

Réseaux de neurones et deep learning : Utilisation et méthodologie

GEOFFREY DANIEL

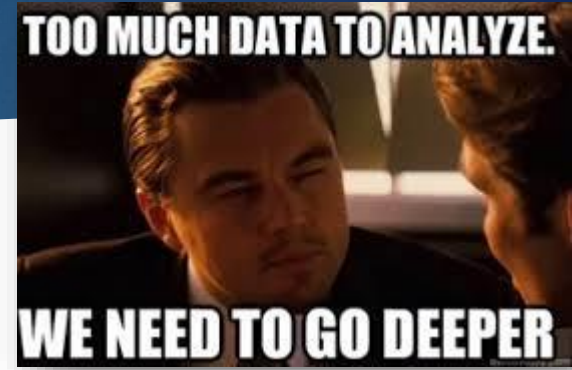
CEA/DES/ISAS/DM2S/STMF/LGLS

GEOFFREY.DANIEL@CEA.FR

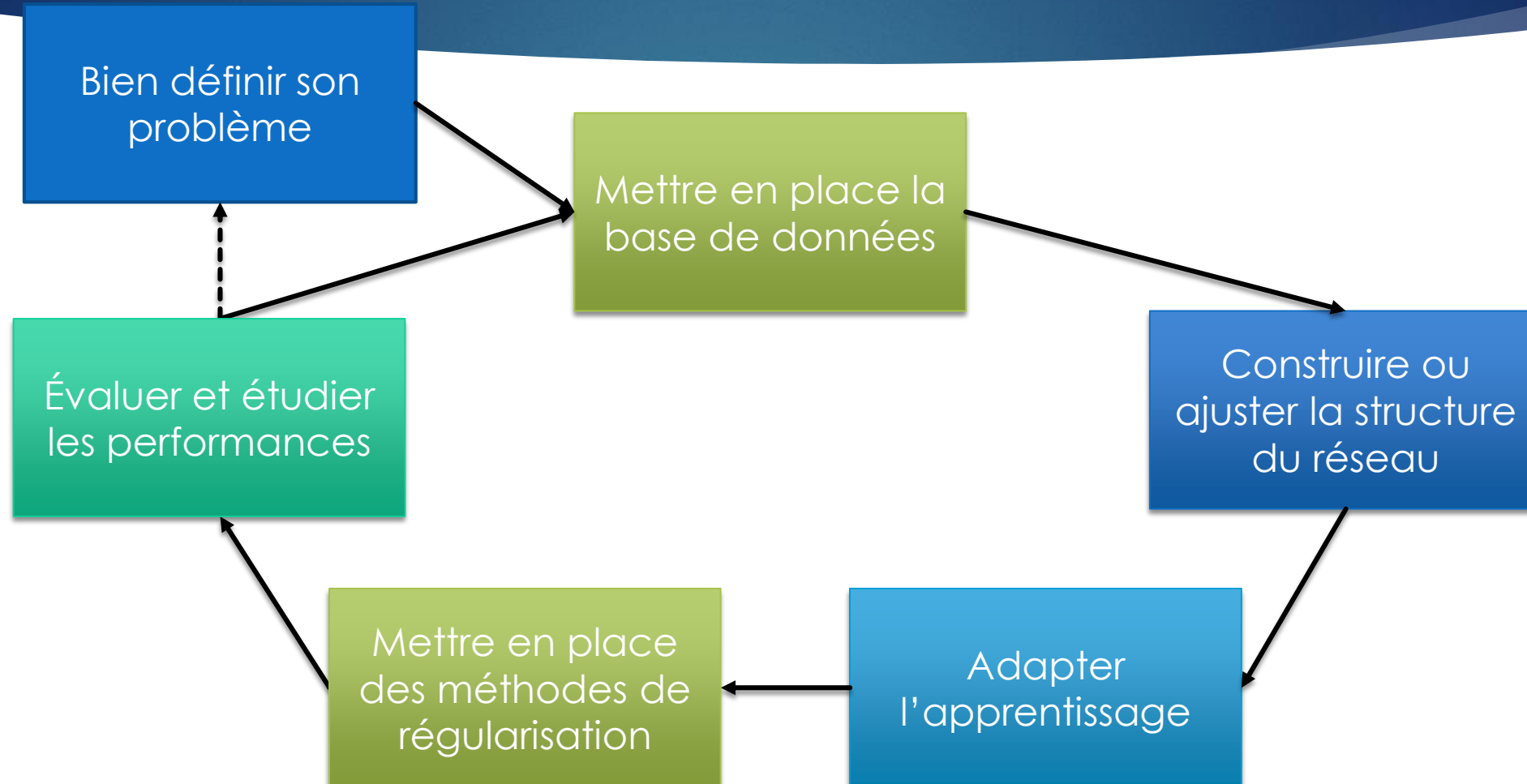


Réseaux de neurones et deep learning

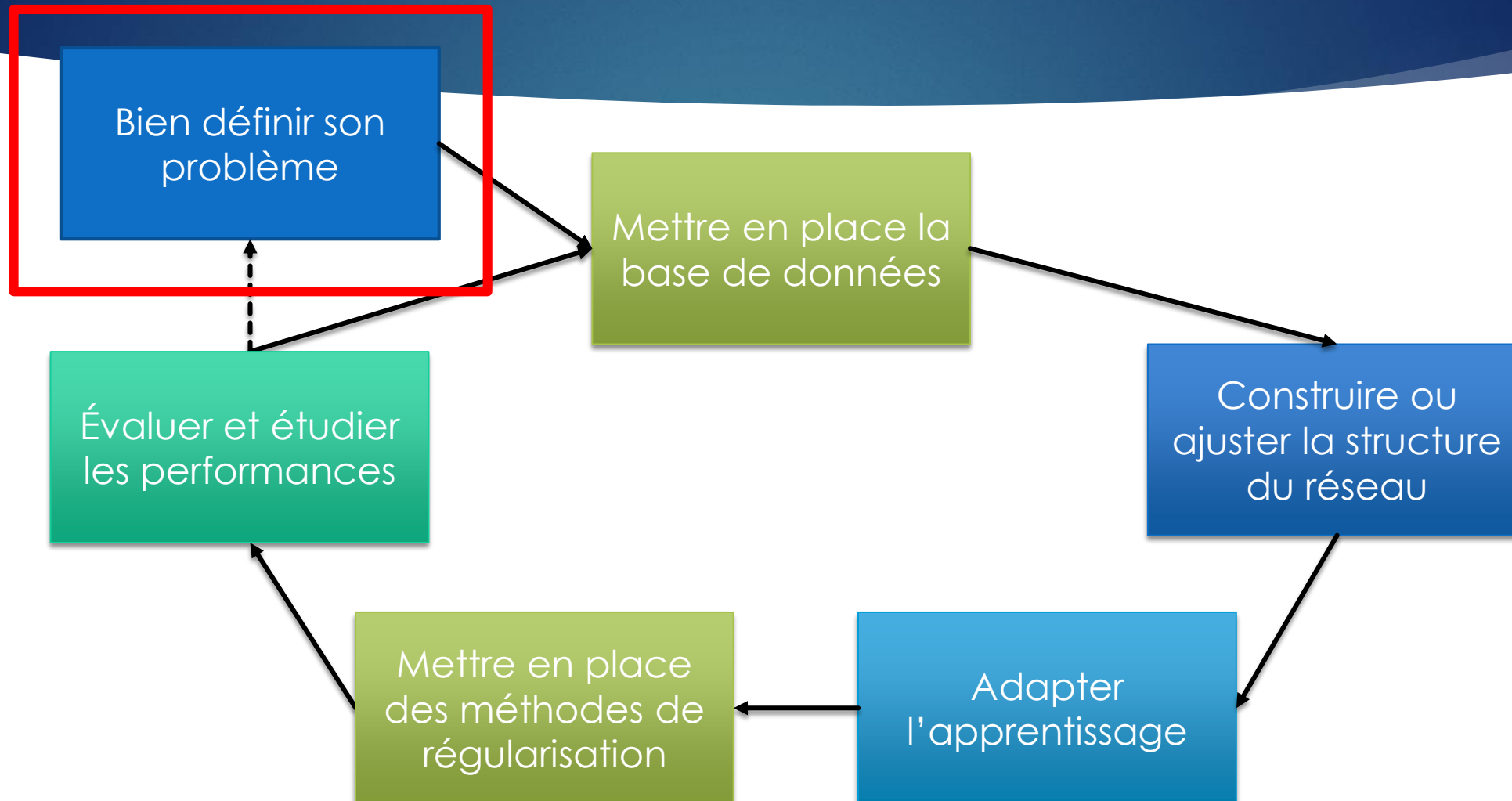
- ▶ Partie 1 : Introduction aux **réseaux de neurones**
 - ▶ Utilisations courantes du deep learning
 - ▶ Bases générales
- ▶ Partie 2 : **Architecture** des réseaux, **hyperparamètres** et évaluation des **performances**
 - ▶ Comment construire mon réseau et adapter la phase d'apprentissage ?
 - ▶ Comment évaluer les performances de mon réseau de neurones ?
- ▶ **Partie 3 : Construction de la base de données**
 - ▶ **Éléments méthodologiques sur la mise en place du problème à résoudre potentiellement par deep learning**
 - ▶ **Comment utiliser l'évaluation des performances pour améliorer la base de données et le réseau ?**
- ▶ Partie 4 : Réseaux de neurones convolutifs
 - ▶ Introduction à des structures plus avancées



Projet de deep learning

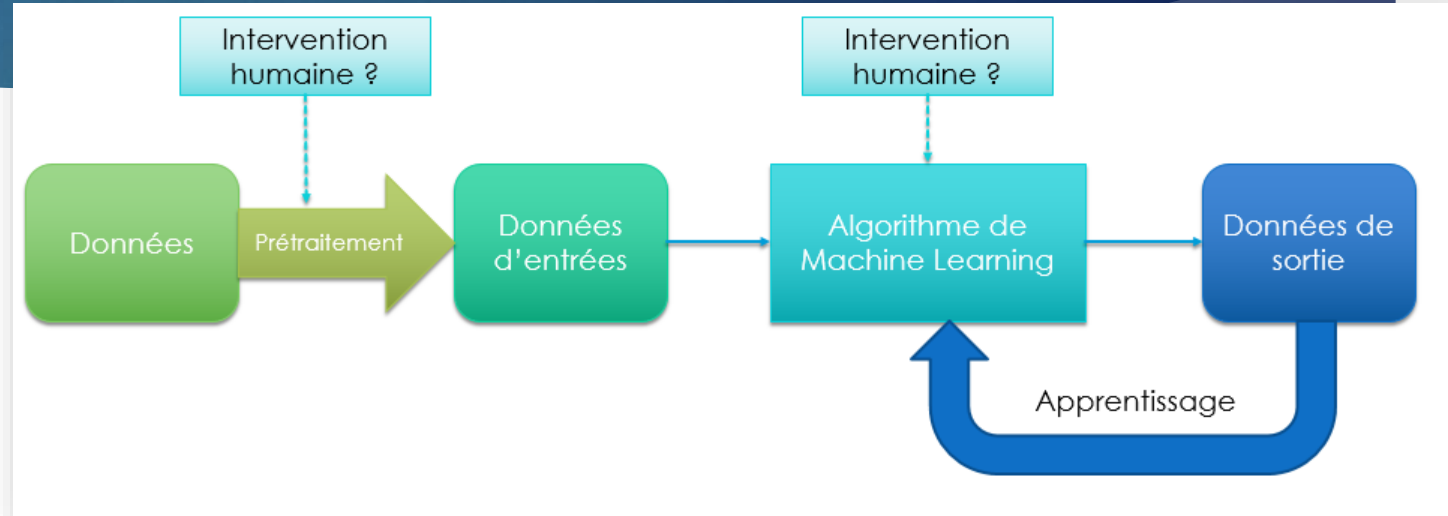


Projet de deep learning



Bien définir le problème

- Pour apprendre, il faut des données : données d'entrée et de sortie !



- De quelles données je dispose ?
- Réfléchir à formuler le problème de la manière suivante (apprentissage supervisé) :

Données d'entrée ?

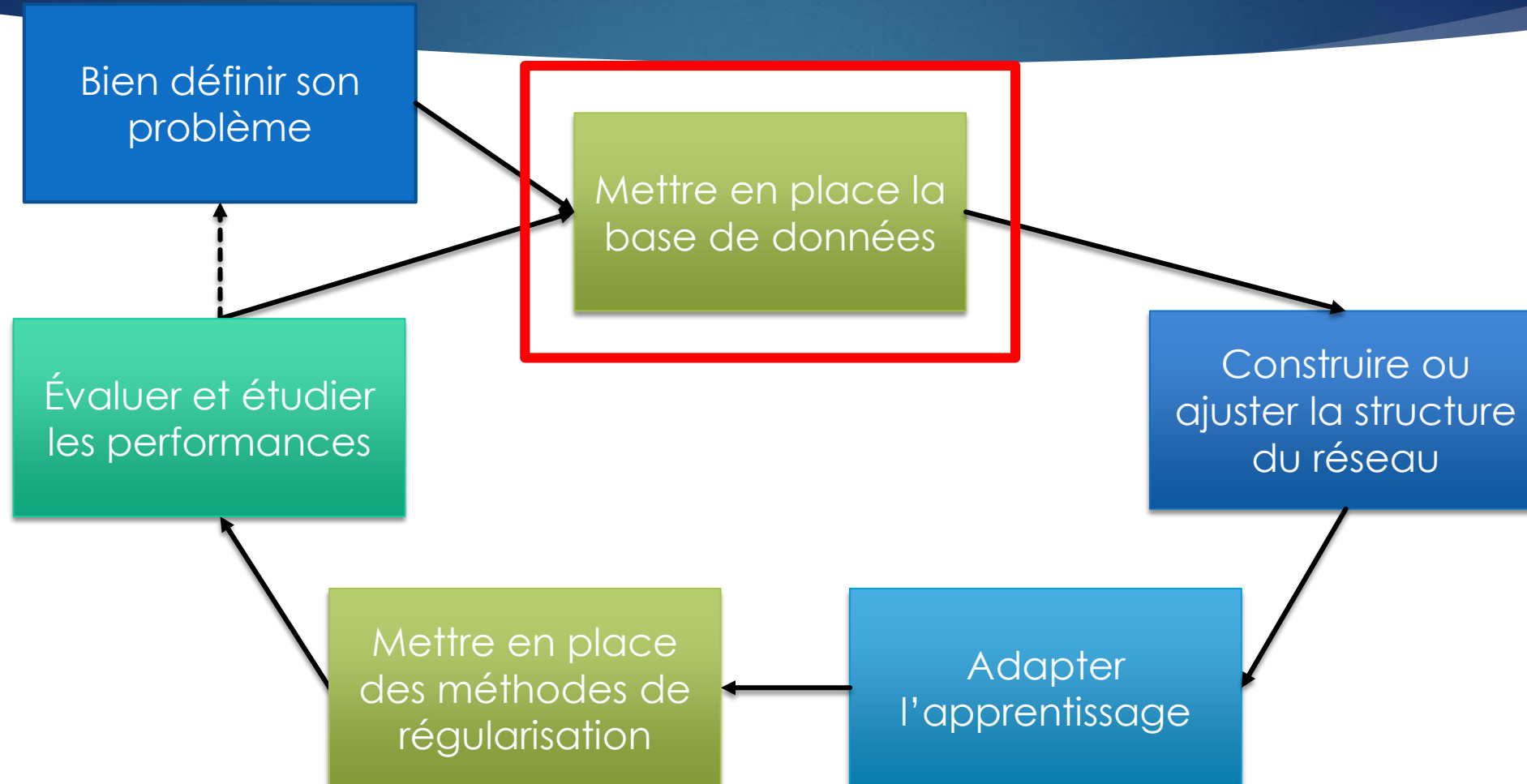
Quelles informations vais-je utiliser pour faire des « prédictions » ?

