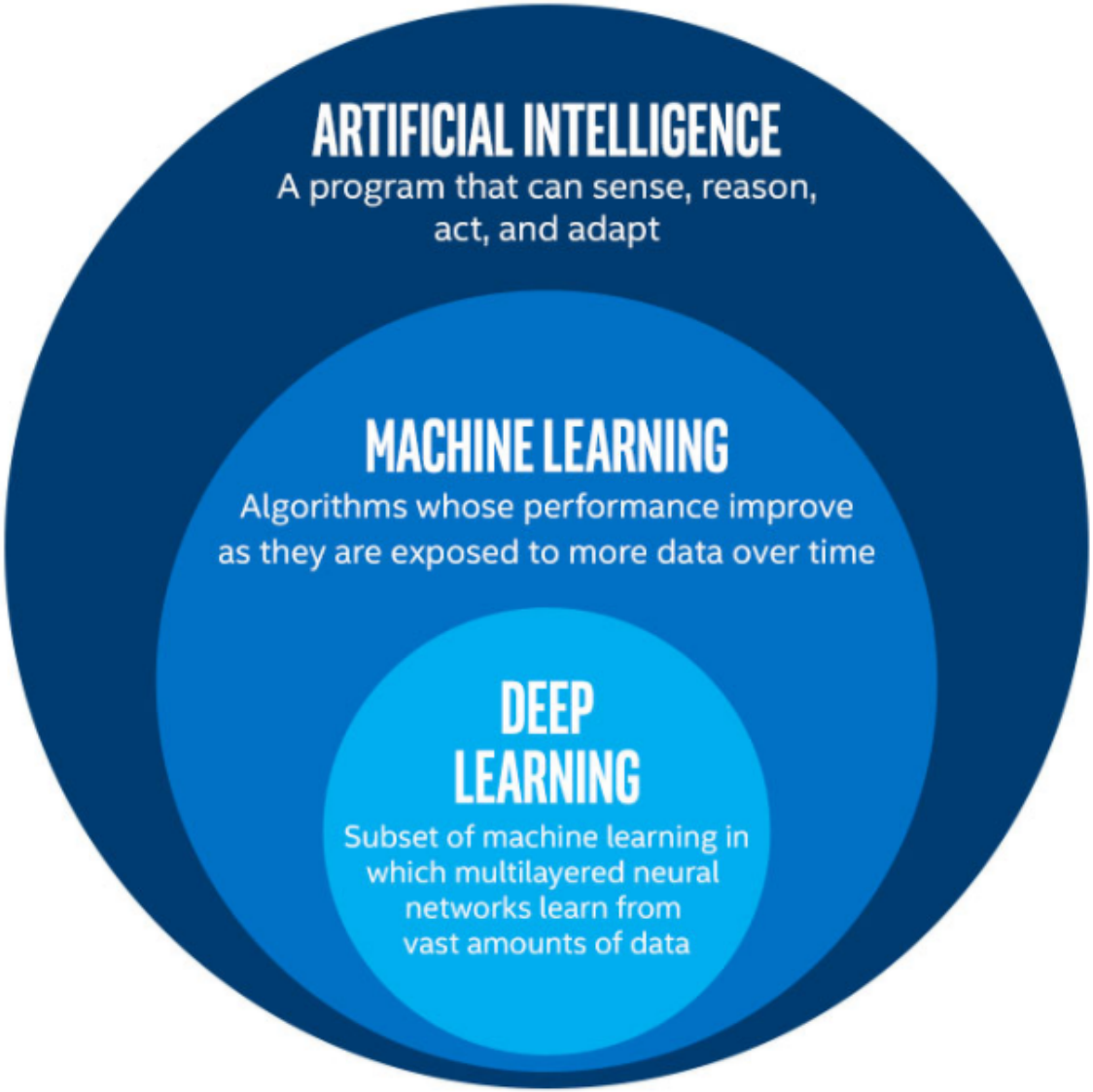
Réseaux de Neurones & Deep Learning – 1/2

Master parcours SSD - UE Apprentissage Statistique II

Pierre Mahé - bioMérieux & Université de Grenoble-Alpes

Introduction

Machine Learning / Intelligence Artificielle / Deep Learning ?

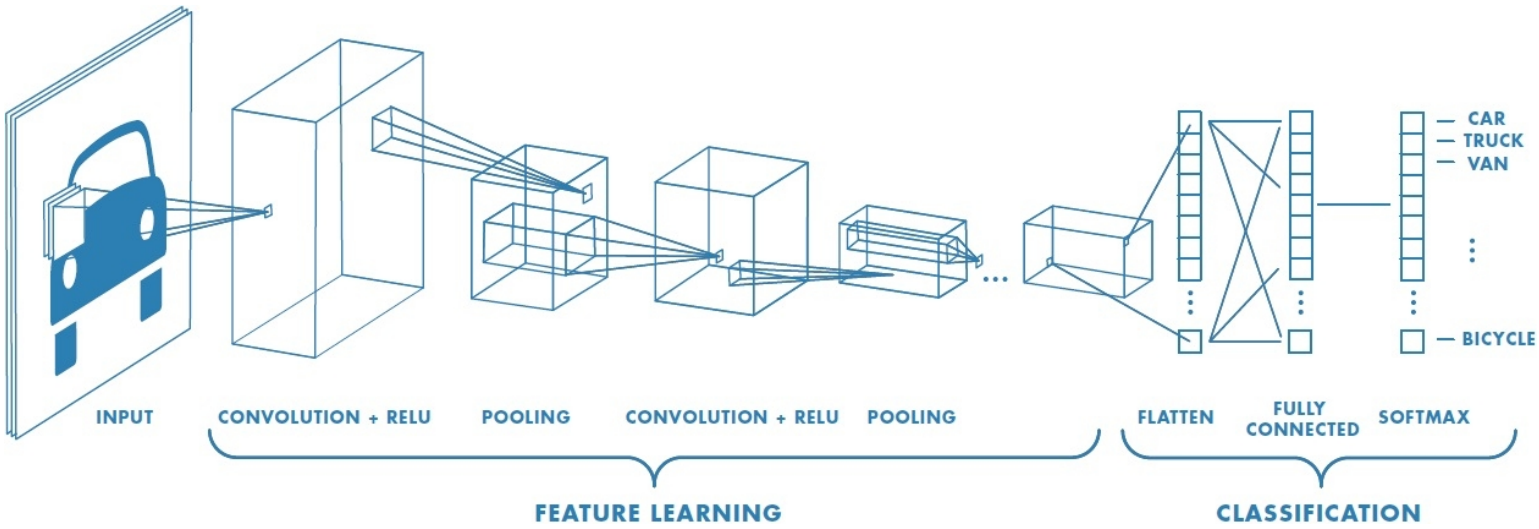


Deep Learning

Wikipedia : L'apprentissage profond est un ensemble de méthodes d'apprentissage automatique tentant de modéliser avec un haut niveau d'abstraction des données grâce à des architectures articulées de différentes transformations non linéaires. Ces techniques ont permis des progrès importants et rapides dans les domaines de l'analyse du signal sonore ou visuel et notamment de la reconnaissance faciale, de la reconnaissance vocale, de la vision par ordinateur, du traitement automatisé du langage.

Deep learning & apprentissage de la représentation

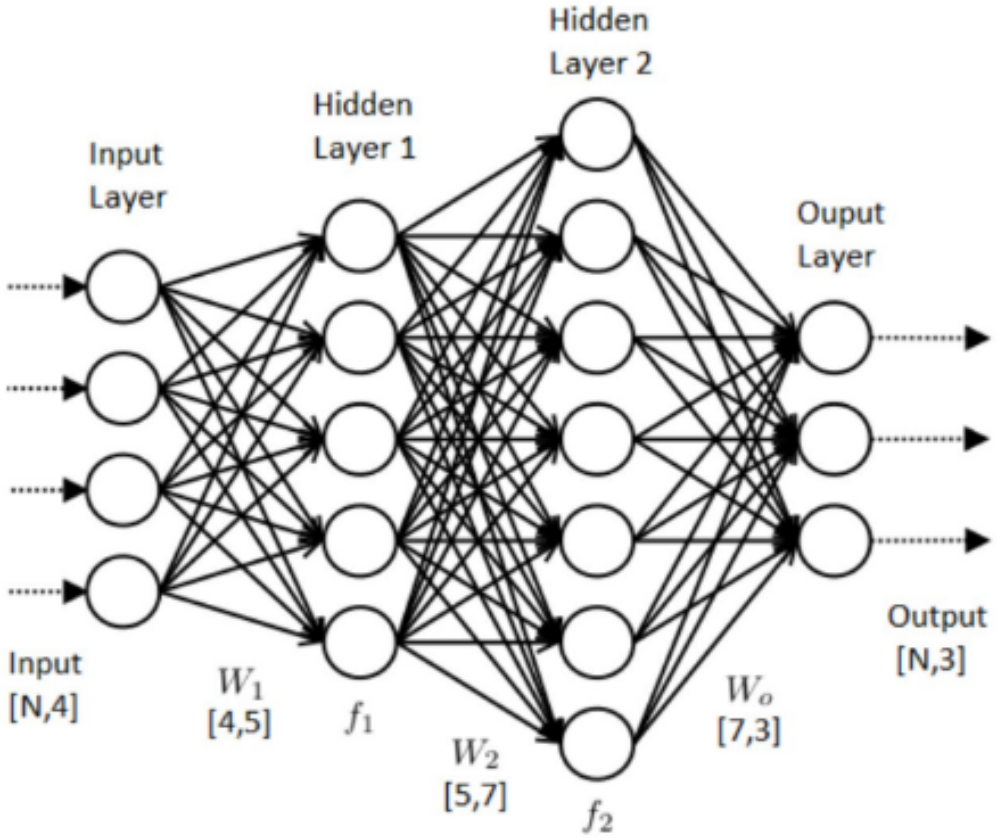
Cascade de transformations (non-linéaires) :



- descripteurs non définis a priori : appris sur les données
  - apprentissage de la représentation
  - approche "end to end"
- basés sur des réseaux de neurones
  - e.g., convolutifs (CNNs), récurrents

https://fr.mathworks.com/discovery/convolutional-neural-network.html

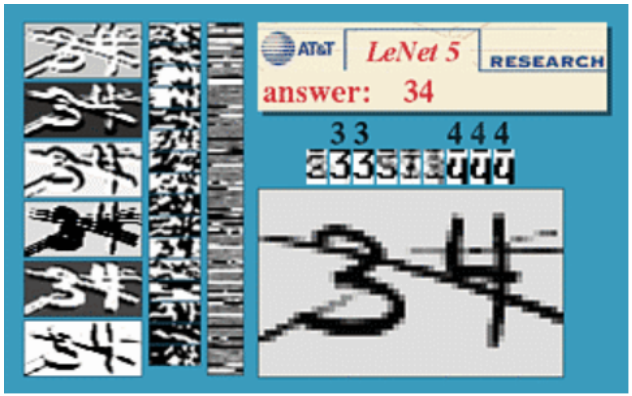
Deep learning & réseaux de neurones



- 1950's : modèle à 1 neurone = perceptron
  - neurone artificiel / formel
- 1980's : plusieurs couches = perceptron multi-couches
  - théorème d'approximation universel

Deep Learning - early days

Années 1980-90's : de nombreux développements et des applications convaincantes (http://yann.lecun.com/exdb/lenet/)



- (réseaux convolutifs & reconnaissance de caractères)

