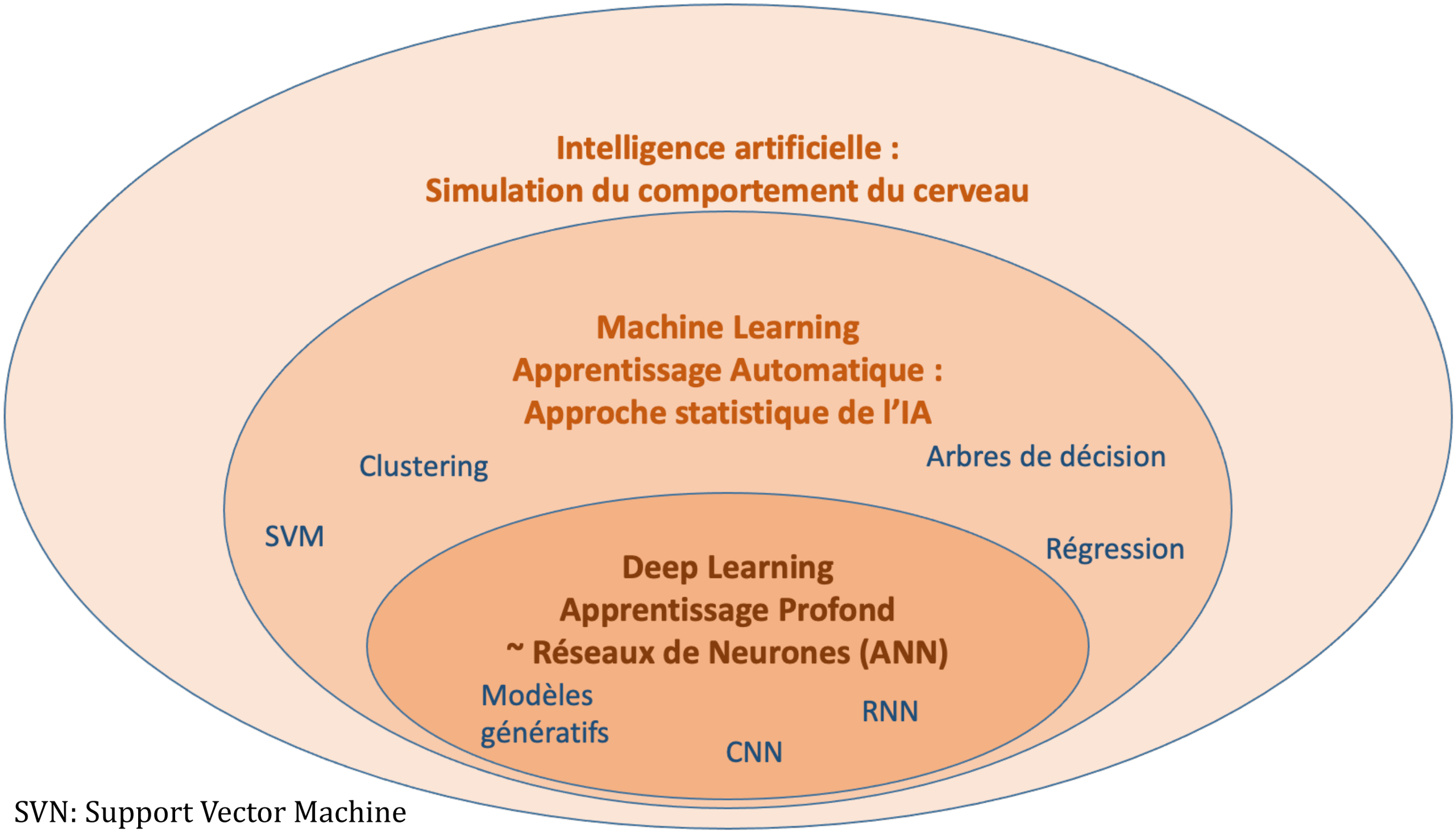
Principe et Applications du Machine Learning

Corentin Allaire

Remerciement à **David Rousseau** et **Françoise Bouvet** pour une partie des figures

Une question de nomenclature



SVN: Support Vector Machine
CNN: Convolution NN
RNN: Recurrent NN

À quoi s'attendre

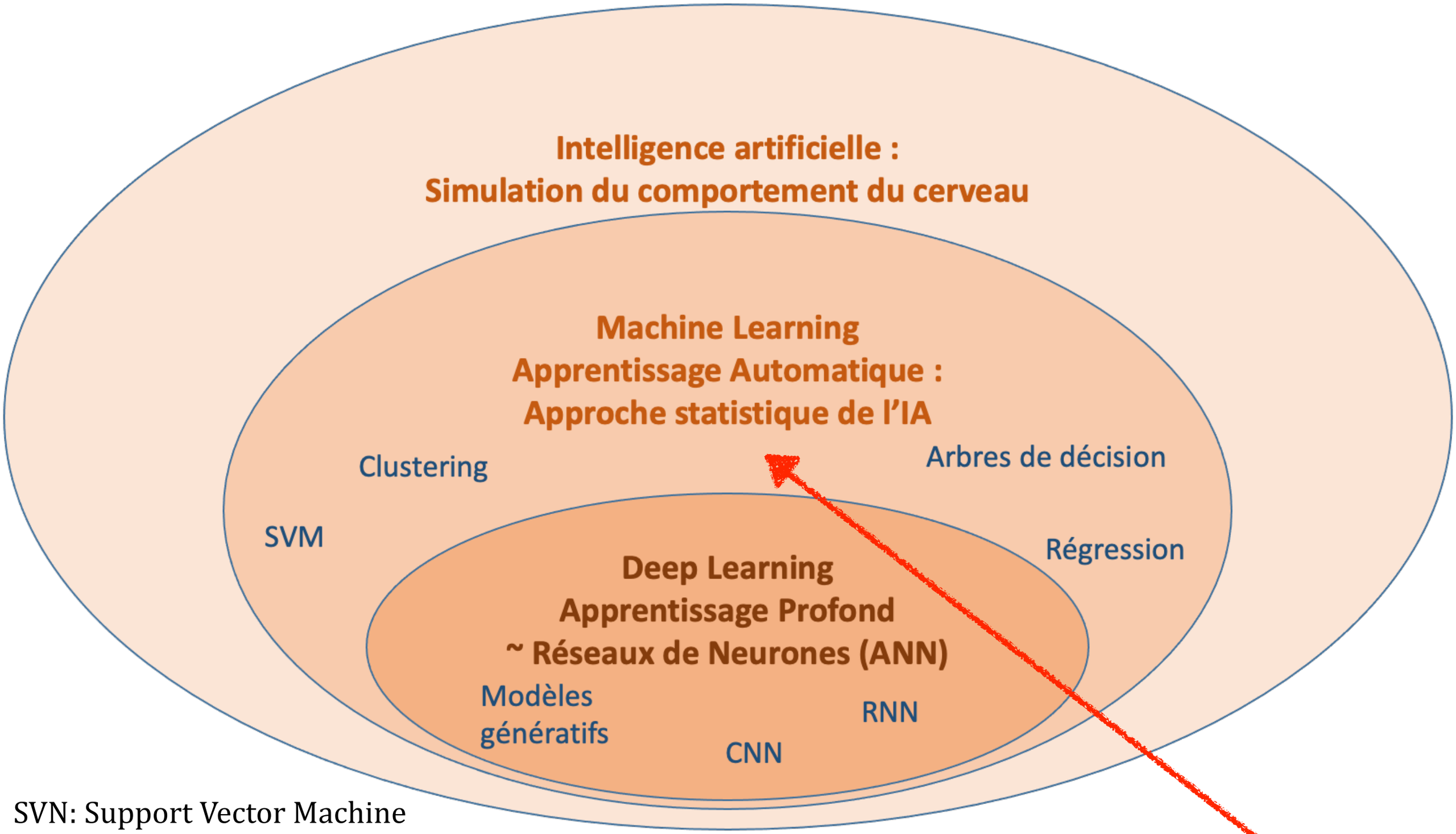
Machine Learning :

- Concepts fondamentaux de « L'IA »
- Les différentes techniques de Machine Learning :
  - Supervisé
  - Non-supervisé
- Application à la recherche du boson de Higgs

Deep Learning :

- Brève histoire du réseau de neurones
- Les différentes techniques de Deep Learning :
  - Perceptron multi-couche
  - Le réseau de convolution
  - Réseaux à base de graphe
- Application à la reconstruction d'objet en physique des hautes énergies

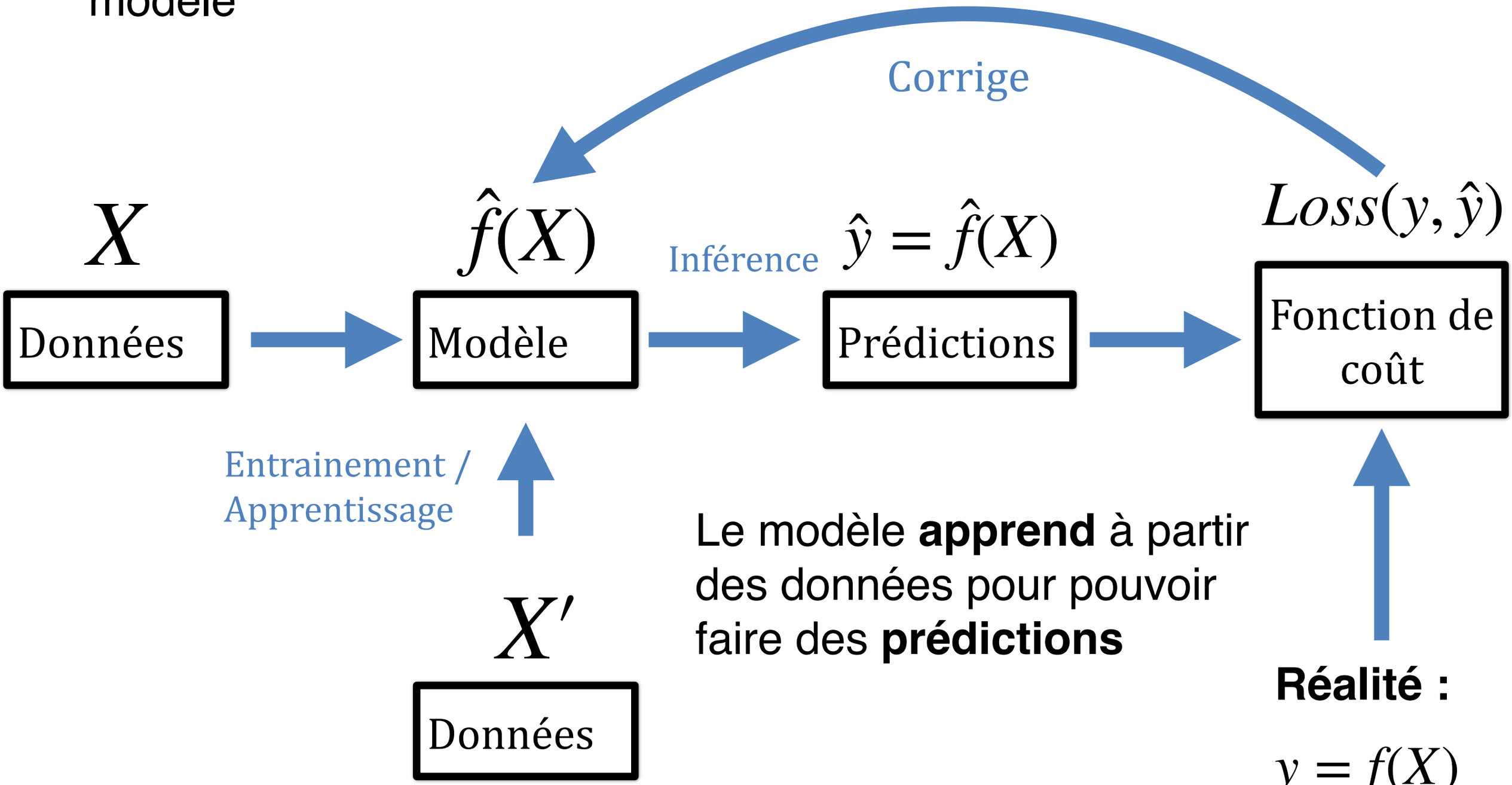
Une question de nomenclature



SVN: Support Vector Machine
CNN: Convolution NN
RNN: Recurrent NN

Le Machine Learning, qu'est ce que c'est ?

Machine Learning : Extraire de l'information de (grandes quantités de) **données** à l'aide d'un modèle



Différents types d'apprentissage possibles

Apprentissage supervisé :

- On dispose de donnée **étiquetées**
- On souhaite effectuer une **prédiction** sur de nouvelle donnée inconnues

Exemples :

- Classification ←
- Régression
- Classement
- Génération de texte, d'image

Chats

Chiens

?

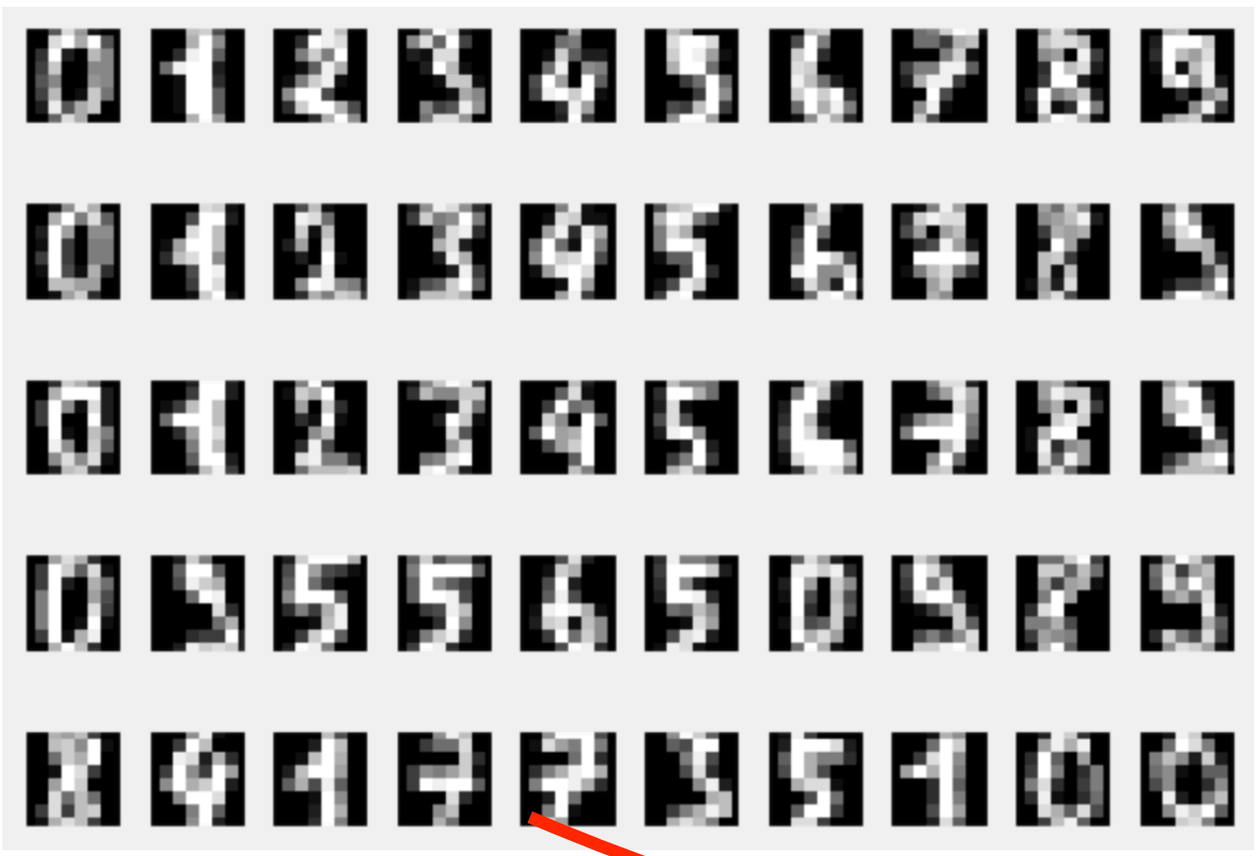
Différents types d'apprentissage possibles

Apprentissage non supervisé :

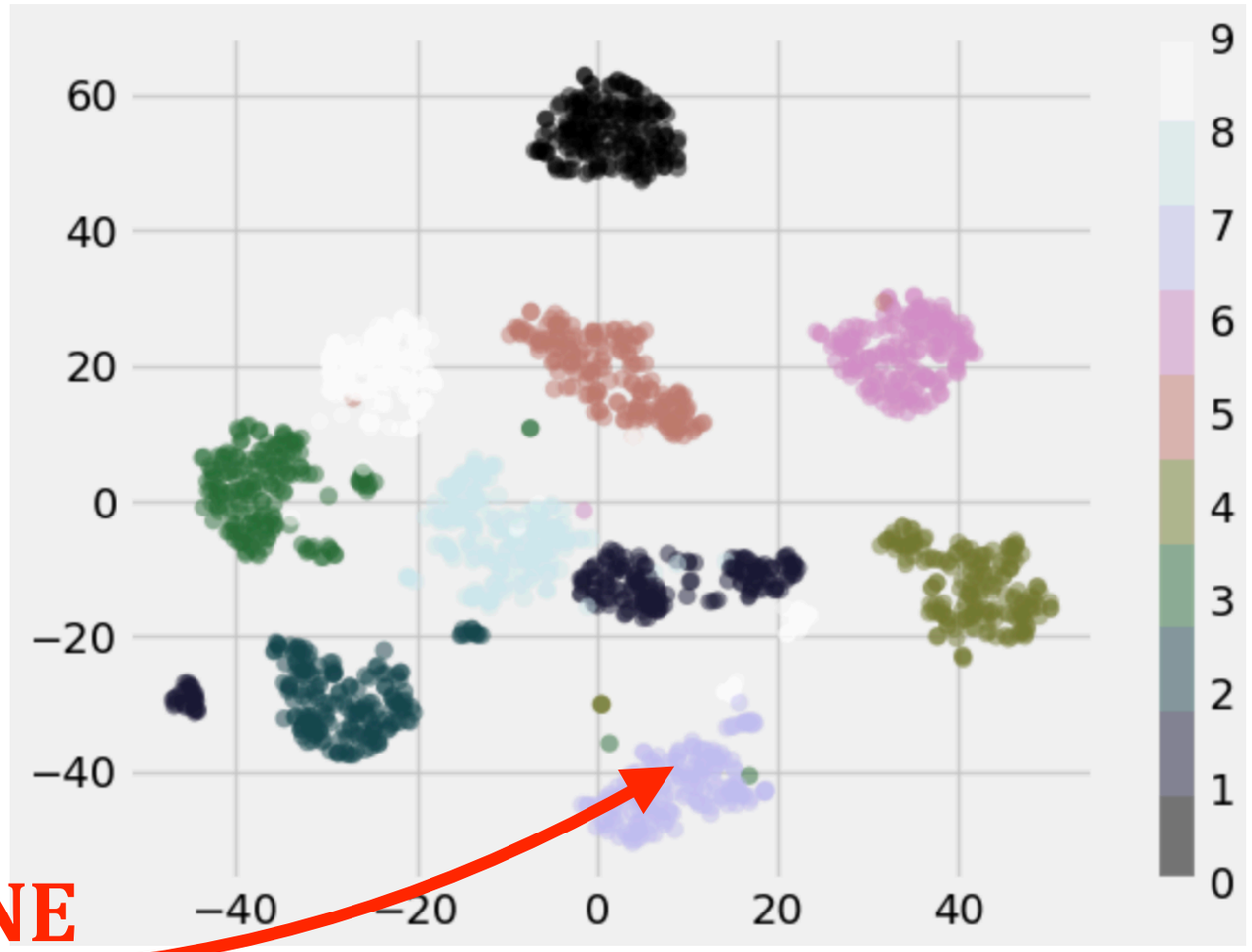
- On dispose de données **non-étiquetées**
- On souhaite obtenir de **nouvelles informations** sur ces données

Exemples :

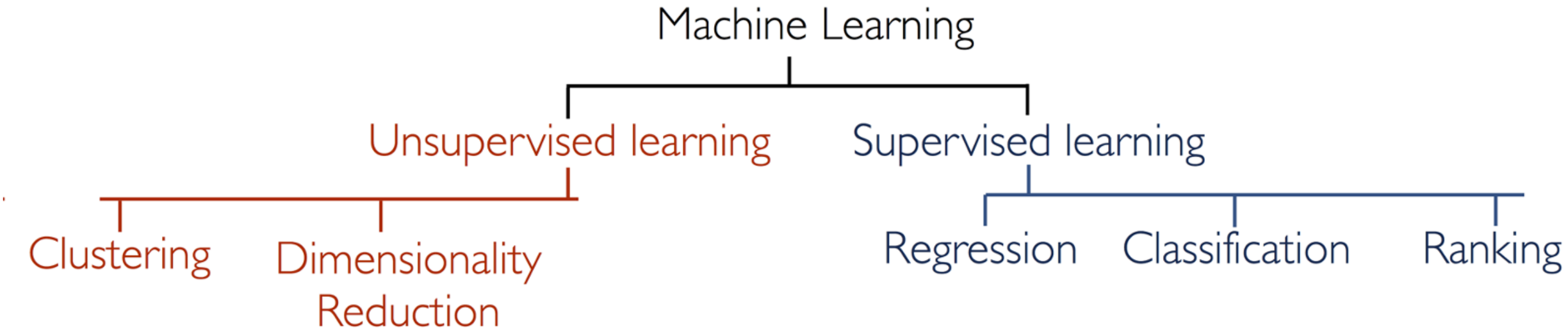
- Clustering
- Reduction de dimension ←
- Détection d'anomalie
- Génération de texte, d'image



t-SNE



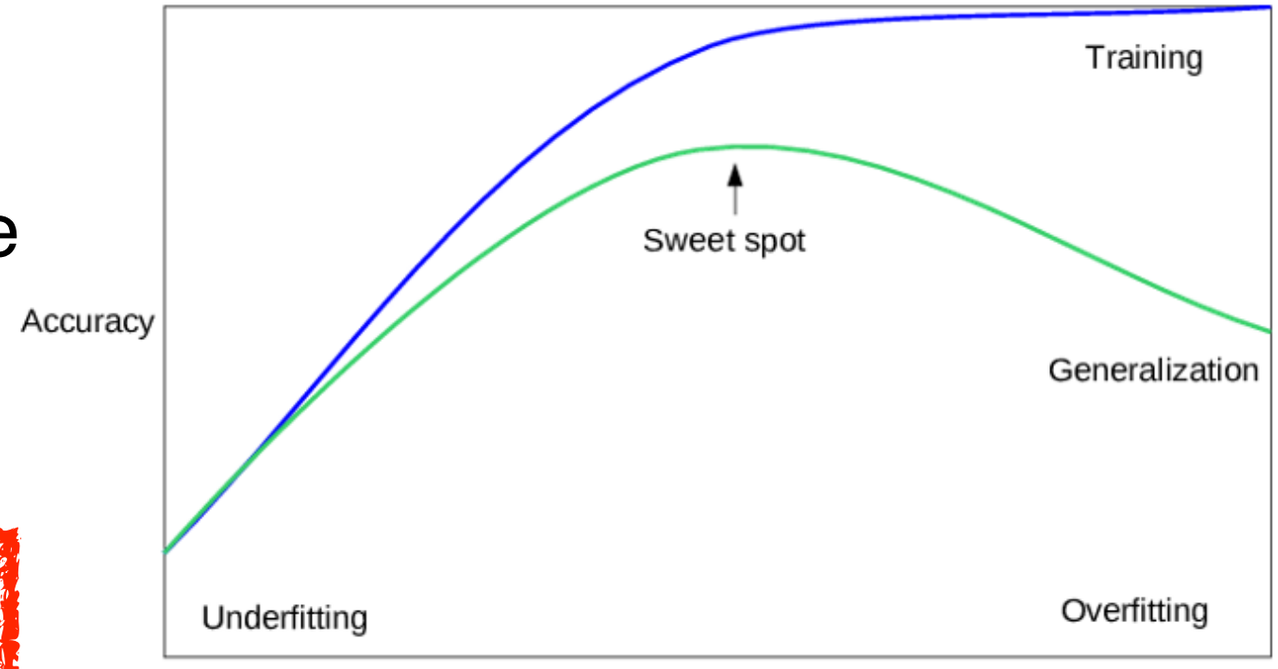
Différents types d'apprentissage possibles



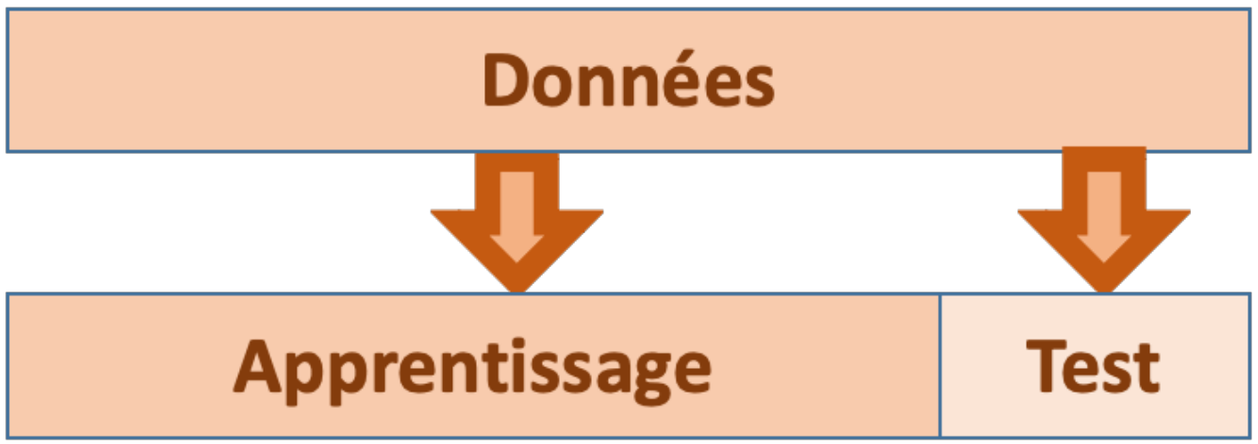
La première étape avant d'utiliser le ML est toujours **d'identifier** le type de problème auquel on est confronté pour déterminer quel type **d'algorithme** utiliser