Données de sortie ?

Que veut-on prédire ?

- Des classes ?
- Des paramètres ?
- Une image ?

Projet de deep learning



La base de données : quel format ?

Données d'entrée

- ▶ Comment représenter les données d'entrée ? → Définit le type des premières couches à utiliser
 - ▶ Ensemble de paramètres : masse d'un objet, type, taille, couleur... sous forme de nombres ou éventuellement de classes → Perceptron (voir parties 1 et 2)
 - ▶ Image, spectre (données localement corrélées) → Réseaux convolutifs (voir partie 4)
 - ▶ Séquences (texte, enregistrement audio/vidéo) → Réseaux récurrents (voir partie 4)

La base de données : quel format ?

Données d'entrée

- ▶ Normaliser les données !
 - ▶ Ne pas avoir des nombres trop grands : calculs plus difficiles à gérer
 - ▶ Faire attention si on a un ensemble de paramètres (masse, taille...) : normaliser indépendamment, il ne faut pas qu'un type de paramètre « domine »
 - ▶ Première possibilité : diviser par un nombre fixe. Exemple : image (intensité des pixels entre 0 et 255) → diviser par 255
 - ▶ Deuxième possibilité : normaliser en moyenne et variance sur la base de données
 - ▶ Calculer $\mu_{\text{train}}, \sigma_{\text{train}}$ sur la base de données d'apprentissage (sur chaque coordonnée des vecteurs d'entrée)
 - ▶ $X := \frac{X - \mu_{\text{train}}}{\sigma_{\text{train}}}$ → sur la base de donnée d'apprentissage, ainsi que sur la base de test
 - ▶ Normalisations plus « exotiques » : en échelle logarithmique...

La base de données : quel format ?

Données de sortie

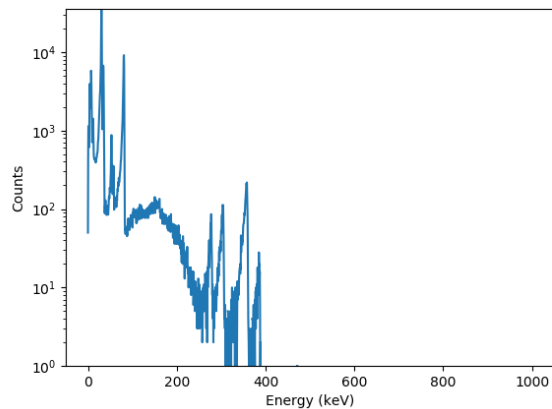
- ▶ Comment représenter les données de sortie ? → Définit la couche de sortie (nombre de neurones et fonction d'activation), la fonction de coût et le type des dernières couches
 - ▶ Classification : définir des classes. Exemple : classer des chiffres manuscrits de 1 à 5, chiffre 4 : (0,0,0,1,0), chiffre 2 : (0,1,0,0,0) → Un neurone par classe, fonction d'activation sigmoïde (non-exclusif), softmax (exclusif). Définir un seuil d'acceptation de la classe. Fonction de coût : binary cross-entropy, distance euclidienne...
 - ▶ Faire attention à ce que toutes les classes pertinentes soient représentées
 - ▶ Il faut éviter qu'une classe soit sur-représentée
 - ▶ Régression → Un neurone par paramètre de sortie, fonction d'activation dépend des paramètres, fonction de coût : distance euclidienne, distance en norme 1
 - ▶ Si plusieurs paramètres → normaliser chaque paramètre indépendamment (éviter qu'un paramètre ne domine pour la fonction de coût)
 - ▶ Image, spectre → Nombre de neurones adapté au format. Fonction d'activation adaptée aux données. Fonction de coût : distance euclidienne, distance en norme 1. Dernières couches éventuellement convolution.
 - ▶ Normaliser les données

La base de données : si je n'ai pas beaucoup de données ?

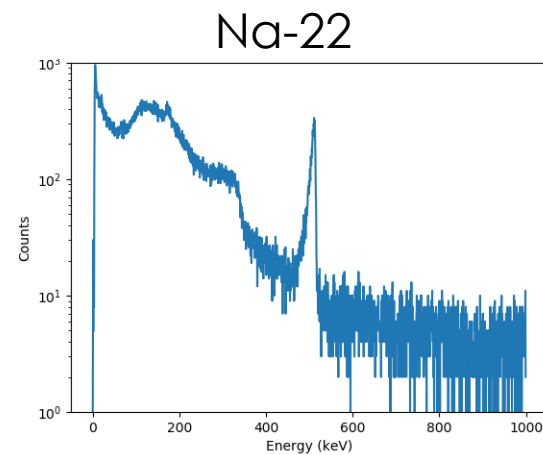
- ▶ Possibilité d'utiliser des données simulées. Attention : test à faire sur des données réelles !
 - ▶ Utiliser la régularisation pour éviter l'overfitting : le réseau de neurones ne doit pas repérer des « biais » dus à la simulation
- ▶ Augmentation de données (data augmentation)
 - ▶ Bruitage volontaire des données (couches « Noise » dans Keras, on peut aussi bruite les couches intermédiaires)
 - ▶ Image : appliquer des rotations, floutage, réflexions...
 - ▶ Superposition linéaire

La base de données : si je n'ai pas beaucoup de données ?

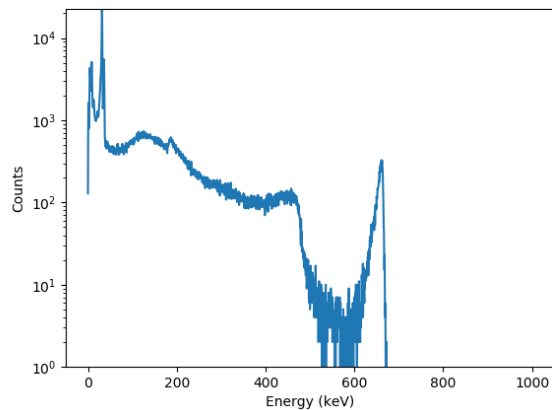
Exemple d'augmentation de données : superposition linéaire



