nn et torch.nn.functional

Motivation

Il est important de différentier un nœud du graphe (aussi appelé couche/layer) et la fonction qui est exécuter par ce nœud. Le nœud contient en plus d'un pointeur sur la fonction à utiliser, les paramètres à utiliser ainsi que d'éventuelles variables internes.

Contenu de torch.nn

- Les fonctions de torch.nn permettent de définir les couches (layer) standard utilisées dans les réseaux de neurones,
- Les fonction torch.nn retourne des objets contenant à la fois les fonctions à appliquer pour effectuer les traitement de la couche et les variables internes pouvant être éventuellement apprisses.

Contenu de torch.nn.functional

- Contient les fonctions qui sont appelés dans les couches correspondantes définies dans torch.nn .
- Ne définit par de variables internes et nécessite les données sur lequel appliquer les opérations.
- Exécute directement l'opération.

Généralement le nom de la couche est en majuscule et le nom de la fonction en minuscule.

Couches entièrement connectées (FC)

La couche entièrement connectée

Cette couche est une couche entièrement connectée. Chaque entrée est connectée à tous les neurones.

```
layer_out = torch.nn.Linear(
    in_features=1024,
    out_features= 32,
    bias=True
)
# Exemple d'utilisation :
x = torch.rand(10,1024)
layer_out(x)

```

Fonction associée à la couche entièrement connectée

La fonction utilisée par le bloc *torch.nn.Linear* est *torch.nn.functional.linear* correspondant au calcul : $y = xA^\top + b$.

```
A = torch.rand(32,1024)
b = torch.rand(32)
x = torch.rand(10,1024)
torch.nn.functional.linear(x,A,b)
```

Les couches de convolution

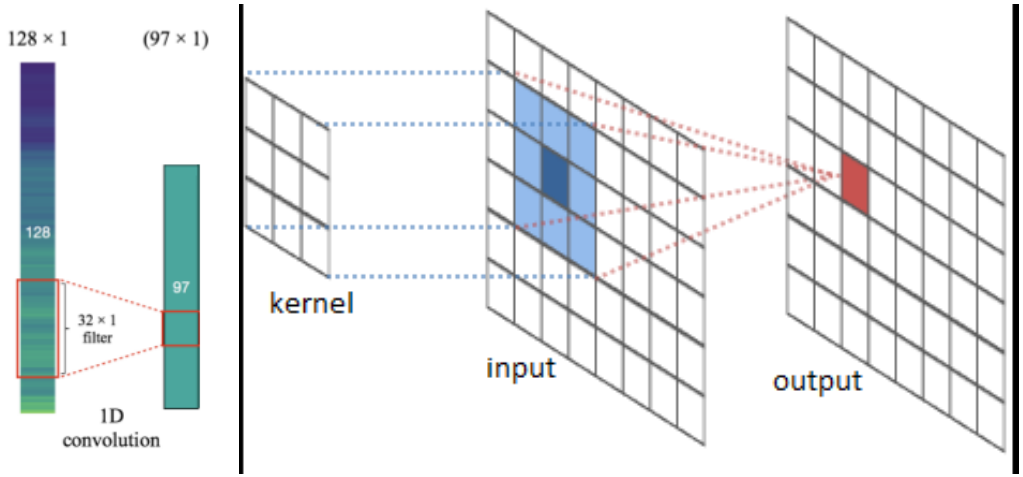
Couches de convolution avec torch.nn

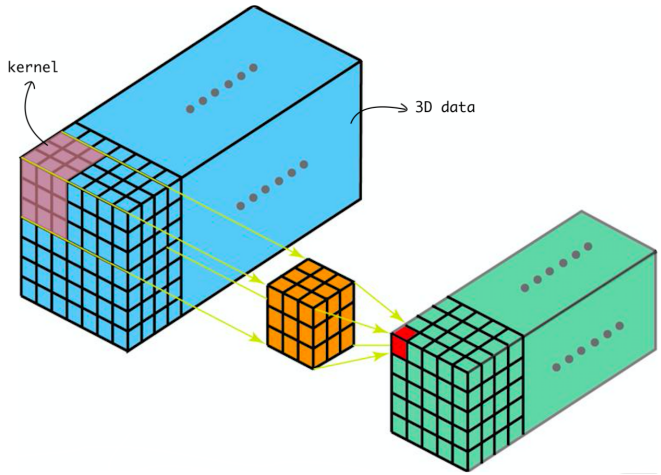
- torch.nn.Conv1d, torch.nn.Conv2d, torch.nn.Conv3d