La gestion des données

Le **succès** d'un algorithme de ML repose sur la bonne utilisation de ses données **d'entraînement** et de **test**



Entrainement supervisé : Ne **<u>jamais tester</u>** un modèle avec les données **d'apprentissage**



Overfitting : Le modèle apprend **par cœur** les données d'entraînement, mais ne sait pas généraliser

L’évaluation du modèle

Un critère d’évaluation est indispensable pour répondre à la question : **quel est le meilleur modèle ?**

Le critère dépendra du problème :

- **Régression** : Erreur aux moindres carrés

$$\mathrm{MSE}(y,\hat{y}) = \frac{1}{N_{samples}} \sum_{i=0}^{N_{samples}-1} (y_i - \hat{y}_i)^2$$

- **Classification** : Précision, Sensibilité, Spécificité, Aire sous la courbe…

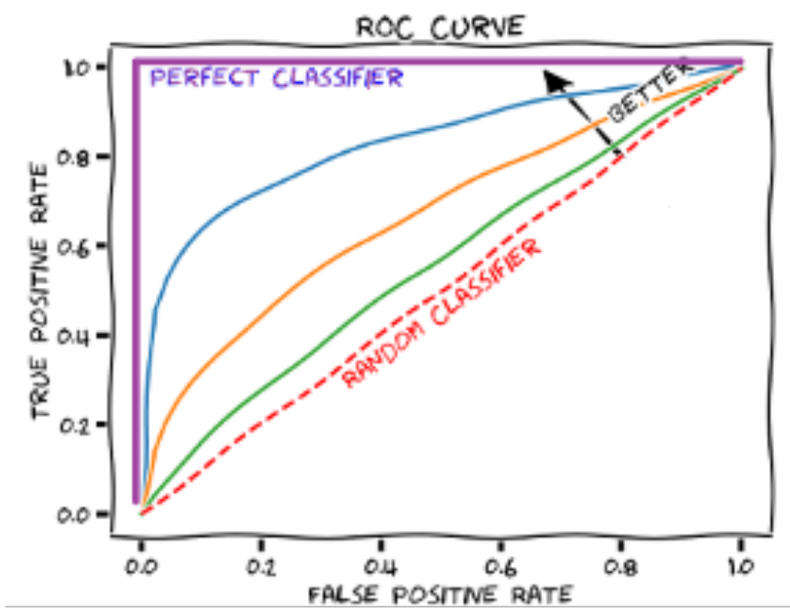
Prédiction

Réalité	FAUX	VRAI
FAUX	VN	FP
VRAI	FN	VP

$$Precision = \frac{VP}{VP + FP}$$

$$Sensibilite = \frac{VP}{VP + FN}$$

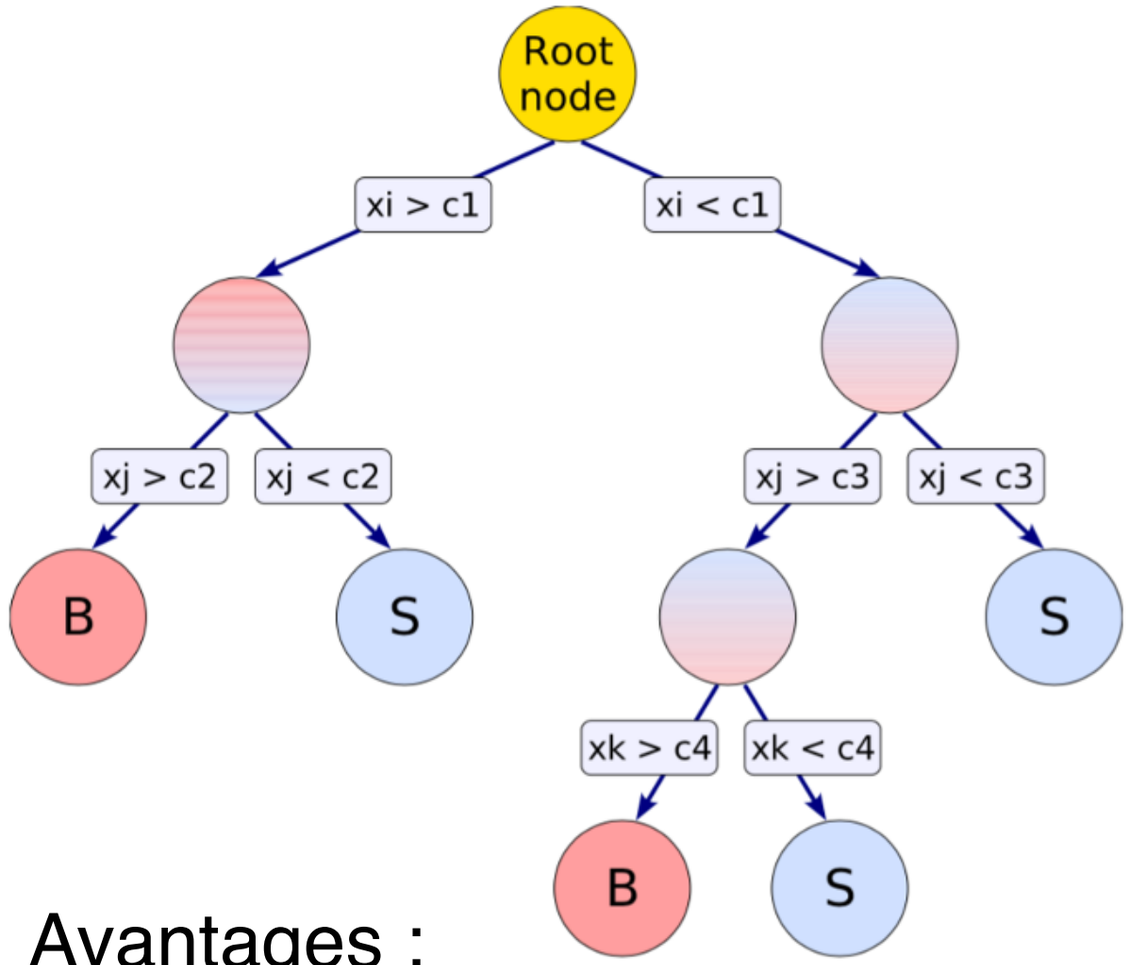
$$Specificite = \frac{VN}{VN + FP}$$



Quelque exemples : Supervisé

Classification : **Arbre de décision**

- Outil central de la physique des particules depuis les années 90
- Classifie des objets décris par N **variables** x_i



Construction de l'arbre :

- **Coupures** successives sur les variables pour maximiser la « pureté » des nœuds suivants
- Répète l'opération jusqu'à atteindre une pureté (ou un nombre de nœuds) cible

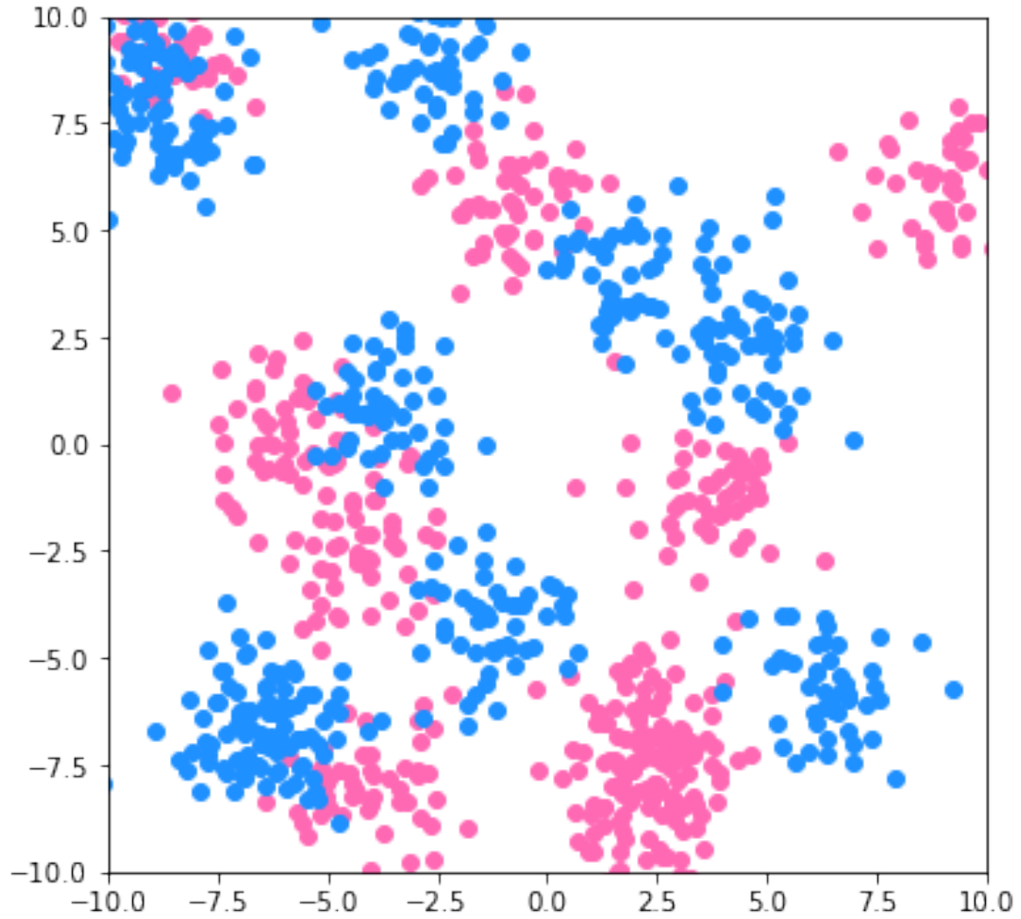
Avantages :