Ba-133

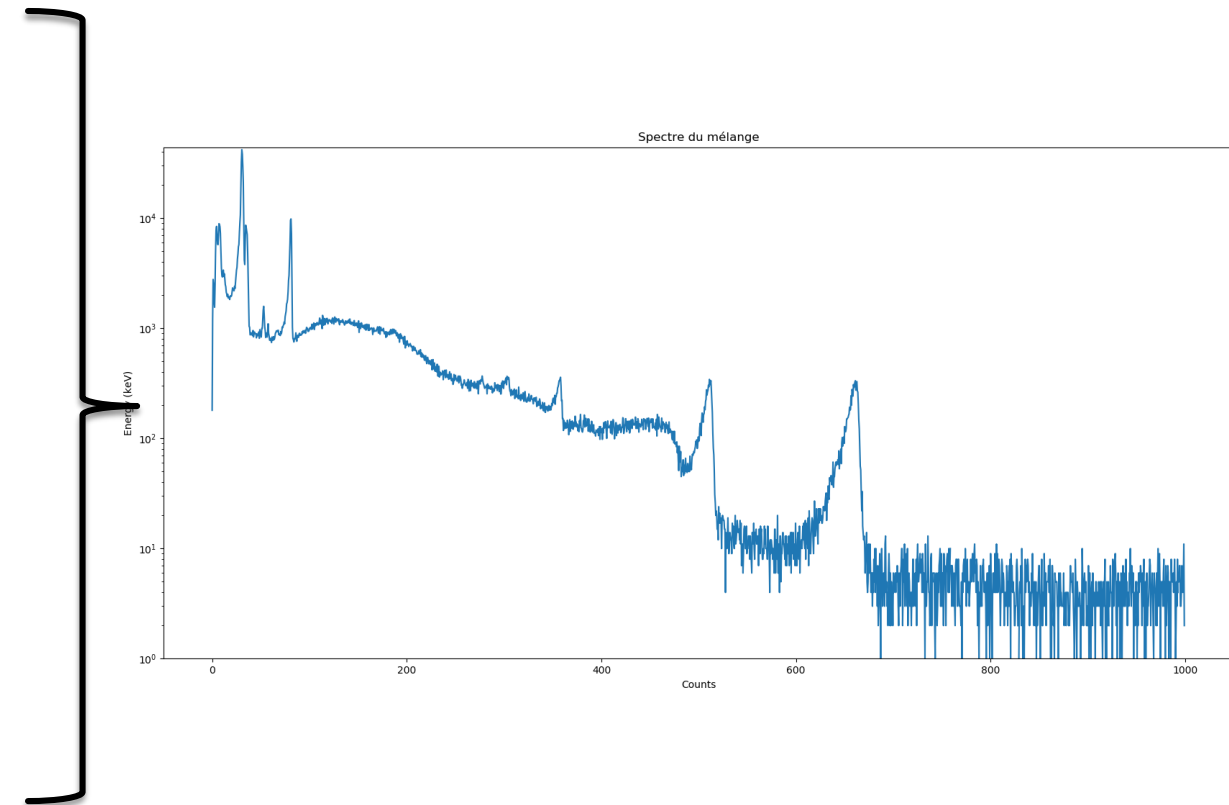


Na-22

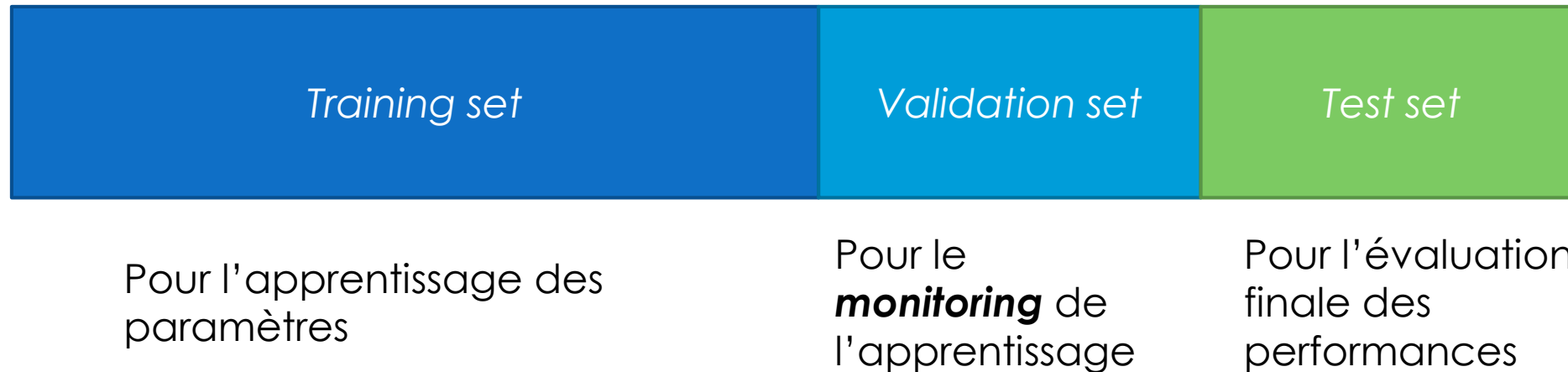


Cs-137

.GLS

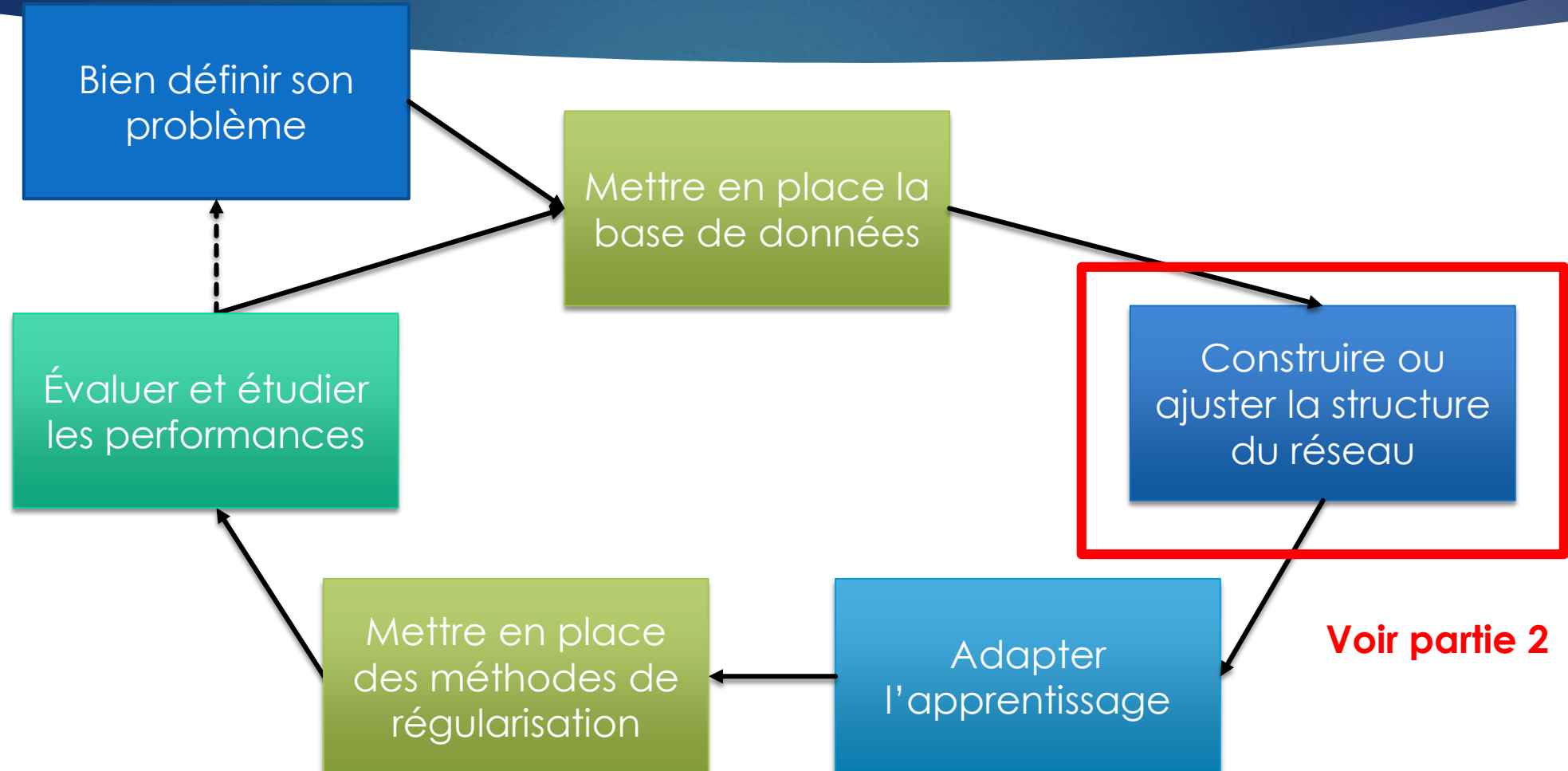


La base de données : le découpage



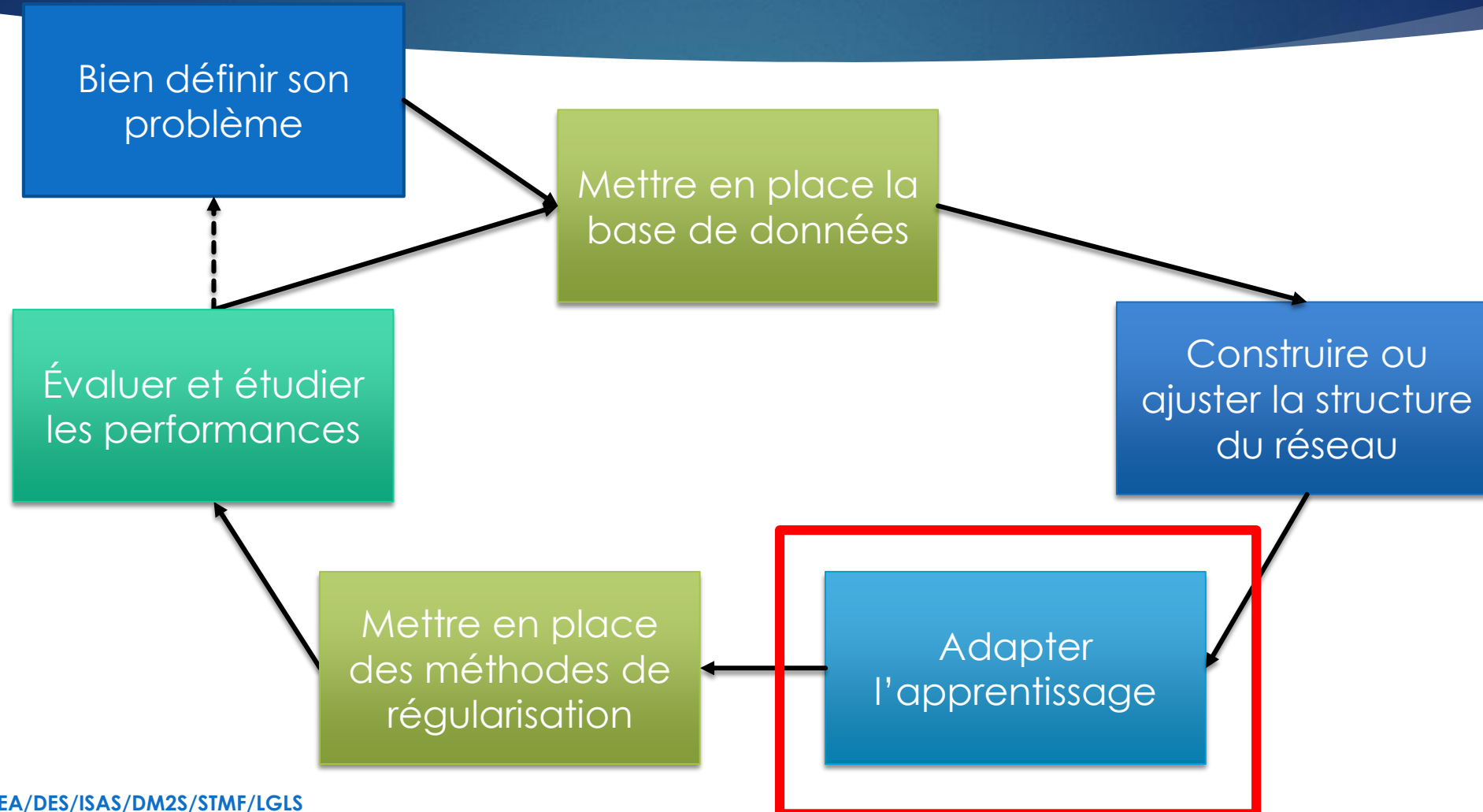
- ▶ *Validation set* : utilisé pour diagnostiquer les performances du réseau de neurones puis adapter le réseau. Le réseau n'apprend pas dessus !
- ▶ *Test set* : utilisé comme évaluation finale, quand on est déjà satisfait du réseau avec le validation set. Le *test set* doit être représentatif des données sur lesquelles le réseau va être appliqué !
- ▶ Pour les proportions :
 - ▶ Avant le *big data* : ordre de grandeur 60%/20%/20%. Avec le *big data* : 98%/1%1% (1% de 1 million = 10 000)

Projet de deep learning



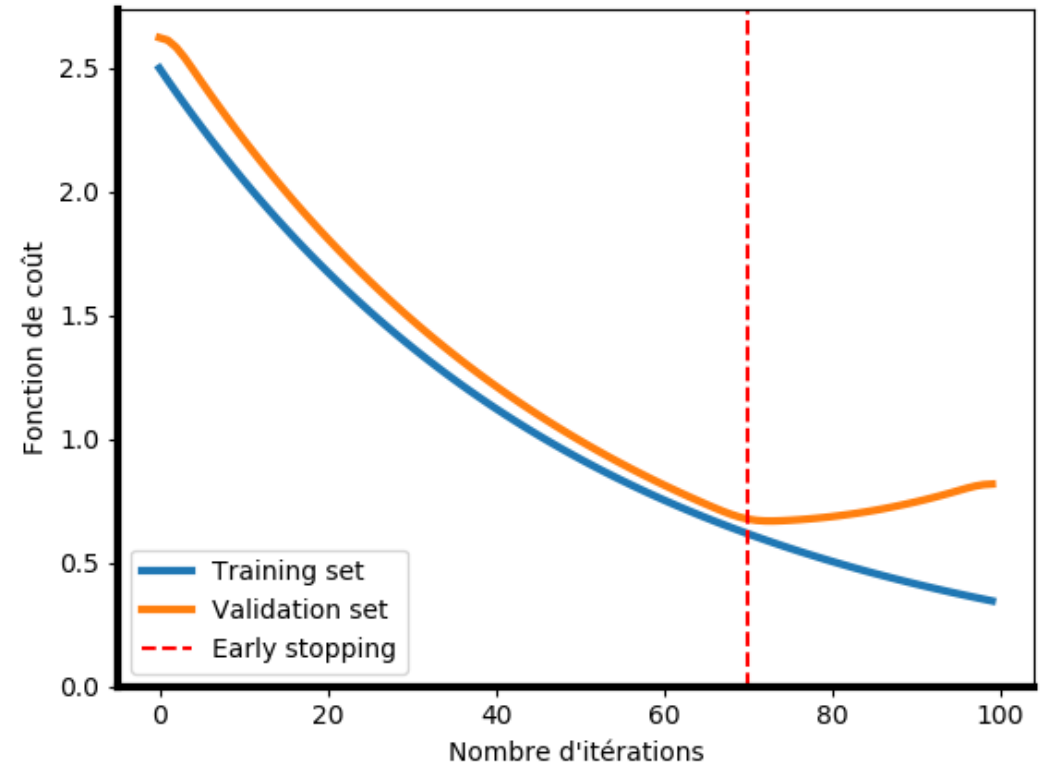
Voir partie 2

Projet de deep learning

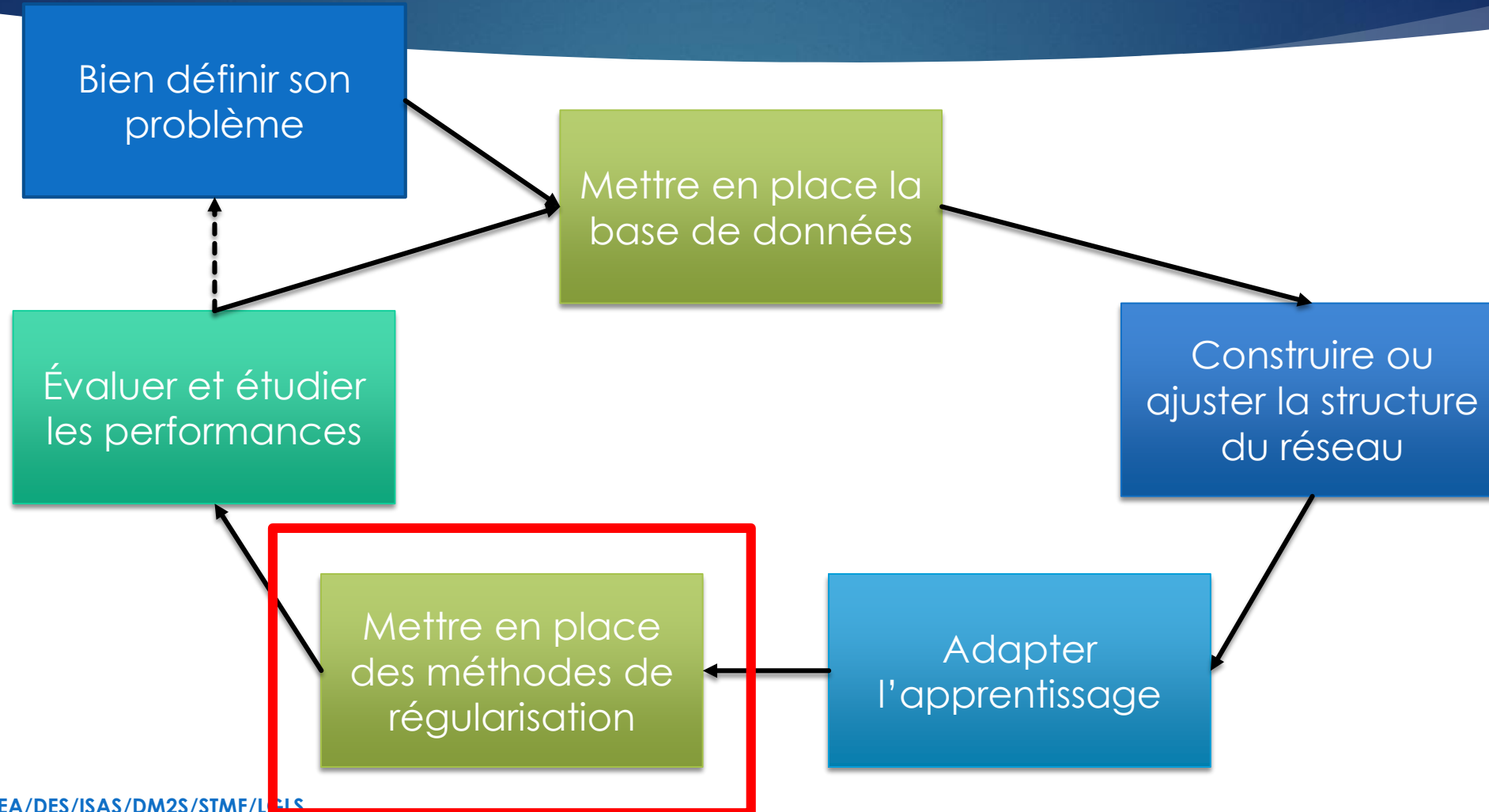


Adapter l'apprentissage : *early stopping*

- ▶ Pour arrêter l'apprentissage, deux possibilités :
 - ▶ Fixer un nombre d'itérations au début
 - ▶ *Early stopping* : arrêter lorsqu'on n'améliore plus la fonction de coût (ou l'*accuracy*) sur le jeu de validation
 - ▶ On regarde l'évolution de la métrique considérée sur les n dernières itérations (n à fixer soi-même)
 - ▶ Si la métrique n'est pas améliorée, on arrête l'apprentissage
 - ▶ Callbacks → `EarlyStopping` sous Keras

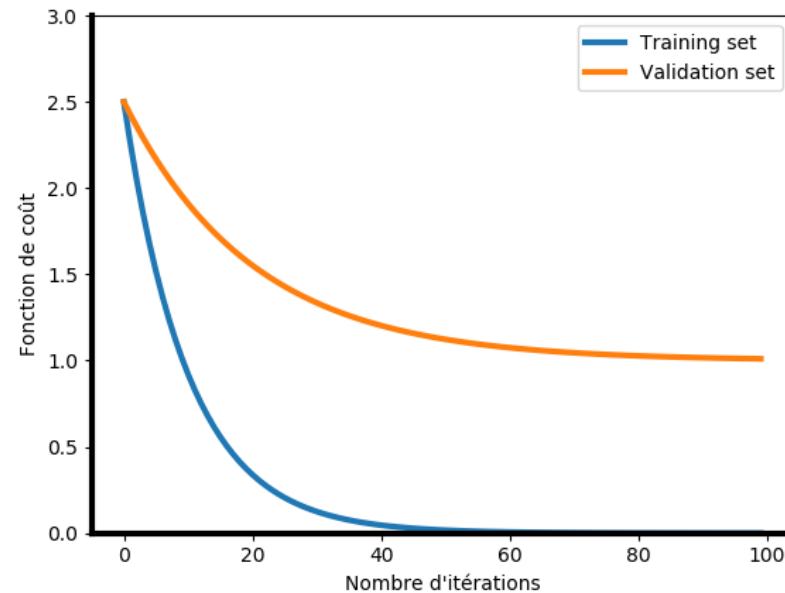


Projet de deep learning



Régularisation : intérêt ?

- ▶ Généraliser au mieux le réseau de neurones
- ▶ Éviter l'*overfitting* : on apprend par cœur sur les données d'apprentissage
 - ▶ Diagnostic avec le monitoring sur le *validation set*



Weight decay

- ▶ Idée : éviter la surinterprétation des données
- ▶ Chaque coordonnée d'un poids W va donner de l'importance aux caractéristiques des données d'entrée. En pratique, on va minimiser $\|W\|_1$ ou 2 grâce à la fonction de coût.
- ▶ On note $L(W, b)$, la fonction de coût de la prédiction. Exemple : $L(W, B) = \|\hat{Y}(W, b) - Y\|_2^2$
- ▶ On va ajouter à cette fonction de coût un terme qui pénalise les « grands » W :

$$L_{\text{tot}}(W, b) = L(W, b) + \alpha \|W\|_2^2$$

α est un coefficient qui donne plus ou moins d'importance à la régularisation : α petit \rightarrow pas de régularisation, α grand \rightarrow on va chercher à avoir des poids petits sans tenir compte de l'apprentissage.