Néanmoins : modèles difficiles à optimiser

- impliquent nombreux paramètres
- fonctions fortement non-linéaires

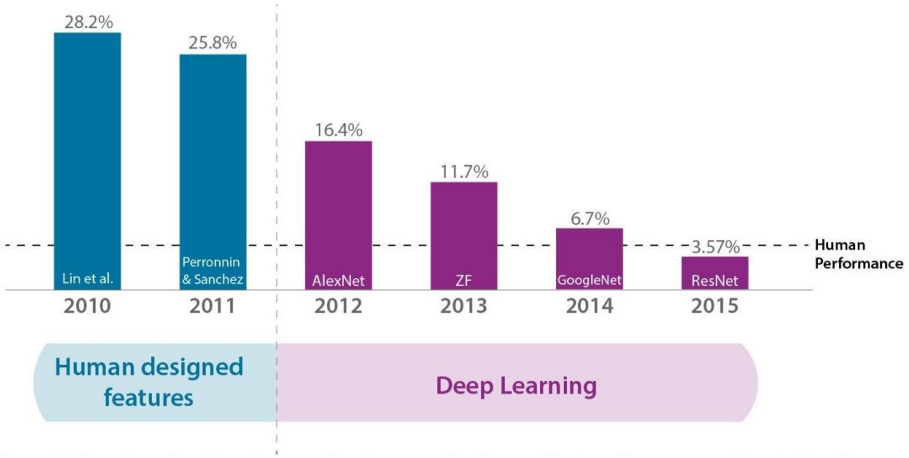
⇒ années 90-200 : état de l'art = modèles convexes

- e.g., SVMs et méthodes à noyaux

Deep Learning - the ImageNet 2012 revolution

Résultats de la compétition ImageNet (2010 à 2015) :

- top-5 error rate



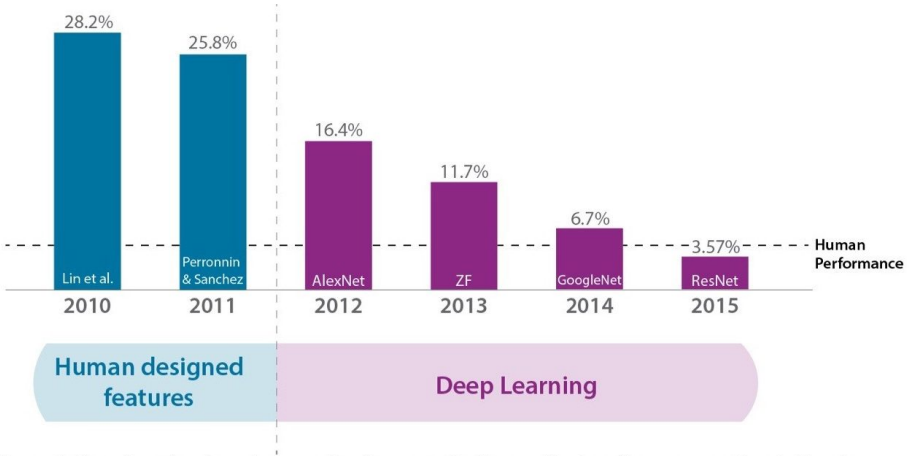
- apport significatif du DL en 2012
- adoption forte par la communauté

https://www.mariner-usa.com/blog/deep-learning-vs-traditional-machine-vision/

Deep Learning - the ImageNet 2012 revolution

Résultats de la compétition ImageNet (2010 à 2015) :

- top-5 error rate



- apport significatif du DL en 2012
- adoption forte par la communauté

https://www.mariner-usa.com/blog/deep-learning-vs-traditional-machine-vision/

Plusieurs raisons :

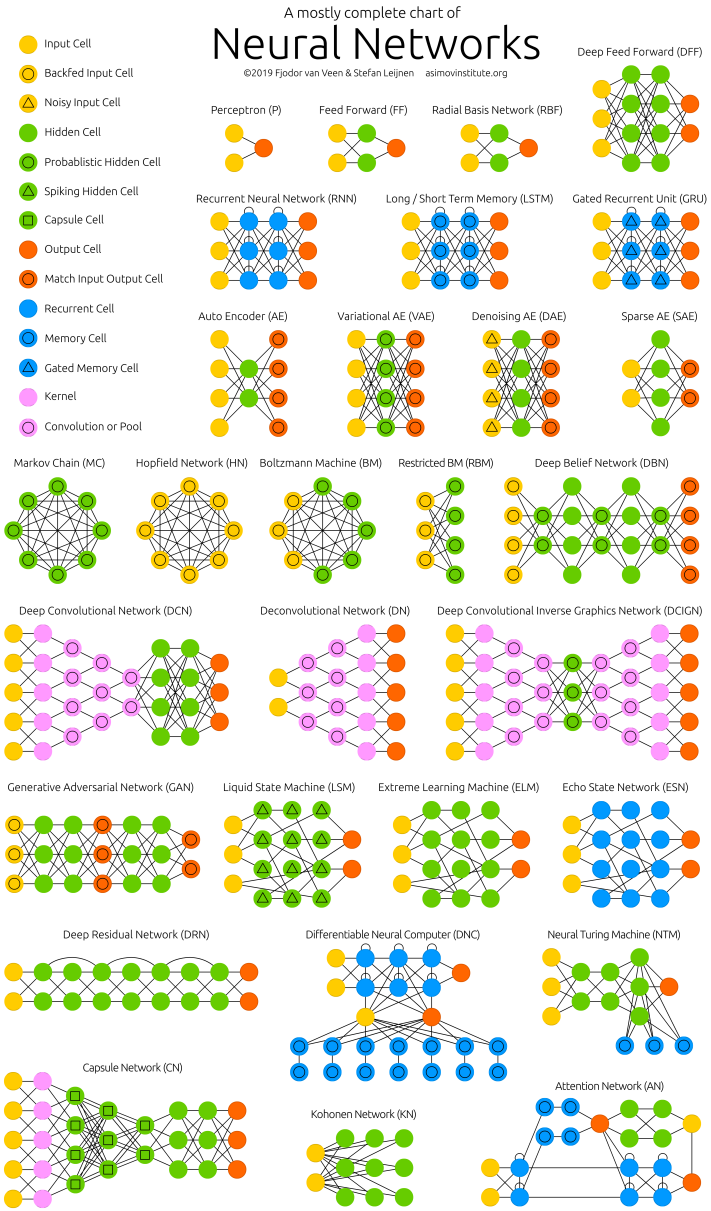
1. jeu de données plus grand (10^6 images pour 10^3 classes)
2. architectures matérielles plus puissantes (GPUs)
3. amélioration des algorithmes (ReLu, DropOut)

Deep Learning - aujourd'hui

Depuis 2012 : domaine très actif

- grande diversité de modèles et d'architectures
  - the Neural Network Zoo
- nombreuses applications
  - domaines "historiques" : image, NLP, parole
  - + biologie / santé, physique / signal, ...
- recherche académique
  - nouveaux modèles, interprétabilité, optimiser l'architecture, comprendre la généralisation, ...
- plusieurs plateformes logicielles

The Neural Network Zoo



https://www.asimovinstitute.org/neural-network-zoo/

Ce cours...

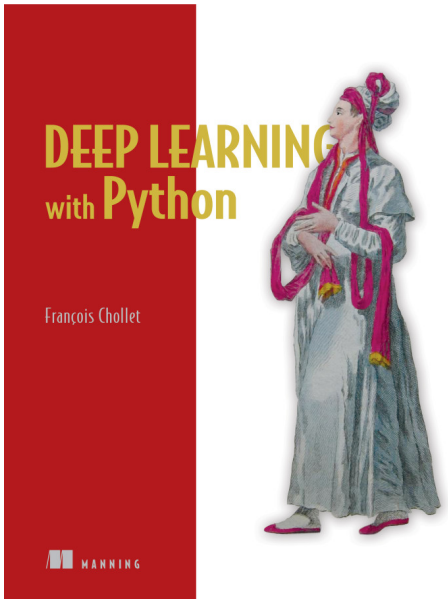
Une **introduction** au domaine du deep-learning.

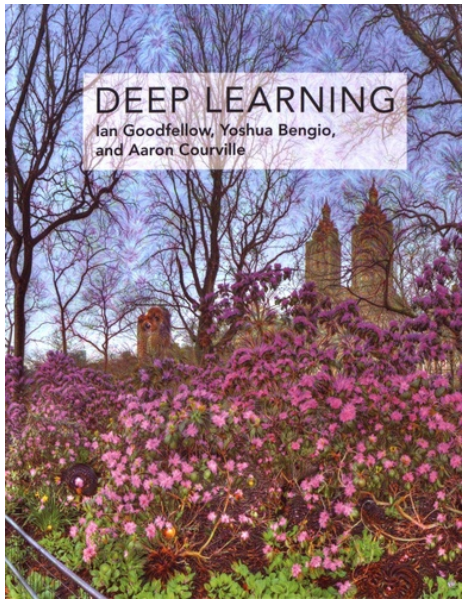
- neurone & réseaux de neurones
- perceptrons multi-couches
- réseaux convolutifs (Convolutional Neural Networks)

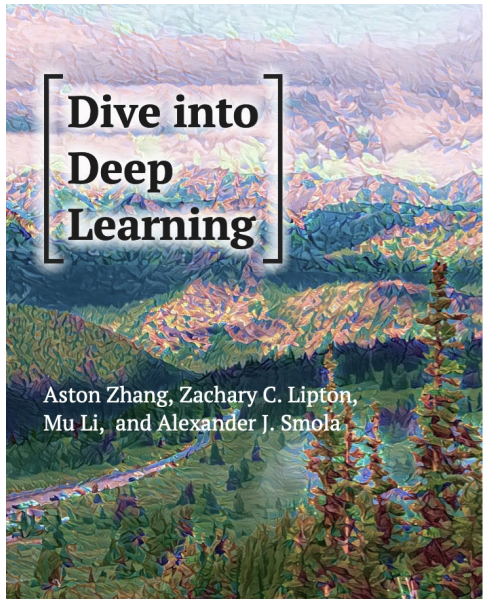
Mise en oeuvre avec Keras

- package python
- interface "haut niveau" à TensorFlow

Sources principales



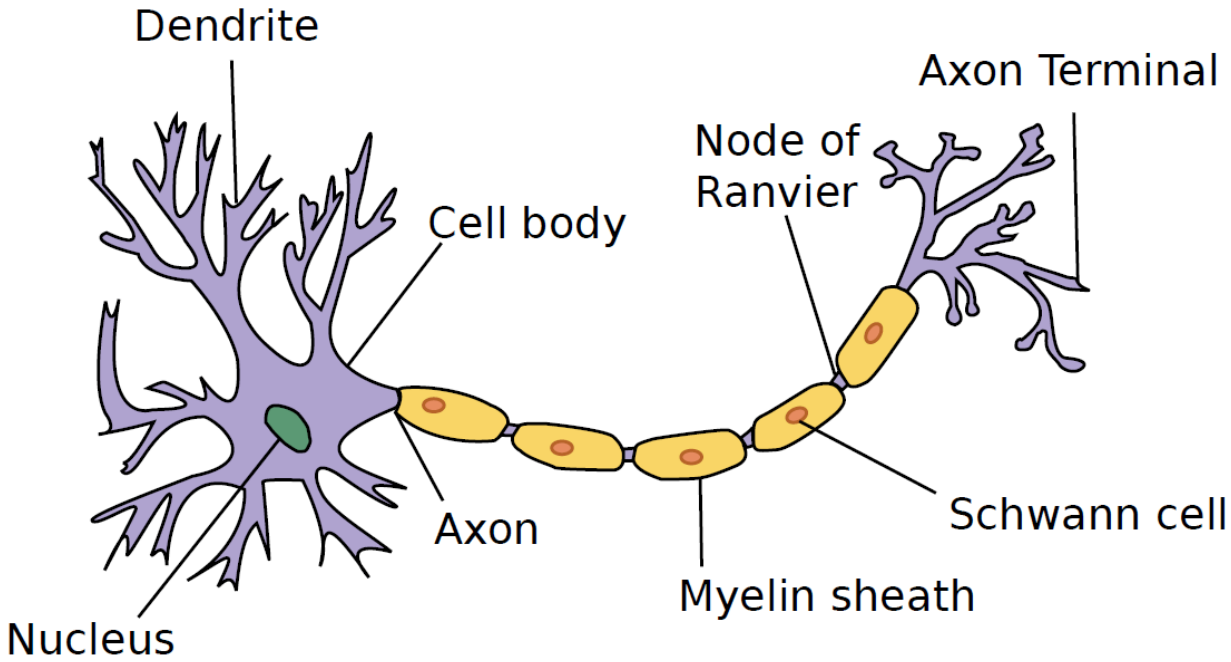




Architectures

- neurones artificiels et modèles linéaires
- perceptrons multi-couches
- réseaux convolutifs