- Simple à **interpréter**
- Demande peu de données

- Désavantages :
  - Risque d'overfit
  - Création de l'arbre instable

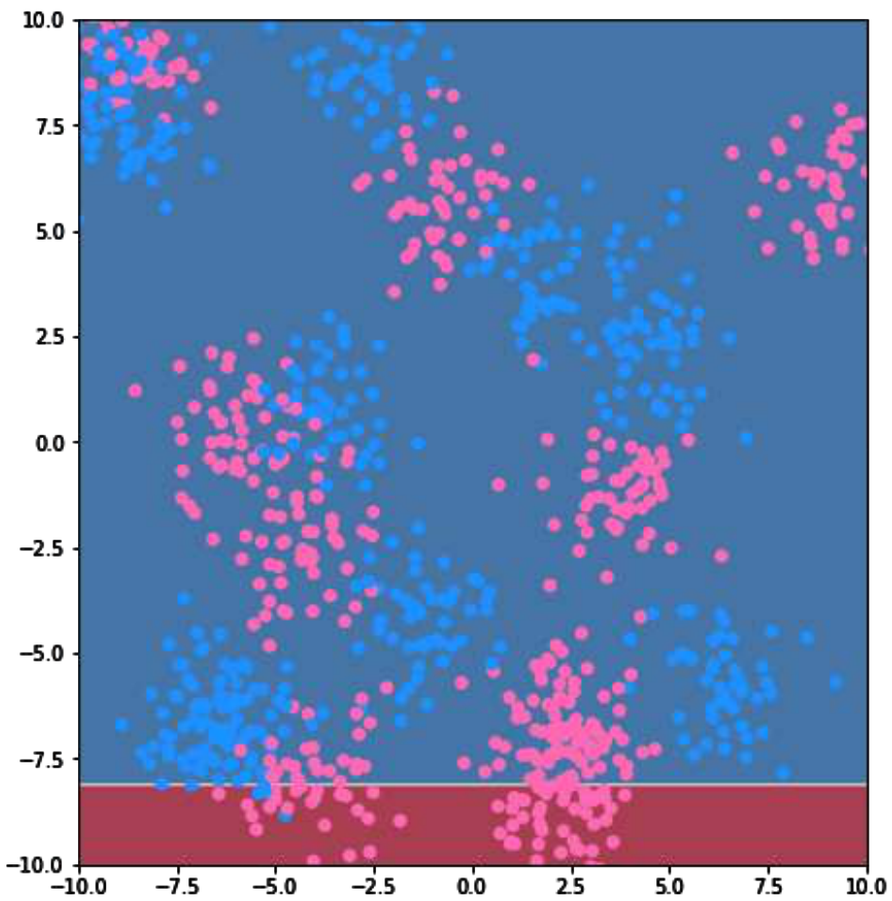
Quelque exemples : Supervisé

Classification : **Arbre de décision**



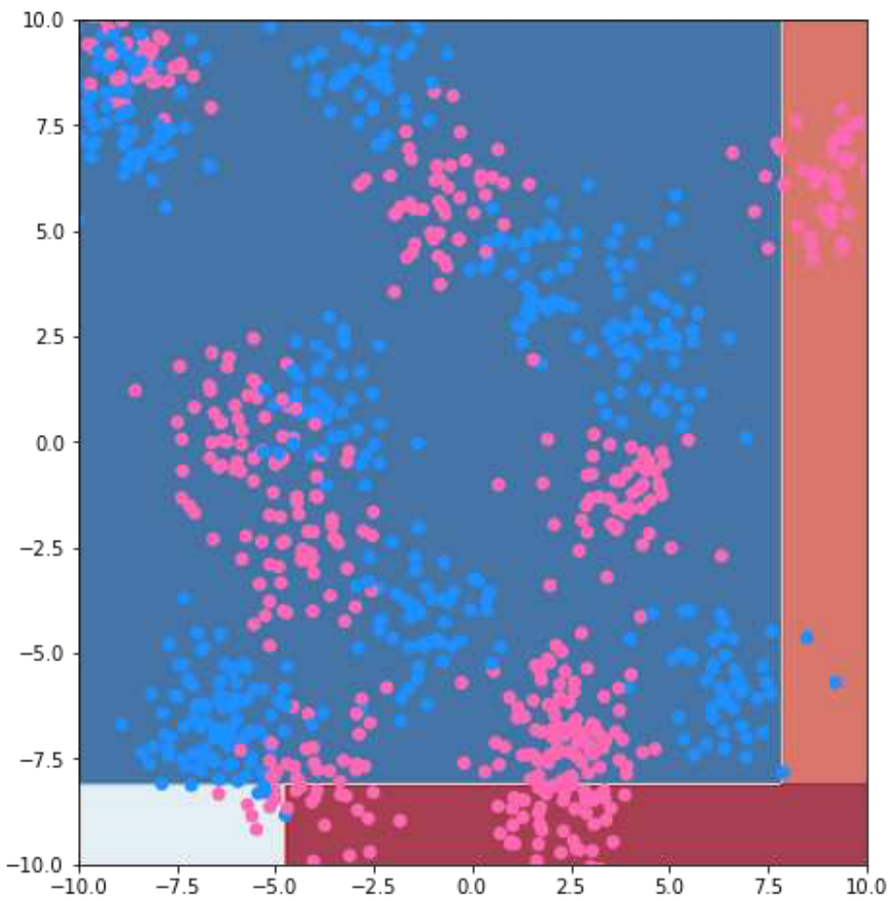
Quelque exemples : Supervisé

Classification : **Arbre de décision**



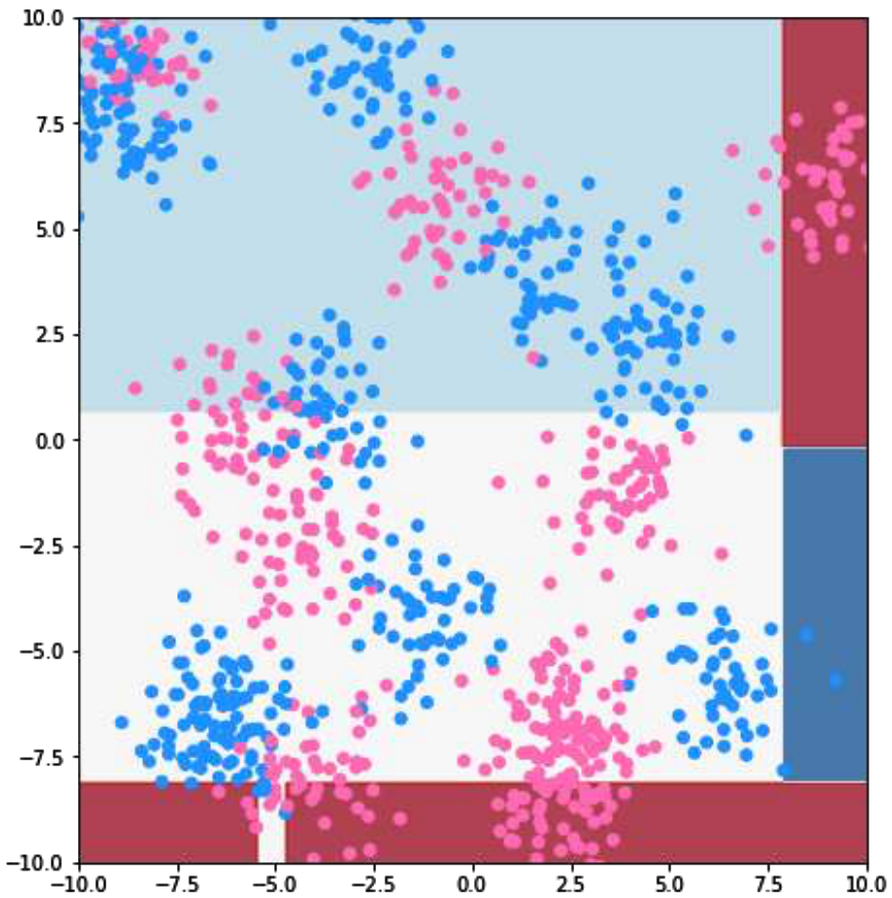
Quelque exemples : Supervisé

Classification : **Arbre de décision**



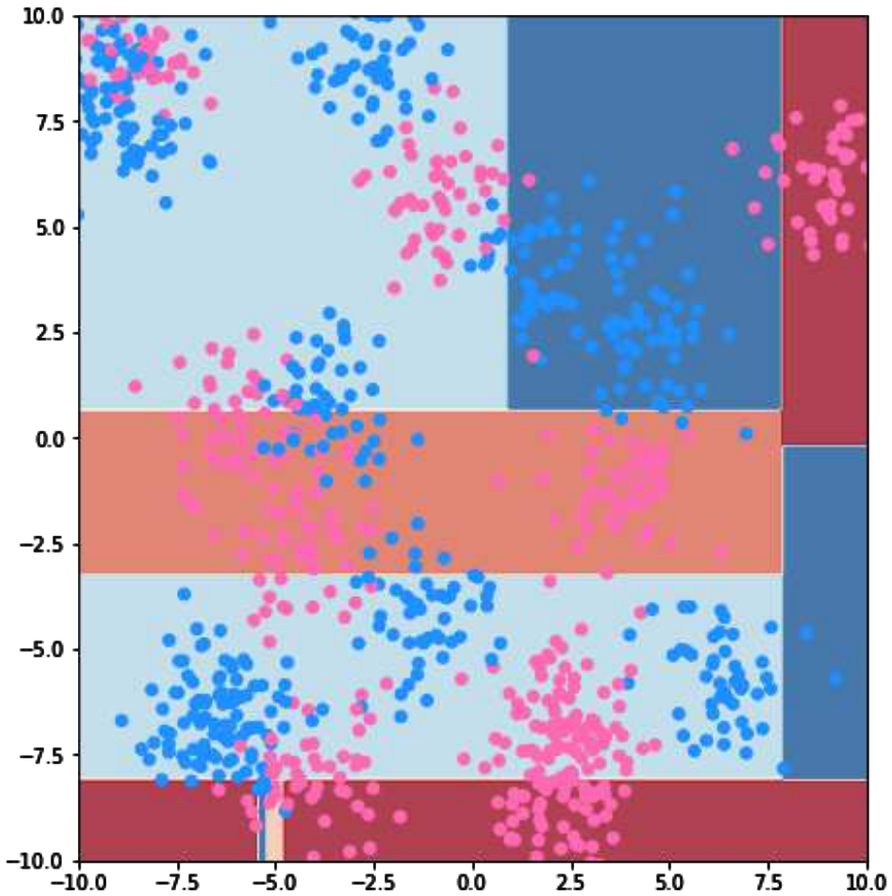
Quelque exemples : Supervisé

Classification : **Arbre de décision**



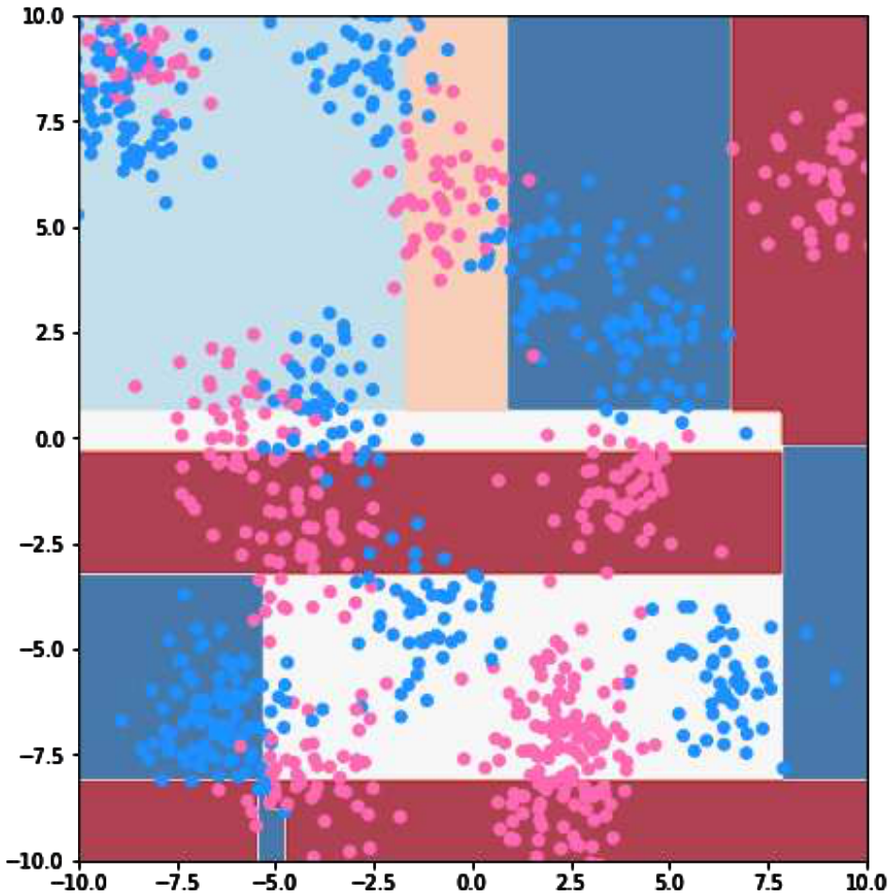
Quelque exemples : Supervisé

Classification : **Arbre de décision**



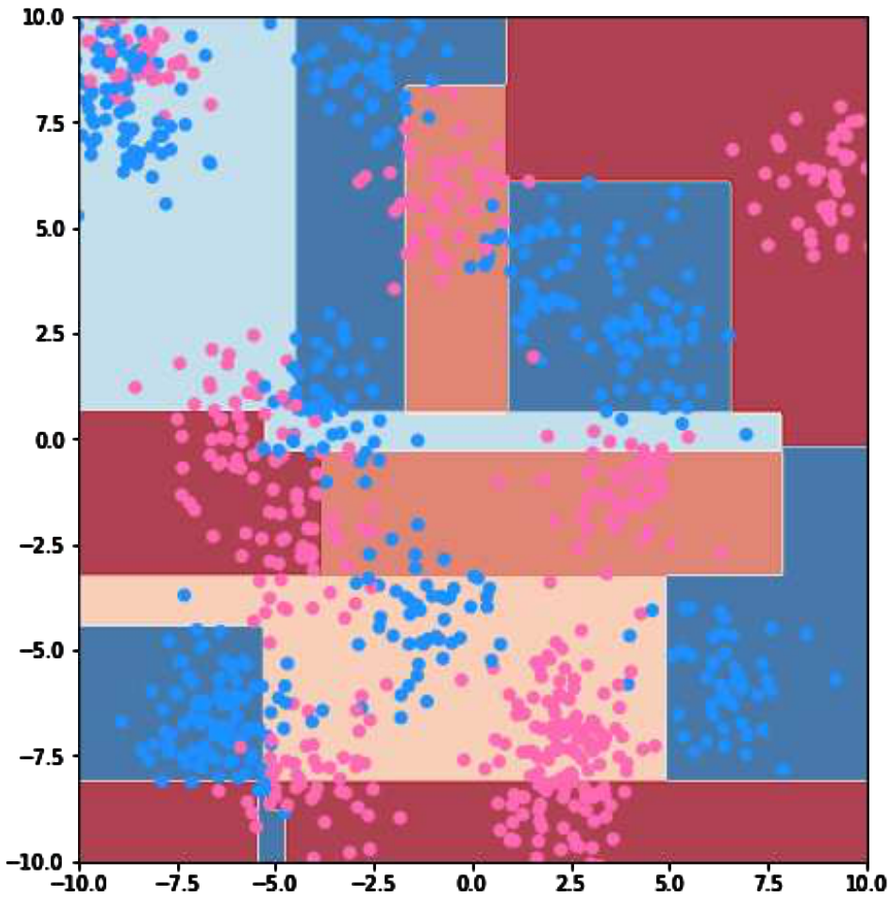
Quelque exemples : Supervisé

Classification : **Arbre de décision**



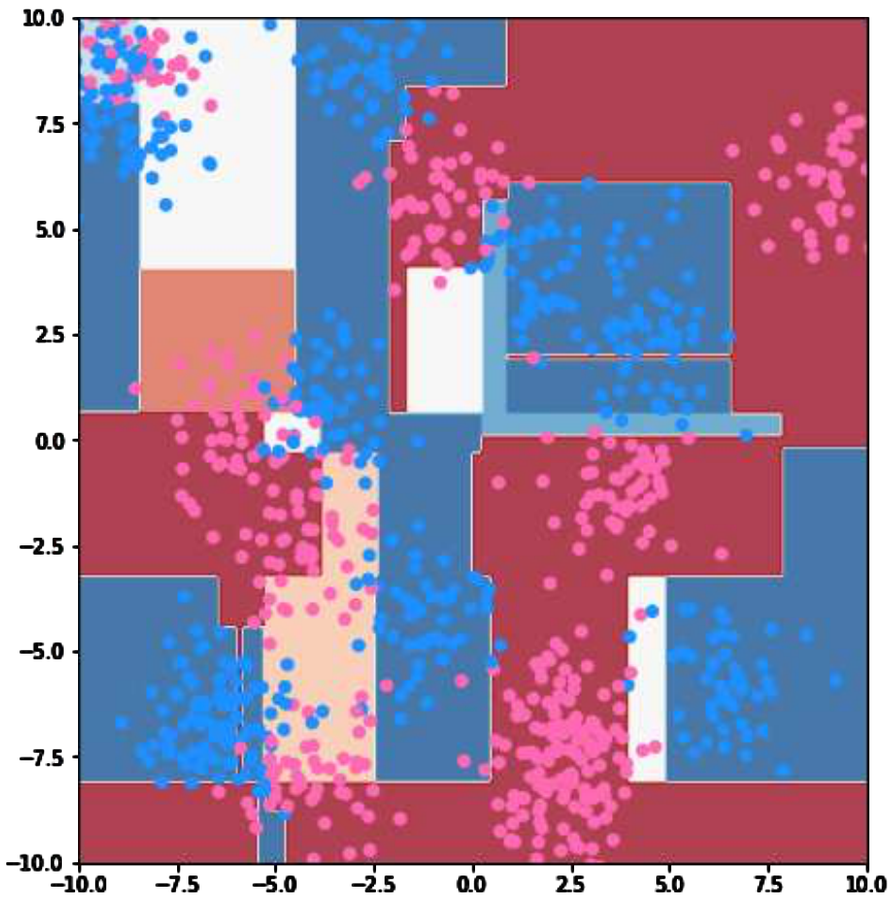
Quelque exemples : Supervisé

Classification : **Arbre de décision**



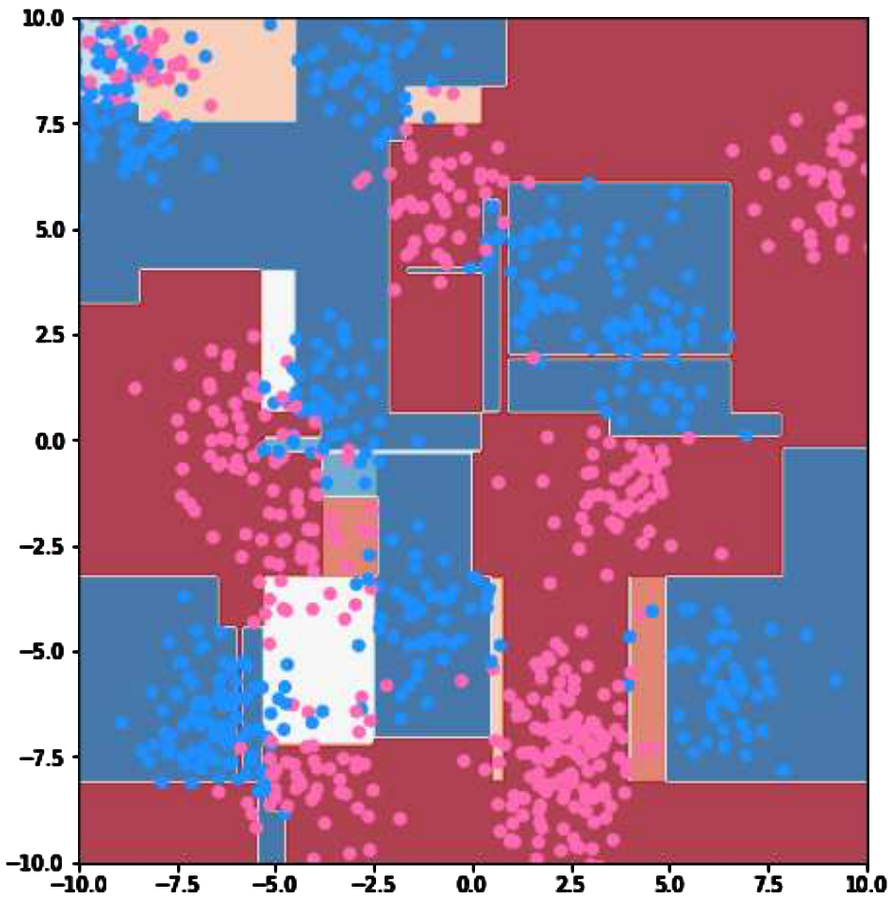
Quelque exemples : Supervisé

Classification : **Arbre de décision**



Quelque exemples : Supervisé

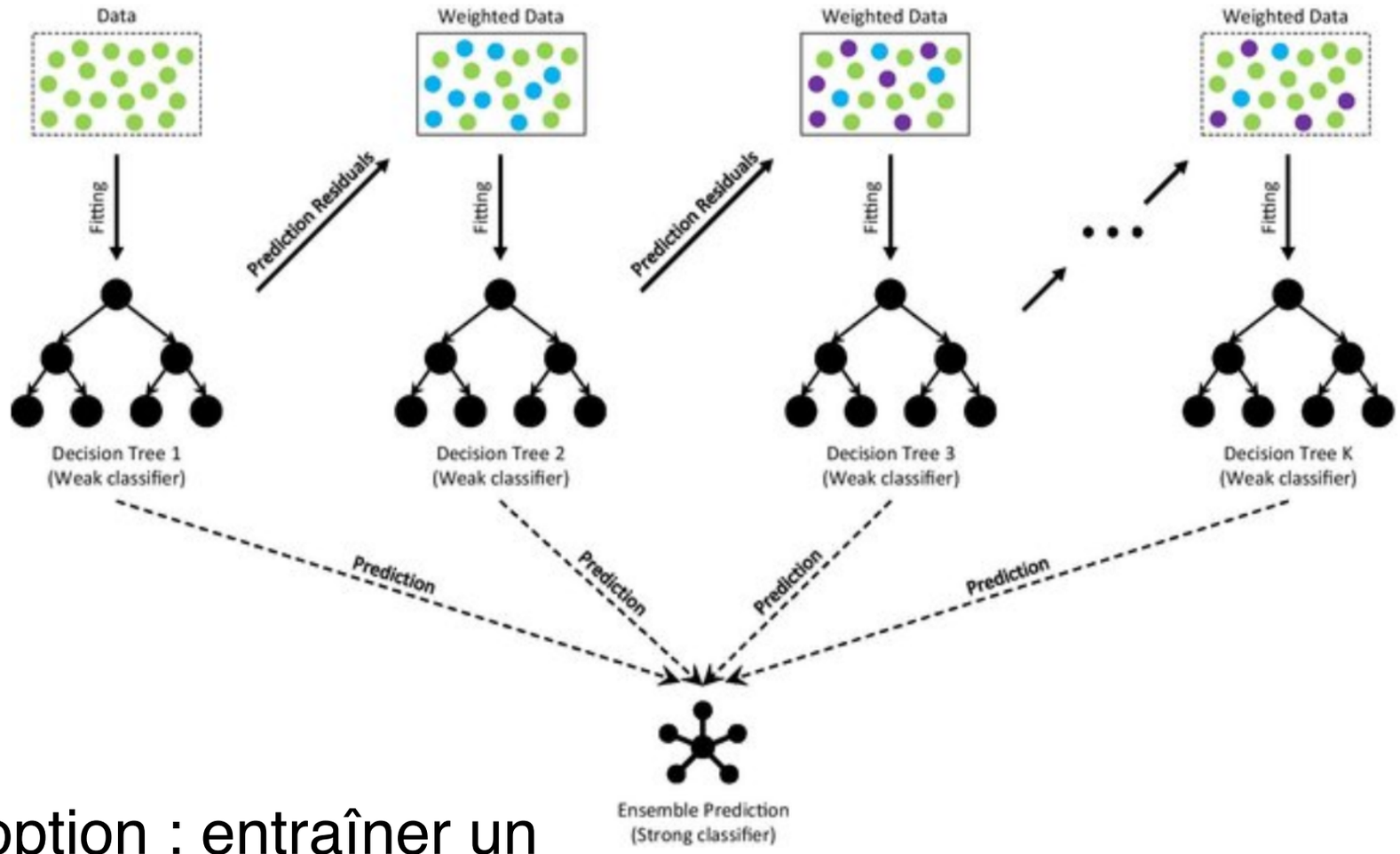
Classification : **Arbre de décision**



Quelque exemples : Supervisé

Classification : **BDT** (boosted decision tree)

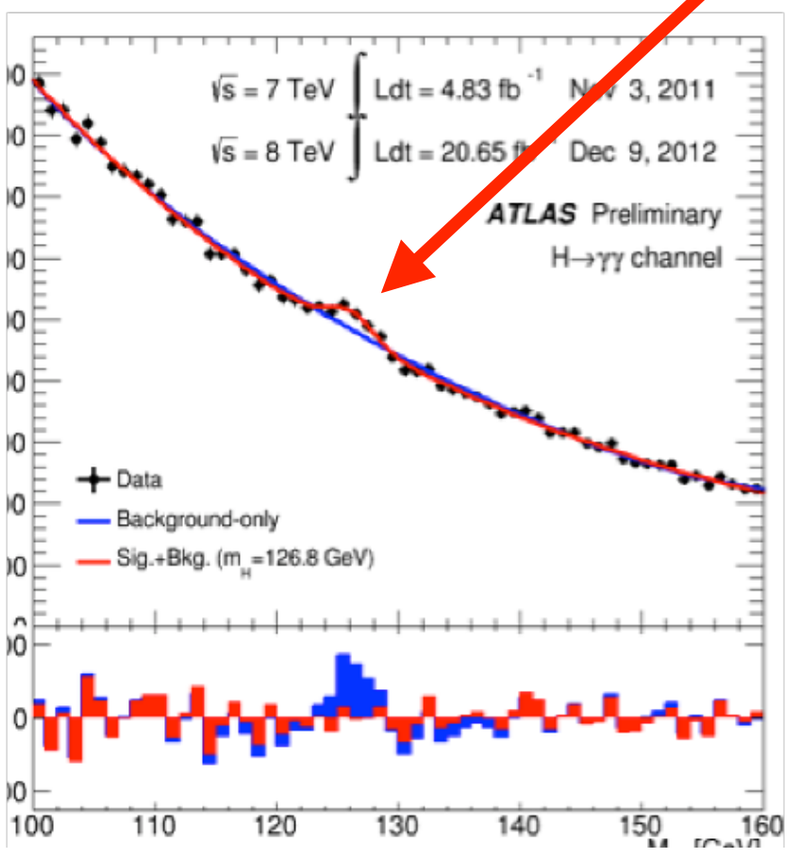
- Individuellement, les arbres peuvent **overfit** et sont très **instables**
- Utilise une **série d'arbres** pour optimiser la précision du système
- Solution idéale pour les problèmes à moins de 100 variables

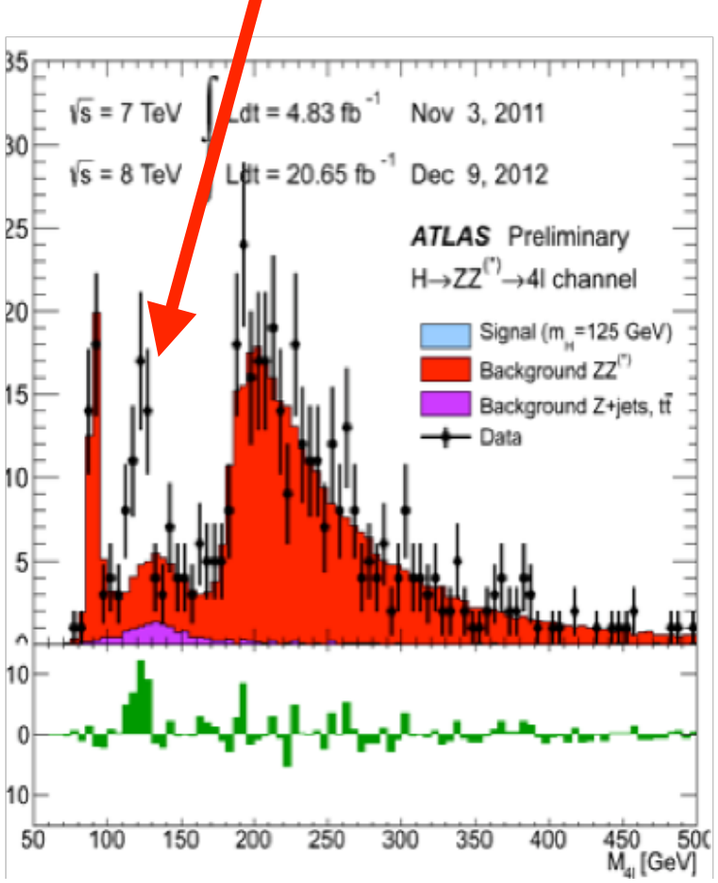


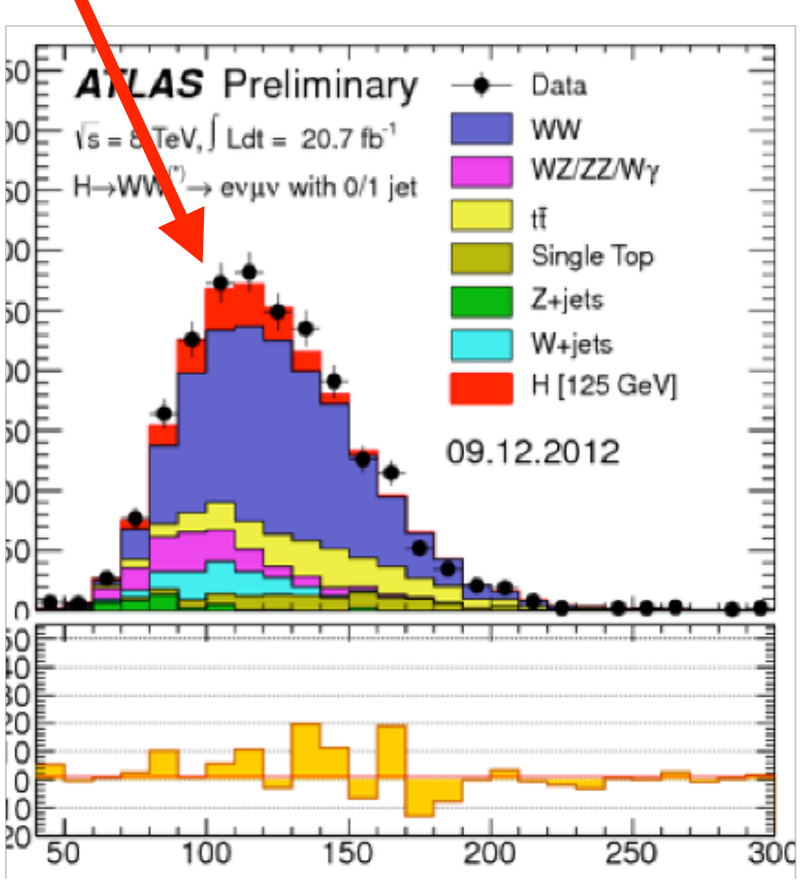
Autre option : entraîner un grand nombre d'arbres et effectuer un **vote**

Application à la recherche du Higgs

- Observation du boson de Higgs : Comparaisons de la **distribution** (histogramme) d'une variable d'intérêt observé au LHC (**signal**) avec sa valeur attendue en absence de Higgs (**bruit de fond**)
- La présence d'un **excès** dénote l'existence d'une nouvelle particule

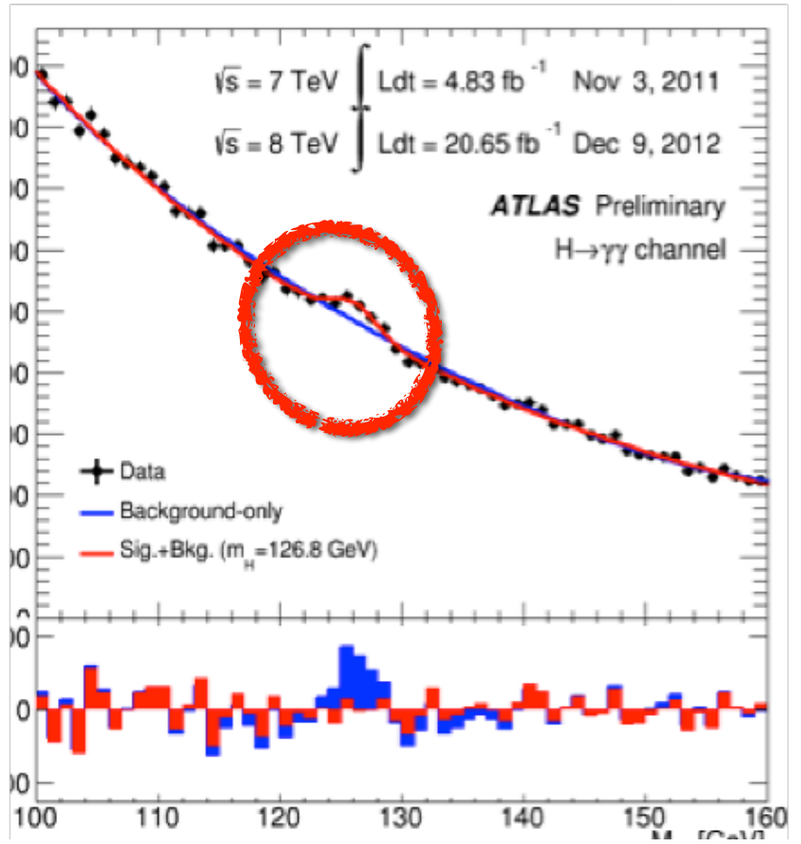


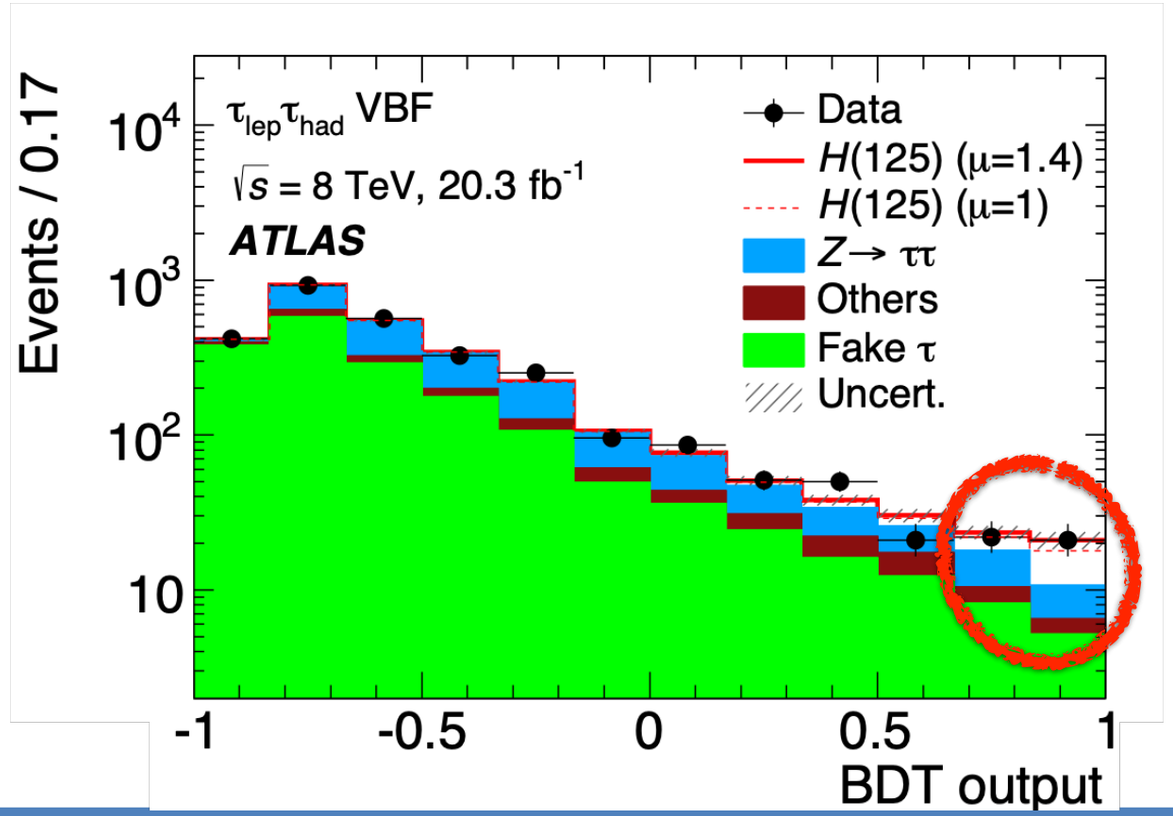




Choix de la variable d'intérêt

- Option évidente : **masse résonante** de production du **Higgs**
- « Facile » à reconstruire à partir de la cinématique des particules produite
- Nous donne une information sur les **propriétés** de la particule reconstruite





- Autre option : score de classification **BDT** des événements
- Le BDT essaie de séparer Higgs / non Higgs
- Idéale pour les **signatures complexes**
- Tire parti de plus de paramètres de l'événement

Quelques exemples : non supervisé

Reduction de dimension: **t-SNE** (t-distributed stochastic neighbour embedding)

Extension des algorithmes de SNE :

- Dans **P** : calcule la probabilité que i et j sois choisis comme **voisin** (choix de voisin basé sur une distribution gaussienne)

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)},$$

Quelques exemples : non supervisé

Reduction de dimension: **t-SNE** (t-distributed stochastic neighbour embedding)

Extension des algorithmes de SNE :

- Dans **P** : calcule la probabilité que i et j sois choisis comme **voisin** (choix de voisin basé sur une distribution gaussienne)
- Même chose dans **Q**

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}}$$

Quelques exemples : non supervisé

Reduction de dimension: **t-SNE** (t-distributed stochastic neighbour embedding)

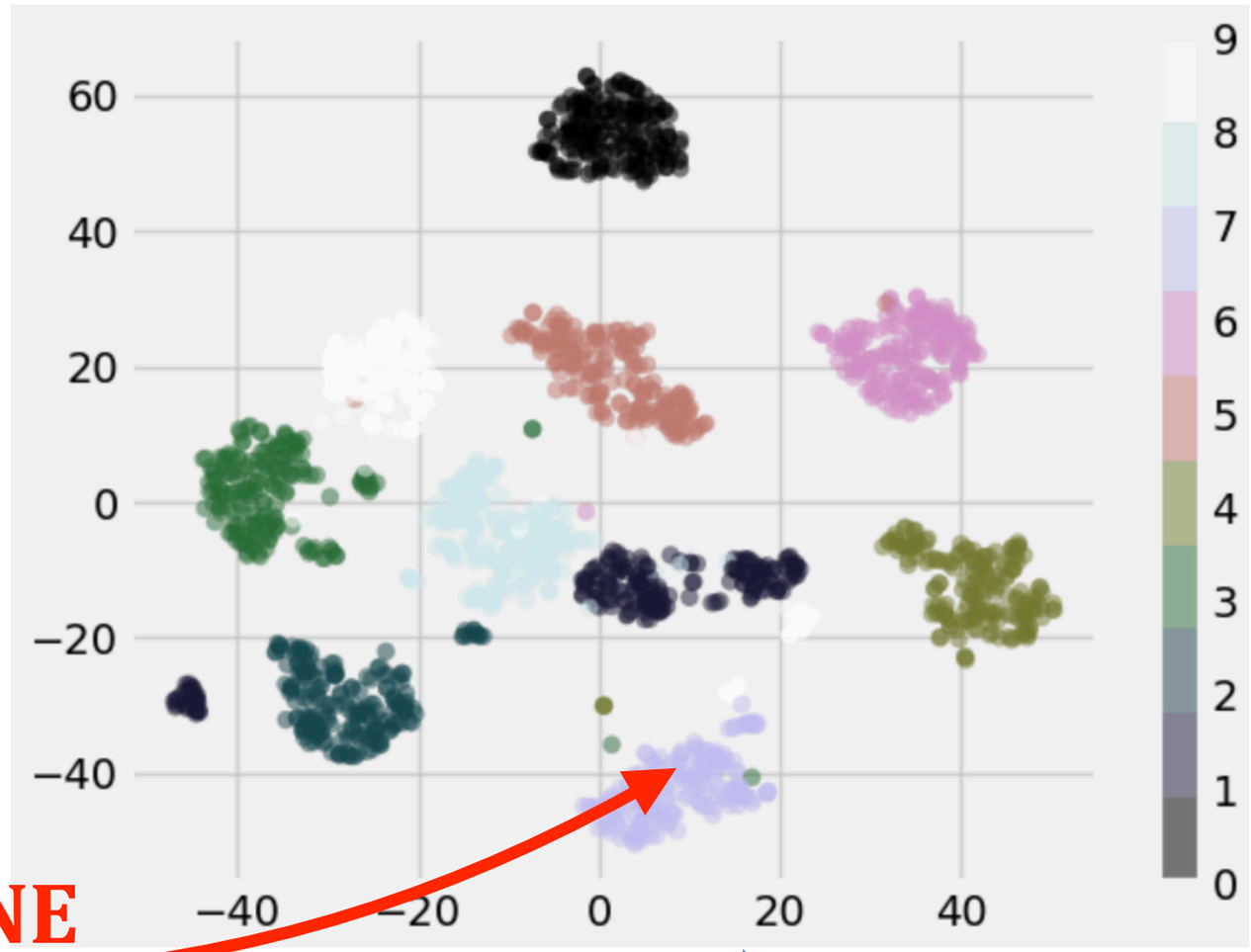
Extension des algorithmes de SNE :