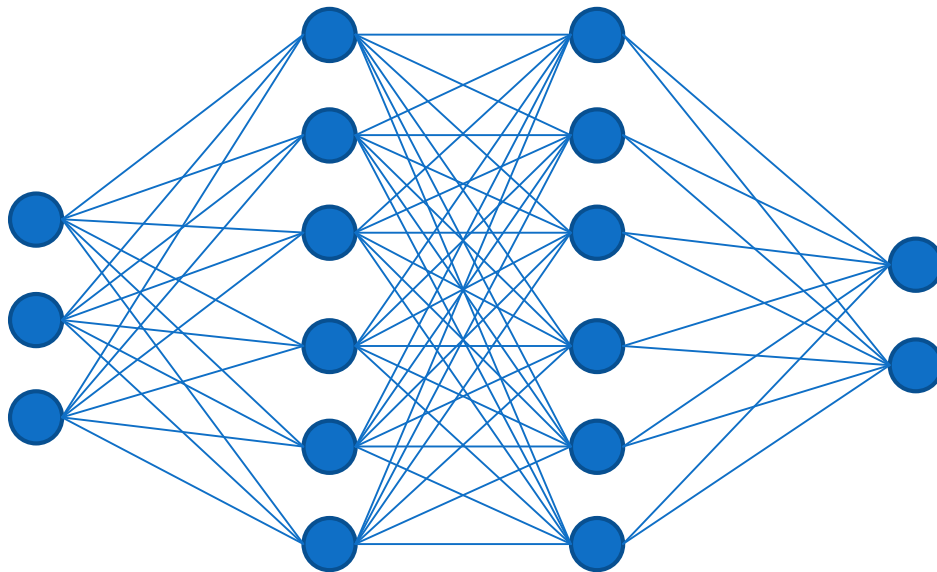
On peut voir l'influence de α grâce au monitoring : on n'arrive plus à *fitter* les données d'apprentissage.

- ▶ Implémentation Keras :

```
from keras import regularizers
model.add(Dense(64, input_dim=64,
                kernel_regularizer=regularizers.l2(0.01),
```

Régularisation : *Dropout*

- ▶ Idée : éviter que certains neurones ne se spécialisent pour un exemple particulier
- ▶ On éteint les neurones aléatoirement pendant l'apprentissage. Un neurone est éteint (output = 0) avec une probabilité p (*dropout rate*) définie par l'utilisateur (couches intermédiaires uniquement)

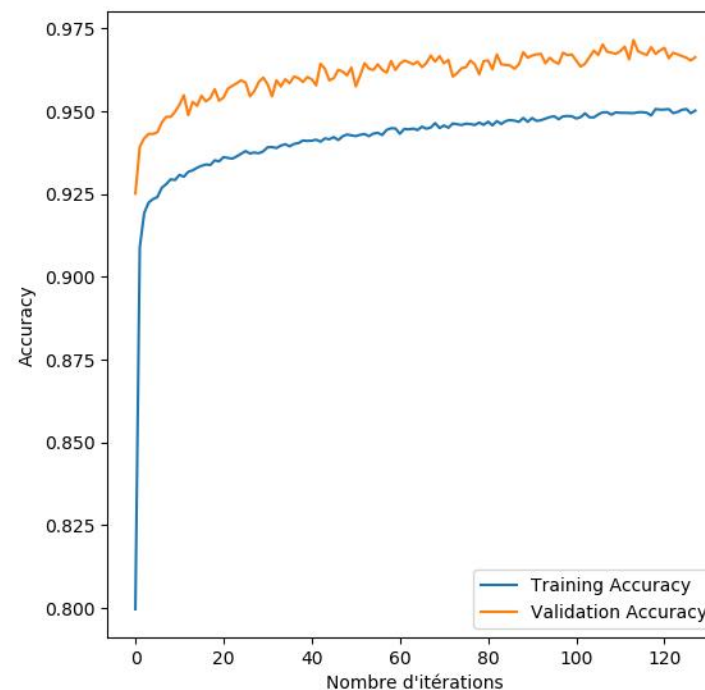
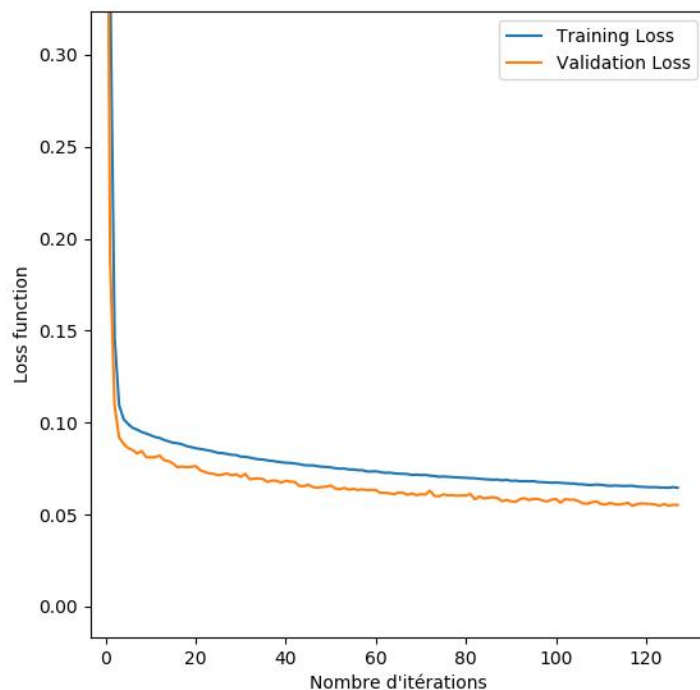


Implémentation Keras :
`model.add(Dropout(dropout_rate))`

- ▶ Dropout seulement pour l'apprentissage (utilisable dans les tests pour les réseaux bayésiens → séance 4)

Régularisation : *Dropout*

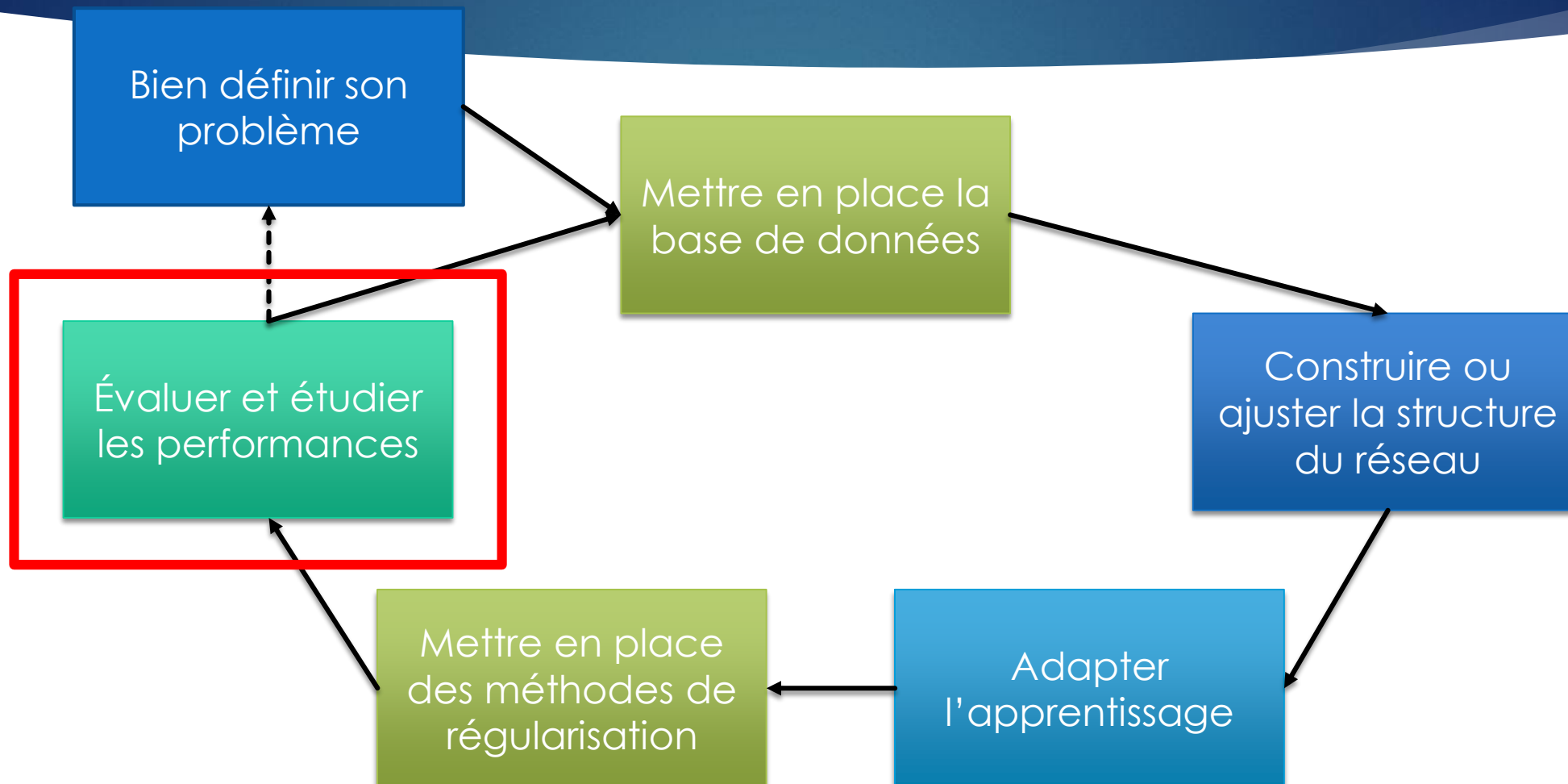
- ▶ Un effet lors du monitoring : meilleure performance sur le jeu de validation que sur le jeu d'apprentissage ! Le jeu de validation utilise tout le réseau, alors que le *dropout* éteint certains neurones pour le jeu d'apprentissage.



Régularisation : *batch-normalization*

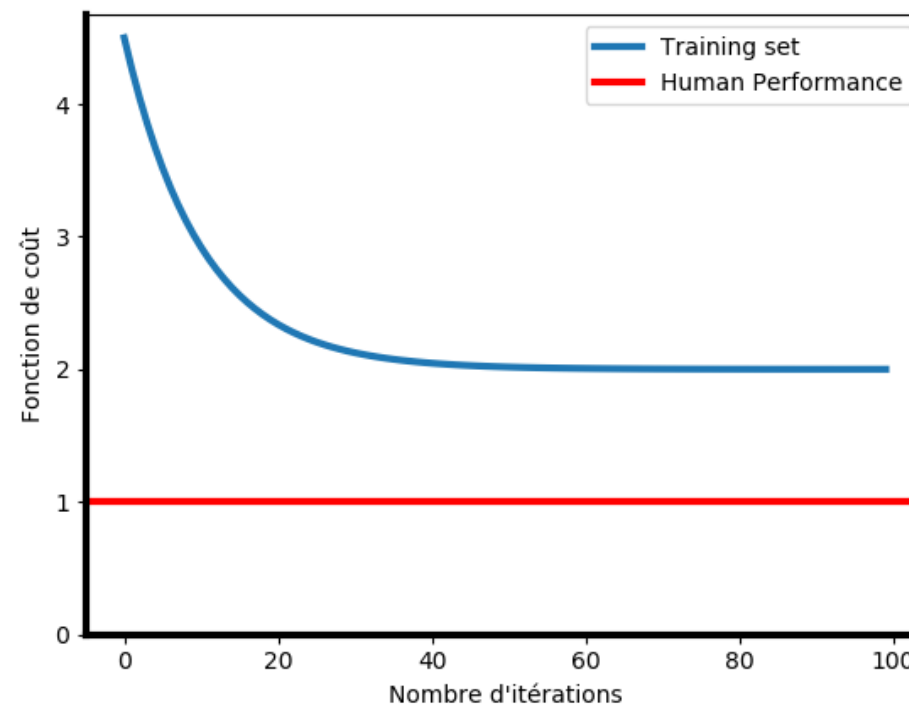
- ▶ Normalise en moyenne et en variance les sorties de chaque couche sur le mini-batch en cours d'apprentissage
 - ▶ Apprentissage de deux paramètres supplémentaires (moyenne et variance)
- ▶ Rend les couches plus indépendantes les unes des autres
- ▶ Conseil : mettre en place une *batch-normalization* après chaque couche intermédiaire
- ▶ Implémentation Keras : `model.add(BatchNormalization)`

Projet de deep learning



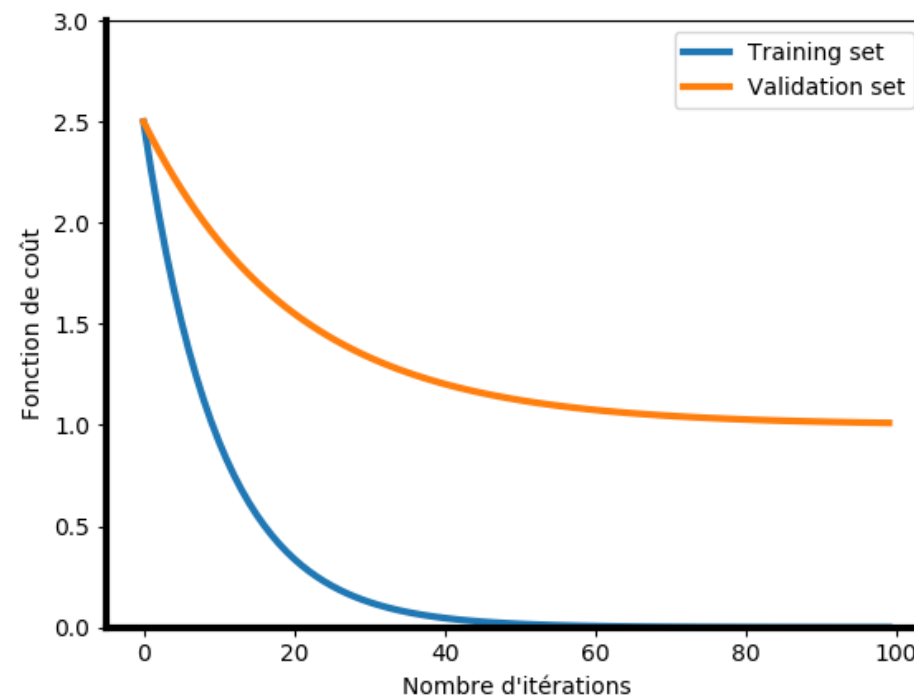
Étude des performances : diagnostic

- ▶ **Problème** : Mauvaises performances sur la base d'apprentissage
- ▶ **Causes possibles** :
 - ▶ Mauvais choix de *learning rate*
 - ▶ Trop de régularisation (α trop grand, *dropout rate* trop grand) → moins de régularisation
 - ▶ Pas assez de couches et/ou de neurones → Complexifier le réseau
 - ▶ N'arrive pas à fitter les données parce qu'il n'y a pas de corrélation entre les données d'entrée et de sortie



Étude des performances : diagnostic

- ▶ **Problème** : Convergence sur le jeu d'apprentissage, pas sur le jeu de validation
- ▶ **Cause** : Cas typique d'*overfitting*
 - ▶ Mettre en place/renforcer la régularisation
 - ▶ Si le problème n'est pas résolu : diminuer la complexité du réseau (moins de couches, moins de neurones)



Étude des performances : diagnostic

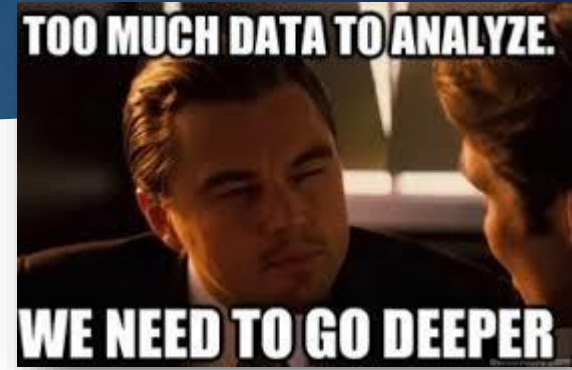
- ▶ **Problème** : Bonne convergence sur le jeu d'apprentissage et le jeu de validation, mais mauvaises performances ensuite sur le jeu de test
- ▶ **Causes possibles** :
 - ▶ *Overfitting* à cause de l'*early stopping* sur la base de validation (peu probable)
 - ▶ Le jeu de test ne provient pas de la même distribution que les données d'apprentissage et de validation
 - ▶ Si c'est volontaire, donner plus de poids aux données qui sont de la même distribution que le jeu de test dans le jeu d'apprentissage et de validation
 - ▶ Sinon, revoir la base de données
 - ▶ Vérifier les erreurs faites sur les données de test : il peut y avoir des erreurs systématiques (mauvaises performances à cause du bruit ; une classe est sur ou sous-représentée ; on a introduit un biais dans les données d'apprentissage...)