Fonctions de convolution avec torch.nn.functional

- torch.nn.functional.conv1d, torch.nn.functional.conv2d, torch.nn.functional.conv3d

Différentes convolutions pour un canal d'entrée et un canal de sortie





Rq : Pytorch ne calcul pas une vraie convolution, mais utilise plutôt une auto-corrélation (le sens de parcours du noyau n'étant pas inversé).

Sommaire

torch.nn : Utilisation de *torch.nn.Sequential*

Définition d'un réseau

Il est possible de définir un réseau ou une partie de réseau comme une séquence de couche grâce à la fonction *torch.nn.Sequential* :

```
model = torch.nn.Sequential(
    torch.nn.Linear(28*28, 256),
    torch.nn.ReLU(),
    torch.nn.Linear(256,32),
    torch.nn.ReLU(),
    torch.nn.Linear(32,10),
)
```

Réseau à deux couches cachées de dimension 256 et 32. La couche de sortie à 10 neurones.

Forward sur le réseau

```
model(x)
```

Attention : x doit être un tenseur ayant les bonnes dimensions soit ici (nombre d'exemples, 28*28).

Définition d'un réseau par surcharge de *torch.nn.Module*

Définition d'un réseau sans *torch.nn.Sequential*

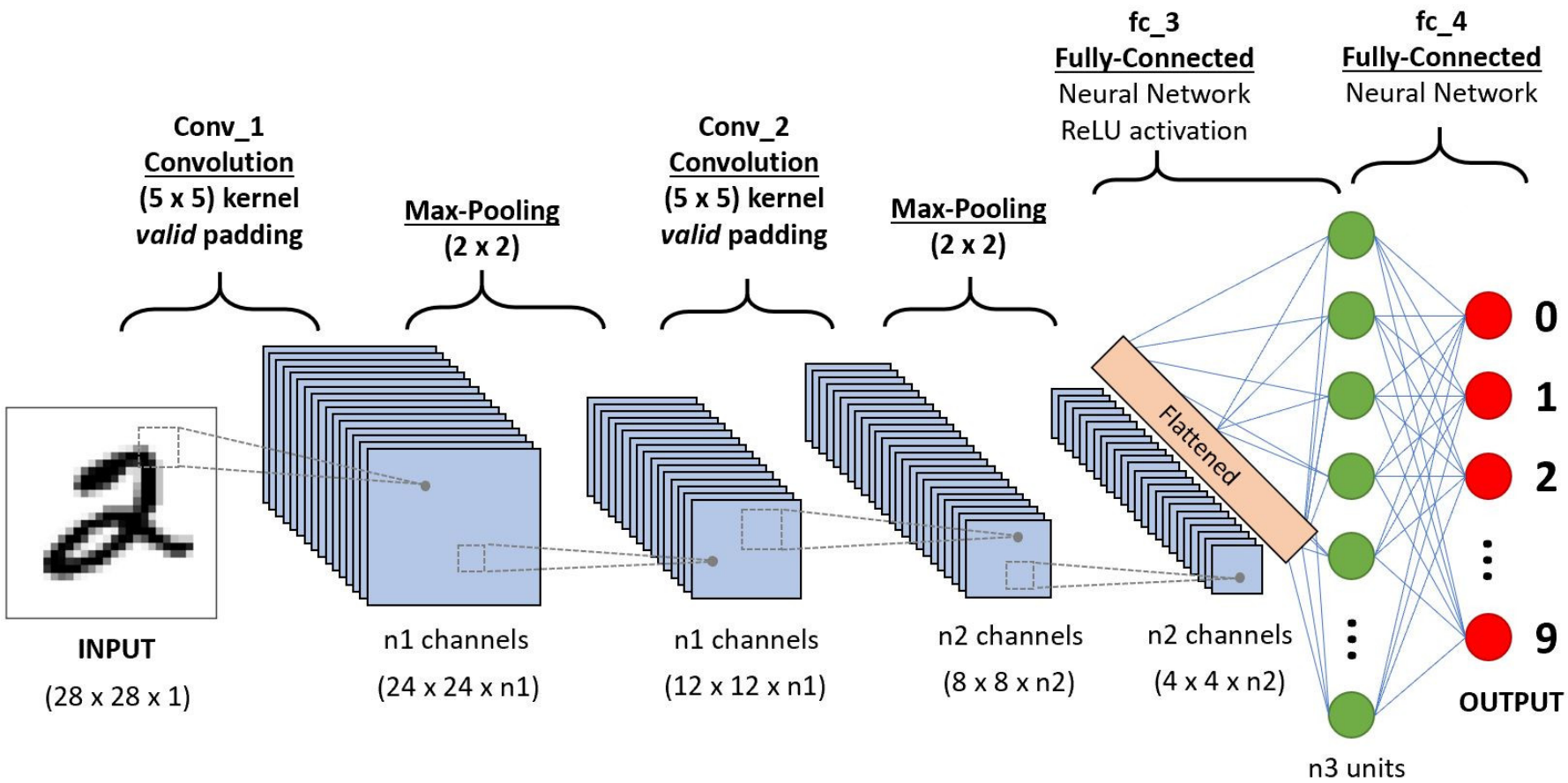
Il est possible de définir entièrement un réseau en créant une classe surchargeant *torch.nn.Module* :

```
1  class Net(nn.Module):
2      def __init__(self):
3          super(Net, self).__init__()
4          self.fc1 = nn.Linear(28*28, 256)
5          self.fc2 = nn.Linear(256,32)
6          self.fc3 = nn.Linear(32,10)
7
8      def forward(self, x):
9          out = nn.functional.relu(self.fc1(x))
10         out = nn.functional.relu(self.fc2(out))
11         out = self.fc3(out)
12     return out
```

Sommaire

Le réseau LeNet5

Architecture d'un réseau LeNet5



torch.nn : Définition d'un réseau sans nn.Sequential