Le neurone biologique [1]

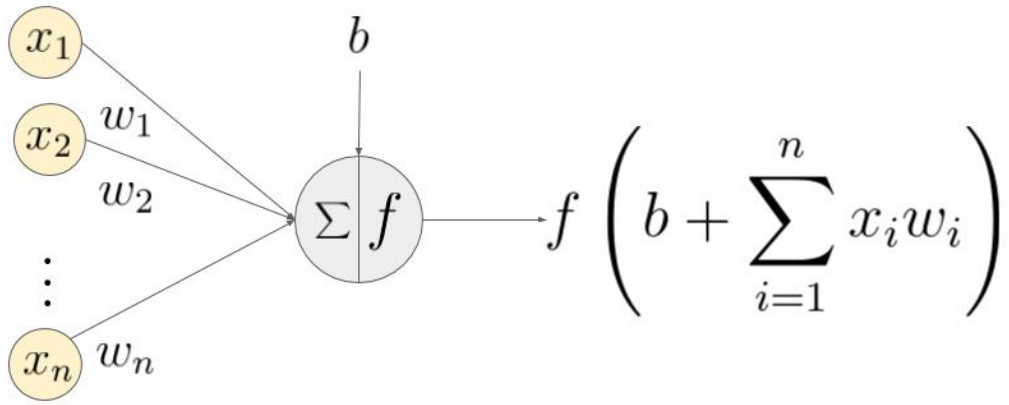


- les dendrites reçoivent l'influx nerveux d'autres neurones
- l'axone libère un influx si le neurone est assez excité
- les synapses relient les neurones entre eux
- le cerveau contient $\sim 80 - 100 \times 10^9$ neurones

1. Si ce n'est pas précisé : figures tirées de `https://d2l.ai/`

Le neurone artificiel

Un modèle mathématique du neurone biologique :



1. reçoit des variables d'entrée $(x_1, ..., x_n)$
2. calcule leur combinaison linéaire avec les poids $(w_1, ..., w_n)$
3. y ajoute un terme de biais b
4. applique une transformation f au score pour fournir une valeur de sortie

Neurone artificiel et régressions linéaires

Pour une observation $x \in \mathbb{R}^p$, un neurone artificiel calcule :

$$h(x) = f\Big(\sum_{i=1}^{p} w_i x_i + b\Big)$$

$\Rightarrow$ il implémente des modèles linéaires (généralisés) :

- régression linéaire si $f = \mathrm{Id}$ (fonction identité)
- régression logistique si $f(x) = \frac{1}{1+\exp(-x)}$

Neurone artificiel et régressions linéaires

Pour une observation $x \in \mathbb{R}^p$, un neurone artificiel calcule :

$$h(x) = f\Big(\sum_{i=1}^{p} w_i x_i + b\Big)$$

$\Rightarrow$ il implémente des modèles linéaires (généralisés) :

- régression linéaire si $f = \mathrm{Id}$ (fonction identité)
- régression logistique si $f(x) = \frac{1}{1+\exp(-x)}$

La fonction f est la fonction d'activation du neurone.

Neurone artificiel et régressions linéaires

Pour une observation $x \in \mathbb{R}^p$, un neurone artificiel calcule :

$$h(x) = f\Big(\sum_{i=1}^{p} w_i x_i + b\Big)$$

$\Rightarrow$ il implémente des modèles linéaires (généralisés) :

- régression linéaire si $f = \mathrm{Id}$ (fonction identité)
- régression logistique si $f(x) = \frac{1}{1+\exp(-x)}$

La fonction f est la fonction d'activation du neurone.

Apprentissage = apprendre ses poids w_i et son biais b.

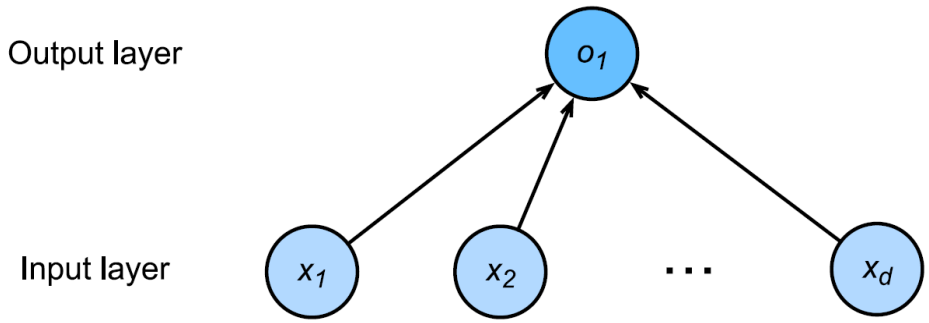
- 1957 : algorithme du perceptron (Rosenblatt)

Neurone artificiel et régressions linéaires

Construction par "couches" des réseaux de neurones :

- couche d'entrée : les variables d'entrée $(x_1, ...x_p)$
- couches internes ou cachées
- couche de sortie : la valeur prédite par le modèle

$\Rightarrow$ neurone artificiel, régressions linéaire et logistique = réseaux de neurones à 1 couche (la couche de sortie)



- avec 1 seul neurone sur la couche de sortie

Neurones artificiels et classification multiclasse

Généralisation de la régression logistique à la classification multi-classe = la régression multinomiale :

$$P(Y = k|x) = \frac{\exp(\langle w^{(k)}, x\rangle + b_k)}{\sum_{l=1}^{K} \exp(\langle w^{(l)}, x\rangle + b_l)}$$

$\Rightarrow$ K classes, classification : $\underset{k=1,\ldots,K}{\operatorname{argmax}} P(Y = k|x)$

- modèle linéaire de classification multi-classe
- un vecteur $w^{(k)} \in \mathbb{R}^p$ par classe

Neurones artificiels et classification multiclasse

Pour un vecteur $z = [z_1, ..., z_K]$ de K valeurs, la fonction

$$f_k(z) = \frac{\exp(z_k)}{\sum_{l=1}^{K} \exp(z_l)}$$

est la fonction softmax (associée à la valeur k).

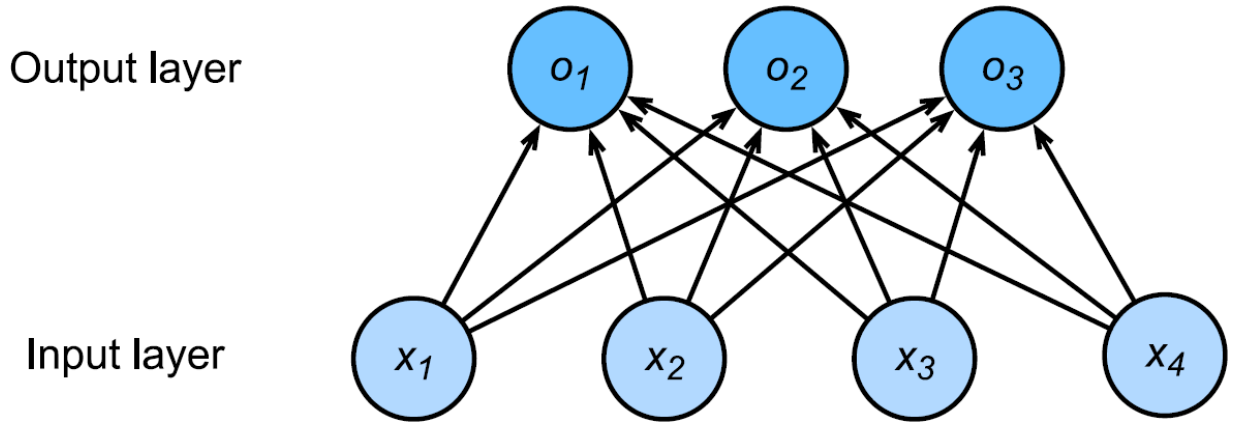
- convertit des "scores" en probabilités (≥ 0, $\sum . = 1$)
- si le score z_k est (bien) + fort que les autres, $f_k(z) \sim 1$
- comportement du max, mais différentiable

$\Rightarrow$ régression multinomiale = "softmax regression"

- avec $z_k = \langle w^{(k)}, x \rangle + b_k$

Neurones artificiels et classification multiclasse

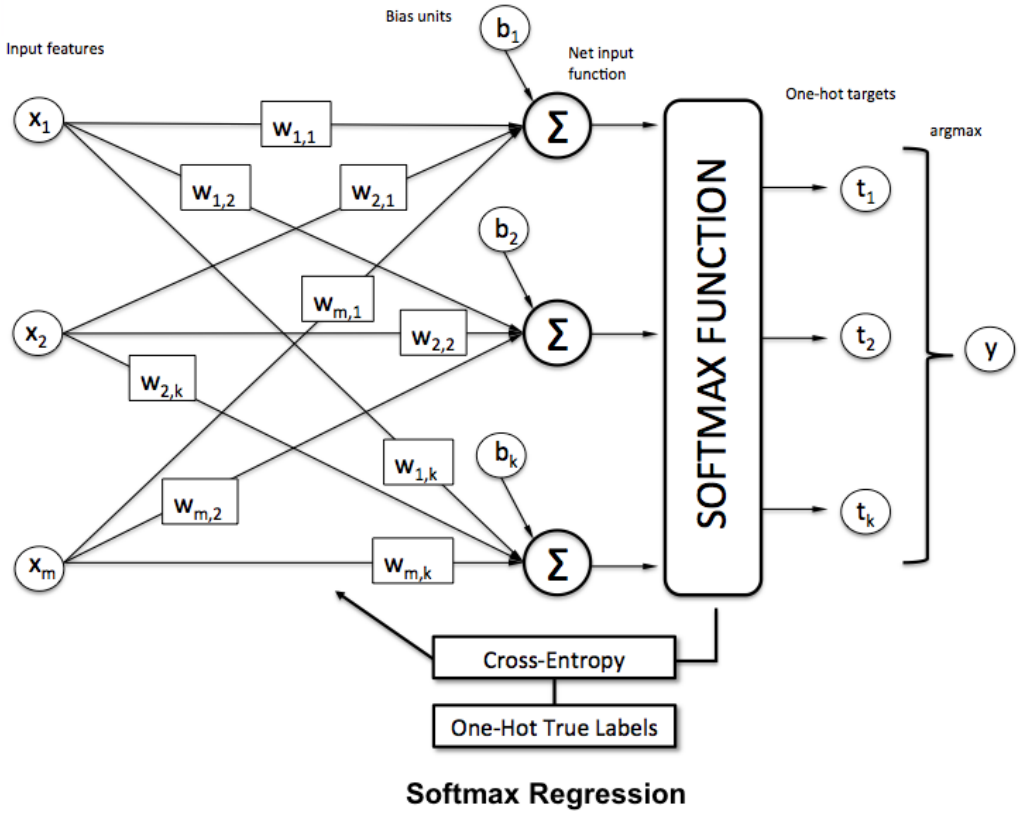
Formalisation en réseaux de neurones :