Pour résumer

- ▶ Bien poser le problème à résoudre
- ▶ Construire la base de données en conséquence
- ▶ Projet de *deep learning* : processus itératif
 - ▶ On développe : mise en place du réseau
 - ▶ On fait l'apprentissage
 - ▶ On teste
 - ▶ On diagnostique et on agit en conséquence : régularisation, complexification du réseau, revoir la base de données
- ▶ Intérêt d'accélérer le calcul (GPU) : Pouvoir exécuter ce processus rapidement

Réseaux de neurones et deep learning

- ▶ Partie 1 : Introduction aux **réseaux de neurones**
 - ▶ Utilisations courantes du deep learning
 - ▶ Bases générales
- ▶ Partie 2 : **Architecture** des réseaux, **hyperparamètres** et évaluation des **performances**
 - ▶ Comment construire mon réseau et adapter la phase d'apprentissage ?
 - ▶ Comment évaluer les performances de mon réseau de neurones ?
- ▶ Partie 3 : Construction de la **base de données**
 - ▶ Éléments méthodologiques sur la mise en place du problème à résoudre potentiellement par deep learning
 - ▶ Comment utiliser l'évaluation des performances pour améliorer la base de données et le réseau ?
- ▶ **Partie 4 : Réseaux de neurones convolutifs**
 - ▶ **Introduction à des structures plus avancées**



Les réseaux de neurones convolutifs (CNN)

- ▶ Motivations sur les images :
 - ▶ Corrélation locale des données
 - ▶ Détection de formes
 - ▶ Invariance par translation : les formes peuvent se trouver à différents endroits de l'image

Les réseaux de neurones convolutifs (CNN)

► L'opération de convolution :

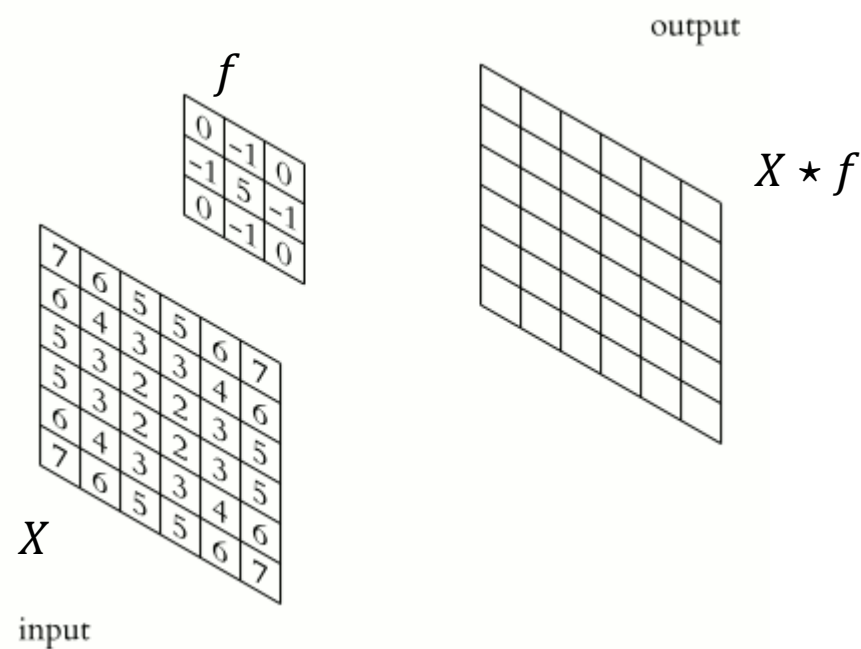
- On note X un tableau $n \times m$, f un filtre $n_f \times m_f$ avec $n_f \leq n$, $m_f \leq m$, la convolution de X par f pour un pixel de coordonnées (i, j) , $i \in \llbracket 0, n - 1 \rrbracket$, $j \in \llbracket 0, m - 1 \rrbracket$ est définie par :

$$(X \star f)[i, j] = \sum_{k=-\frac{n_f}{2}}^{\frac{n_f}{2}} \sum_{l=-\frac{m_f}{2}}^{\frac{m_f}{2}} X[i + k, j + l] \times f\left[k + \frac{n_f}{2}, l + \frac{m_f}{2}\right]$$

- Dans le cas ci-dessus, si $i + k$ ou $j + l$ sort de l'image X (c'est-à-dire, $i + k \geq n$ par exemple) alors on prend $X[i + k, j + l] = 0$, on parle de « zéro-padding ». D'autres padding sont possibles (en reprenant la valeur du pixel le plus proche par exemple).
- On peut aussi choisir de ne faire le calcul que sur les pixels (i, j) tels que $i + k$ et $j + l$ sont toujours dans l'image : il y aura alors réduction de la dimension de sortie

Les réseaux de neurones convolutifs (CNN)

- ▶ Plus visuellement :
- ▶ Le filtre ou noyau f est appelé filter ou kernel en anglais



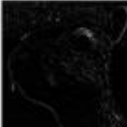
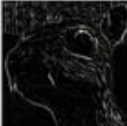





Michael Plotke,
https://upload.wikimedia.org/wikipedia/commons/1/19/2D_Convolution_Animation.gif

Les réseaux de neurones convolutifs (CNN)

- ▶ Effet des convolutions, exemple sur des images :



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

[Deep Learning Methods for Vision, CVPR 2012 Tutorial](https://www.cse.cmu.edu/~pabdey/courses/687/lectures/lecture11/Deep_Learning_Methods_for_Vision_CVPR_2012_Tutorial)

<https://ujwkarn.files.wordpress.com/2016/08/giphy.gif>

[Wikipedia article on Kernel \(image processing\)](#)

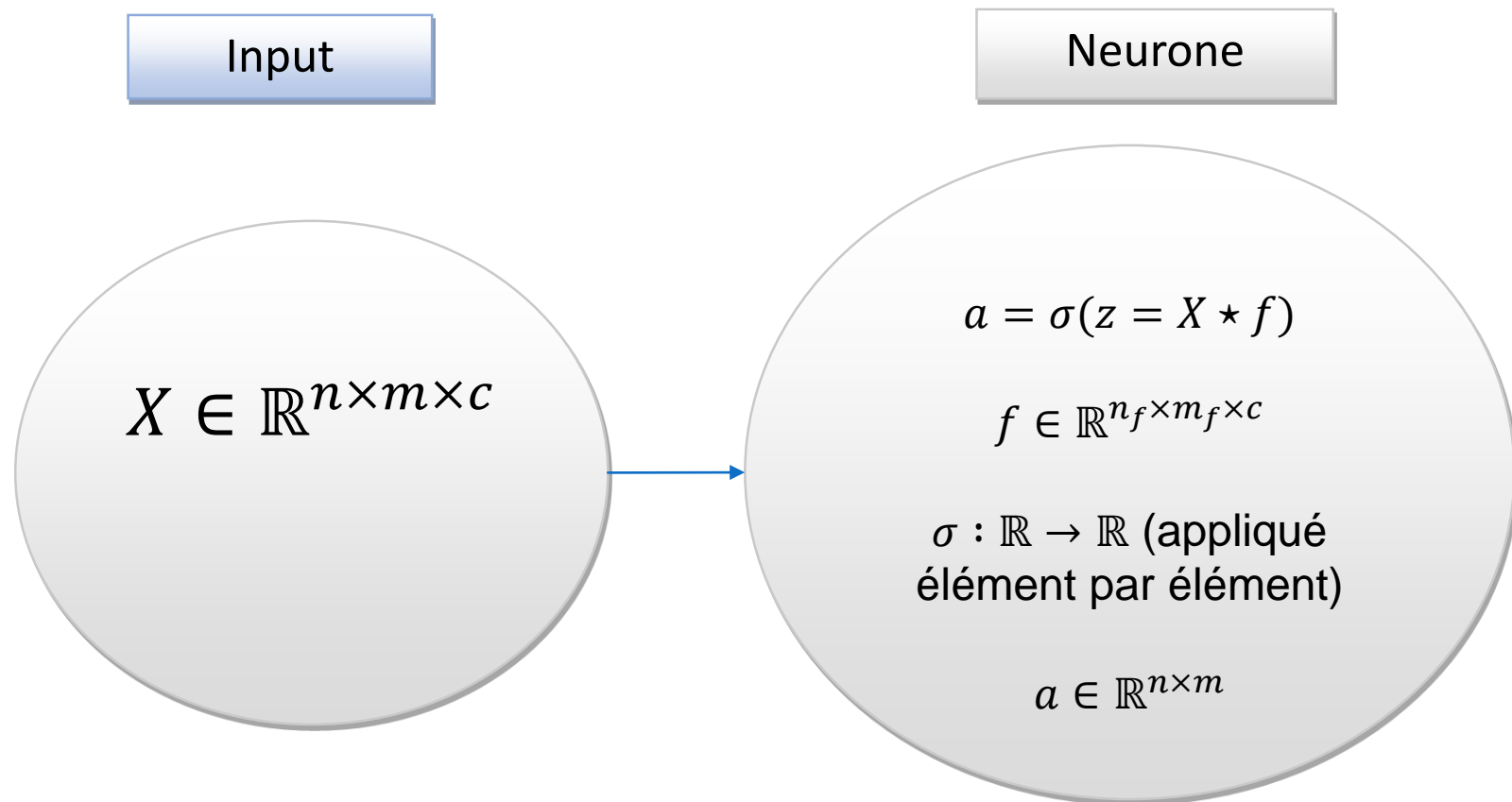
Les réseaux de neurones convolutifs (CNN)

► Un neurone de convolution :

On ne choisit pas les **filtres**, on les apprend !!! Ce sont des **paramètres entraînables** du réseau

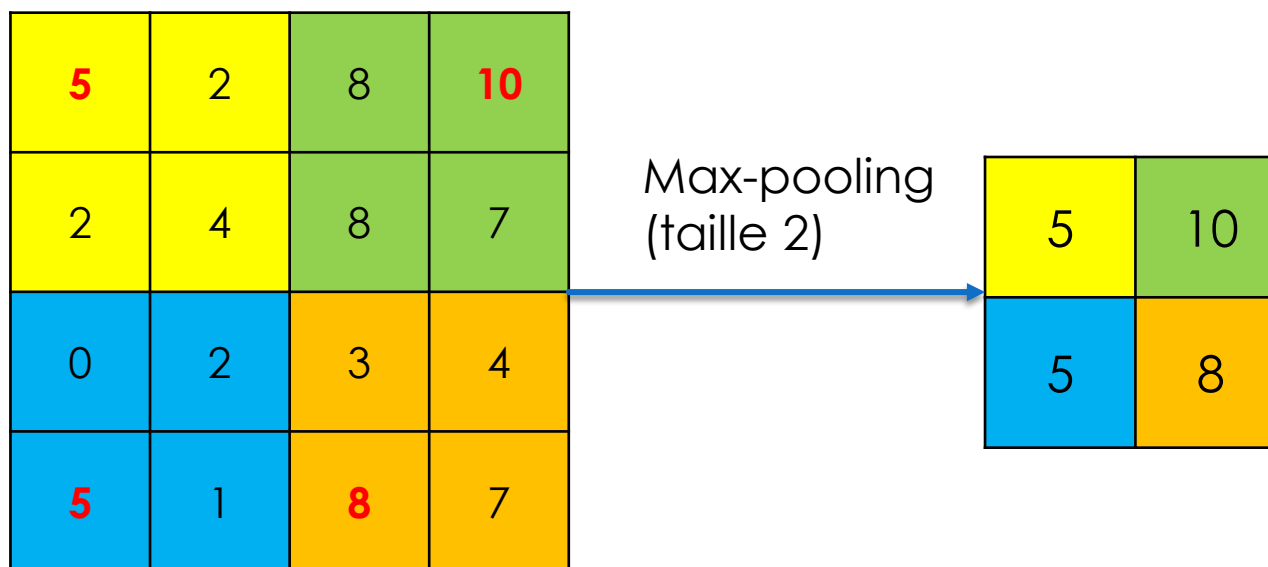
c s'appelle nombre de channels ou de canaux :

- $c = 1$ pour une image en niveaux de gris
- $c = 3$ pour une image RGB (une channel par couleur)
- Dans les couches intermédiaires, c correspond au nombre de neurones de la couche précédente



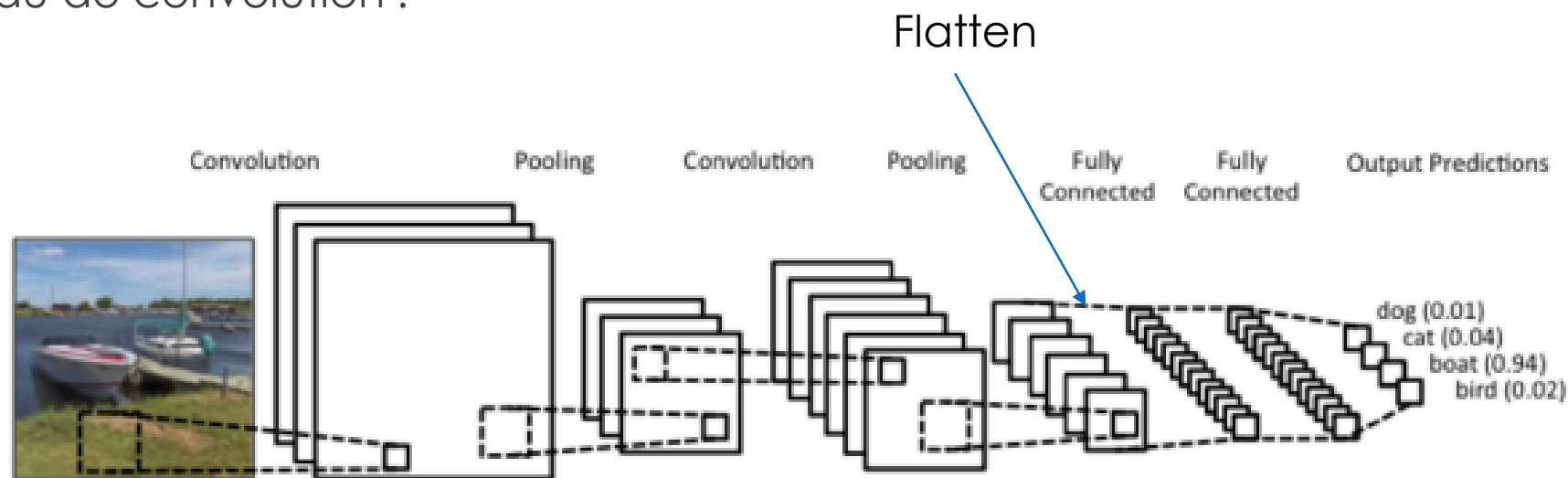
Les réseaux de neurones convolutifs (CNN)