- Dans **P** : calcule la probabilité que i et j sois choisis comme **voisin** (choix de voisin basé sur une distribution gaussienne)
- Même chose dans **Q**
- Si Q est une bonne représentation de P, alors $\frac{p_{j|i}}{q_{j|i}}$ tends vers 1
- On trouve les positions dans Q par **descente de gradient**

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

Quelques exemples : non supervisé

Exemple avec des chiffres



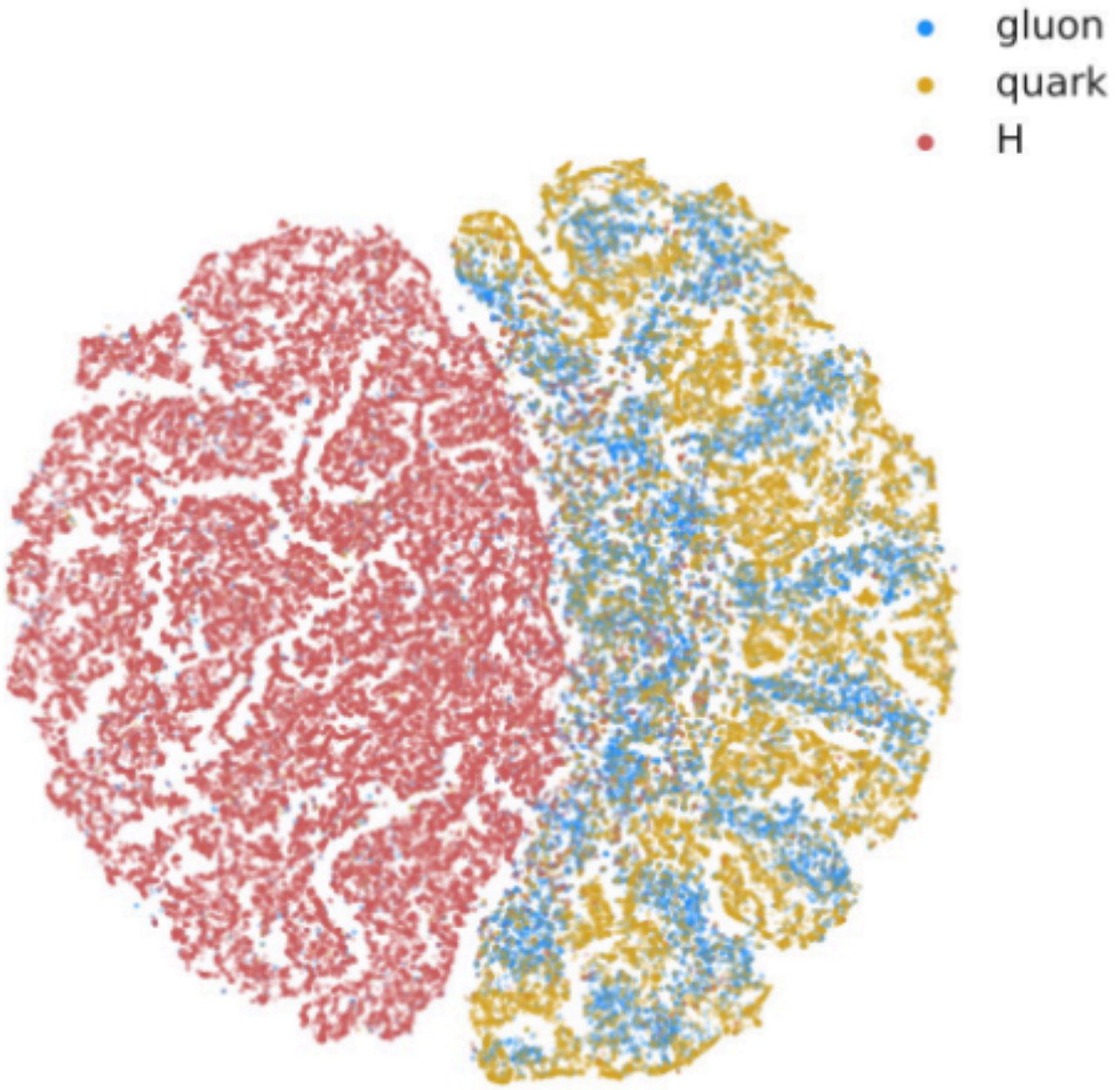
t-SNE

Image 8*8 = 64D

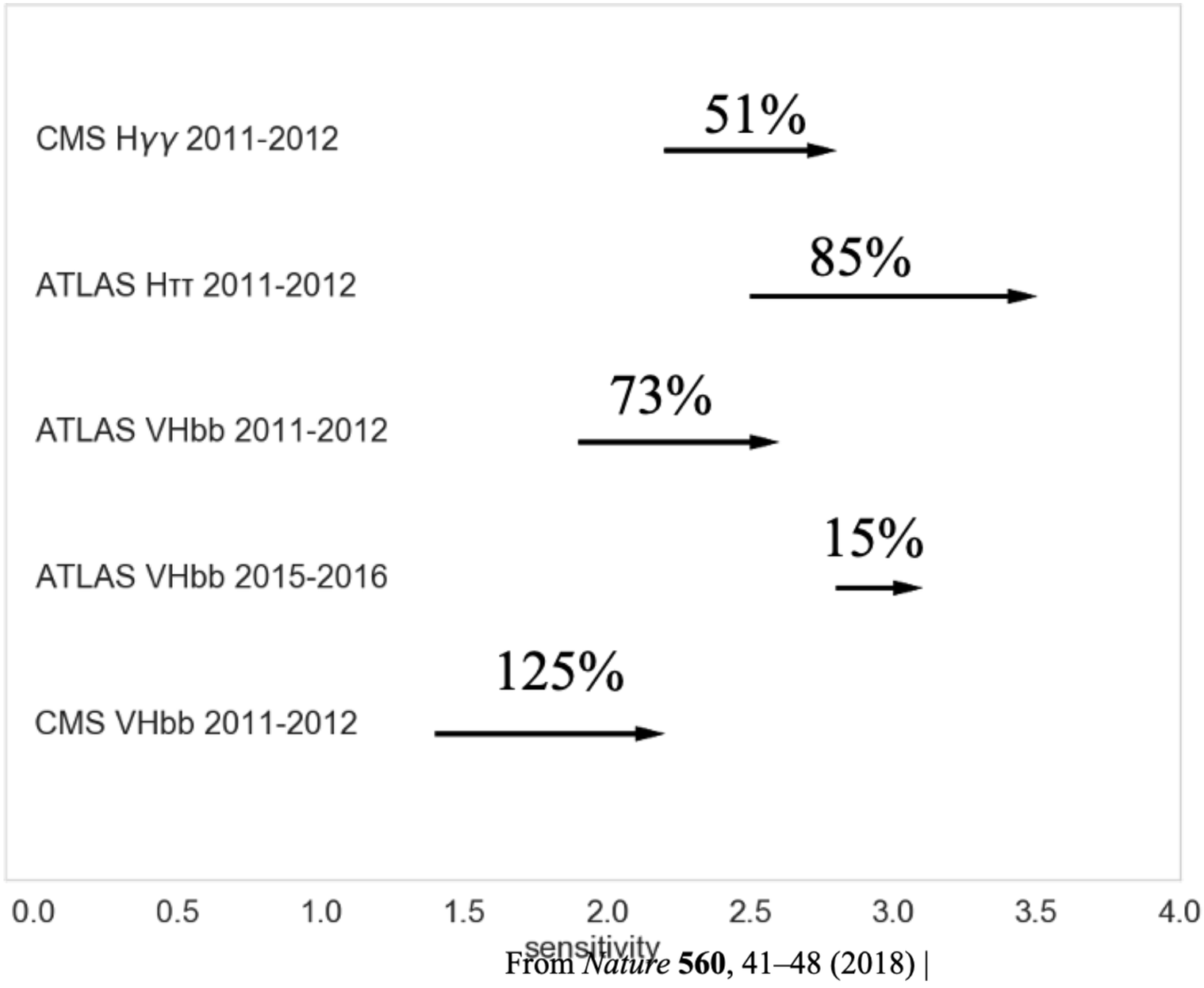
2D plot

t-SNE pour l'explicabilité des réseaux de neurones

- Les **réseaux de neurones** (cf demain) sont aussi utilisés pour la classification des **particules**
- Beaucoup plus complexe (projection en dimension > 100)
- L'utilisation d'algorithme de **t-SNE** nous permet de « **visualiser** » partiellement le fonctionnement de notre algorithme
- Visualisation de **l'encodage** de jet de particules en fonction de leurs sources



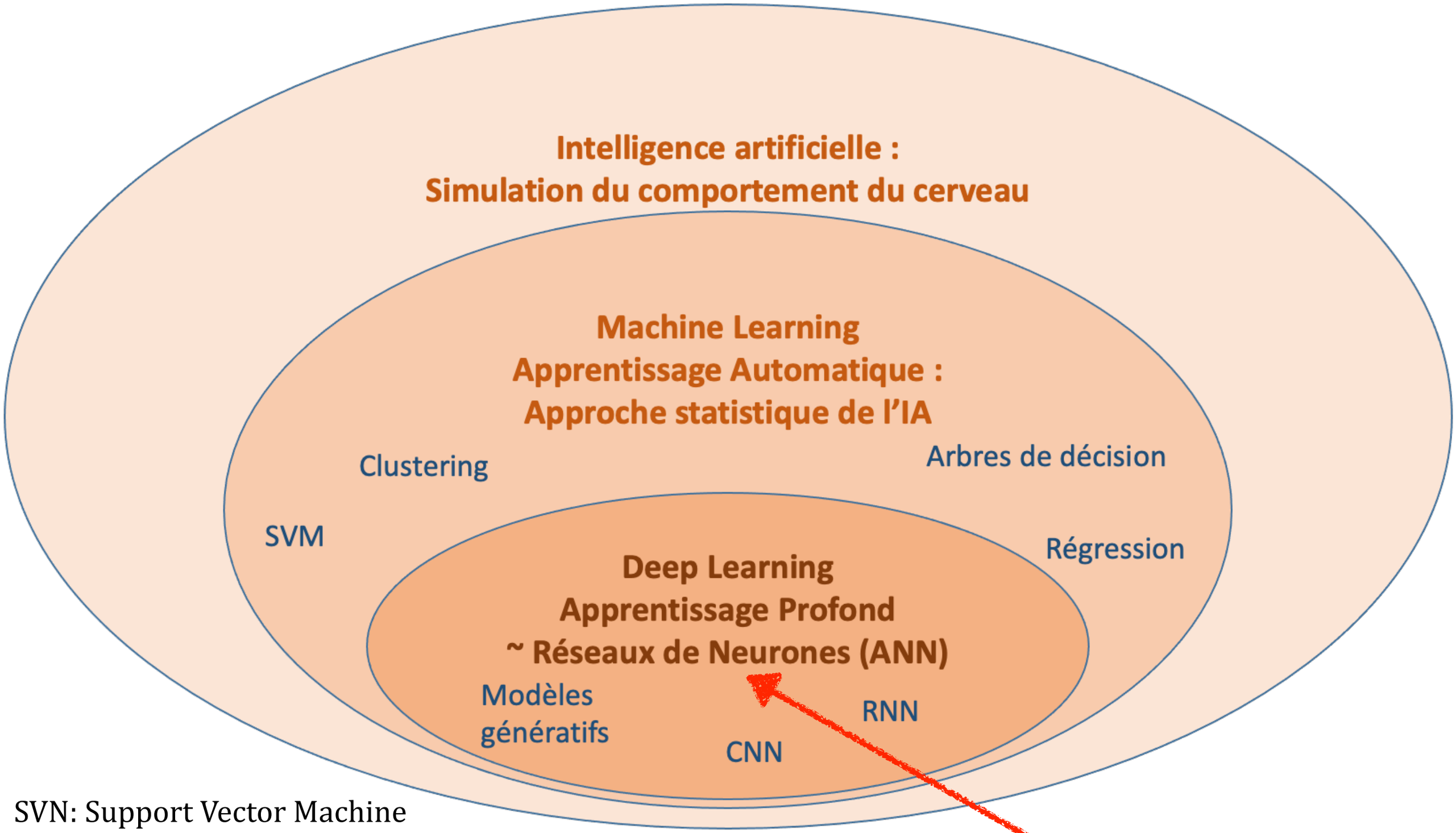
Pourquoi utiliser du ML ?



From *Nature* **560**, 41–48 (2018) |

- Principalement des **BDT** à **~10 variables**
- En moyenne, l'utilisation de BTD nous permet d'atteindre des **sensibilités** à l'existence du Higgs **~50%** plus élevés
- Maximisation de notre usage du LHC

Une question de nomenclature

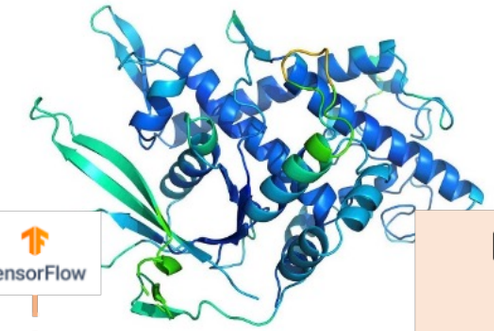


SVN: Support Vector Machine
CNN: Convolution NN
RNN: Recurrent NN

Une brève histoire du Deep Learning

Perceptron

Deep Blue



AlphaFold2

Diffusion models

AlexNet

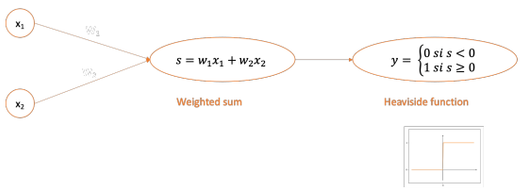
TensorFlow

PyTorch

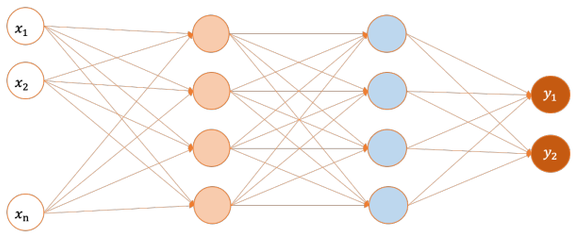
Expert systems

1943 | 1957 | *First winter* | 1980 | 1986 | *Second winter* | 1997 | 2011 | 2012 | ..17 | ..18 | ..20 | ..21 | ..22

Formal neuron



Multi-Layer Perceptron
Gradient Back Propagation

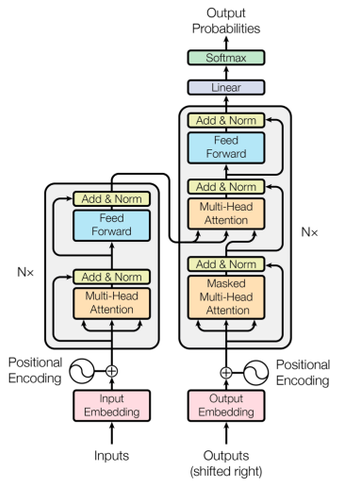


- GoogleBrain
- Watson
- Siri

AlphaGo (DeepMind)

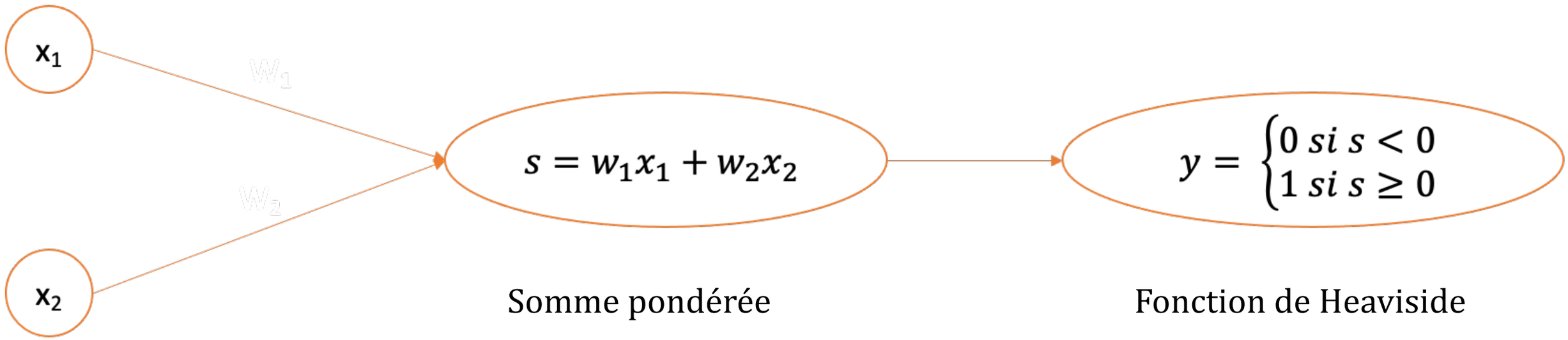
GPT3

Transformers

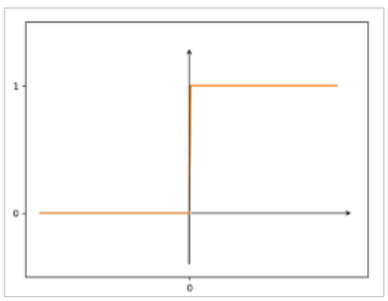


Le neurone formel

Idée : Crée une construction mathématique qui « simule » les **neurones biologiques**

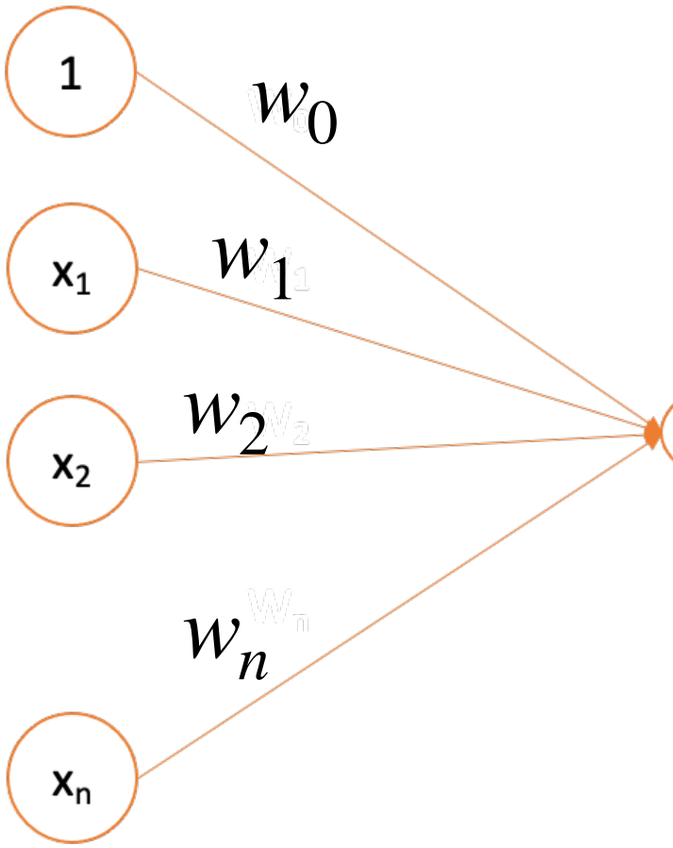


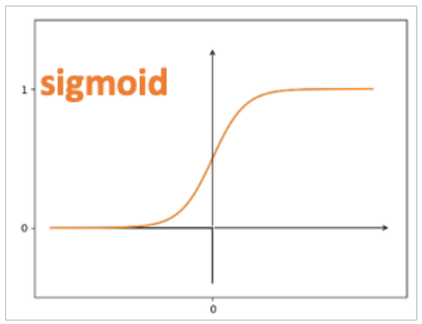
1943 : McCulloch & Pitts

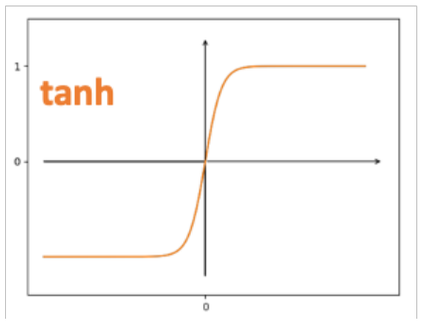


- Retourne uniquement des entrée **booléennes** (0,1)
- Assemblage de neurone : **Turing complets**

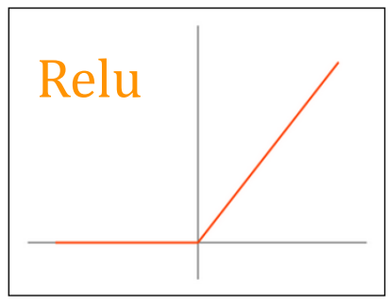
Le neurone formel : généralisation



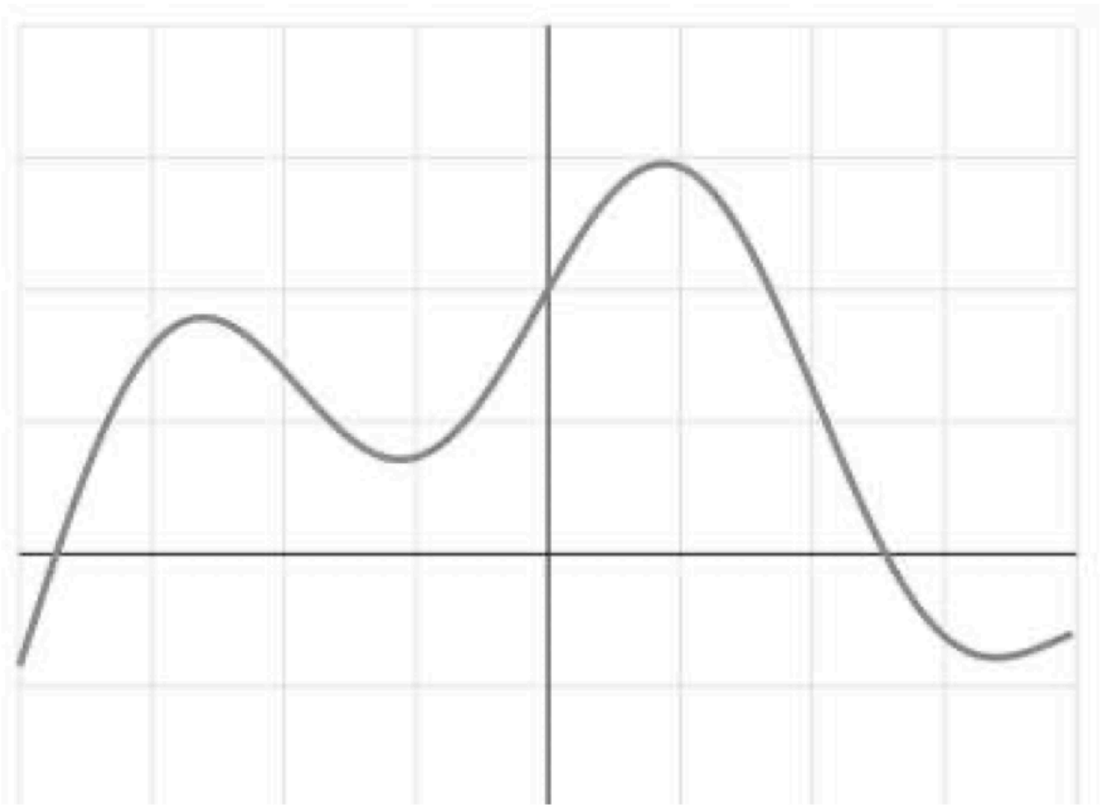


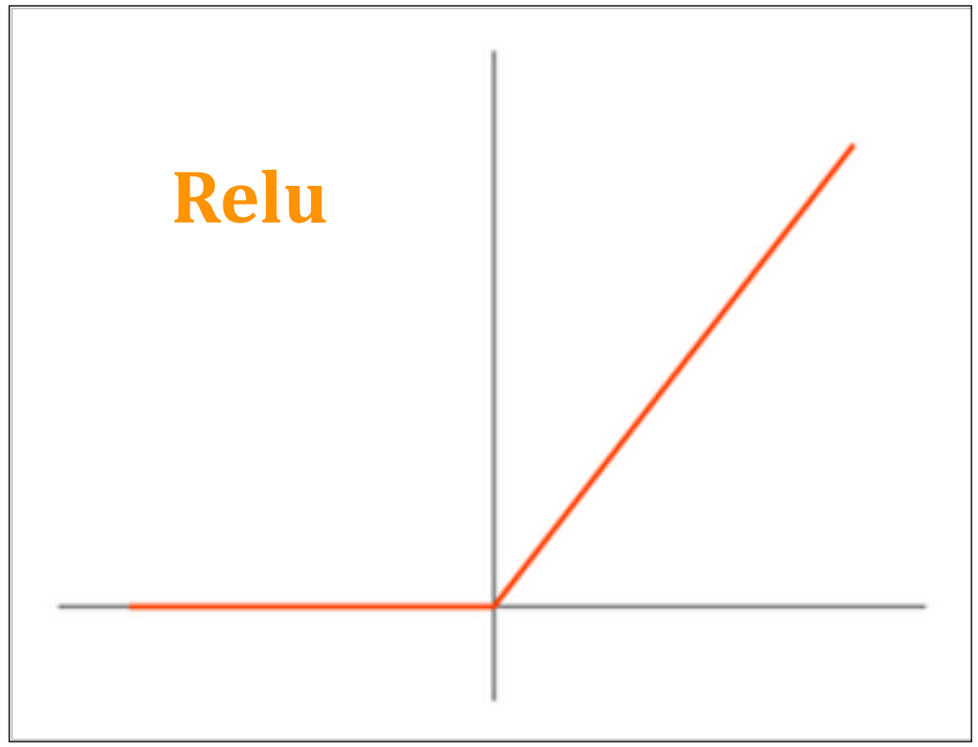


- Nombre d'entrées arbitraire
- Addition d'une constante par neurone : w_0
- **Fonction d'activation** appliquée au résultat du neurone : différents intervalles de valeur possible