CNN pour MNIST

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, 5, 1)
        self.conv2 = nn.Conv2d(16, 32, 5, 1)
        self.fc1 = nn.Linear(4*4*32, 512)
        self.fc2 = nn.Linear(512, 10)

    def forward(self, x):
        x = nn.functional.relu(self.conv1(x))
        x = nn.functional.max_pool2d(x, 2, 2)
        x = nn.functional.relu(self.conv2(x))
        x = nn.functional.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*32)
        x = nn.functional.relu(self.fc1(x))
        x = self.fc2(x)
        return x #nn.functional.log_softmax(x, dim=1)

net1 = Net()
```

torch.nn : Définition d'un réseau avec nn.Sequential

CNN pour MNIST

```
class ConvNet(nn.Module):
    def __init__(self, num_classes=10):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=5, stride=1, padding=0),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=5, stride=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.fc = nn.Sequential(nn.Linear(4*4*32, 512),nn.ReLU(),nn
    .Linear(512, num_classes))
    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        return out
net2 = ConvNet()
```

torch.nn : Définition d'un réseau 100% avec nn.Sequential

CNN pour MNIST

```
net3 = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=5, stride=1),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(16, 32, kernel_size=5, stride=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Flatten(),
            nn.Linear(7*7*32, 512),
            nn.ReLU(),
            nn.Linear(512, 10),
)
```

Architecture PyTorch et Lightning Flash

Ajout d'une architecture dans la base des réseaux de Lightning Flash

```python
from flash.image.classification.backbones import
    IMAGE_CLASSIFIER_BACKBONES