- ▶ Le *Pooling* : opération utilisée pour réduire la dimension
 - ▶ Recherche de détails plus « grossiers », de plus grandes « structures » dans l'image
 - ▶ *Max-pooling* de taille l : on prend l'élément maximal de chaque sous-tableau de taille l
 - ▶ *Sum-pooling* de taille l : on fait la somme de tous les éléments de chaque sous tableau de taille l



Les réseaux de neurones convolutifs (CNN)

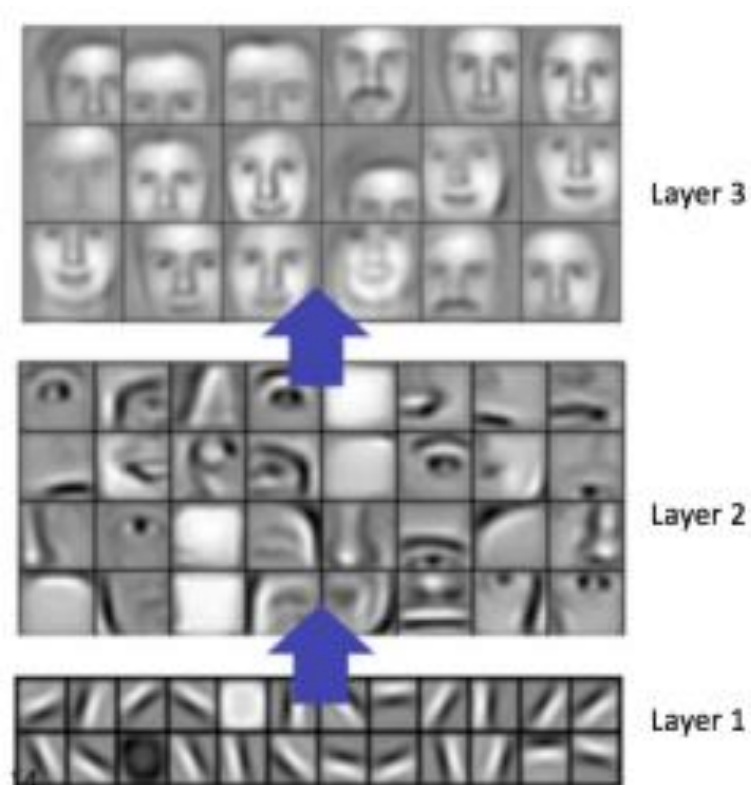
- ▶ Un réseau de convolution :



<https://towardsdatascience.com/build-your-own-convolution-neural-network-in-5-mins-4217c2cf964f>

Les réseaux de neurones convolutifs (CNN)

- ▶ Les filtres apprennent des formes de plus en plus complexes, globales



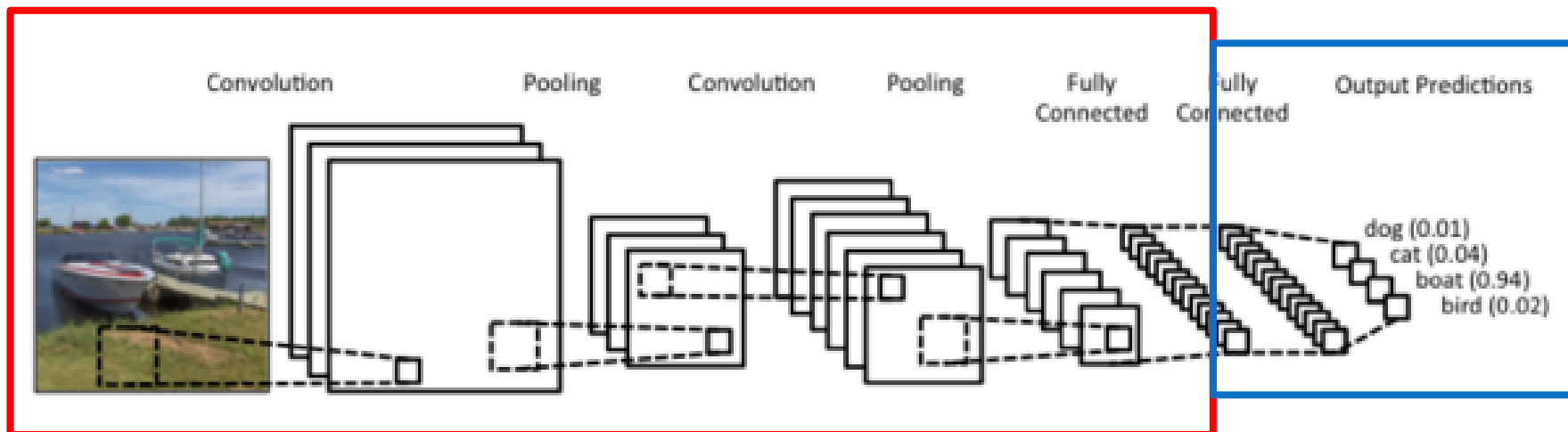
Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations, Lee et al., Proceedings of the 26th International Conference on Machine Learning, 2009

Les réseaux de neurones convolutifs (CNN)

- ▶ Sous Keras (en **gras** : ce qui est nouveau, en italique, ce qui est modifiable) :
 - ▶ `keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None)`
 - ▶ `keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid')`
 - ▶ `keras.layers.UpSampling2D(size=(2, 2), interpolation='nearest')` : opération inverse du pooling
 - ▶ Ces trois couches existent aussi en version 1D et 3D
 - ▶ `keras.layers.Flatten()` (pour mettre les données sur une seule ligne avant des fully-connected layers)
- ▶ Quelques ressources :
 - ▶ LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.
 - ▶ Zeiler, Matthew D., et al. "Deconvolutional networks." *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. IEEE, 2010.*
 - ▶ <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/> (explication générale sur les CNN)
 - ▶ <http://scs.ryerson.ca/~aharley/vis/conv/flat.html> (représentation d'un réseau de neurones et des couches intermédiaires sur les chiffres MNIST)

Transfer learning

- ▶ Idée : conserver l'extraction des caractéristiques apprise sur d'autres problématiques
 - ▶ Revient à conserver des couches de convolution apprises sur un problème similaire
 - ▶ On ne change que la (ou éventuellement les) dernière(s) couche(s) d'identification

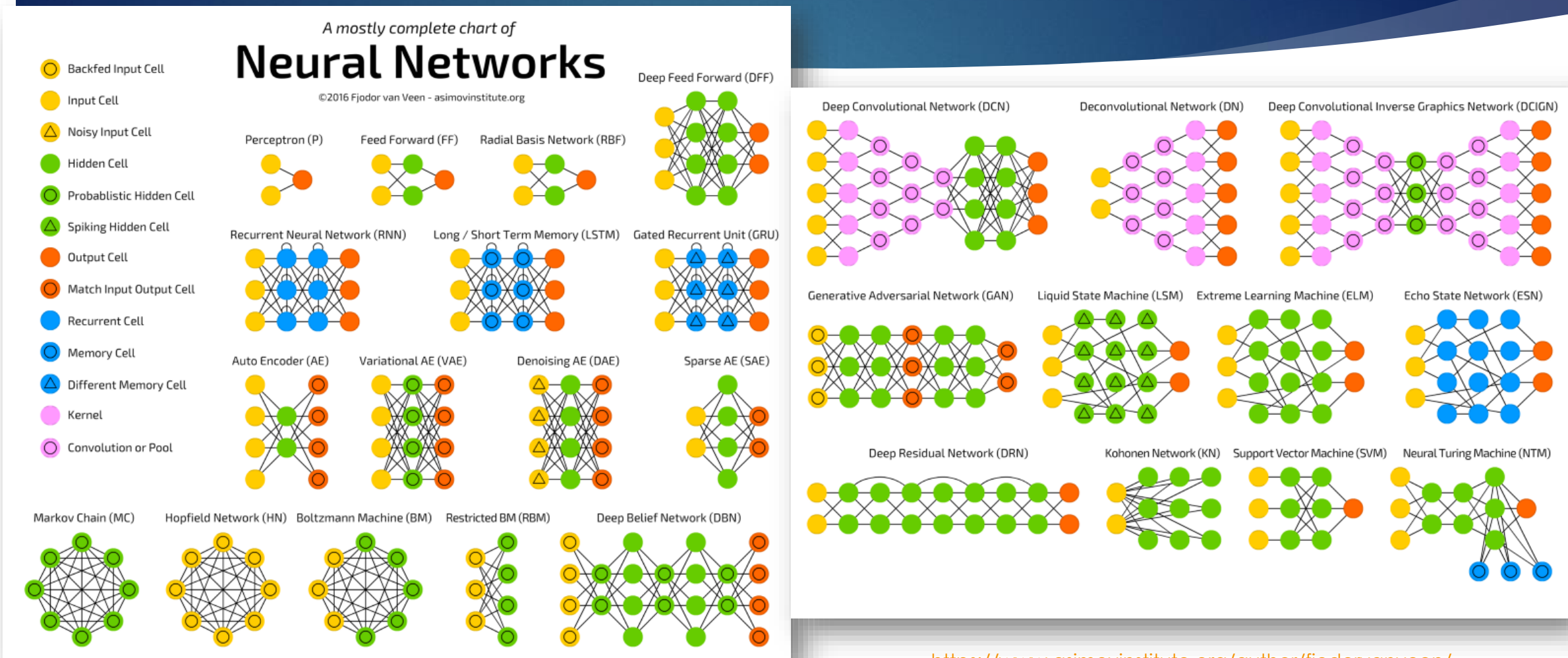


Fixée
(déjà
apprise)

On apprend
seulement ces
couches

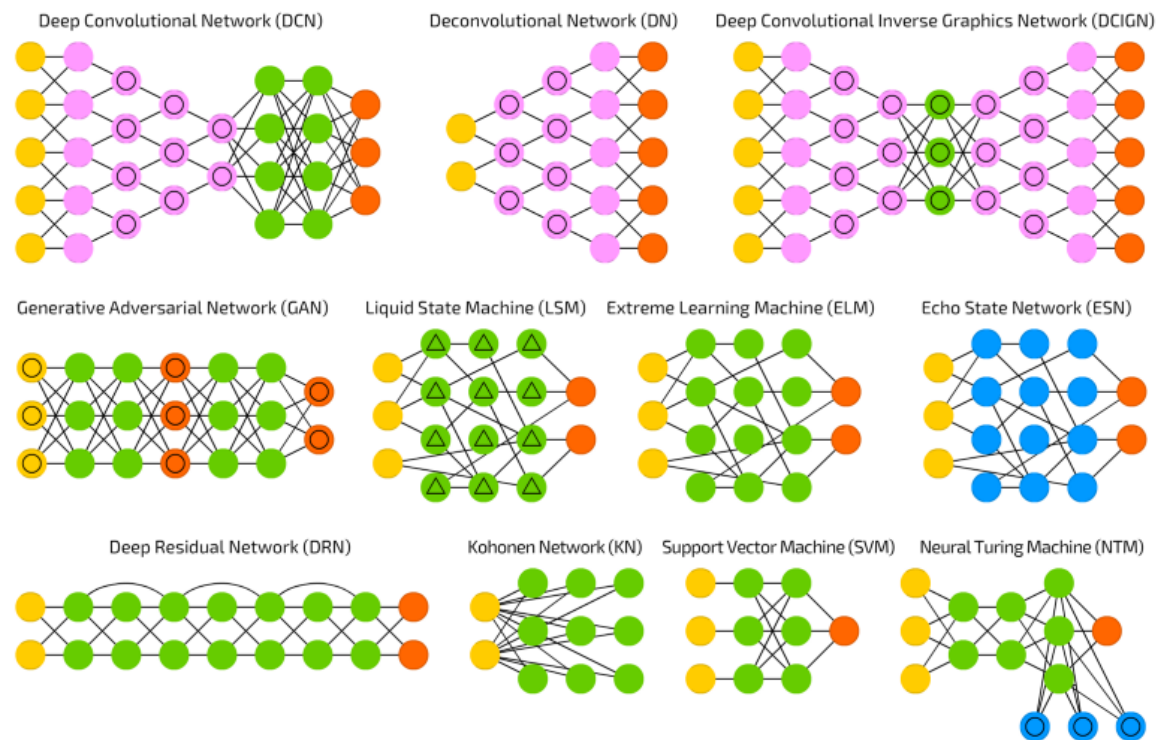
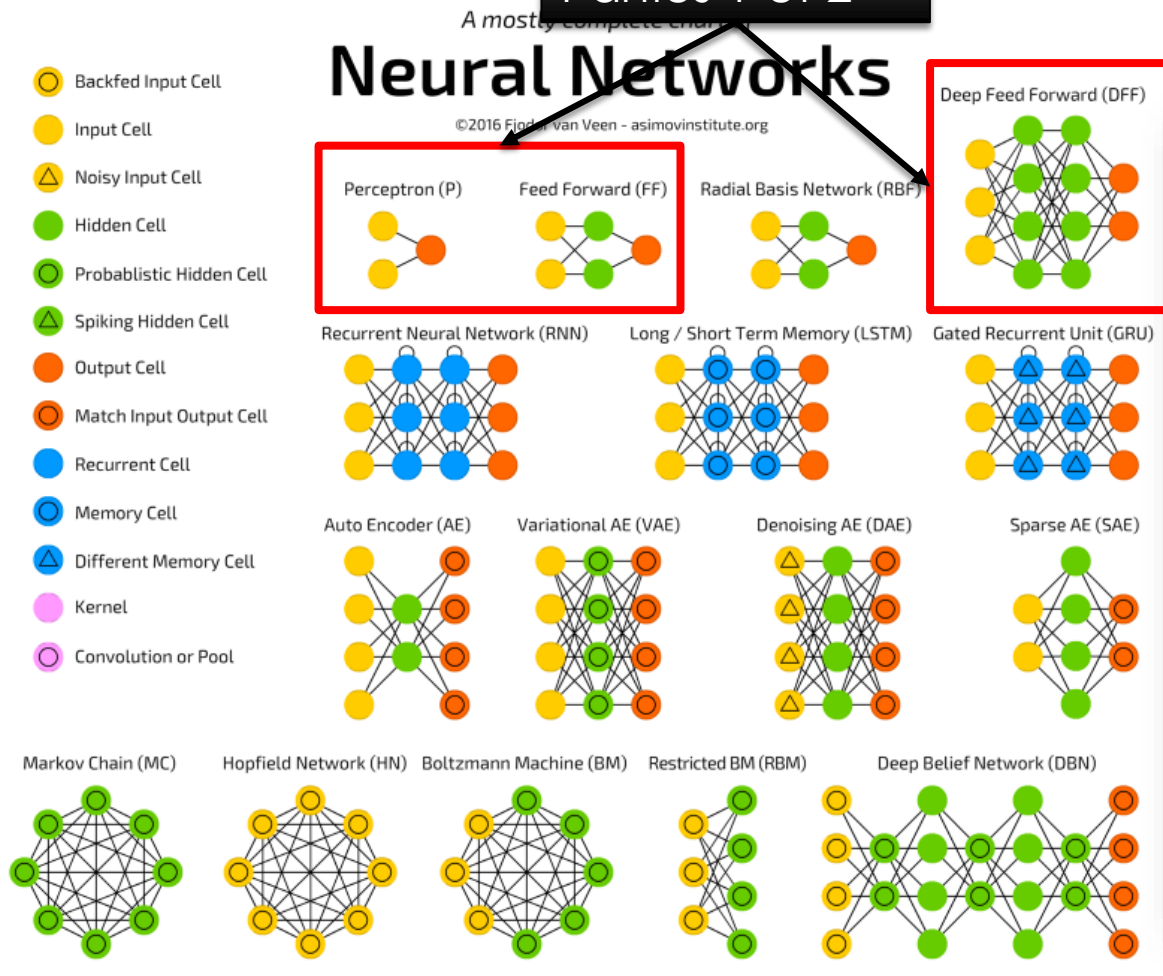
<https://towardsdatascience.com/build-your-own-convolution-neural-network-in-5-mins-4217c2cf964f>