Le neurone formel : approximateur universel

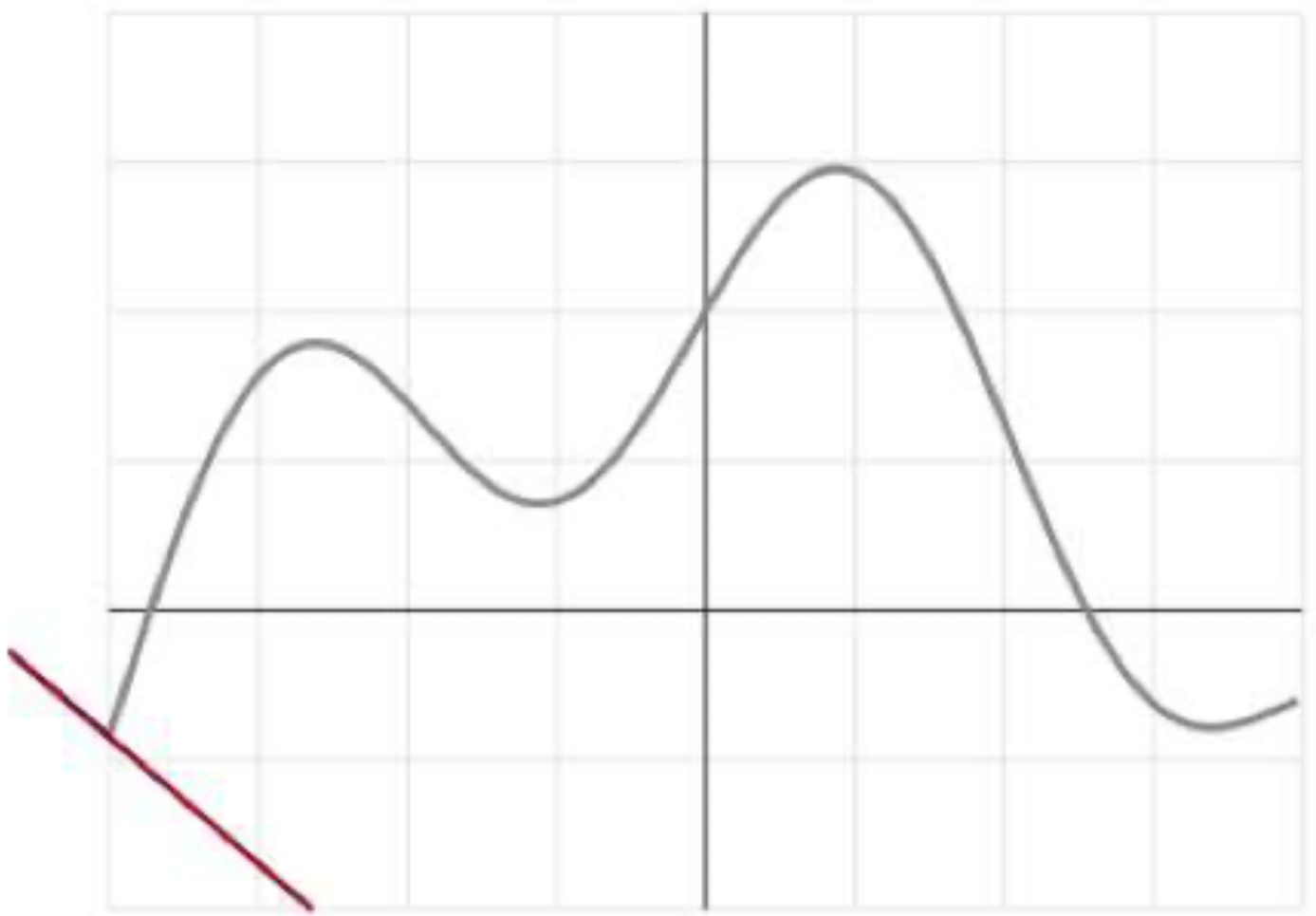




Rectified Linear Unit

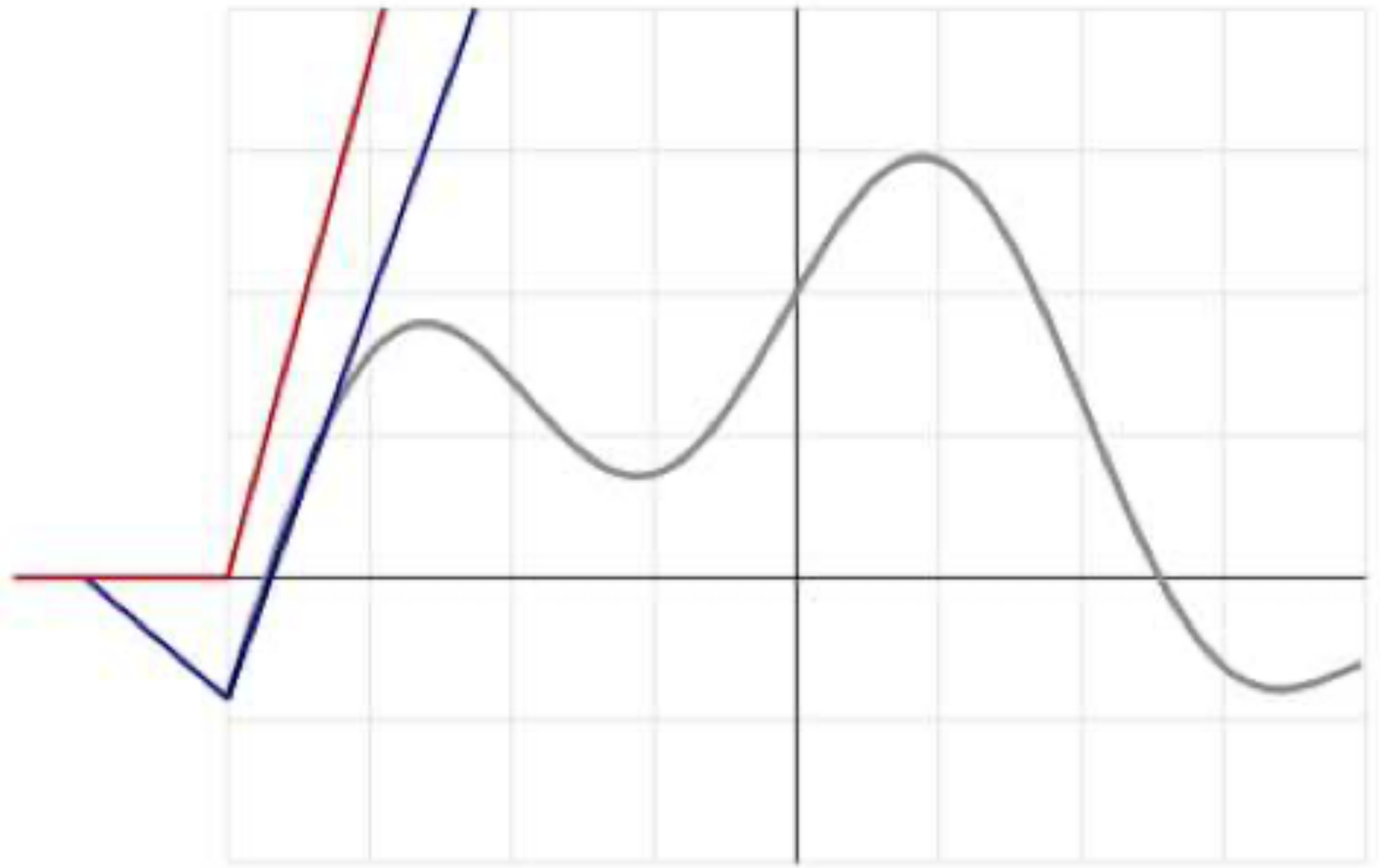
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



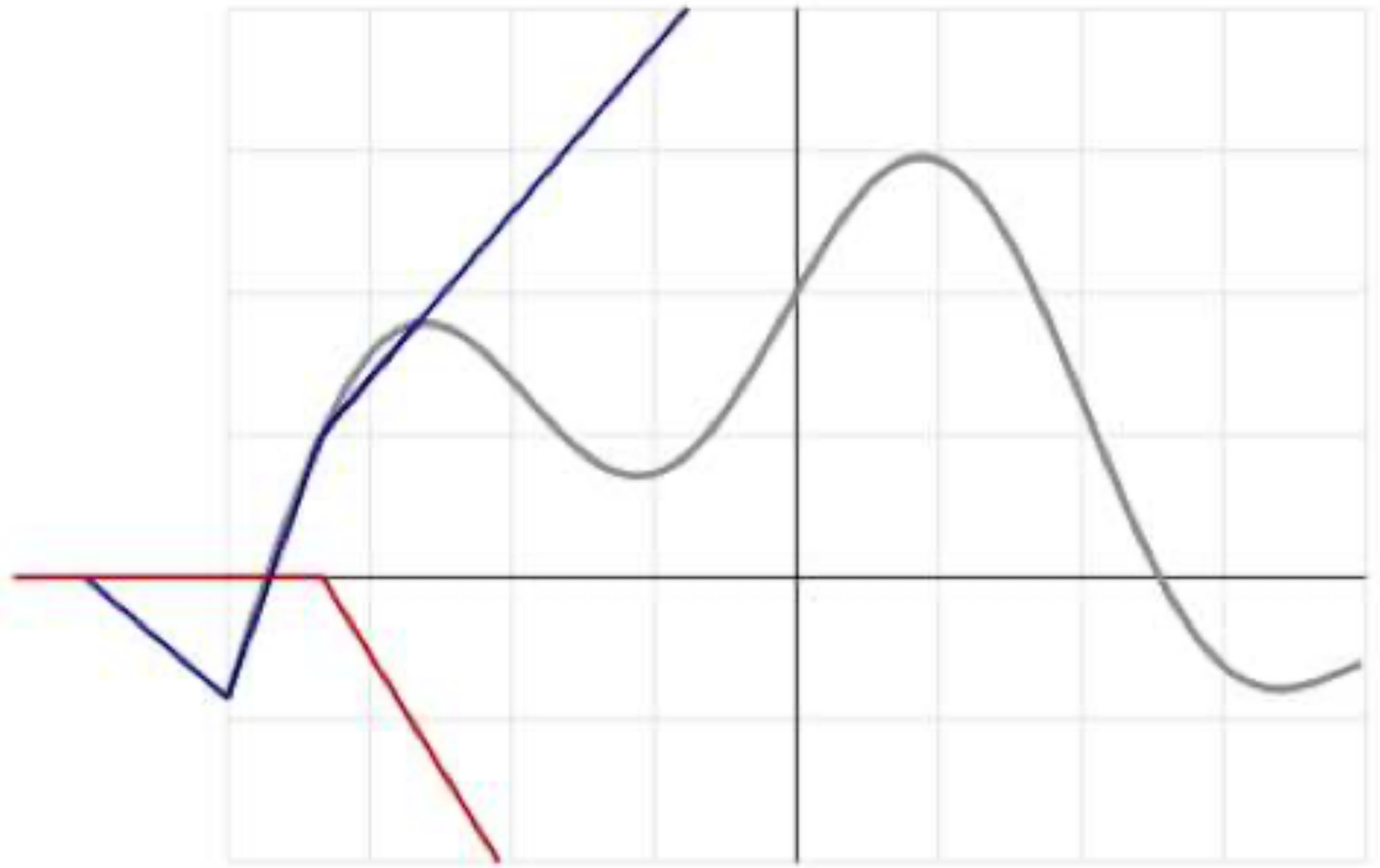
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



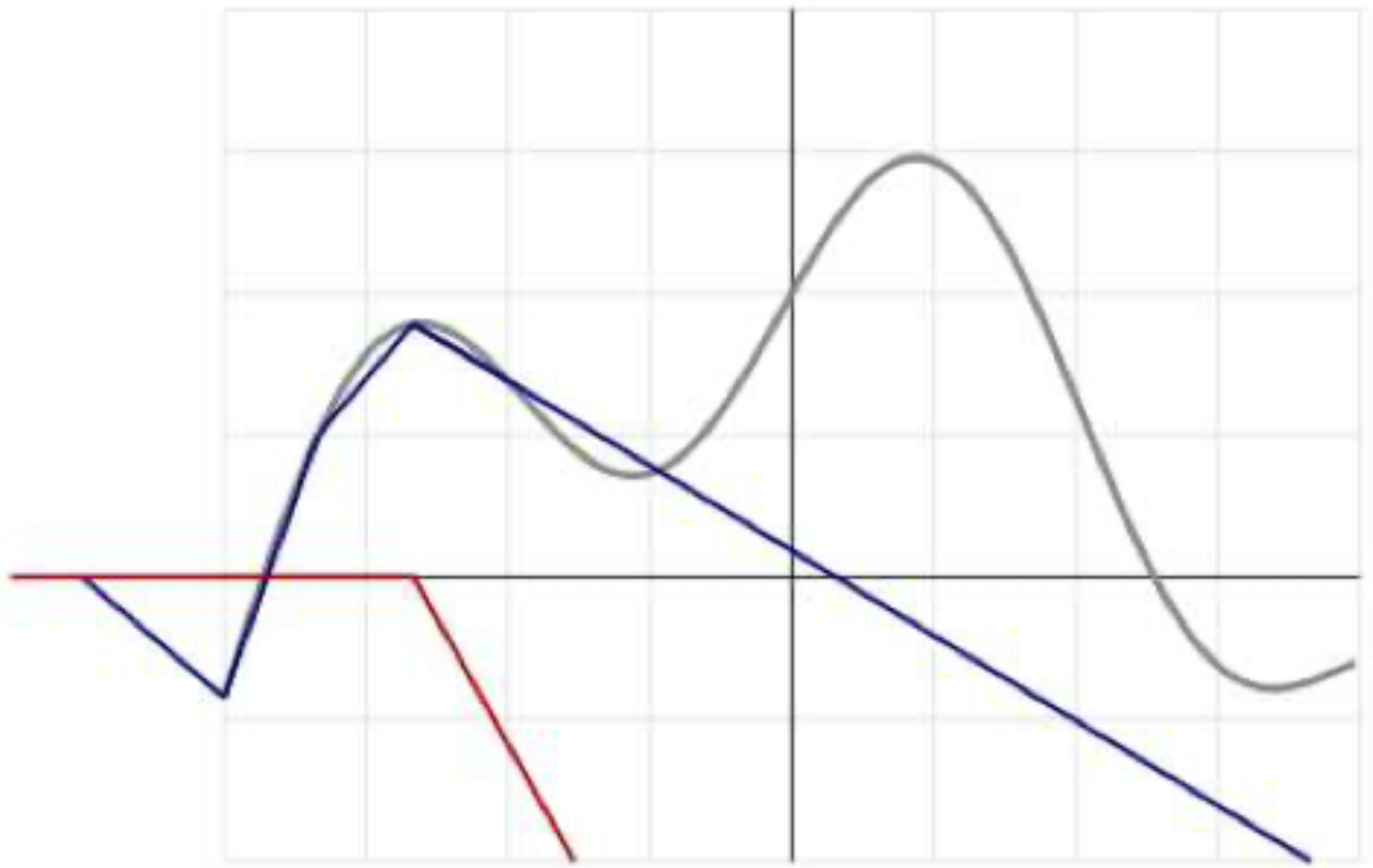
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



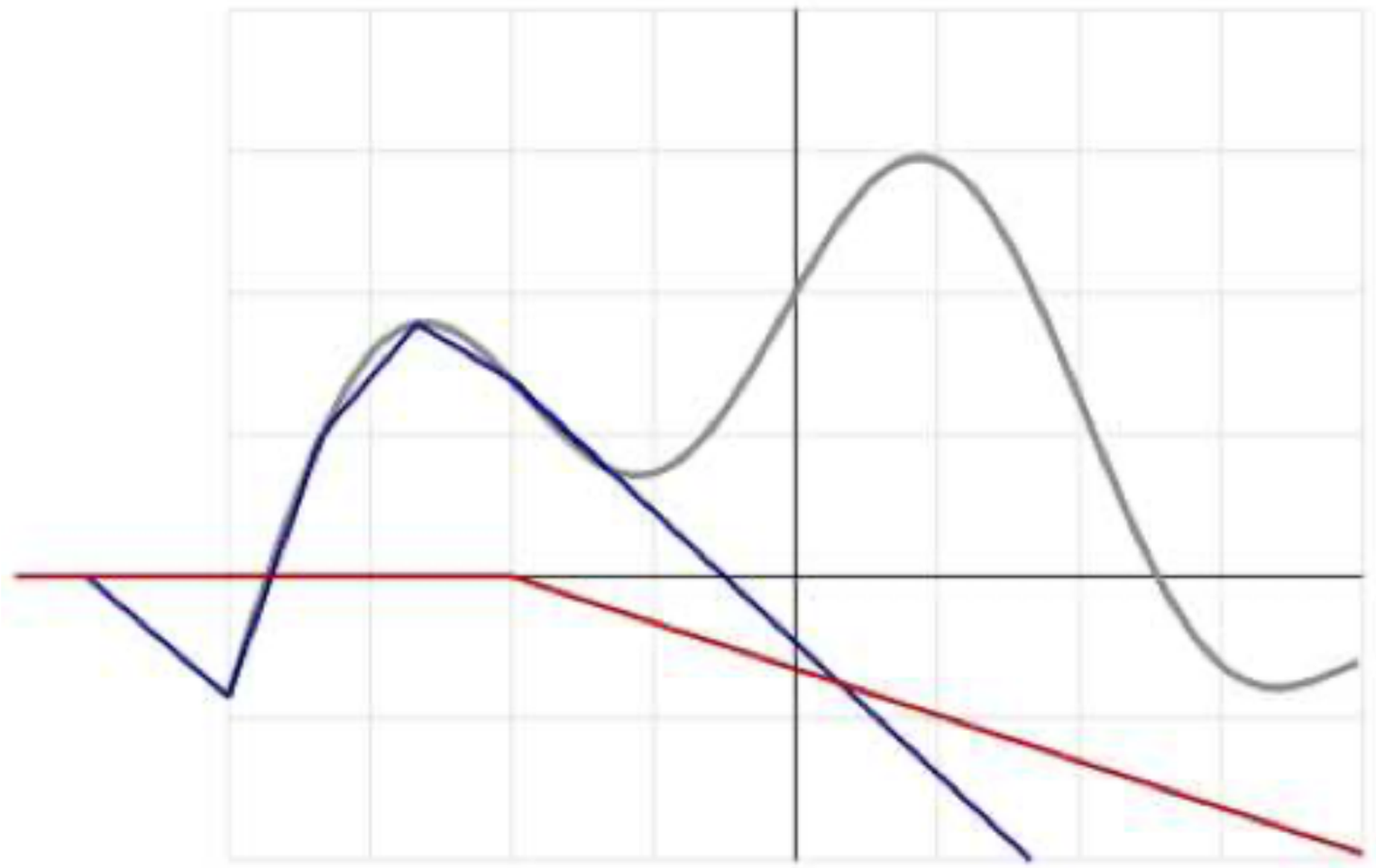
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



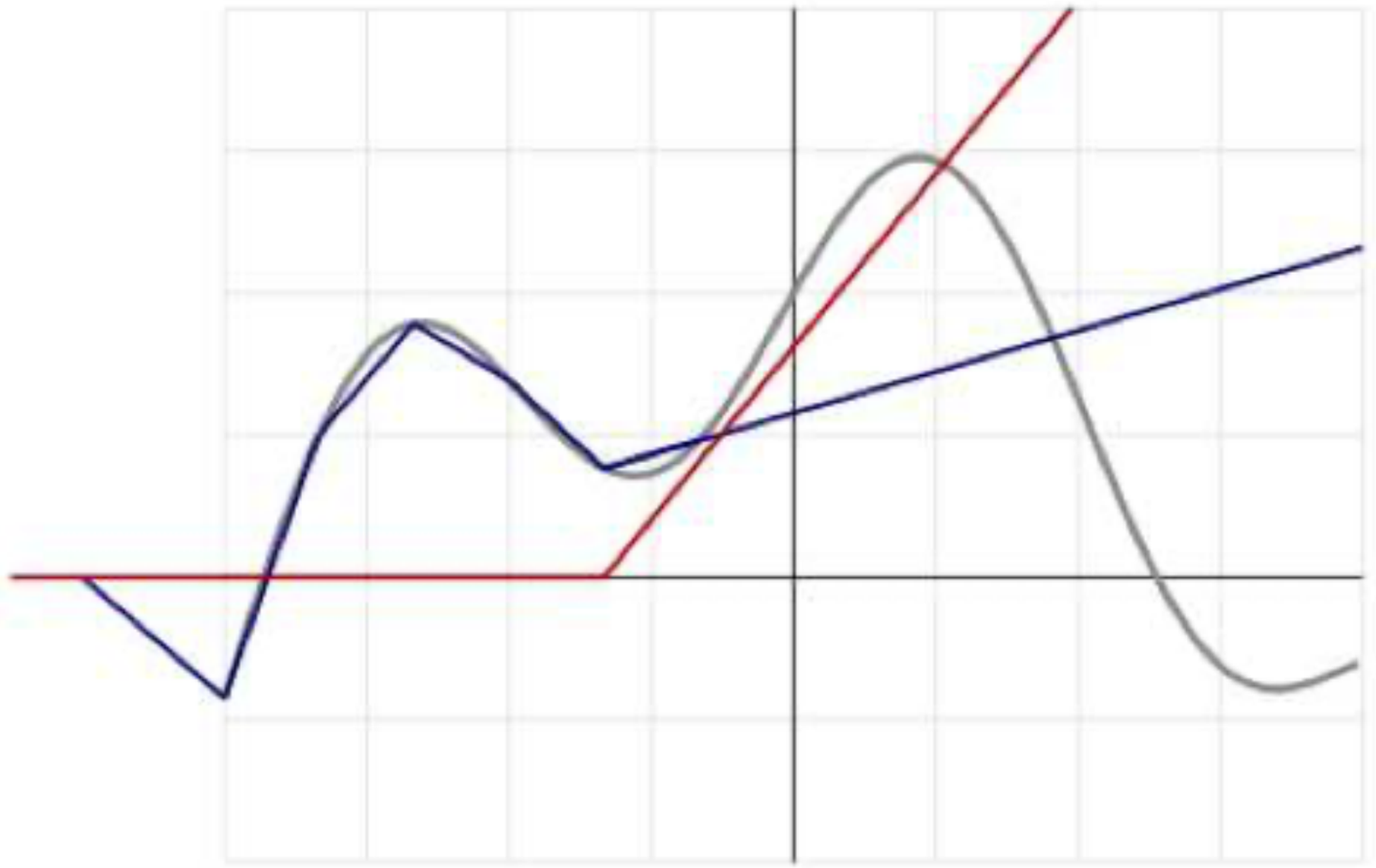
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