- 1 neurone par classe dans la couche de sortie
- nombre de connections = $(p+1) \times K$ (+1 : les biais)
- softmax = fonction d'activation de la couche de sortie

Neurones artificiels et classification multiclasse

Formalisation en réseaux de neurones :



- softmax = fonction d'activation de la couche de sortie

https://sebastianraschka.com/faq/docs/softmax_regression.html

Exemples de mise en oeuvre Keras

Régression linéaire :

```python
# import
from keras.models import Sequential
from keras.layers import Dense
# define parameters
n_units = 1
p = 100
# build model
model = Sequential()
model.add(Dense(n_units, input_dim=p, activation='linear'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 101
Trainable params: 101
Non-trainable params: 0
_________________________________________________________________
```

- "Sequential" model, "Dense" layer
- # paramètres $= (p + 1)$

Exemples de mise en oeuvre Keras

Régression logistique :

```
# import
from keras.models import Sequential
from keras.layers import Dense
# define parameters
n_units = 1
p = 100
# build model
model = Sequential()
model.add(Dense(n_units, input_dim=p, activation='sigmoid'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 101
Trainable params: 101
Non-trainable params: 0
_________________________________________________________________
```

- activation = sigmoid
- # paramètres = $(p + 1)$

Exemples de mise en oeuvre Keras

Régression softmax / multinomiale :

```
# import
from keras.models import Sequential
from keras.layers import Dense
# define parameters
n_classes = 10
p = 100
# build model
model = Sequential()
model.add(Dense(n_classes, input_dim=p, activation='softmax'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10)                1010
=================================================================
Total params: 1,010
Trainable params: 1,010
Non-trainable params: 0
_________________________________________________________________
```

- `n_classes` neurones + activation = softmax
- # paramètres = `n_classes` $\times (p+1)$

Bilan intermédiaire...

- neurone artificiel= { poids w_i, biais b, activation f }
- modèles linéaires = réseaux de neurones à 1 couche
  - 1 neurone pour régressions linéaire et logistique
  - K neurones pour régression multinomiale
- fonctions d'activation : identité, logistique, softmax
- premières couches `Keras`
  - Dense ou "fully connected"

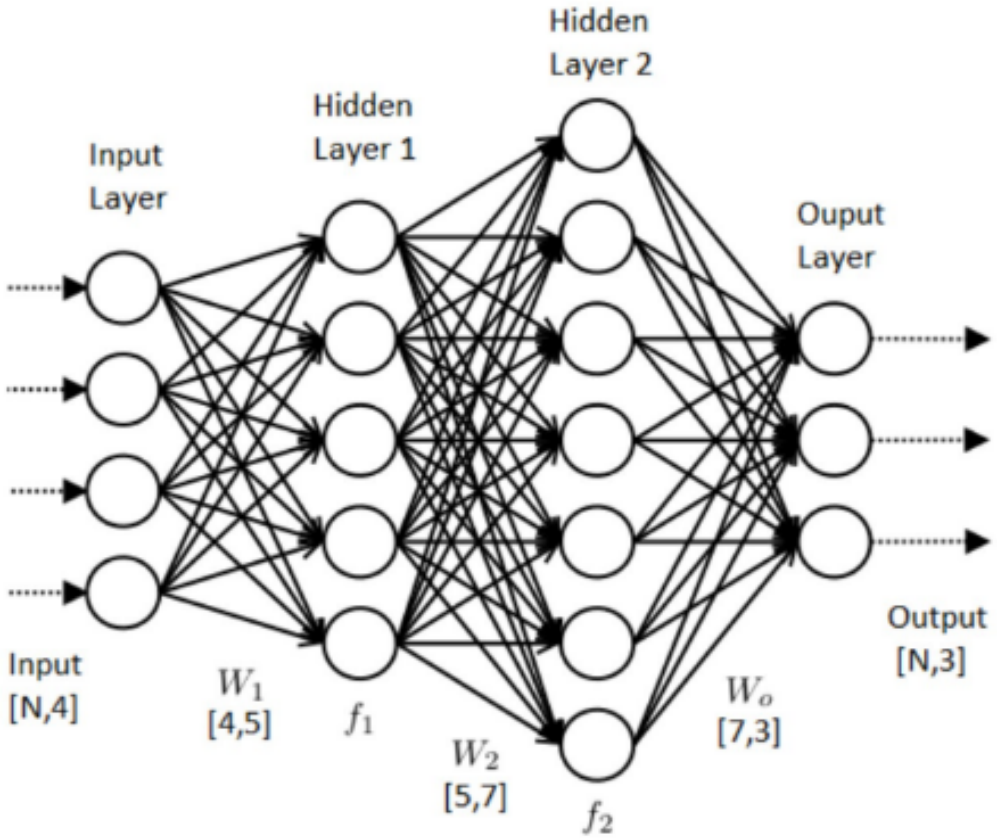
$\Rightarrow$ éléments de base des architectures plus complexes...

Architectures

- neurones artificiels et modèles linéaires
- perceptrons multi-couches
- réseaux convolutifs

Perceptron multi-couches

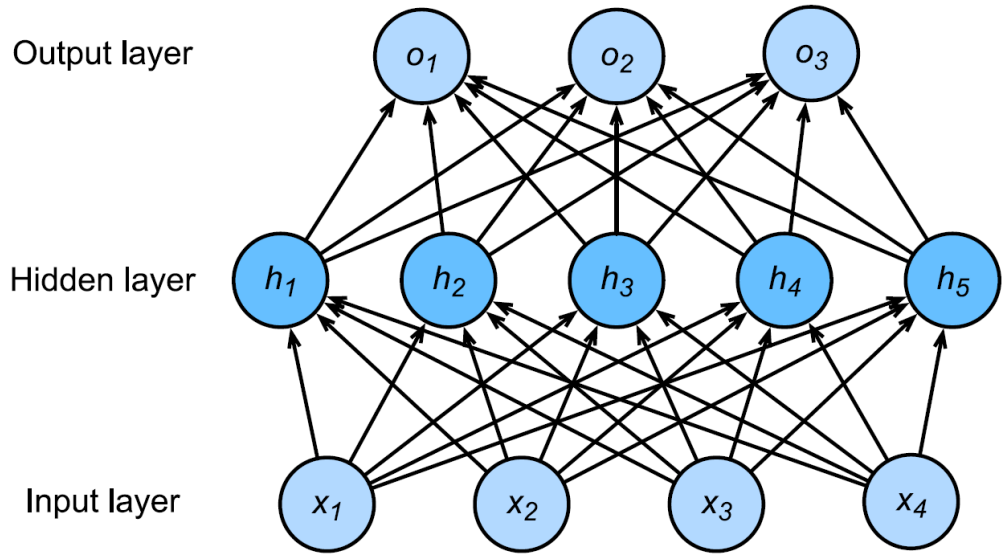
Perceptron multi-couches (multilayer perceptron - MLP) :



$\Rightarrow$ deux couches internes / cachées de 5 et 7 neurones

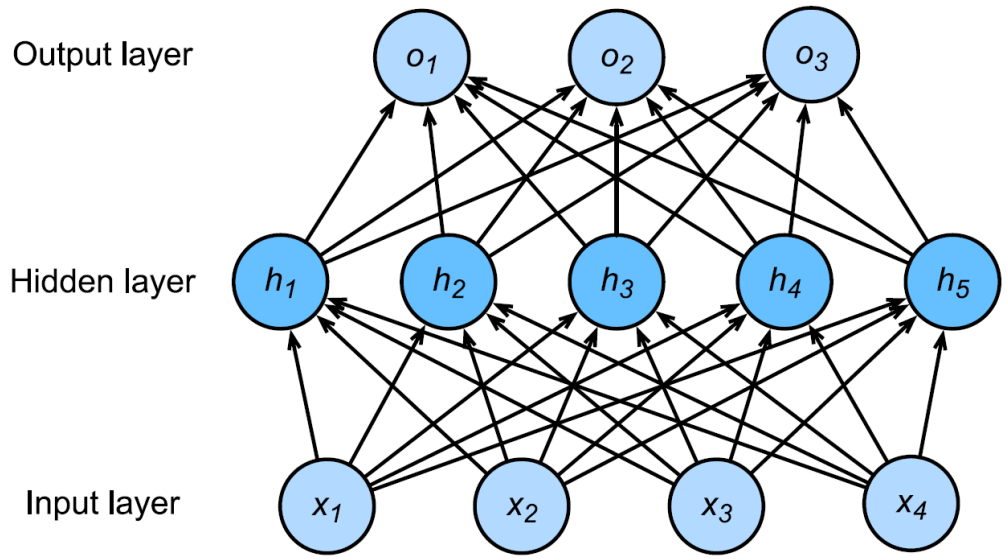
$\Rightarrow$ structure entièrement connectée (fully connected)

Perceptron multi-couches - formalisation



- couche cachée : $h_j = f\left(\sum_{i=1}^{4} w_{j,i}^{(1)} x_i + b_j^{(1)}\right), j = 1, ..., 5.$
- couche de sortie : $o_k = \sum_{j=1}^{5} w_{k,j}^{(2)} h_j + b_k^{(2)}, \ k = 1, ..., 3.$
- prédiction : $\hat{y}_k = \text{softmax}(o_k)$

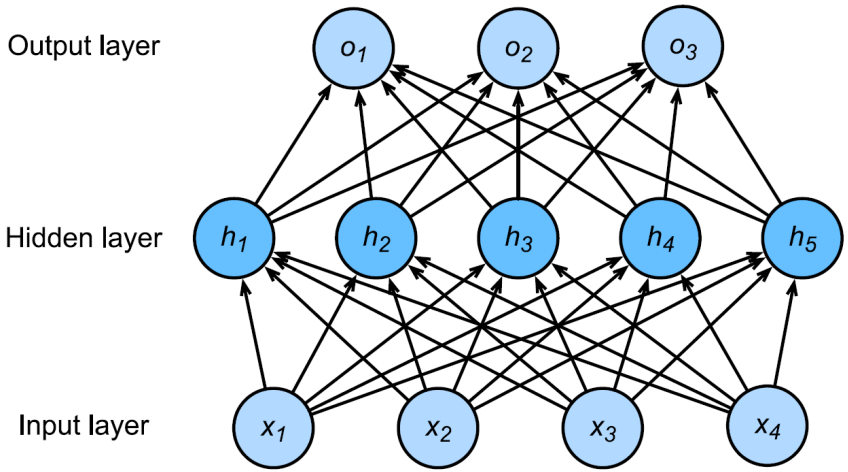
Perceptron multi-couches - formalisation



$\Rightarrow$ en écriture matricielle :