D'autres structures : le zoo des réseaux



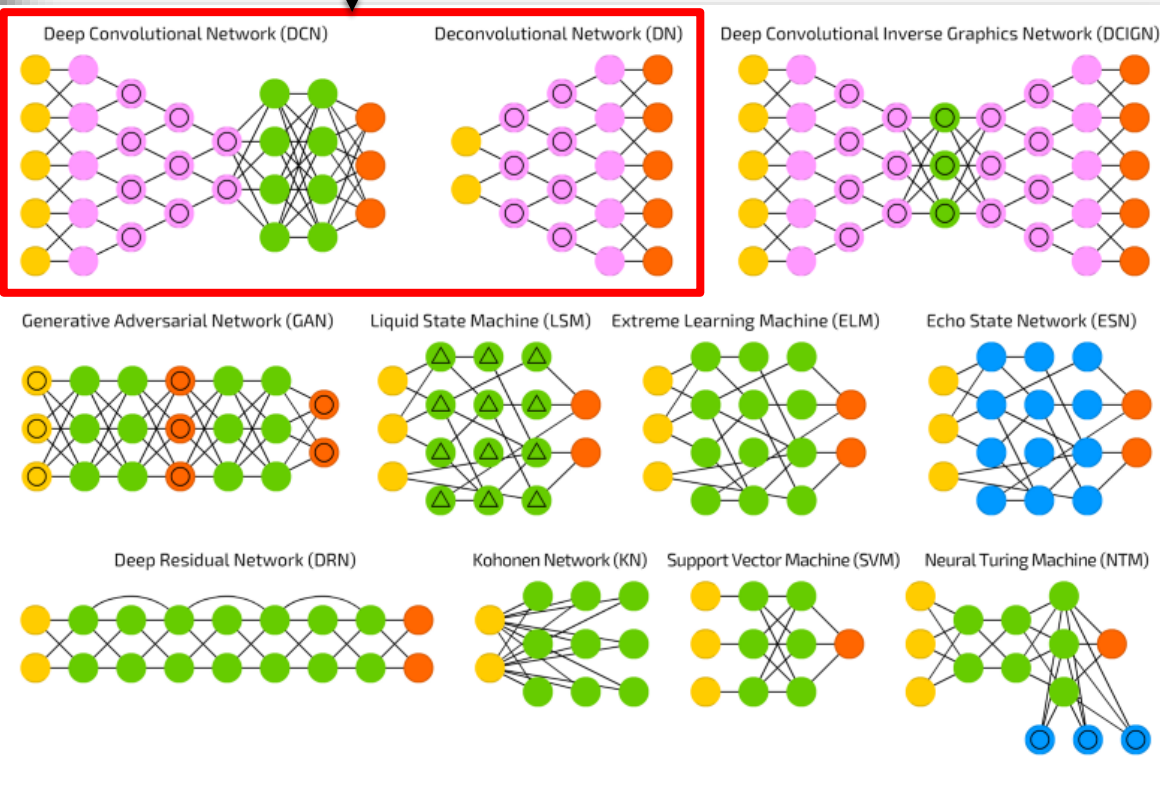
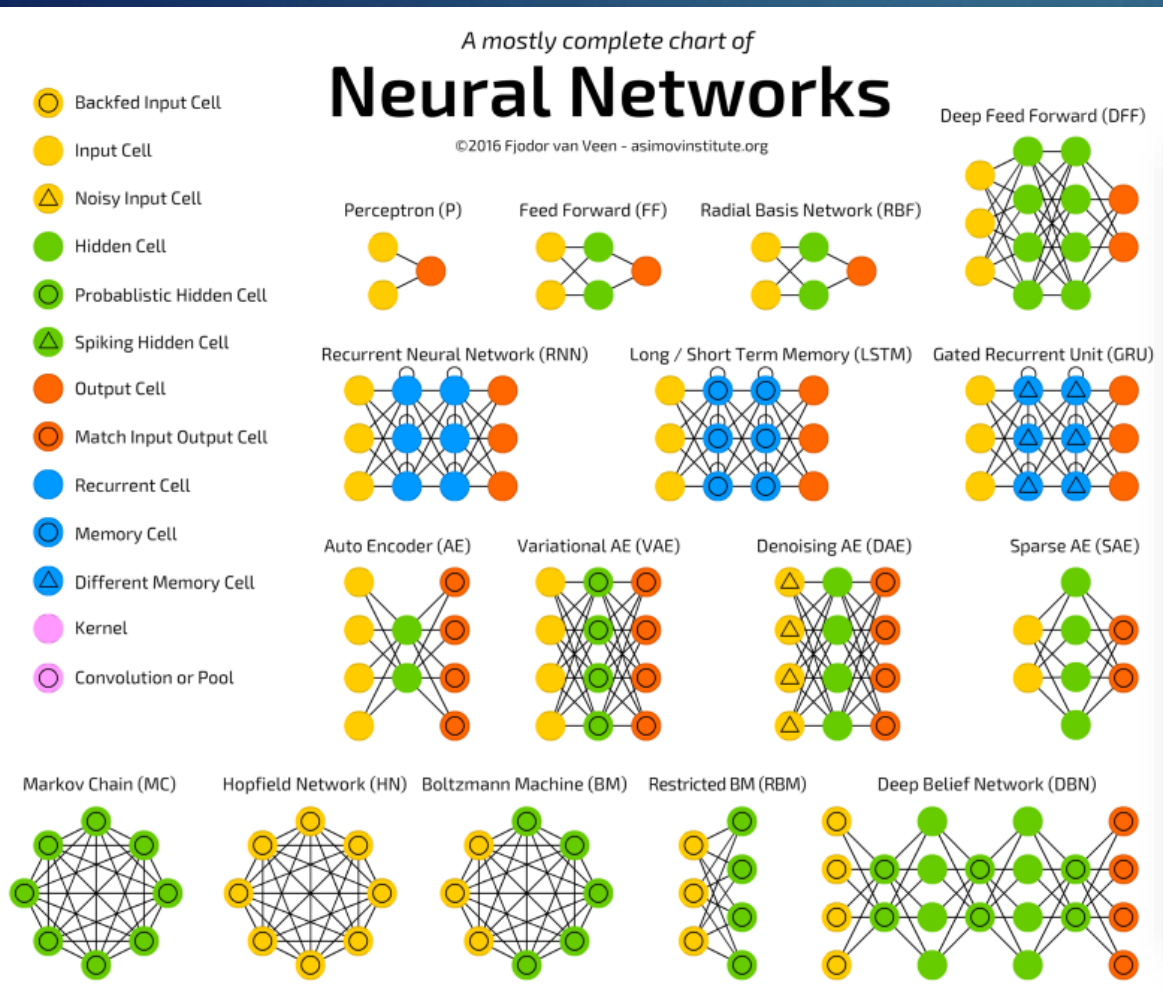
D'autres structures : le zoo des réseaux

Parties 1 et 2

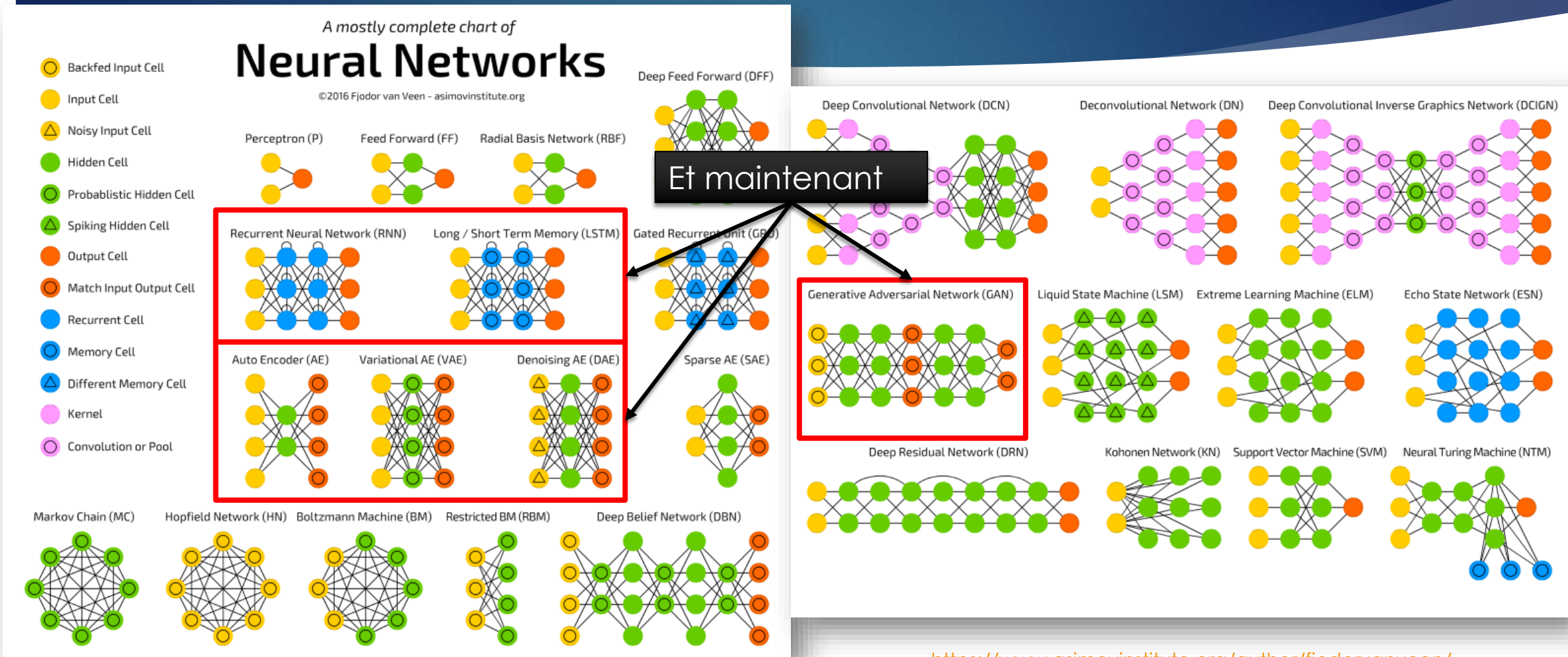


D'autres structures : le zoo des réseaux

Partie 4



D'autres structures : le zoo des réseaux



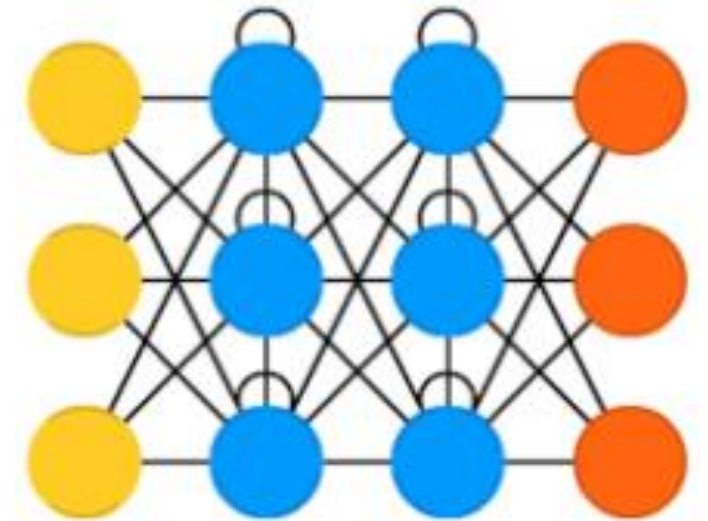
D'autres structures : les réseaux récurrents

- ▶ Problème à traiter : analyse de séquences (textes, enregistrement audio/vidéo)
 - ▶ Input de tailles différentes
 - ▶ Ordre dans les données

- ▶ Recurrent Neural Network (+ LSTM) :
 - ▶ Les neurones récurrents prennent en input :
 - ▶ La nouvelle information (le nouveau mot, la nouvelle image...)
 - ▶ Sa propre sortie précédente (ce que le neurone avait calculé avant)

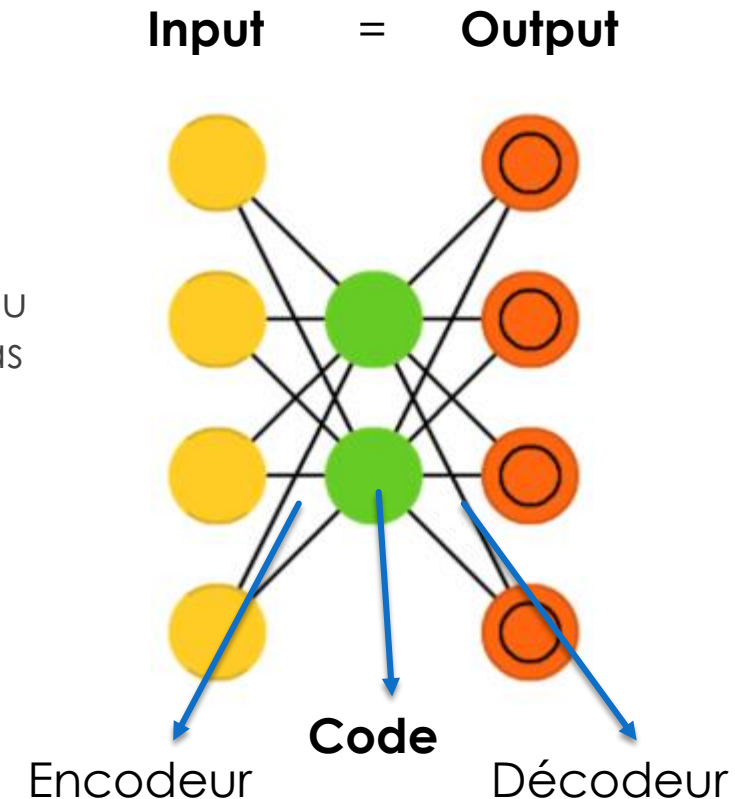
- ▶ Quelques références :