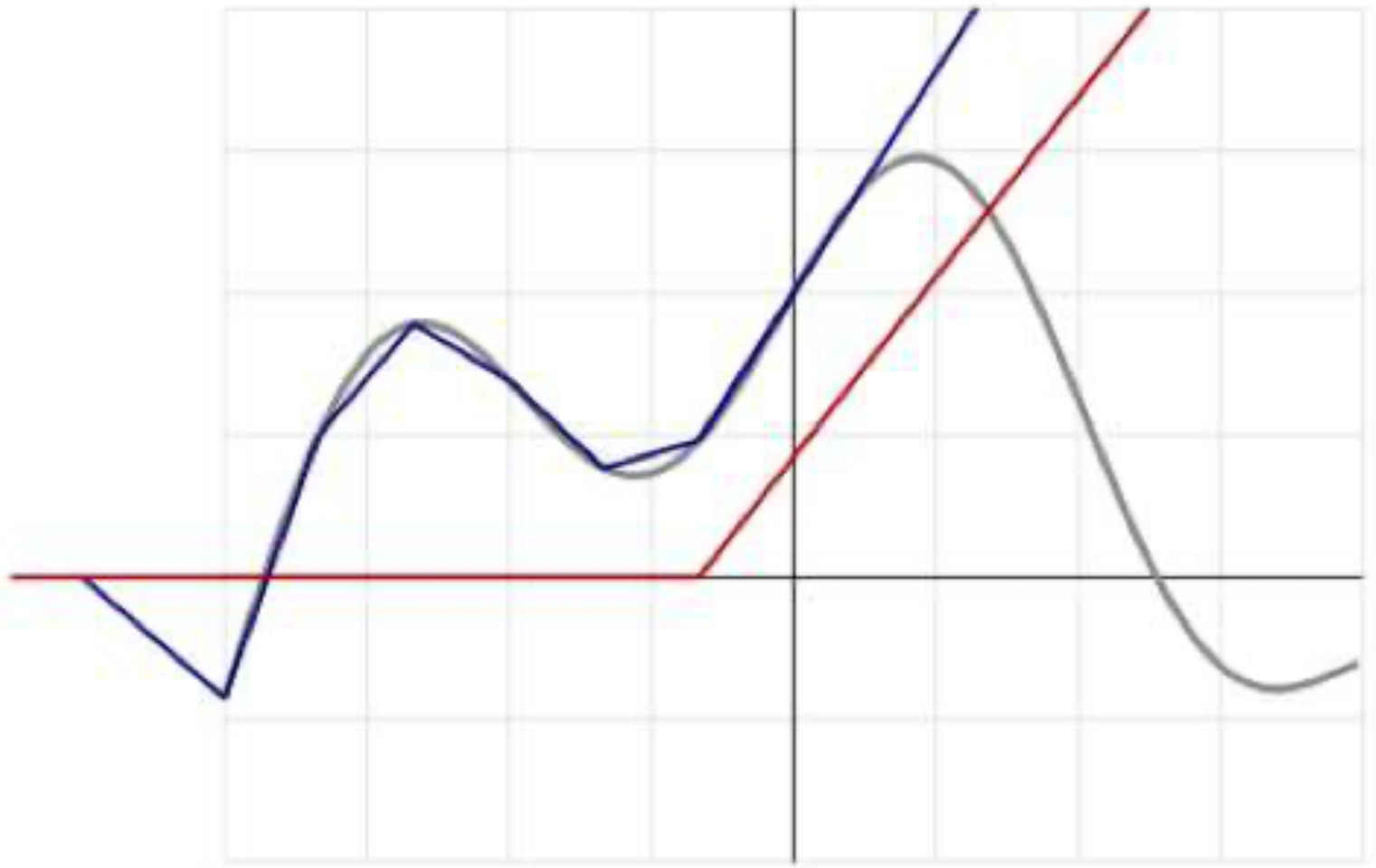
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



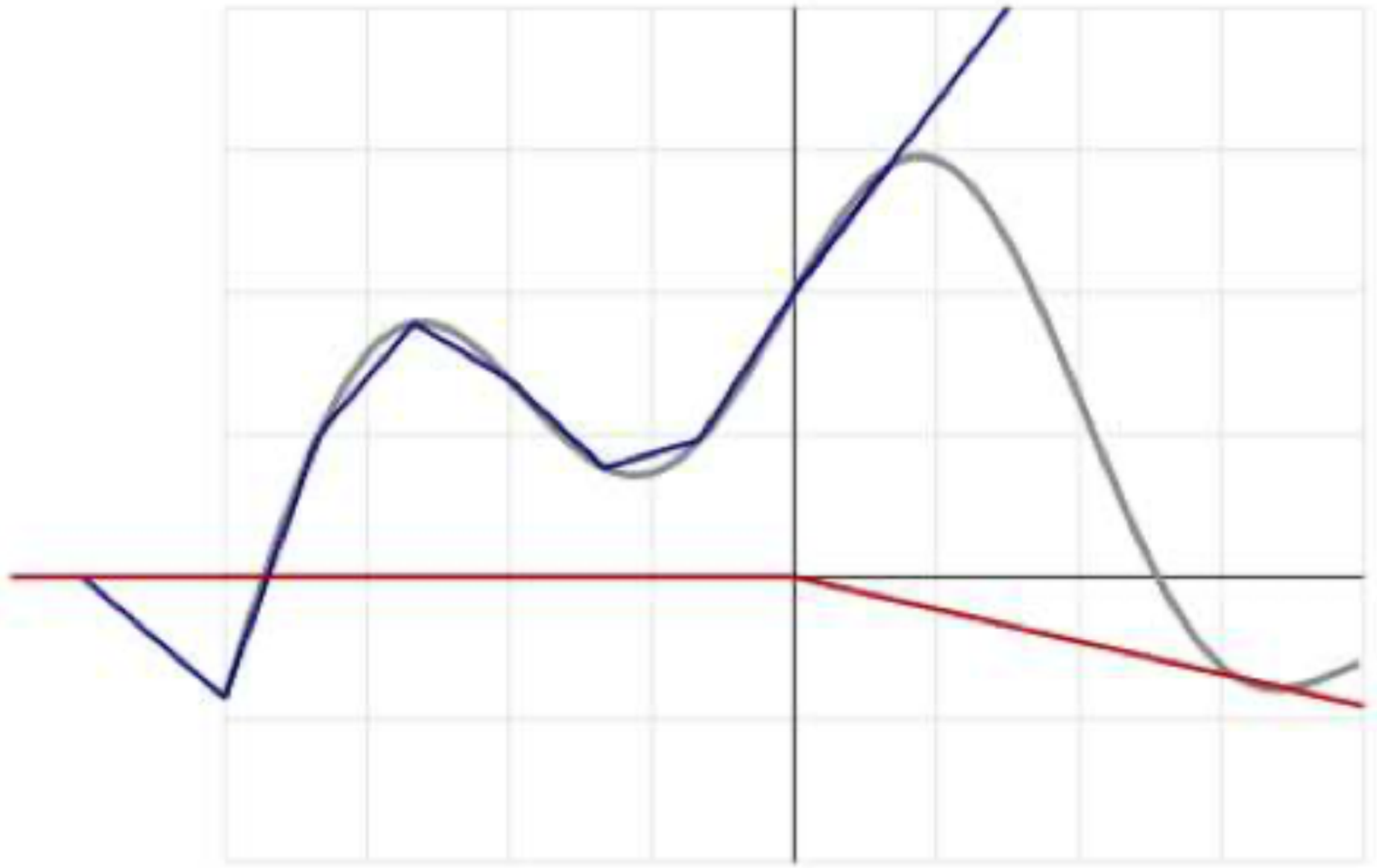
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



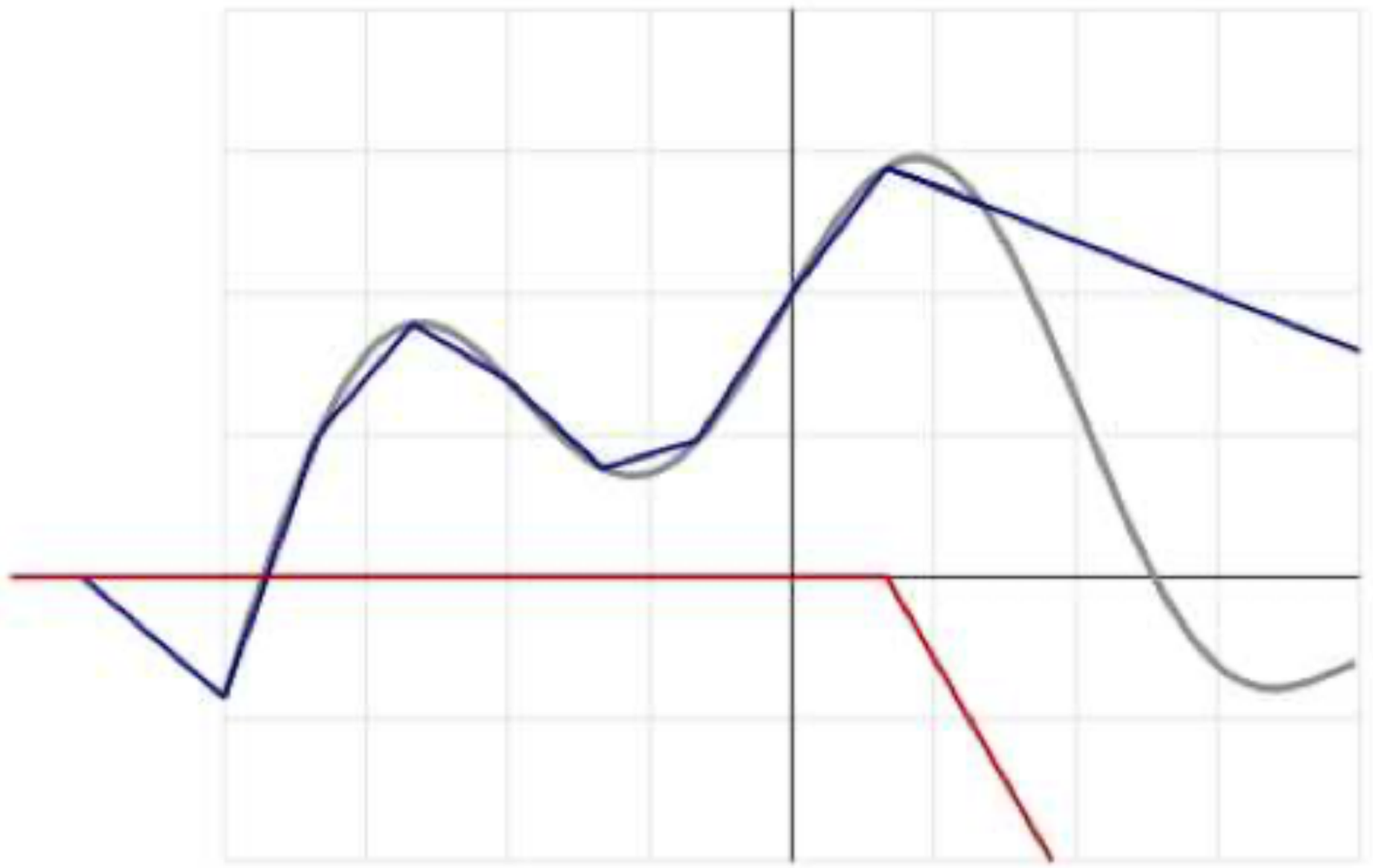
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



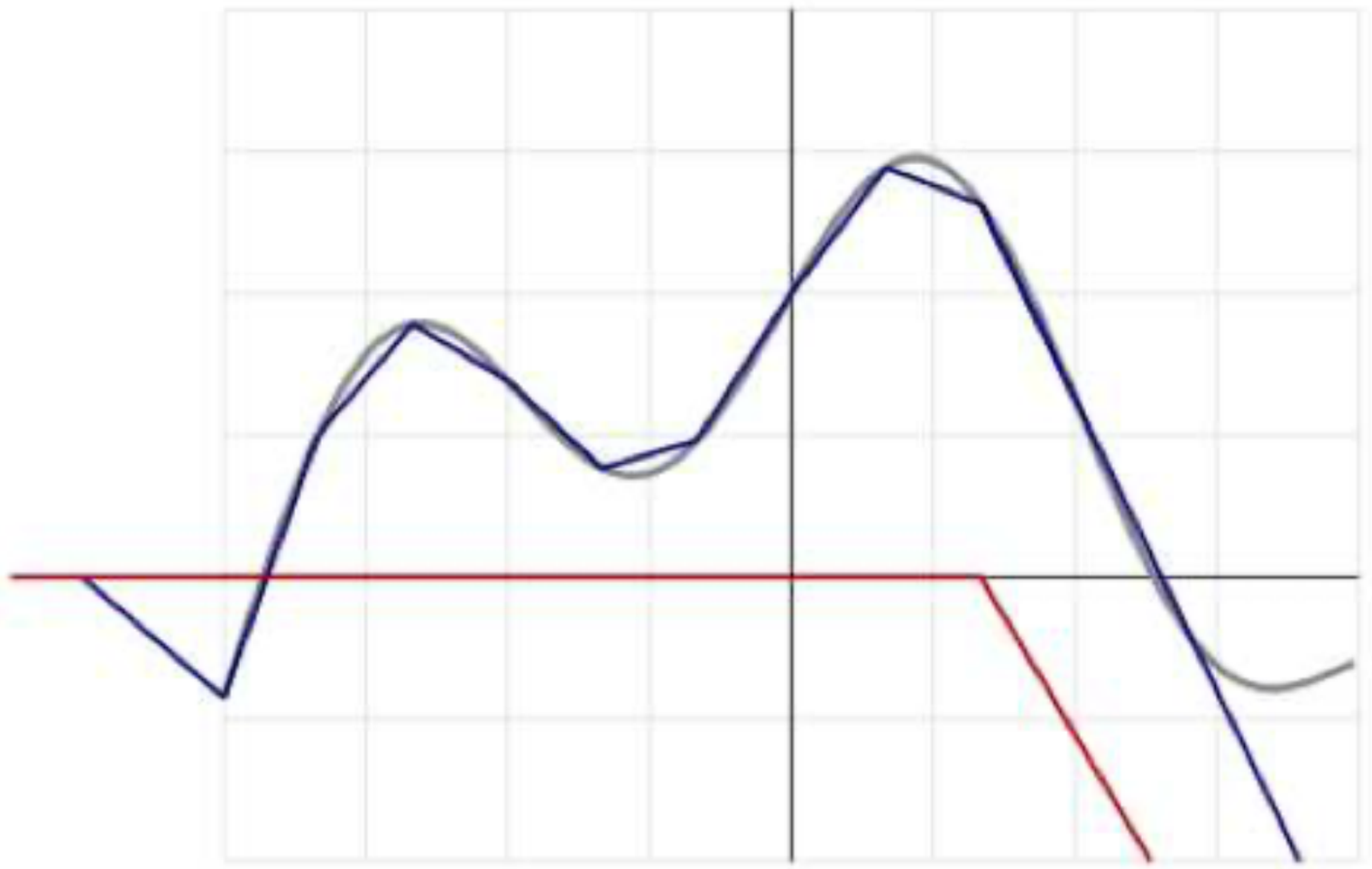
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



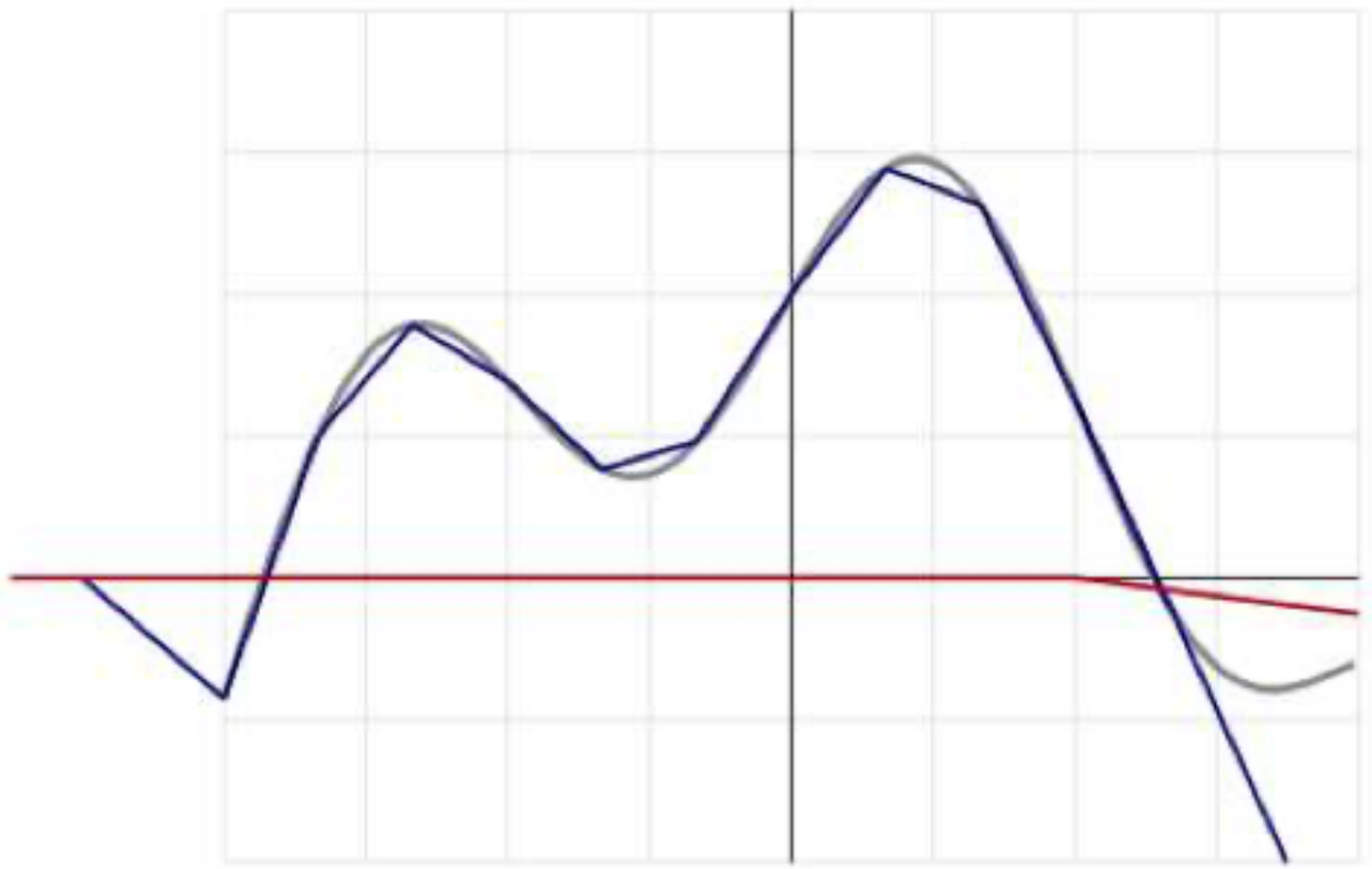
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



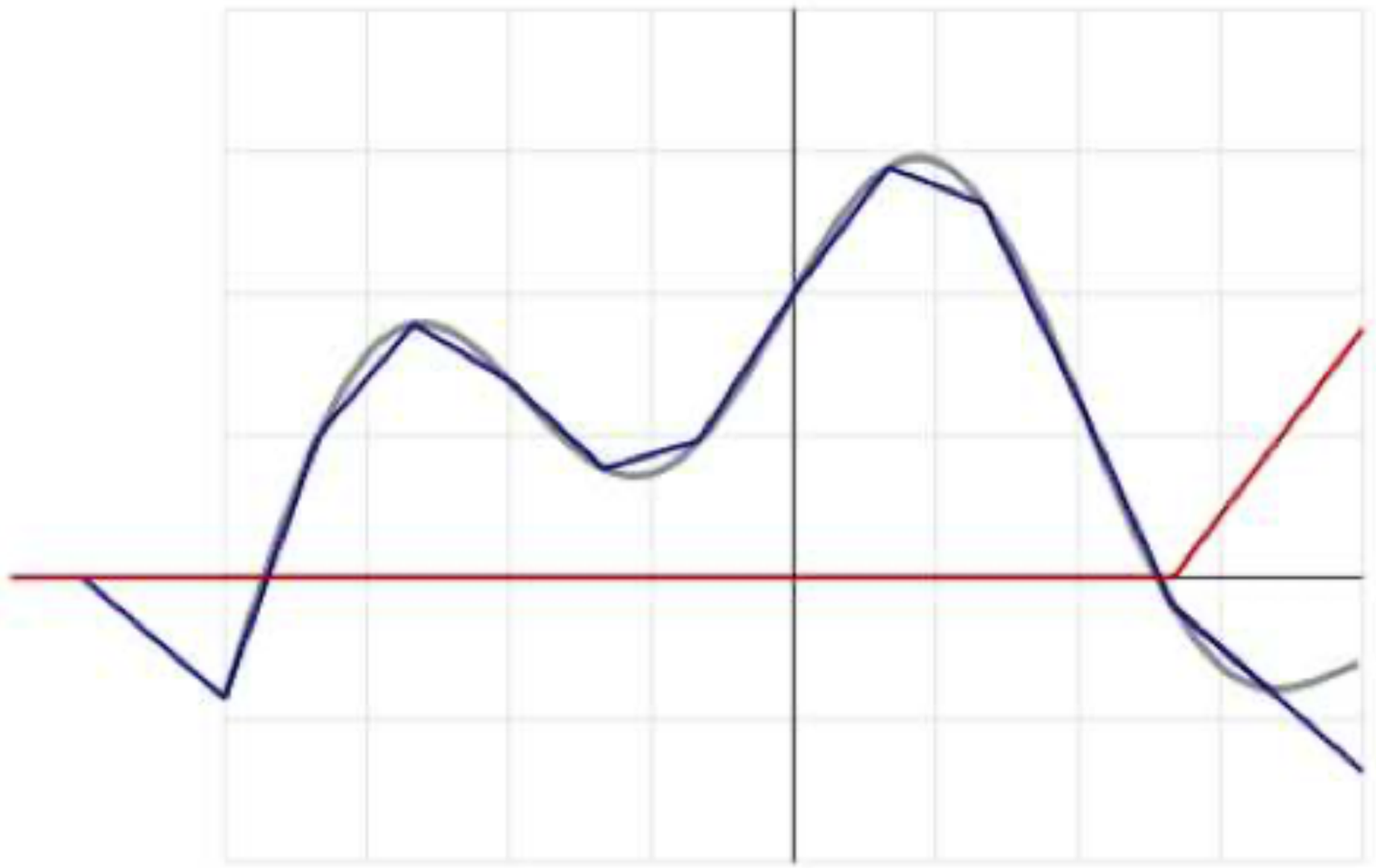
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



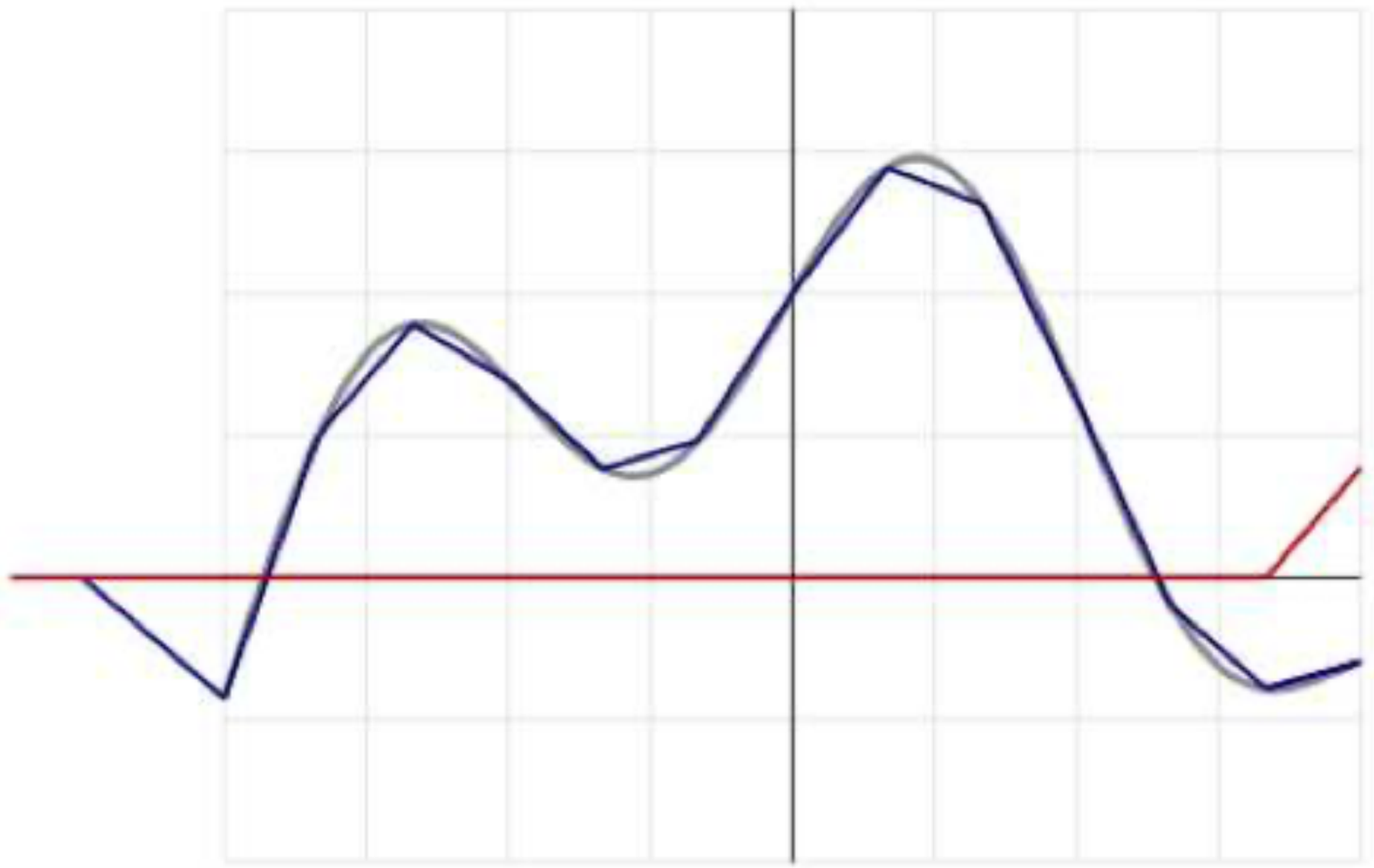
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