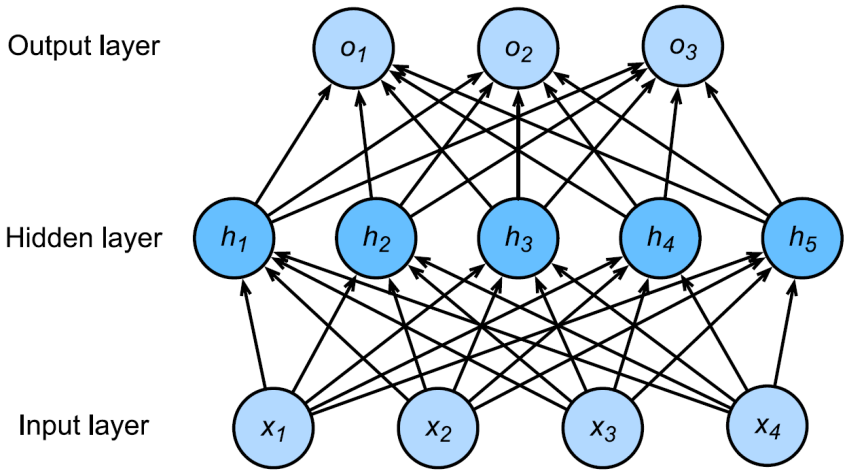
- couche cachée : $\mathbf{h} = f\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right)$
  - $\mathbf{W}_1 \in \mathbb{R}^5 \times \mathbb{R}^4$; $\mathbf{h}, \mathbf{b}_1 \in \mathbb{R}^5$
- couche de sortie : $\mathbf{o} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$.
  - $\mathbf{W}_2 \in \mathbb{R}^3 \times \mathbb{R}^5$; $\mathbf{o}, \mathbf{b}_2 \in \mathbb{R}^3$
- prédiction : $\hat{\mathbf{y}} = \text{softmax}(\mathbf{o})$

Perceptron multicouche - formalisation



- couche cachée : $\mathbf{h} = f\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right)$
- couche de sortie : $\mathbf{o} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$.
- prédiction : $\mathbf{y} = \text{softmax}(\mathbf{o})$

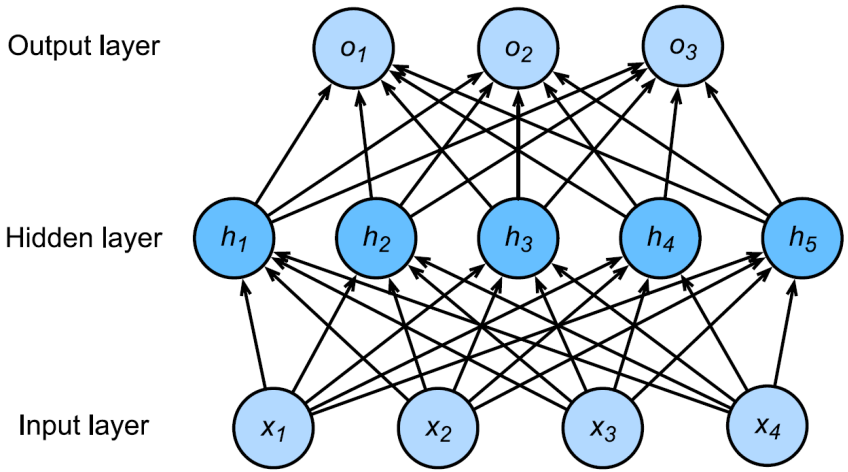
Perceptron multicouche - formalisation



- couche cachée : $\mathbf{h} = f\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right)$
- couche de sortie : $\mathbf{o} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$.
- prédiction : $\mathbf{y} = \text{softmax}(\mathbf{o})$

Quizz : quel est l'élément clé dans tout ça ?

Perceptron multicouche - formalisation

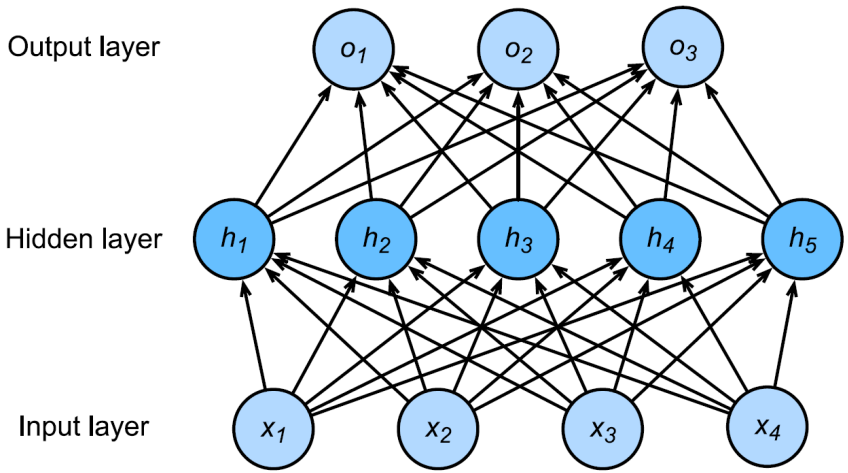


- couche cachée : $\mathbf{h} = f\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right)$
- couche de sortie : $\mathbf{o} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$.
- prédiction : $\mathbf{y} = \text{softmax}(\mathbf{o})$

Quizz : quel est l'élément clé dans tout ça ?

$\Rightarrow$ la fonction d'activation f

Perceptron multicouche - formalisation



- couche cachée : $\mathbf{h} = f\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right)$
- couche de sortie : $\mathbf{o} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$.
- prédiction : $\mathbf{y} = \text{softmax}(\mathbf{o})$

Avec une fonction d'activation linéaire de la couche interne :

$$\begin{aligned}\mathbf{o} &= \mathbf{W}_2\mathbf{h} + \mathbf{b}_2 \\ &= \mathbf{W}_2\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right) + \mathbf{b}_2 \\ &= \mathbf{W}_2\mathbf{W}_1\mathbf{x} + \left(\mathbf{b}_2 + \mathbf{W}_2\mathbf{b}_1\right)\end{aligned}$$

$\Rightarrow$ le réseau implémente un modèle linéaire !

MLP et non-linéarité

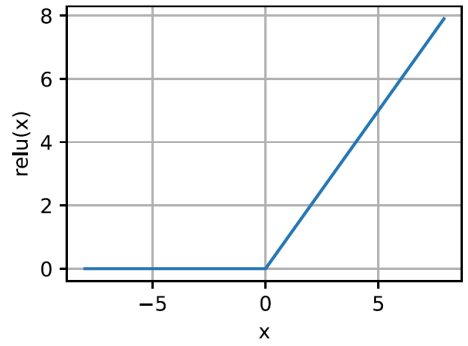
$\Rightarrow$ C'est l'utilisation de fonctions d'activation (non-linéaires) qui conduit à des modèles non-linéaires

MLP et non-linéarité

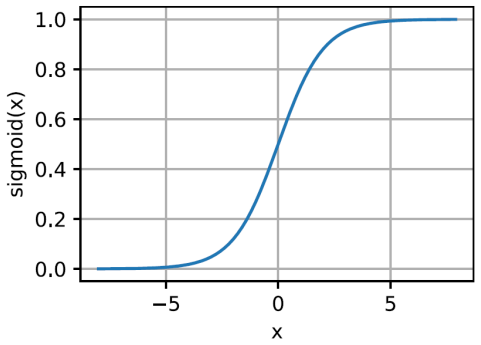
⇒ C'est l'utilisation de fonctions d'activation (non-linéaires) qui conduit à des modèles non-linéaires

Fonctions d'activation usuelles :

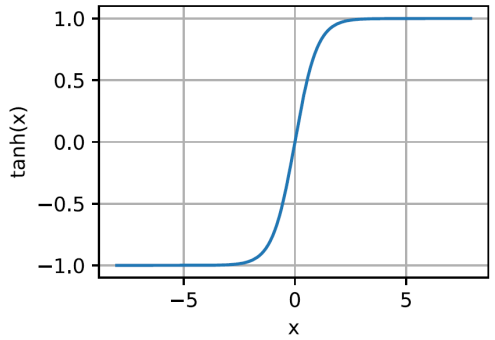
ReLU



sigmoid



tanh



- ReLU (Rectified Linear Unit) : $f(x) = \max(0, x)$
  - on "coupe" les neurones négatifs
- sigmoide / logistique : $f(x) = \frac{1}{1+\exp(-x)}$
  - chaque neurone $\sim$ une régression logistique
- tangente hyperbolique : $f(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}$
  - l'activation peut-être négative

MLP et non-linéarité

Fonction d'activation - remarques :

- ReLU, sigmoide, tanh : activation des couches internes
  - activation de la **couche de sortie** = linéaire, logistique ou softmax en fonction de la problématique
- ReLU : souvent maintenant le choix par défaut
- sigmoide et tanh = "squashing functions"
  - le neurone émet un influx ("tire") ou pas
- plusieurs variantes du ReLU
  - leaky-ReLU, parametrized ReLU, ...
  - problèmes de "vanishing gradient" et de "dead neurons"

Quelle structure pour un MLP ?

Question clé : définir la structure du réseau

1. nombre de couches
2. nombre de neurones dans chaque couche
3. fonction(s) d'activation

2. voir back-up slides.

Quelle structure pour un MLP ?

Question clé : définir la structure du réseau

1. nombre de couches
2. nombre de neurones dans chaque couche
3. fonction(s) d'activation

Théorème d'approximation universelle (simplifié) : un réseau à 1 couche cachée avec suffisamment de neurones peut approximer n'importe quelle fonction [2].

2. voir back-up slides.

Quelle structure pour un MLP ?

Question clé : définir la structure du réseau

1. nombre de couches
2. nombre de neurones dans chaque couche
3. fonction(s) d'activation

Théorème d'approximation universelle (simplifié) : un réseau à 1 couche cachée avec suffisamment de neurones peut approximer n'importe quelle fonction [2].

Pourquoi chercher plus loin ?

- le "suffisamment" peut-être très grand
- on peut être plus efficace avec plusieurs petites couches
  - moins de paramètres, construction incrémentale

⇒ essayer différentes structures et s'inspirer de l'existant !

2. voir back-up slides.

Exemple de mise en oeuvre Keras

```python
# import
from keras.models import Sequential
from keras.layers import Dense
# define parameters
n_classes = 10
p = 100
# build model
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=p))
model.add(Dense(100, activation='relu'))
model.add(Dense(n_classes, activation='softmax'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                6464
_________________________________________________________________
dense_2 (Dense)              (None, 100)               6500
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1010
=================================================================
Total params: 13,974
Trainable params: 13,974
Non-trainable params: 0
_________________________________________________________________
```

- deux couches cachées + couche de sortie = 3 couches
- 64 neurones + 100 neurones + 10
- # paramètres $\sim 14\times$ modèle linéaire.

Bilan intermédiaire...