@IMAGE_CLASSIFIER_BACKBONES(name="lenet5", namespace="vision")
def load_lenet5(num_features,*_, **__):
  return (Lenet5(num_features),512)

model_lenet5 = ImageClassifier(
  backbone="lenet5",
  backbone_kwargs={'num_features':3},
  num_classes=10,
)

```

Sommaire

Tensorboard

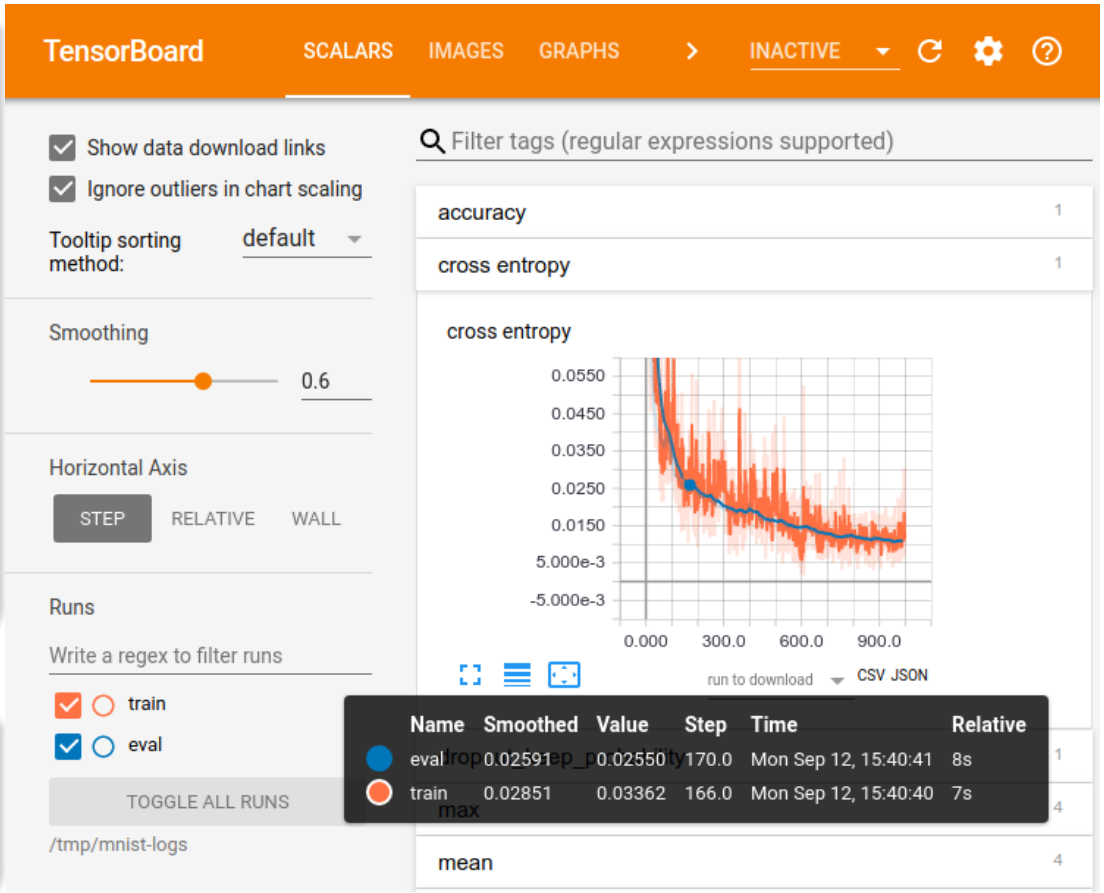
Présentation

Tensorflow met à disposition un outil de monitoring : Tensorboard. Cet