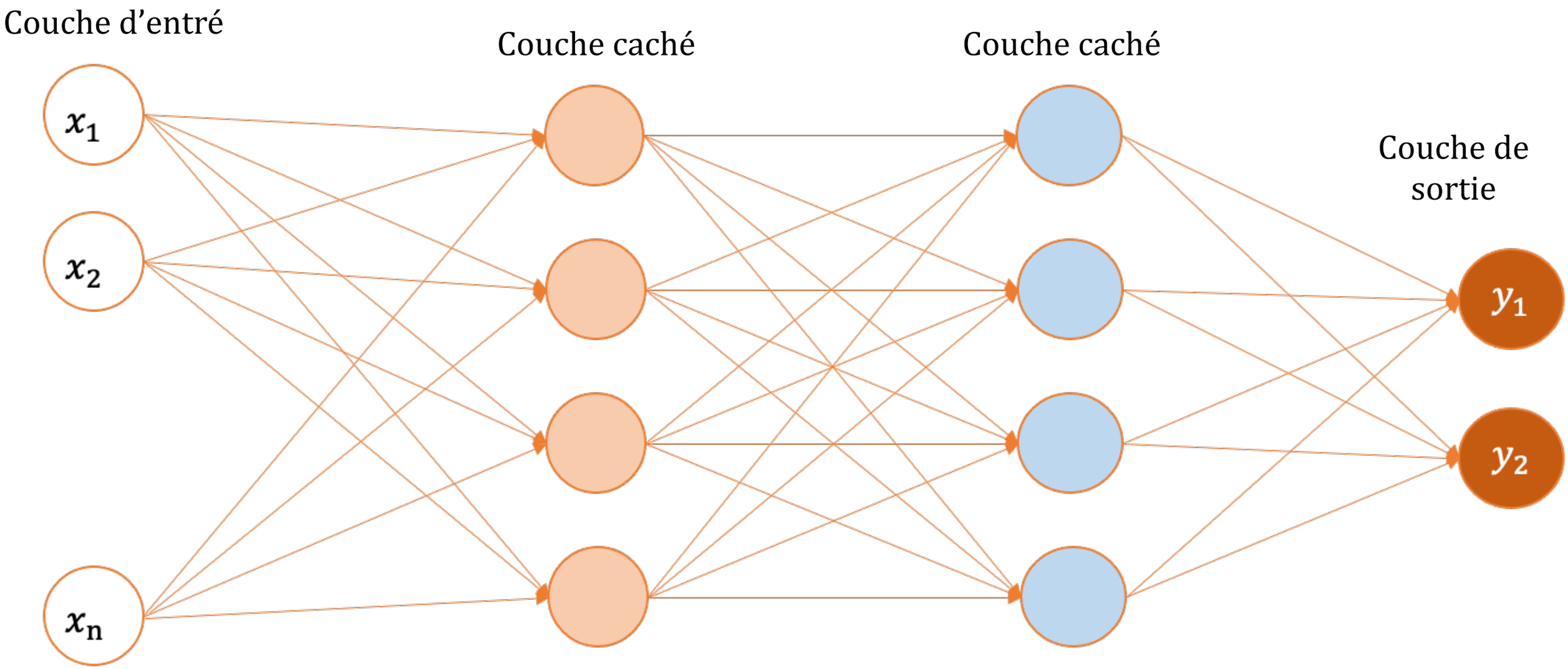
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



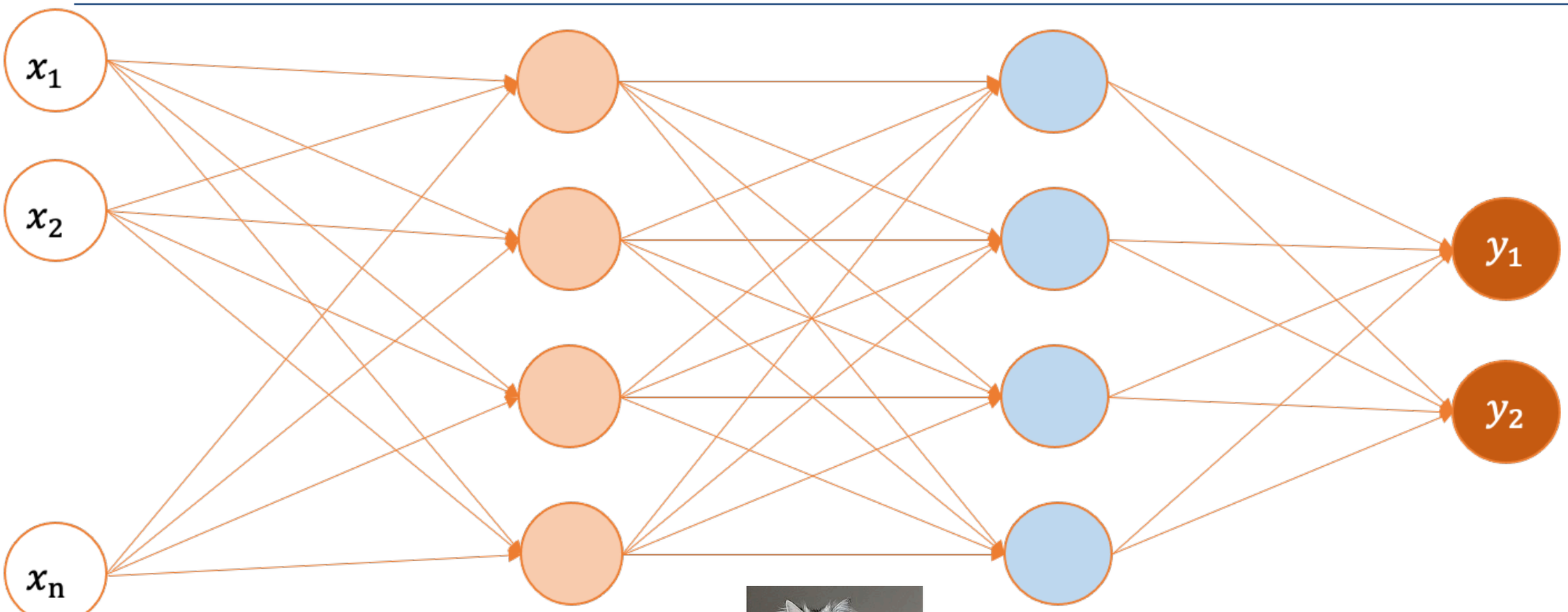
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones peuvent naturellement remplir ce rôle

Perceptron multicouche (MLP)



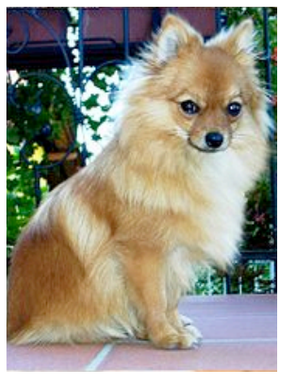
- Neurones assemblés ensemble pour **approximer** une fonction inconnue
- Taille des couches cachées et profondeur (nombre de couches) au choix du développeur

Perceptron multicouche : example



- Example Chat/Chien
- x_i : pixel de mon image
- y_1 : 0 ➜ pas chat; 1 ➜ chat
- y_2 : 0 ➜ pas chien; 1 ➜ chien

(0.85, 0.1)
Clairement Chat



(0.35, 0.78)
Plutôt chien

Perceptron multicouche : mathématiquement

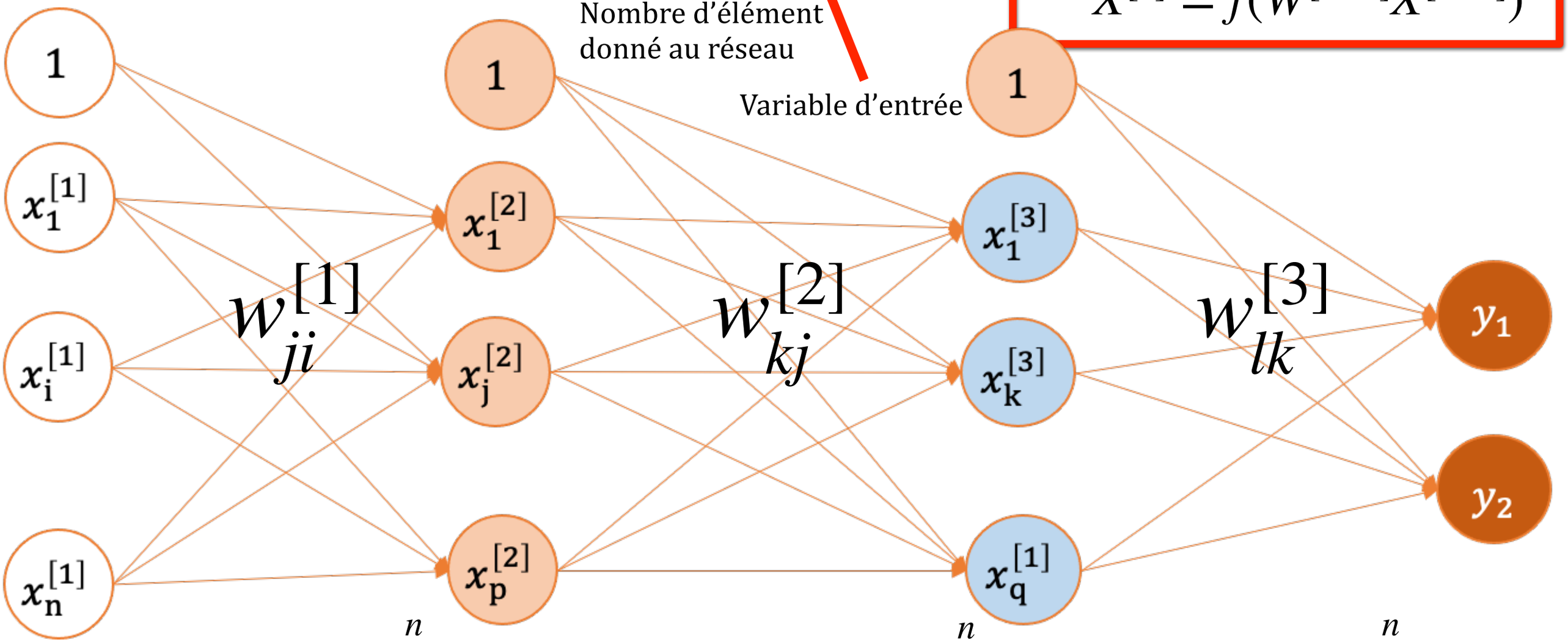
$$X^{[0]} : N \times n$$

Nombre d'élément donné au réseau

Variable d'entrée

D'un point de vue matricielle :

$$X^{[c]} = f(W^{[c-1]} X^{[c-1]})$$



$$s_j^{[2]} = \sum_{i=0}^{n} w_{ji}^{[1]} x_i^{[1]}$$

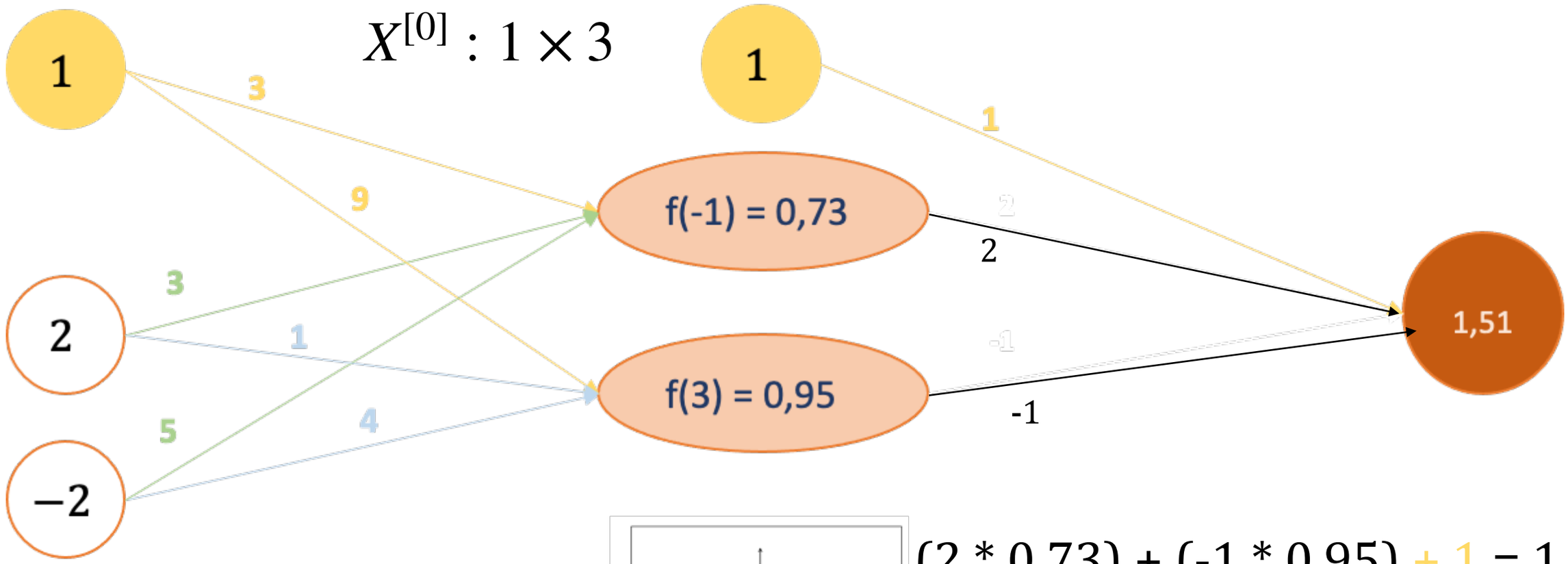
$$x_j^{[2]} = f(s_j^{[2]})$$

$$s_k^{[3]} = \sum_{j=0}^{n} w_{kj}^{[2]} x_j^{[2]}$$

$$x_k^{[2]} = f(s_k^{[2]})$$

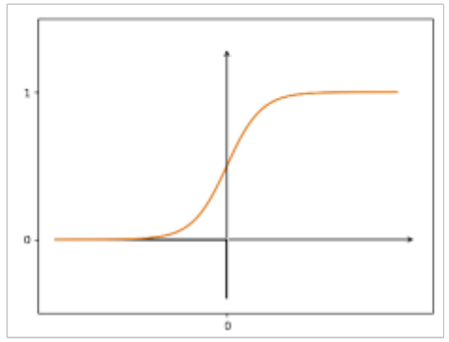
$$y_l = \sum_{k=0}^{n} w_{lk}^{[3]} x_k^{[3]}$$

Perceptron multicouche : Example



$(3 * 2) + (5 * -2) + 3 = -1$

$(1 * 2) + (4 * -2) + 9 = 3$



$(2 * 0.73) + (-1 * 0.95) + 1 = 1.51$

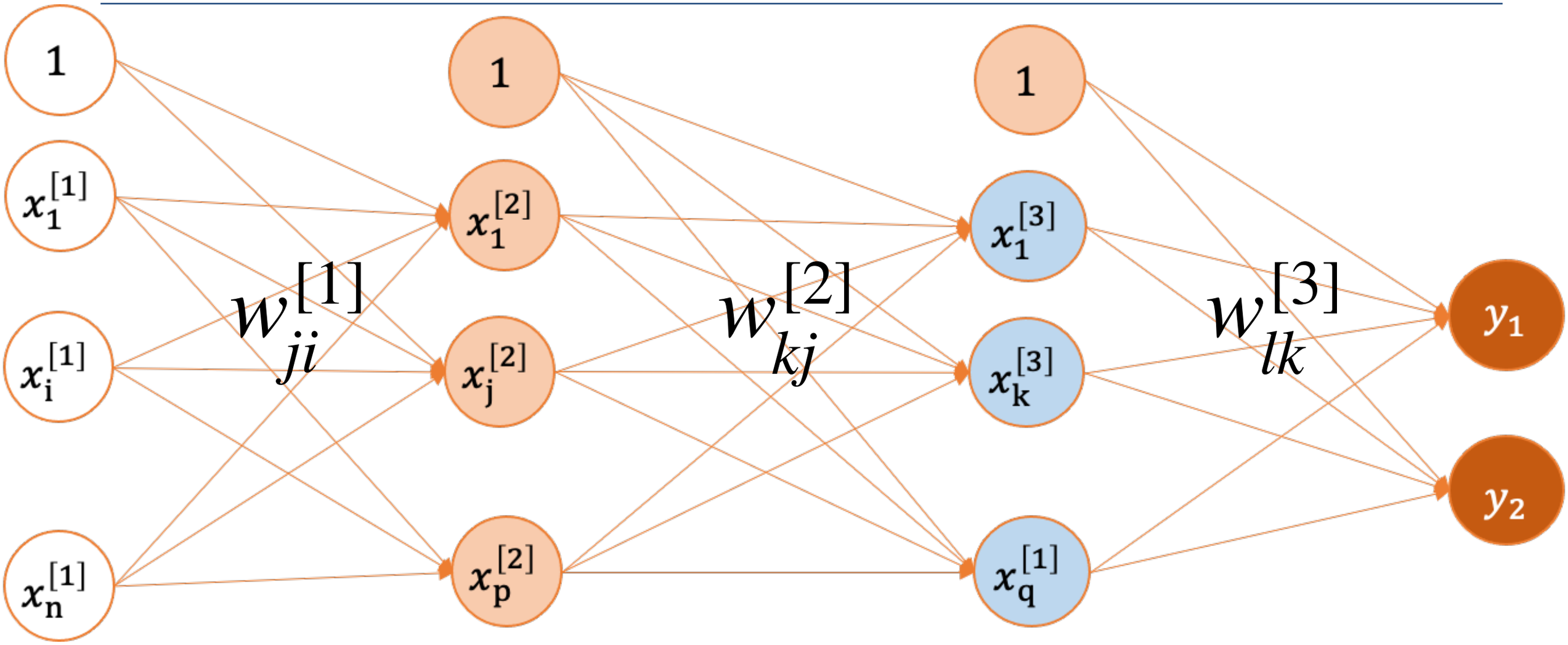
En notation matriciel :

$$\begin{pmatrix} 3 & 3 & 5 \\ 9 & 4 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ -2 \end{pmatrix} = \begin{pmatrix} -1 \\ 3 \end{pmatrix}$$

$$f\begin{pmatrix} -1 \\ 3 \end{pmatrix} = \begin{pmatrix} 0.73 \\ 0.95 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0.73 \\ 0.95 \end{pmatrix} = 1.51$$

Rétropropagation du gradient

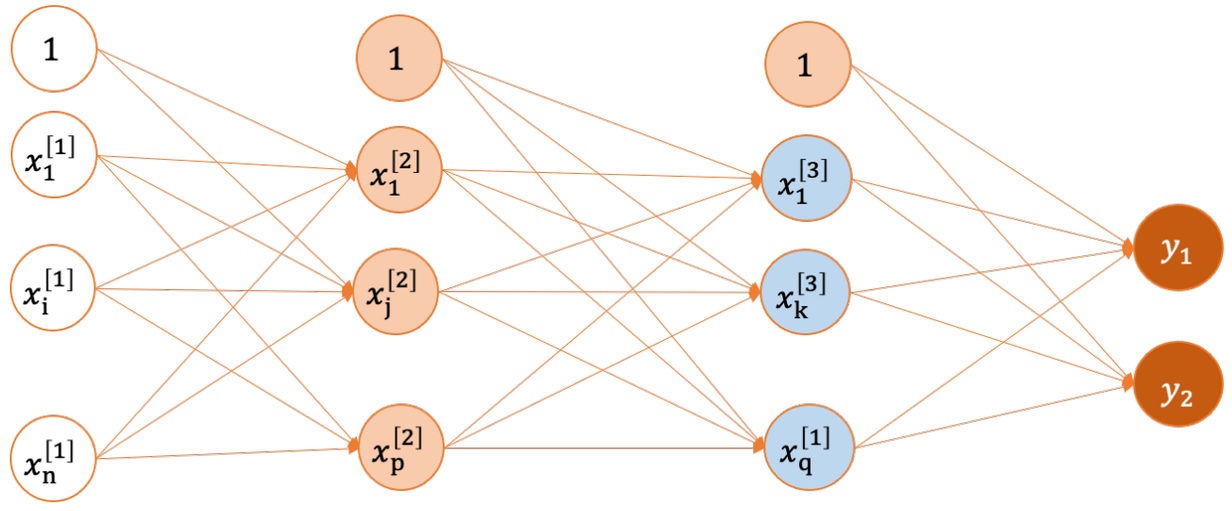


- Nombre de **paramètres** : $(n+1) \times p + (p+1) \times q + (q+1) \times 2$
- Si n = p = q =10 ➜ **242** paramètre à optimiser
- Comment effectuer cette **optimisation** de manière efficace ?

Rétropropagation du gradient

Fonction de coût pour évaluer les performances de notre algorithme (**adapté** à l'objectif) :

$$Loss(y, \hat{y}) = \frac{1}{2} \sum_{i=0}^{N} (\hat{y}_i - y_i)^2$$



Algorithme de **rétropropagation** :

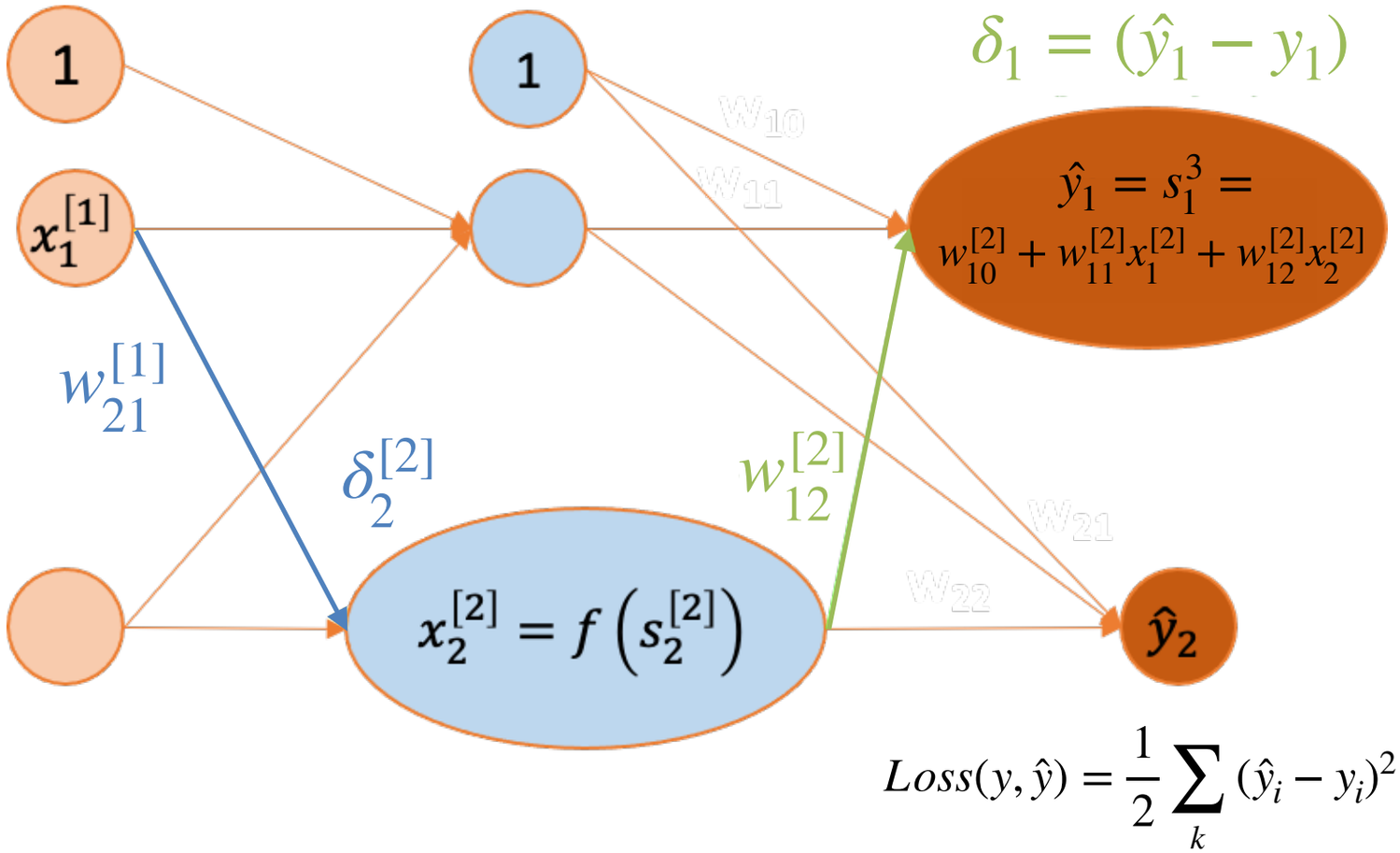
- Calcule $\dfrac{\partial Loss(y, \hat{y})}{\partial w_{ji}}$
- Mise à jour des poids : $w_{ji} \leftarrow w_{ji} - \alpha \dfrac{\partial Loss(y, \hat{y})}{\partial w_{ji}}$

avec α le **taux d'apprentissage** (peut évoluer au cours du temps)