- MLP= réseaux de neurones à couches "cachées"
- fonction d'activations et modèles non-linéaires
- fonctions d'activation : ReLU, sigmoide, tanh
- Question ouverte : optimiser la structure du réseau
  - attention au sur-apprentissage...
- Mêmes couches `Keras`
  - succession de couches "Dense"

Architectures

- neurones artificiels et modèles linéaires
- perceptrons multi-couches
- réseaux convolutifs

Réseaux de neurones et imagerie

Motivation :

1. réseaux de neurones pour l'imagerie
2. sans passer par l'extraction de descripteurs pré-définis
   - e.g., visual words

$\Rightarrow$ approche générique, descripteurs optimisés pour la tâche en question, pas d'expertise en traitement d'image

Réseaux de neurones et imagerie

Motivation :

1. réseaux de neurones pour l'imagerie
2. sans passer par l'extraction de descripteurs pré-définis
   - e.g., visual words

$\Rightarrow$ approche générique, descripteurs optimisés pour la tâche en question, pas d'expertise en traitement d'image

1ere idée : utiliser directement les pixels en entrée d'un MLP

- image $p \times q \rightarrow$ vecteur de longueur $p \times q$

Réseaux de neurones et imagerie

Motivation :

1. réseaux de neurones pour l'imagerie
2. sans passer par l'extraction de descripteurs pré-définis
   - e.g., visual words

⇒ approche générique, descripteurs optimisés pour la tâche en question, pas d'expertise en traitement d'image

1ere idée : utiliser directement les pixels en entrée d'un MLP

- image $p \times q \rightarrow$ vecteur de longueur $p \times q$

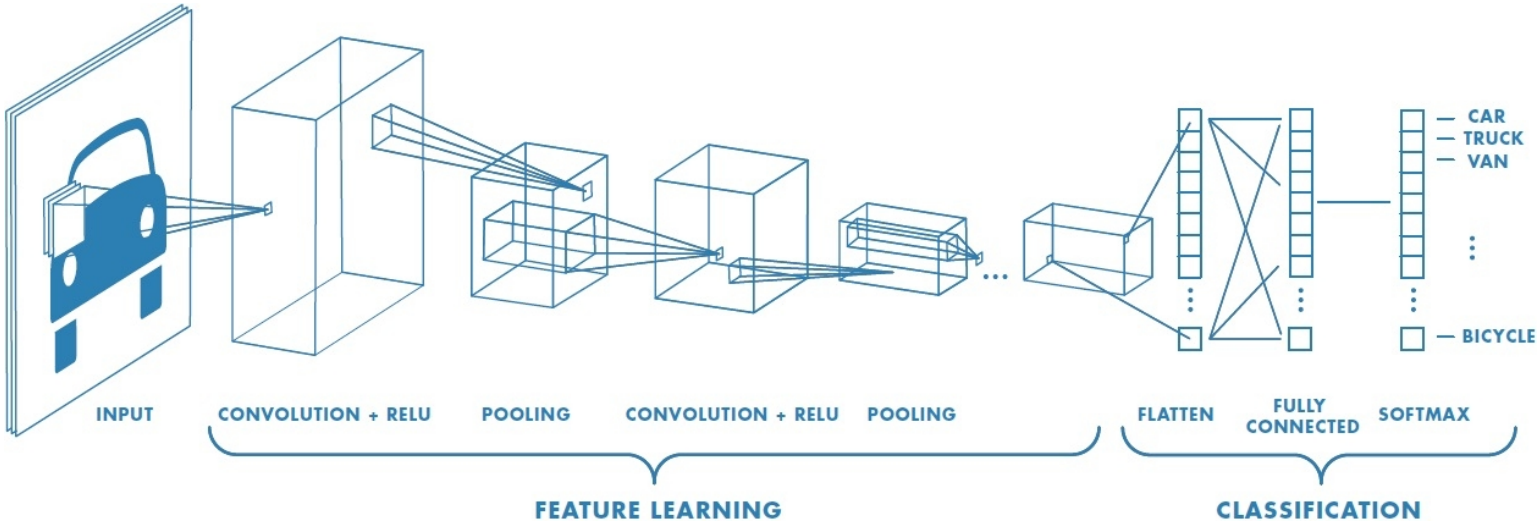
Plusieurs problèmes :

- beaucoup (beaucoup) de paramètres !
- pas de propriétés d'invariance
- comment gérer la couleur

⇒ introduction de structures dédiées : les réseaux convolutifs

Deep learning & apprentissage de la représentation

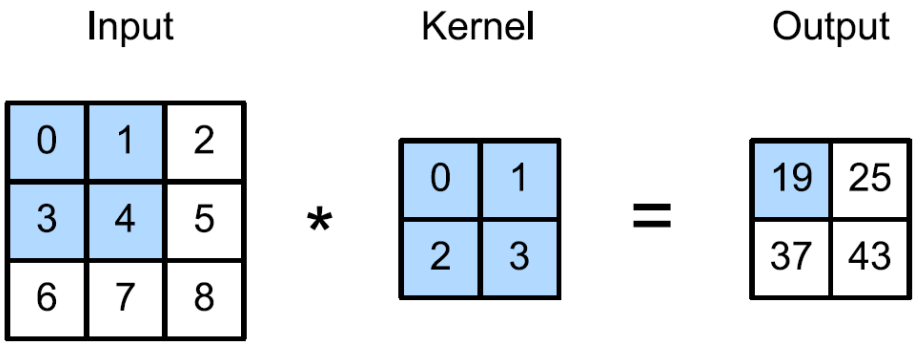
Réseaux convolutifs :



- descripteurs non définis a priori : appris sur les données
  - apprentissage de la représentation
  - approche "end to end"
- basés sur des couches de convolution et de "pooling"
- classification finale basé sur un MLP

Couches de convolutions

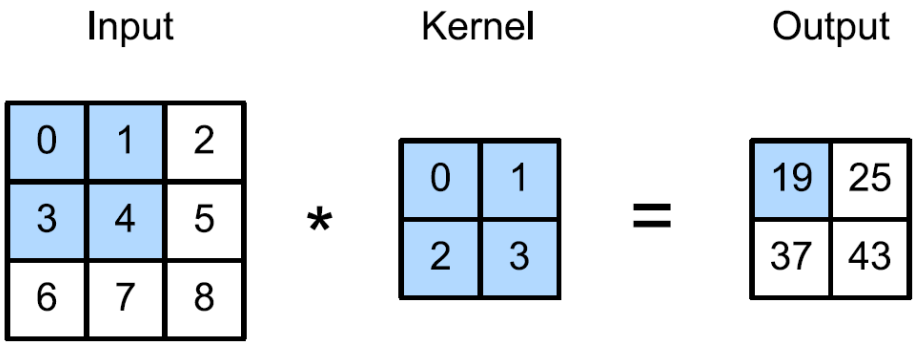
Filtre de convolution (image 1D) :



- combinaison linéaire des pixels d'une région
- on "balaye" le filtre sur l'image : **mêmes paramètres**
- apprentissage = apprendre les poids du filtre

Couches de convolutions

Filtre de convolution (image 1D) :



- combinaison linéaire des pixels d'une région
- on "balaye" le filtre sur l'image : **mêmes paramètres**
- apprentissage = apprendre les poids du filtre

1 couche de convolution = un ensemble de filtres

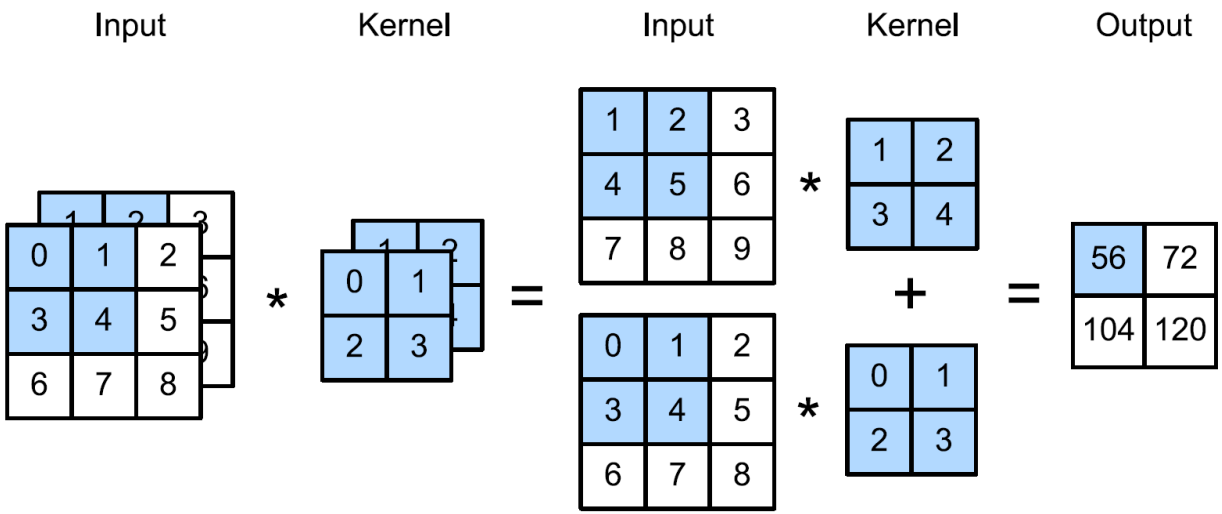
- d'une taille donnée

$\Rightarrow$ détectent des caractéristiques dans les images

- contours, textures,...

Couches de convolutions

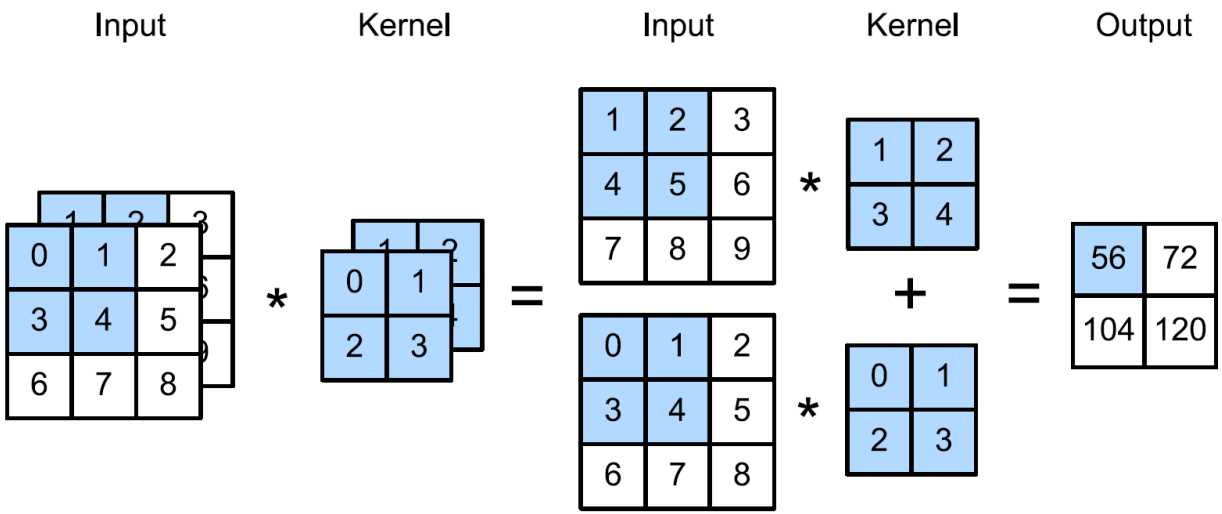
Filtre de convolution si plusieurs canaux :



- combine l'information des différents canaux
- nombre de paramètres = taille du filtre $\times$ # canaux

Couches de convolutions

Filtre de convolution si plusieurs canaux :



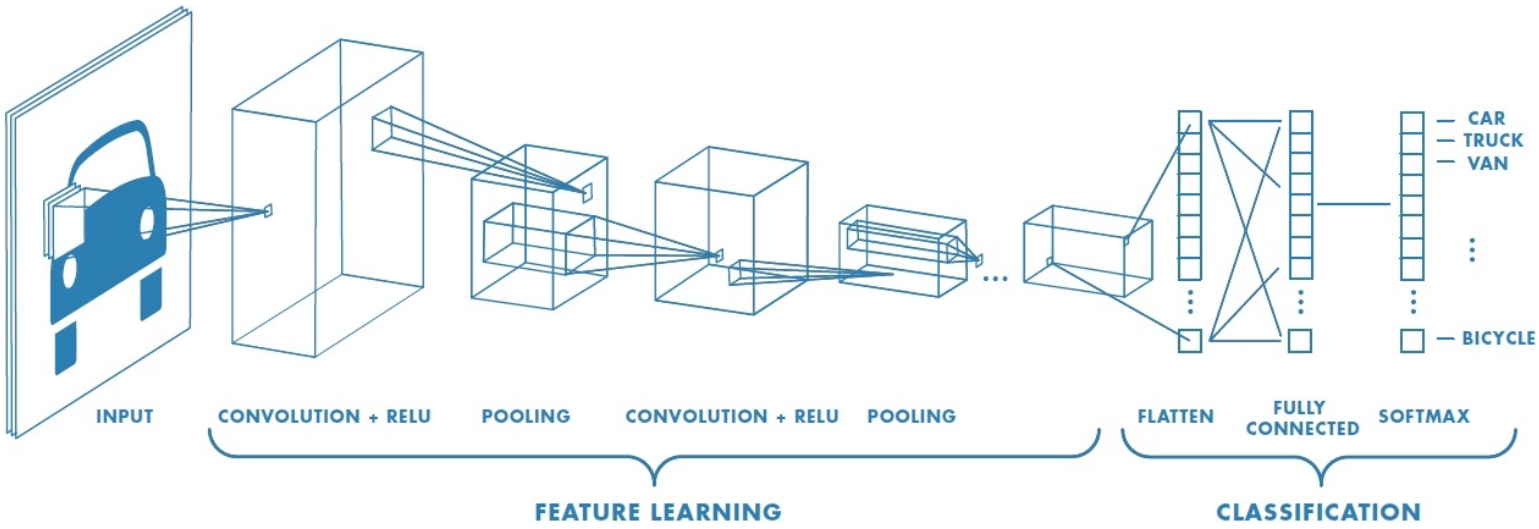
- combine l'information des différents canaux
- nombre de paramètres = taille du filtre $\times$ # canaux

$\Rightarrow$ Toujours le cas en pratique (ou presque) !