outil est désormais compatible avec un code PyTorch. Il permet de visualiser facilement l'apprentissage d'un réseau et de régler par la suite ses différents paramètres.

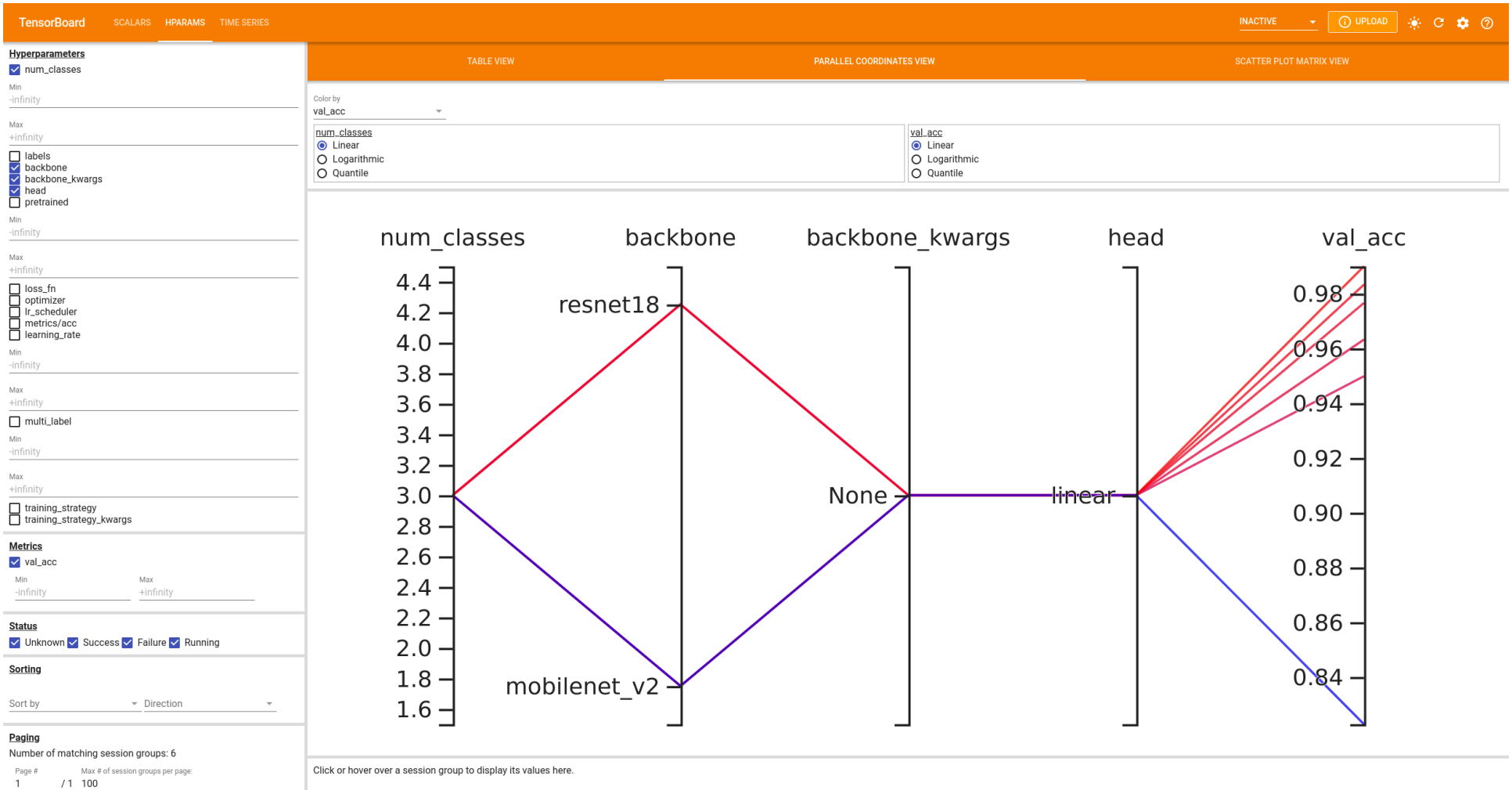
Lancement de Tensorboard

Dans un terminal, tapez :

```
tensorboard -logdir=chemin_des_logs
```



Tensorboard - Comparaison des hyperparamètres



Tensorboard - Visualisation du graphe de calcul