- ▶ Elman, Jeffrey L. "Finding structure in time." *Cognitive science* 14.2 (1990)
- ▶ Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997)
- ▶ Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." *arXiv preprint arXiv:1412.3555* (2014)



D'autres structures : les auto-encodeurs

- ▶ Problème : on veut réduire la dimension de nos données
 - ▶ Compresser les données
- ▶ Auto-encodeur :
 - ▶ On prend les mêmes données comme input et output ! Pas besoin de labels ou de « pré-travail ». Attention : on ne fait que compresser les données, il n'y a pas de classification à proprement parler.
 - ▶ Réseaux symétriques en général (en termes de nombre de neurones et de couches dans l'encodeur et le décodeur)
 - ▶ Il n'y a pas forcément une unique couche, on peut aussi combiner avec des couches de convolution
 - ▶ La couche du milieu correspond au « code »



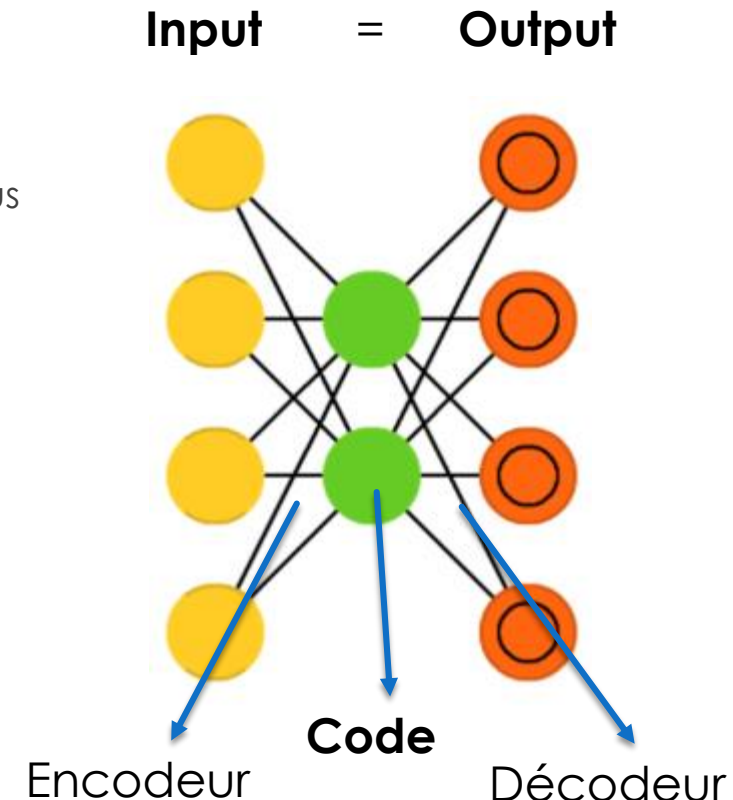
D'autres structures : les auto-encodeurs

► Utilisation :

- Débruiter les données
- Réduire la dimension avant d'appliquer un autre algorithme :
 - Mettre des couches d'identification à la place du décodeur : le « code » peut être plus simple à traiter
 - Garder le code et faire du clustering, de la classification non supervisée sur le « code »

► Quelques ressources :

- *Bourlard, Hervé, and Yves Kamp. "Auto-association by multilayer perceptrons and singular value decomposition." Biological cybernetics 59.4-5 (1988)*
- *Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann LeCun. "Efficient learning of sparse representations with an energy-based model." Proceedings of NIPS. 2007*
- *Vincent, Pascal, et al. "Extracting and composing robust features with denoising autoencoders." Proceedings of the 25th international conference on Machine learning. ACM, 2008.*



D'autres structures : les GANs

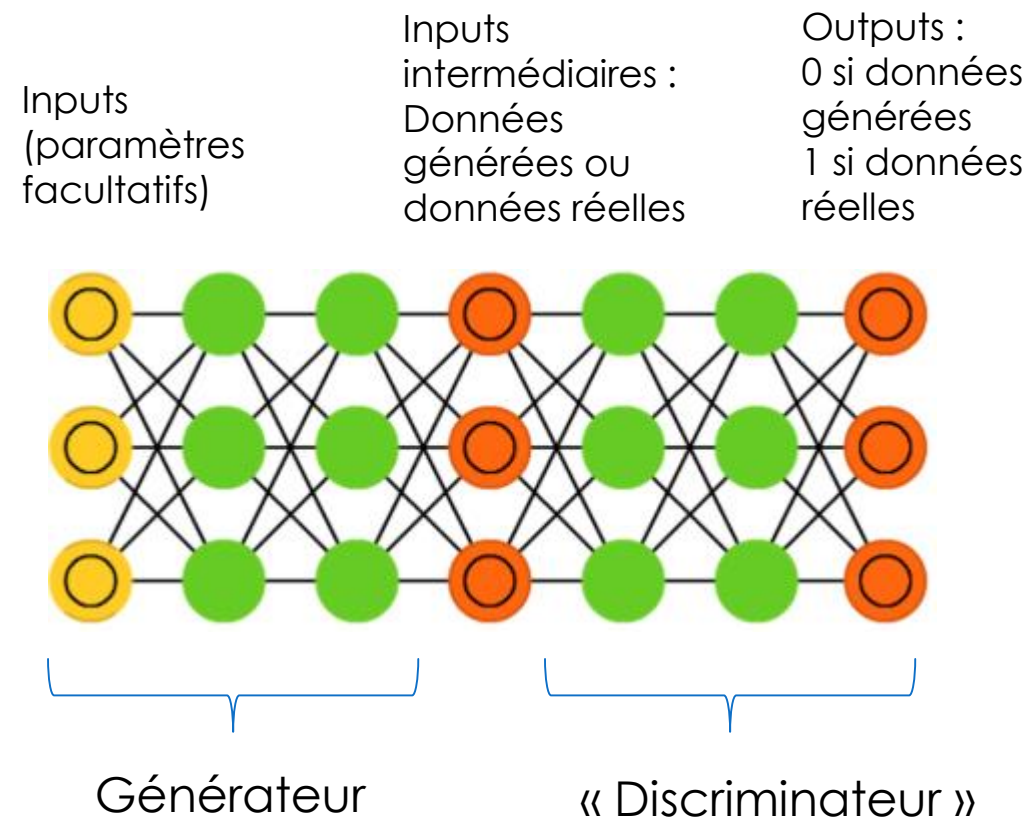
► **Generative Adversarial Networks**

► Problème : on veut générer des données qui semblent « vraies »

- Exemple : générer des images de personnes qui n'existent pas mais qui ont l'air réelles
- Utilisation possible en simulation

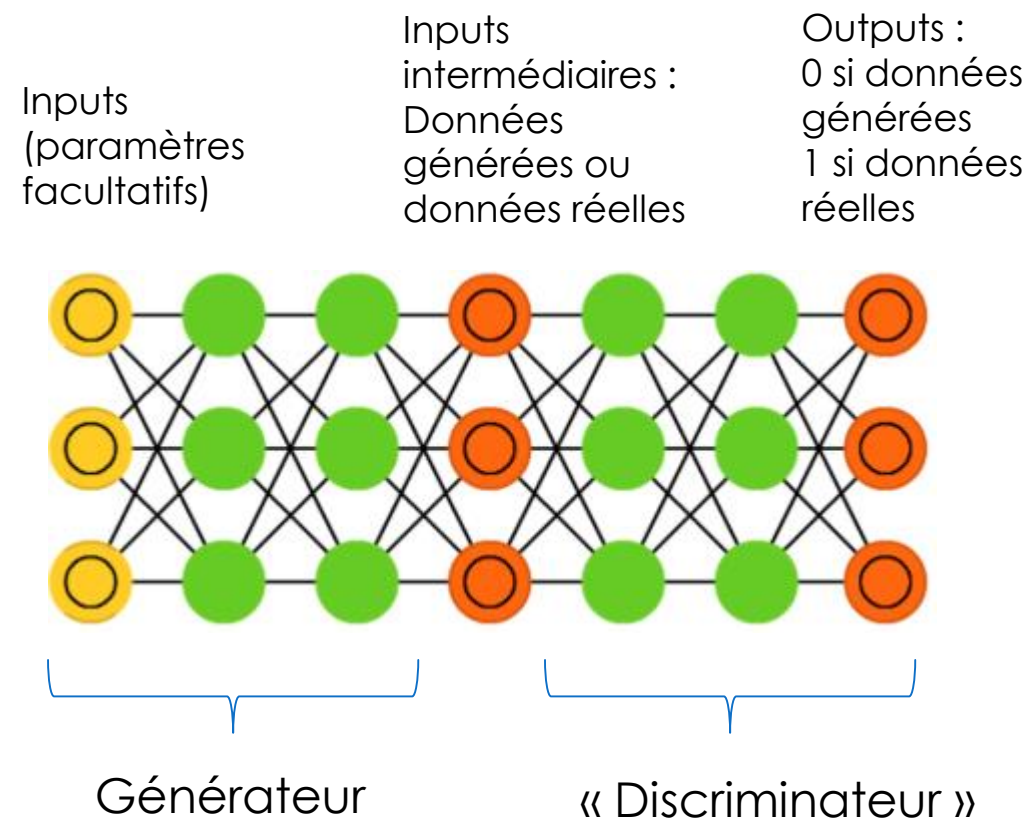
► Deux réseaux travaillent en compétition :

- Un réseau générateur essaie de générer des inputs qui ont l'air « vrais »
- Un réseau discriminateur essaie de distinguer les données générées des données réelles



D'autres structures : les GANs

- ▶ Méthode : on dispose de données réelles
 - ▶ Première étape : générateur initialisé aléatoirement, le discriminateur est entraîné (Inputs : données réelles ou générées, Outputs : 0 si généré, 1 si réel)
 - ▶ Deuxième étape : on fixe le discriminateur, et on entraîne le générateur à tromper le discriminateur (il faut que le discriminateur donne 1 au maximum de données générées)
 - ▶ Ensuite : on reprend l'étape 1 avec le nouveau générateur entraîné, puis l'étape 2 et ainsi de suite
- ▶ Quelques ressources :
 - ▶ "Generative adversarial nets." Goodfellow, Ian, et al. Advances in Neural Information Processing Systems. 2014.
 - ▶ Generative models for fast simulation, S.Vallecorsa, CERN
 - ▶ Generative adversarial networks simulate gene expression and predict perturbations in single cells, Ghahramani, Watt, Luscombe (2018)



Pour résumer

- ▶ Réseaux de neurones convolutifs :
 - ▶ Permet d'extraire les caractéristiques pertinentes dans les inputs : caractéristiques qui sont invariantes par translation (forme dans les images), corrélation locale
- ▶ De nombreuses autres structures existent :
 - ▶ Récurrentes : pour les séquences
 - ▶ Auto-encodeurs, GANs...
- ▶ À vous de jouer !

Quelques ressources

- ▶ Cours en ligne :

- ▶ Coursera, spécialisation deep learning :

- <https://www.coursera.org/specializations/deep-learning>

- Quiz pour s'entraîner et exercices de programmation (Python) : fortement recommandé pour ceux qui veulent vraiment faire du deep learning

- ▶ Cours de Yann Le Cun au Collège de France sur l'apprentissage profond (vidéos) :

- <https://www.college-de-france.fr/site/yann-lecun/course-2015-2016.htm>

- ▶ Open Course MIT (1^{ère} vidéo, les autres sont normalement proposées à la suite par Youtube)

- <https://www.youtube.com/watch?v=TjZBTDzGeGg>

Quelques ressources

▶ Vidéos Youtube :

- ▶ Science Étonnante : Le deep learning (présentation générale)

<https://www.youtube.com/watch?v=trWrEWfhTVg>

- ▶ 3blue1brown : Calcul des réseaux de neurones illustrés (4 vidéos) :

<https://www.youtube.com/watch?v=aircAruvnKk> (Introduction aux ANN)

<https://www.youtube.com/watch?v=IHZwWFHWa-w> (Descente de gradient)

<https://www.youtube.com/watch?v=llg3gGewQ5U> (Backpropagation version friendly)

<https://www.youtube.com/watch?v=tIeHLnjs5U8> (Backpropagation calculs)

- ▶ Science4all : Playlist Intelligence artificielle, presque 50 vidéos sur l'IA (parfois un peu « philosophiques » et « sociologiques », d'autres plus techniques, réseaux de neurones à partir de la vidéo 40)

<https://www.youtube.com/watch?v=DrjkjPVf7Bw&list=PLtzmb84AoqRTI0m1b82gVLcGU38miqdrC>

(vidéos 42 et 43 non accessibles dans la playlist, mais toujours accessibles depuis la chaîne)

Quelques ressources

- ▶ Livre d'Andrew Ng (co-fondateur de Coursera, professeur en charge du cours de spécialisation Deep Learning) :
 - ▶ En libre accès : <https://github.com/xiaqunfeng/machine-learning-yearning>
 - ▶ Possibilité de s'inscrire pour recevoir une version gratuite : <http://www.mlyearning.org/>
- ▶ Dropout :
 - ▶ *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, Srivastana et al. (2014)
 - ▶ *Improving neural networks by preventing co-adaptation of feature detectors*, Hinton et al. (2012)
- ▶ Batch-Normalization :
 - ▶ *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, Ioffe & Szegedy (2015)

Pour installer Keras

- ▶ Installation d'Anaconda et Python :

<https://www.anaconda.com/download/> (Pour Windows, Mac et Linux)

- ▶ Pour Python, j'utilise « Spyder » comme environnement (intégré à Anaconda), mais d'autres peuvent être utilisés aussi

- ▶ Installation de Keras-GPU (prendre par défaut la version GPU, même si l'ordinateur n'est pas équipé de carte graphique ad-hoc, cette version marche aussi bien sur CPU) :

- ▶ Ouvrir une console « Anaconda Prompt »
- ▶ Taper : `conda install -c anaconda keras-gpu`
- ▶ Pour plus de détails : <https://anaconda.org/anaconda/keras-gpu>

- ▶ Utilisation des GPU :

- ▶ Pour l'utilisation sur GPU : vérifier que la carte graphique supporte l'utilisation de Cuda, installer Cuda et ajouter le dossier d'installation au Path de Python
- ▶ Et aussi toujours de l'aide : <https://stackoverflow.com/questions/45662253/can-i-run-keras-model-on-gpu>