- si entrée = image couleur
- si entrée = sortie de la couche de convolution précédente

Couches de convolutions

Filtre de convolution si plusieurs canaux :

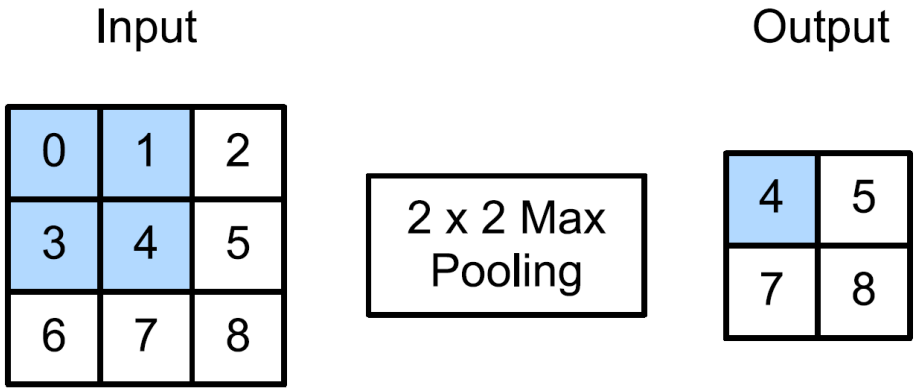


$\Rightarrow$ bien noter la profondeur des masques :

- 1ère couche : 1 ou 3 (niveaux de gris ou RGB)
- couches suivantes : # filtres de la couche précédente

Couches de "pooling"

Pooling = réduire la dimension spatiale



Stratégies principales : max-pooling et average-pooling

- max-pooling recommandé en général

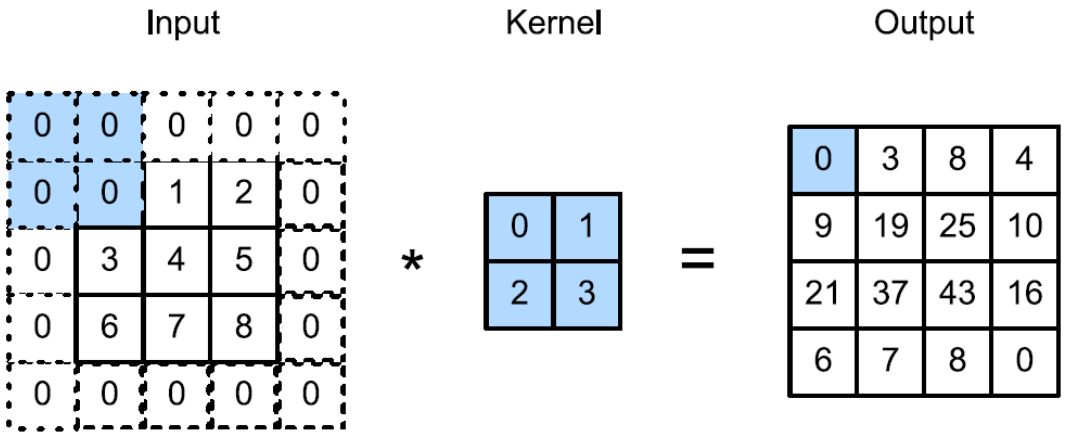
⇒ s'intercalent entre les couches de convolution

- pas forcément systématiquement

⇒ induisent des propriétés d'invariance par translation

En pratique : padding & stride

En pratique : on utilise du padding

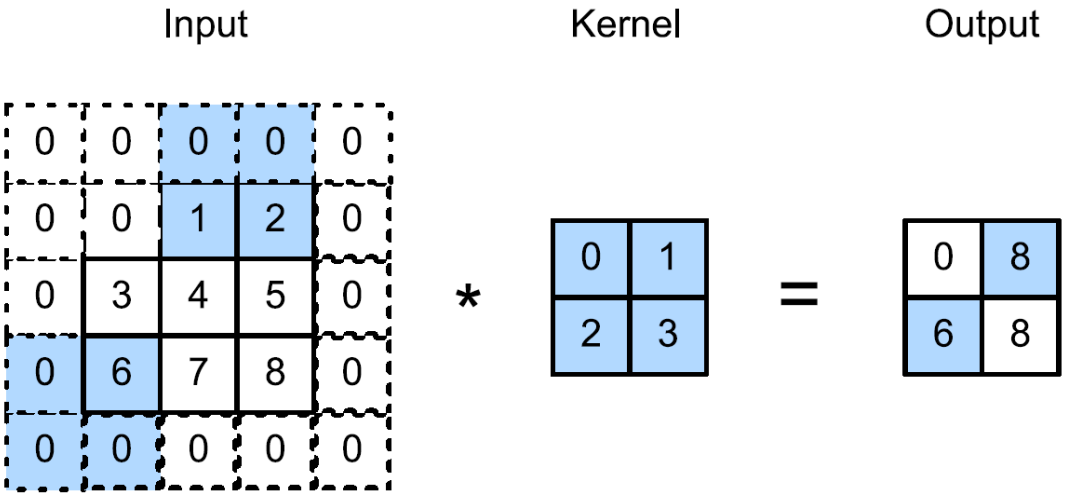


$\Rightarrow$ permet d'éviter de "rogner" les bords de l'image

- marginal sur une couche si les filtres sont petits...
- ...mais s'accumule au fil des couches

En pratique : padding & stride

En pratique : on peut modifier le stride

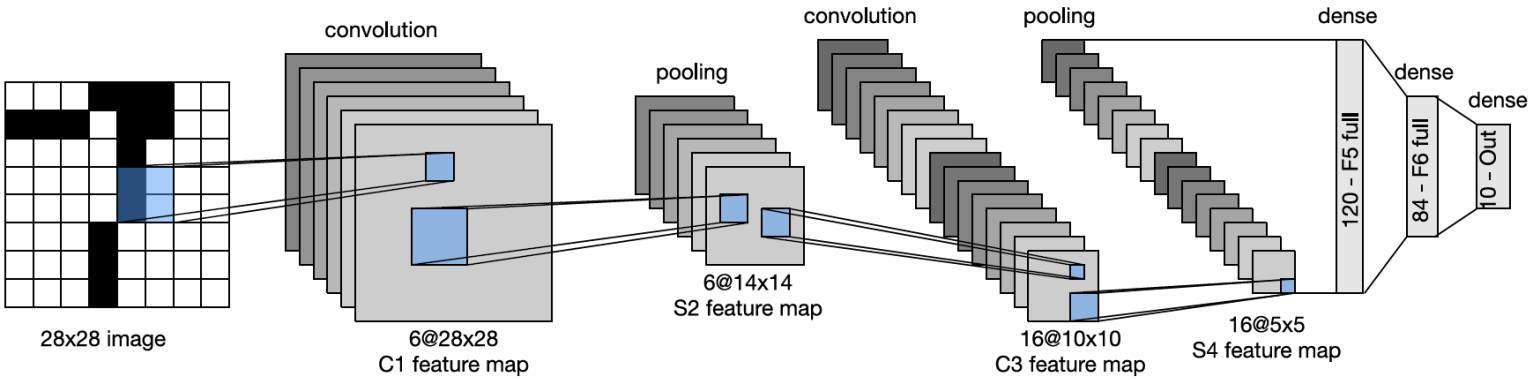


- si stride = 1, on déplace le filtre pixel par pixel
- ici, stride horizontal = 2 et stride vertical = 3

⇒ permet de réduire la résolution (même à l'étape de convolution)

⇒ peut s'appliquer à convolution ou max-pooling

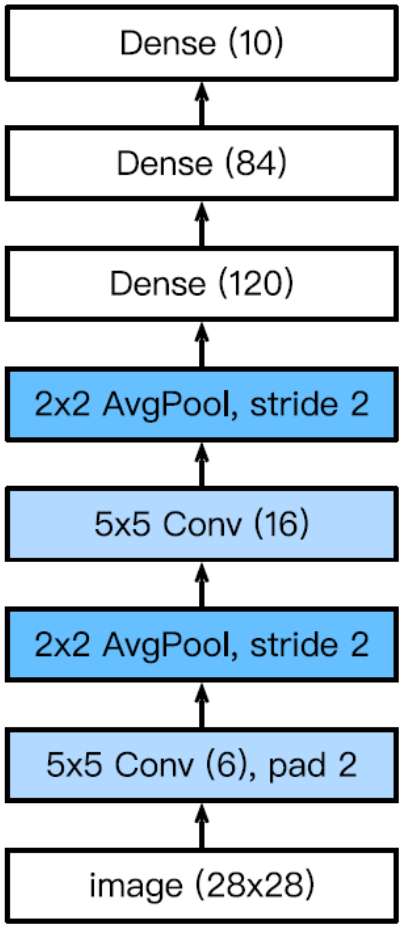
L'exemple fondateur : le réseau LeNet



- développé dans les 1990's pour les "digits"
  - LeCun, Bottou, Bengio and Haffner.
- deux "blocs" de convolution + pooling
  - conv1 : 6 filtres de taille 5×5 ($\times 1$)
  - conv2 : 16 filtres de taille 5×5 ($\times 6$)
  - pooling : taille 2×2 avec stride de 2 $\Rightarrow$ taille / 4
  - activation sigmoide + average-pooling
- "flattening" + MLP à 2 couches cachées
  - (120 et 84 neurones)

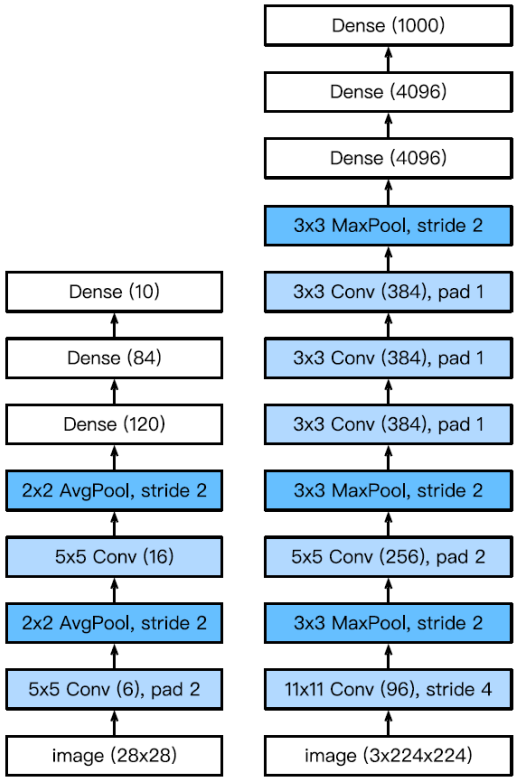
L'exemple fondateur : le réseau LeNet

Représentation "compressée :



Architectures modernes

LeNet vs AlexNet - vainqueur ImageNet 2012 :

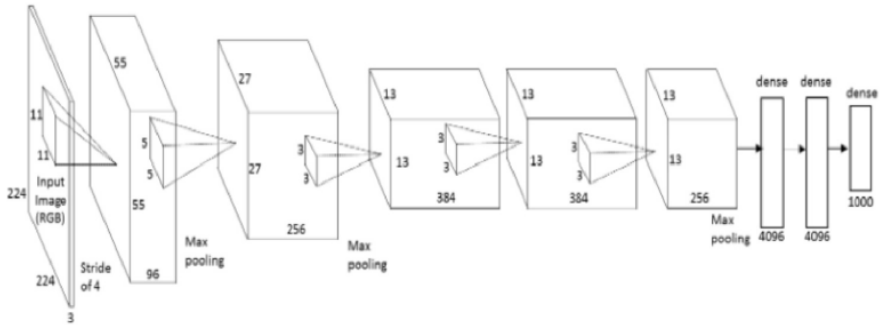


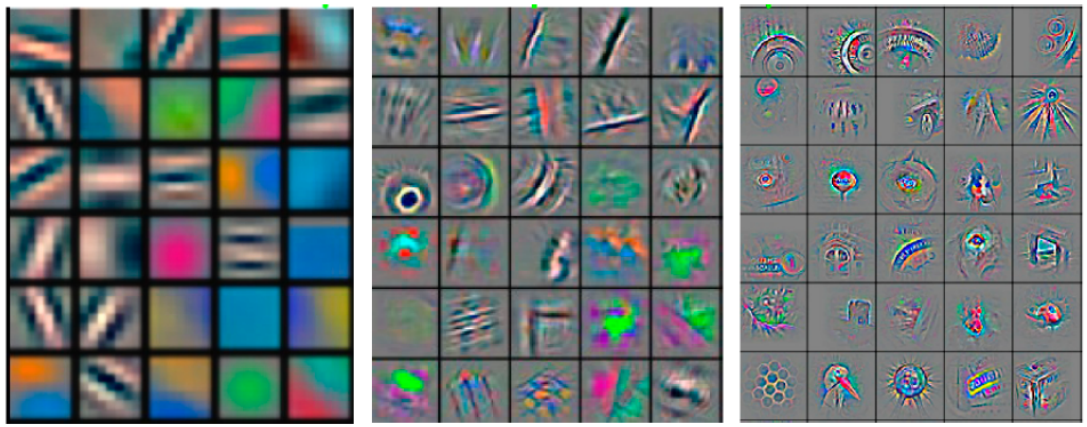
- architectures très similaires !
- {sigmoid + Avg-Pooling} $\rightarrow$ {ReLU + Max-Pooling}

Architectures modernes

AlexNet : visualisation des "features" appris par le modèle [3]

- low-, mid- and high-level features





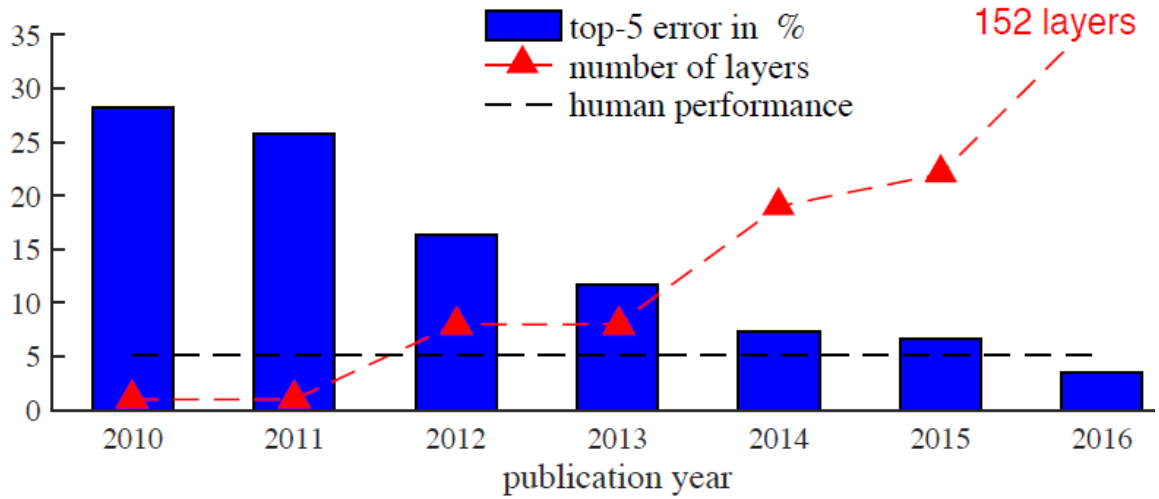
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Y. LeCun StatLearn tutorial

3. https://github.com/StephaneCanu/Deep_learning_lecture

Architectures modernes

Depuis AlexNet [4] :



2012 Alex Net

2013 ZFNet

2014 VGG

2015 GoogLeNet / Inception

2016 Residual Network

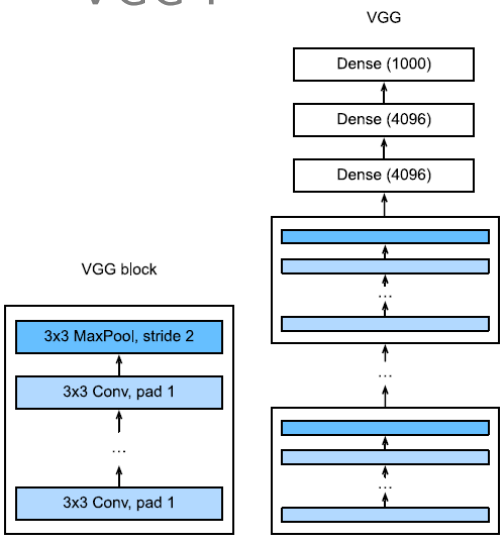
⇒ explosion du nombre de couches !

4. https://github.com/StephaneCanu/Deep_learning_lecture

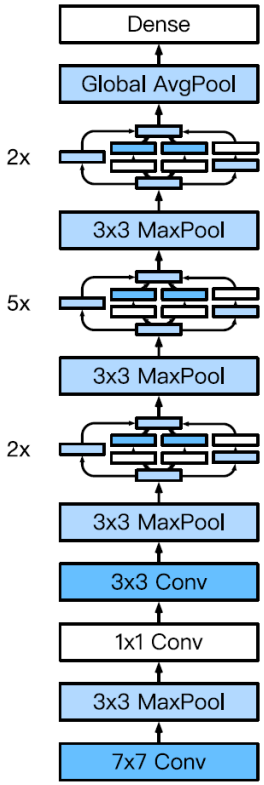
CNN - Architectures modernes

Depuis AlexNet :

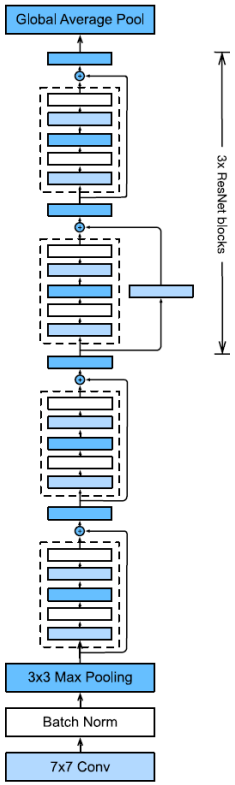
VGG :



GoogLeNet :



ResNet :



⇒ voir "Dive into Deep Learning" (chap. 9) pour une très bonne présentation des différentes architectures.

Exemple de mise en oeuvre Keras

Réseau LeNet original :

```
# import
from keras.models import Sequential
from keras.layers import Conv2D, AveragePooling2D, Flatten, Dense
# build model
model = Sequential()
model.add(Conv2D(6, (5,5), padding = 'same', activation='sigmoid', input_shape=(28,28,1)))
model.add(AveragePooling2D(pool_size=(2,2)))
model.add(Conv2D(16, (5,5), activation='sigmoid'))
model.add(AveragePooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(120, activation='sigmoid'))
model.add(Dense(84, activation='sigmoid'))
model.add(Dense(10, activation='softmax'))
# show summary
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_1 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_2 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_1 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_2 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_3 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

- couches "Conv2D", "AveragePooling" et "Flatten"

Exemple de mise en oeuvre Keras

Réseau LeNet "modernisé" :

```
# import
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
# build model
model = Sequential()
model.add(Conv2D(6, (5,5), padding = 'same', activation='relu', input_shape=(28,28,1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(16, (5,5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(120, activation='relu'))
model.add(Dense(84, activation='relu'))
model.add(Dense(10, activation='softmax'))
# show summary
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_1 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_2 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_3 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

- {sigmoid + Avg-Pooling} $\rightarrow$ {ReLU + Max-Pooling}

Bilan intermédiaire...

- CNN et "apprentissage de la représentation"
  - Couches de convolution et pooling pour représentation
  - MLP pour classification
- Exemple fondateur : LeNet (1990)
- Modernisé par :
  - ReLU + MaxPooling : AlexNet
  - Profondeur : VGG, GoogLeNet, ResNet, ...
- Quelques lignes de code dans `Keras`
  - pour LeNet / AlexNet (API "séquentielle")
  - un peu plus complexe pour GoogLeNet, ResNet, etc.. (passage à l'API "fonctionnelle")

Apprentissage

- Descente de gradient & rétro-propagation
- Régularisation
- Augmentation de données
- Transfer learning
- En pratique

⇒ **A suivre la semaine prochaine....**

Références

Back-up

Théorème d'approximation universelle

Tiré d'un cours de Stéphane Canu [5] :

MLP with one hidden layer as universal approximator

Universal approximation theorem for MLP

- given any $\varepsilon > 0$
- for any continuous function f on compact subsets of $\mathbb{R}^p$
- for any admissible activation function σ (not a polynomial)
- there exists h, $W_1 \in \mathbb{R}^{p \times h}$, $b \in \mathbb{R}^h$, $c \in \mathbb{R}$ and $w_2 \in \mathbb{R}^h$ such that

$$\|f(x) - w_2 \sigma(W_1 x + b) + c\|_\infty \leq \varepsilon$$

SVM, Boosting and Random Forest also

Approximation theory of the MLP model in neural networks, A Pinkus - Acta Numerica, 1999
The power of depth for feedforward neural networks, R. Eldan and O. Shamir, 2015.

5. https://github.com/StephaneCanu/Deep_learning_lecture

CNN - Architectures modernes

Depuis AlexNet [6] :

2012 Teams	%error
Supervision (Toronto)	15.3
ISI (Tokyo)	26.1
VGG (Oxford)	26.9
XRCE/INRIA	27.0
UvA (Amsterdam)	29.6
INRIA/LEAR	33.4

2013 Teams	%error
Clarifai (NYU spinoff)	11.7
NUS (singapore)	12.9
Zeiler-Fergus (NYU)	13.5
A. Howard	13.5
OverFeat (NYU)	14.1
UvA (Amsterdam)	14.2
Adobe	15.2
VGG (Oxford)	15.2
VGG (Oxford)	23.0

2014 Teams	%error
GoogLeNet	6.6
VGG (Oxford)	7.3
MSRA	8.0
A. Howard	8.1
DeeperVision	9.5
NUS-BST	9.7
TTIC-ECP	10.2
XYZ	11.2
UvA	12.1

shallow approaches

deep learning

Y. LeCun StatLearn tutorial

⇒ forte adhésion de la communauté "computer vision" !

6. https://github.com/StephaneCanu/Deep_learning_lecture