Rétropropagation du gradient

Dernière (P) couche :

$$\frac{\partial Loss}{\partial w_{12}^{[2]}} = \frac{\partial Loss}{\partial s_1^3}\frac{\partial s_1^3}{\partial w_{12}^{[2]}}$$



$$Loss(y, \hat{y}) = \frac{1}{2}\sum_k (\hat{y}_i - y_i)^2$$

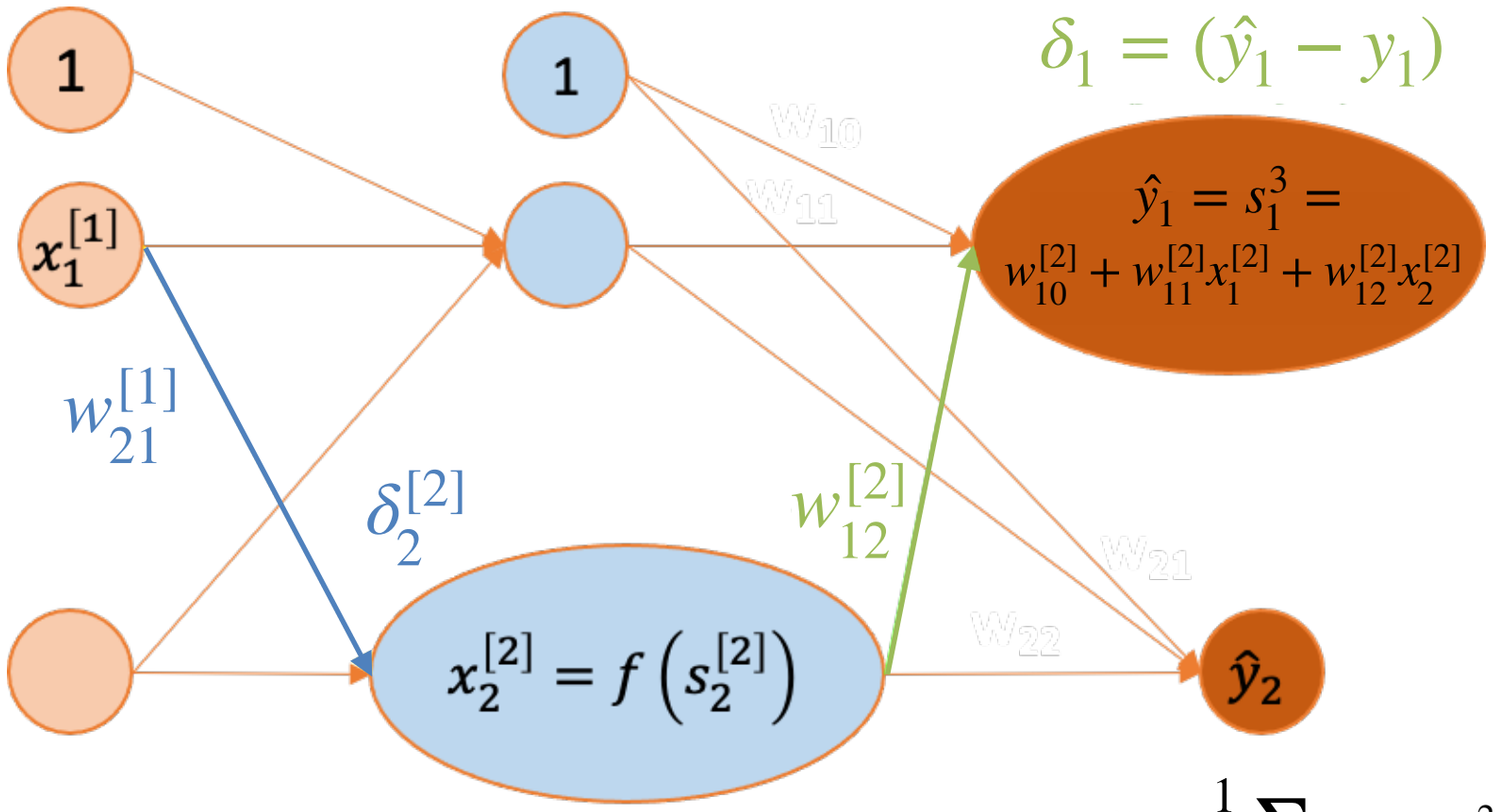
Rétropropagation du gradient

Dernière (P) couche :

$$\frac{\partial Loss}{\partial w_{12}^{[2]}} = \frac{\partial Loss}{\partial s_1^3}\frac{\partial s_1^3}{\partial w_{12}^{[2]}}$$

$\delta_1 = (\hat{y}_1 - y_1)$ $\quad$ $x_2^{[2]}$

$$w_{12}^{[2]} = w_{12}^{[2]} - \alpha\delta_1 x_2^{[2]}$$



$$Loss(y, \hat{y}) = \frac{1}{2}\sum_k (\hat{y}_i - y_i)^2$$

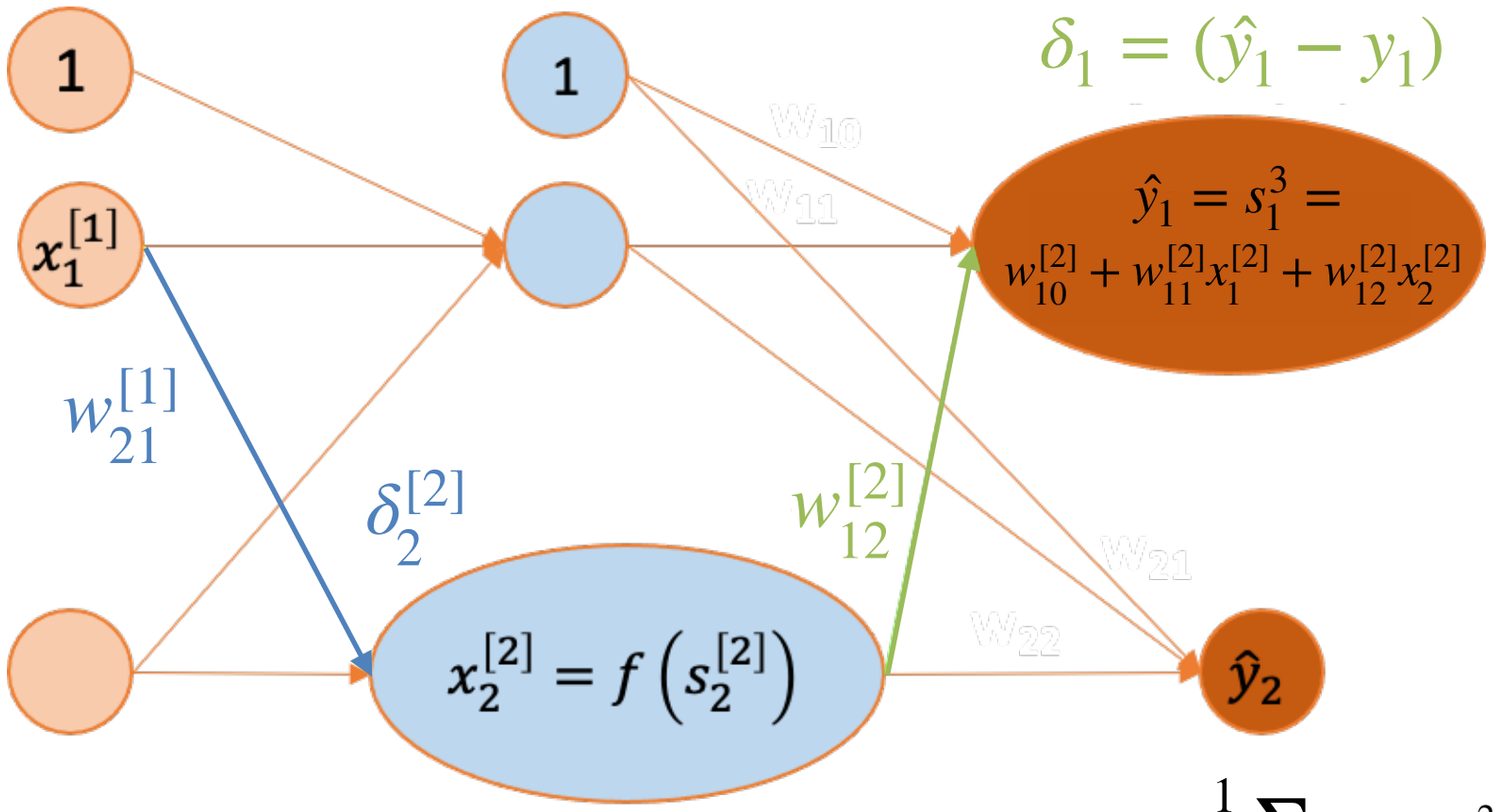
Rétropropagation du gradient

Dernière (P) couche :

$$\frac{\partial Loss}{\partial w_{12}^{[2]}} = \frac{\partial Loss}{\partial s_1^3}\frac{\partial s_1^3}{\partial w_{12}^{[2]}}$$

$\delta_1 = (\hat{y}_1 - y_1)$ $\quad$ $x_2^{[2]}$

$$w_{12}^{[2]} = w_{12}^{[2]} - \alpha\delta_1 x_2^{[2]}$$



$$Loss(y, \hat{y}) = \frac{1}{2}\sum_k (\hat{y}_i - y_i)^2$$

Couche P-1 :

$x_1^{[1]}$

$$\frac{\partial Loss}{\partial w_{21}^{[1]}} = \frac{\partial Loss}{\partial s_2^{[2]}}\frac{\partial s_2^{[2]}}{\partial w_{21}^{[1]}}$$

$$\frac{\partial Loss}{\partial s_2^{[2]}} = \sum_k \delta_k \frac{\partial \delta_k}{\partial s_2^{[2]}}$$

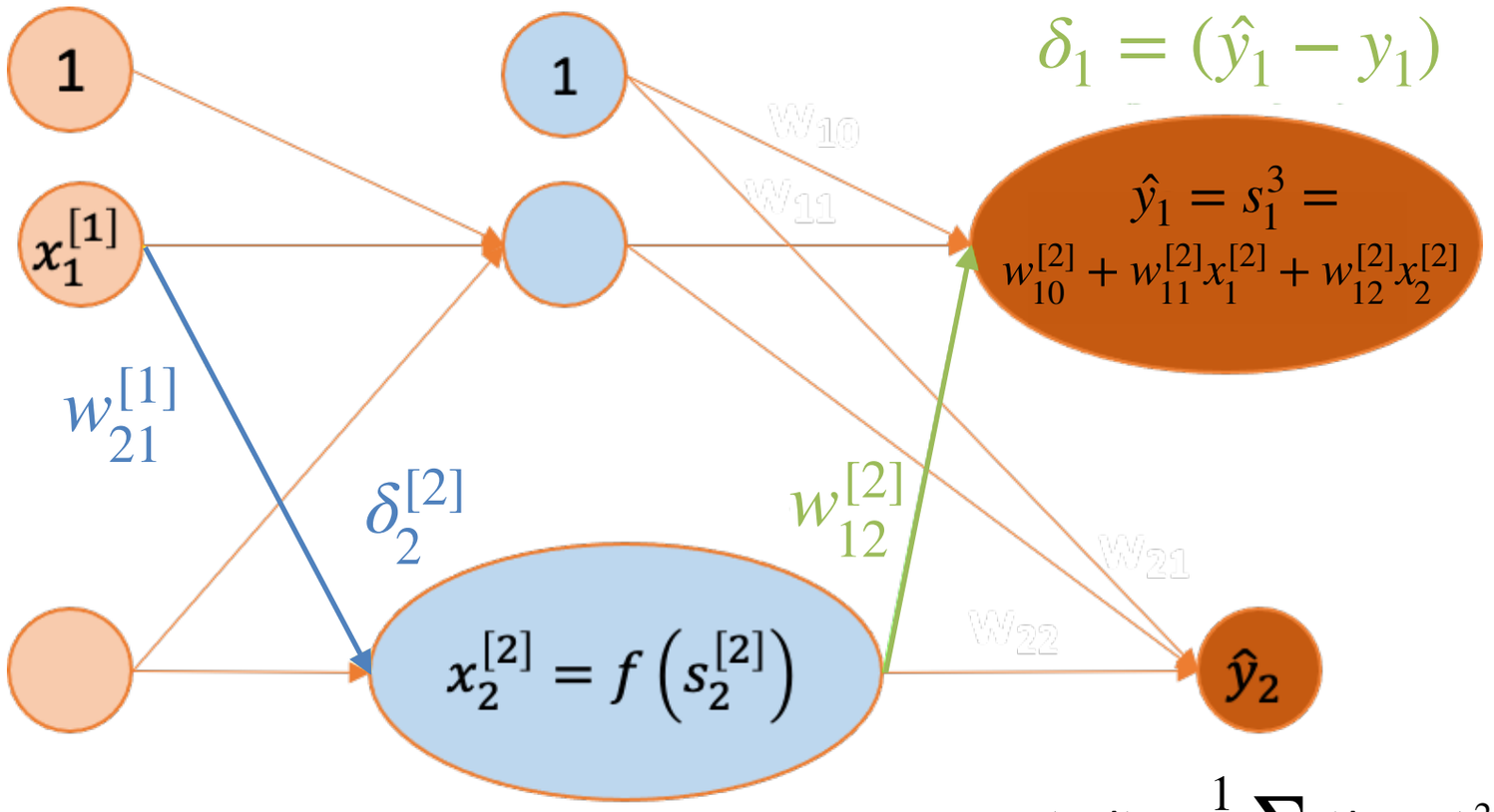
Rétropropagation du gradient

Dernière (P) couche :

$$\frac{\partial Loss}{\partial w_{12}^{[2]}} = \frac{\partial Loss}{\partial s_1^3}\frac{\partial s_1^3}{\partial w_{12}^{[2]}}$$

$\delta_1 = (\hat{y}_1 - y_1)$ $\quad$ $x_2^{[2]}$

$$w_{12}^{[2]} = w_{12}^{[2]} - \alpha\delta_1 x_2^{[2]}$$



$$Loss(y, \hat{y}) = \frac{1}{2}\sum_k (\hat{y}_i - y_i)^2$$

Couche P-1 :

$x_1^{[1]}$

$$\frac{\partial Loss}{\partial w_{21}^{[1]}} = \frac{\partial Loss}{\partial s_2^{[2]}}\frac{\partial s_2^{[2]}}{\partial w_{21}^{[1]}}$$

$$\frac{\partial Loss}{\partial s_2^{[2]}} = \sum_k \delta_k \frac{\partial \delta_k}{\partial s_2^{[2]}}$$

Au moins dérivable par morceaux

$$\frac{\partial \delta_k}{\partial s_2^{[2]}} = w_{k2}^{[2]} f'(s_2^{[2]}) \qquad \frac{\partial Loss}{\partial s_2^{[2]}} = f'(s_2^{[2]}) \sum_k \delta_k w_{k2}^{[2]}$$

$$\frac{\partial Loss}{\partial w_{21}^{[1]}} = x_1^{[1]} f'(s_2^{[2]}) \sum_k \delta_k w_{k2}^{[2]}$$

Rétropropagation du gradient

Si vous vous ennuyez cet été essayez de calculer P-2 !

$$(g \circ f)' = (g' \circ f) \times f'$$

Plus **généralement**, on peut ensuite montrer que :

$$\frac{\partial Loss}{\partial w_{ji}^{[p-1]}} = x_i^{[p-1]} f'(s_j^{[p]}) \sum_k \frac{\partial Loss}{\partial s_k^{[p]}} w_{kj}^{[p]}$$

Précédent Gradient !

Grâce à cette formule on peut facilement calculer **de proche en proche** la dérivée de chaque poids pour mettre à jour le réseau sans difficulté !

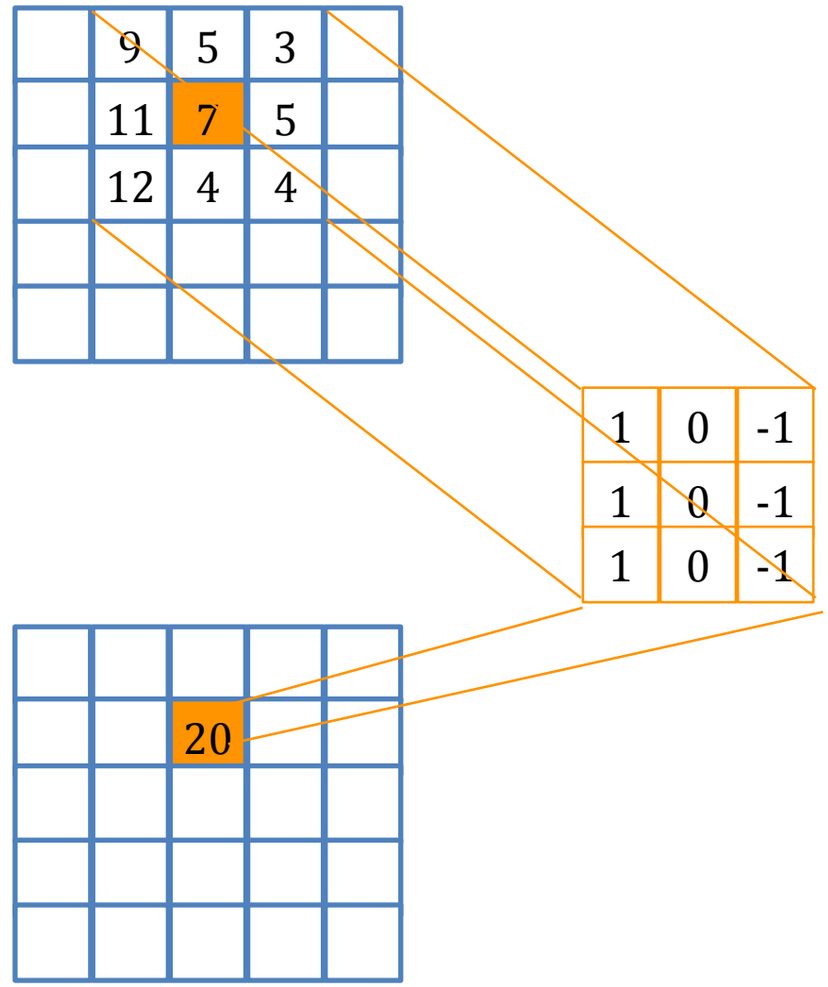
Les réseaux de convolution (CNN)

MLP : Peu adapté aux images

- Le nombre paramètre augmente très vite avec la taille de l'entrée
- Pas **d'invariance** par translation/rotation

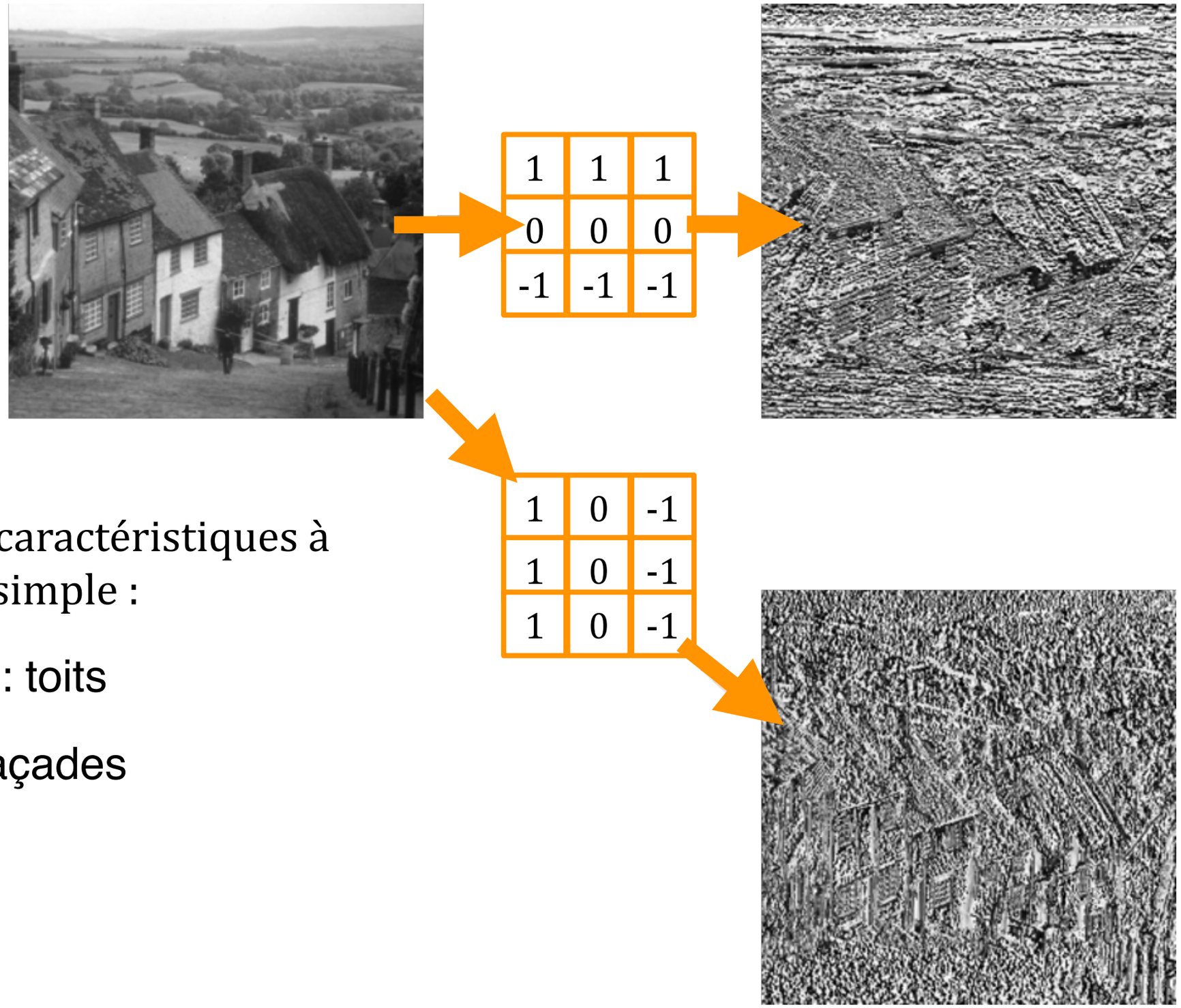
Utilisation de filtre de convolution :

- Boucle sur tous les pixels et calcule une nouvelle valeur basée sur les voisins
- Les **poids** du filtre sont **optimisés** à l'entraînement
- La taille du filtre est au choix du développeur
- Essaie de capturer les **traits caractéristiques** de l'image
- Peut remplacer les MLP dans un grand nombre de réseaux



Les réseaux de convolution

Illustration de l'effet des filtres :



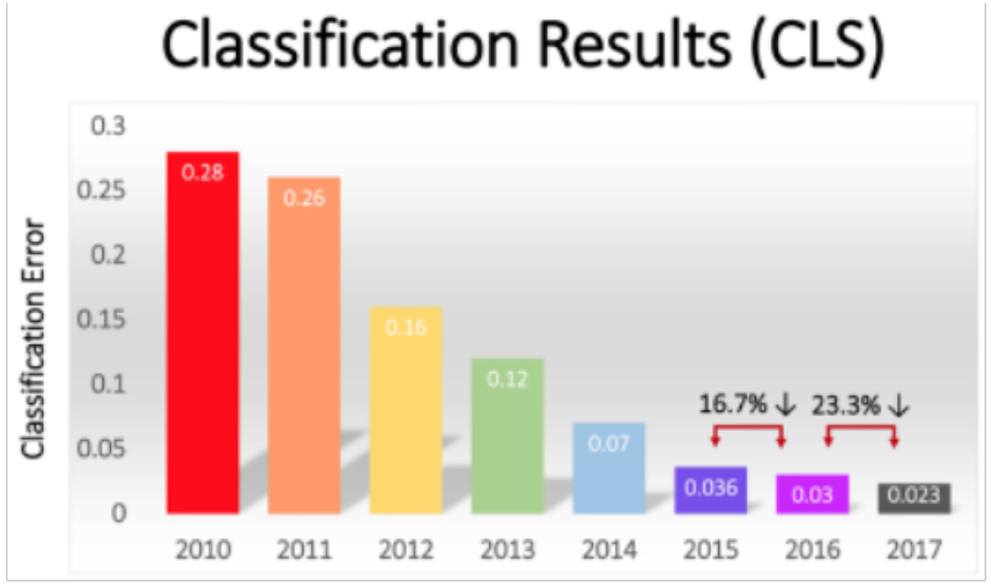
Extraction des caractéristiques à partir de filtre simple :

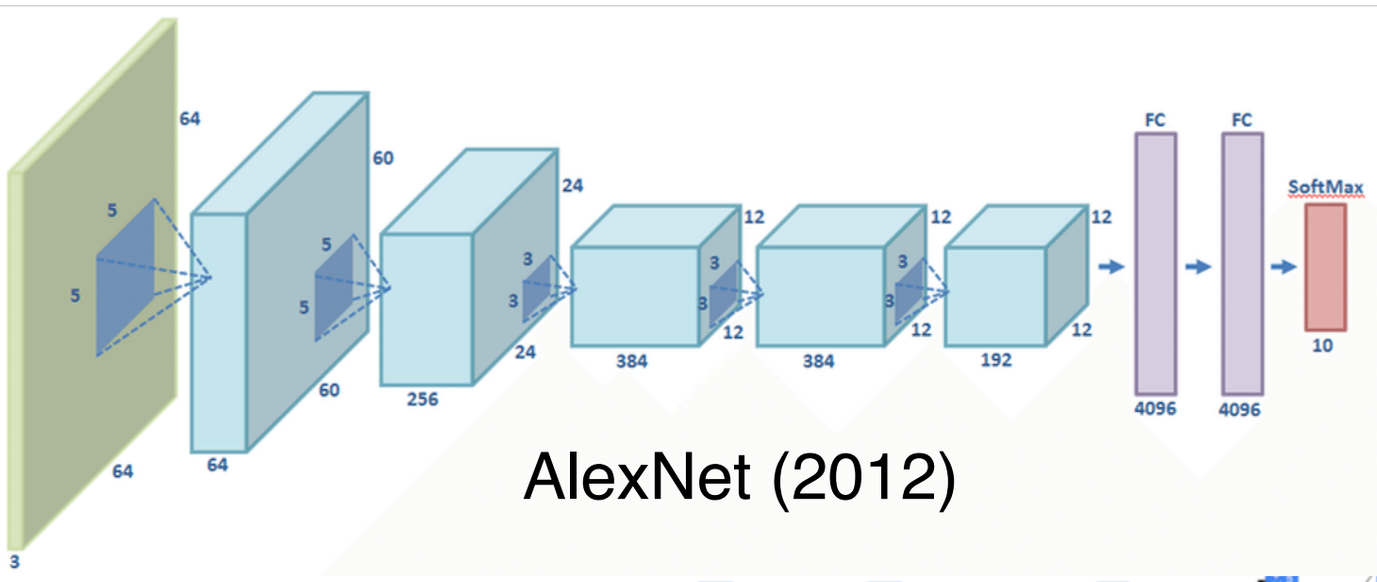
- Horizontale : toits
- Verticale : façades

AlexNet, 2012

ImageNet Visual Recognition Chalenge :

- ~10^6 images et ~1000 catégories
- 2 avancées majeures :
  - AlexNet (2012)
  - GoogleNet (2014)





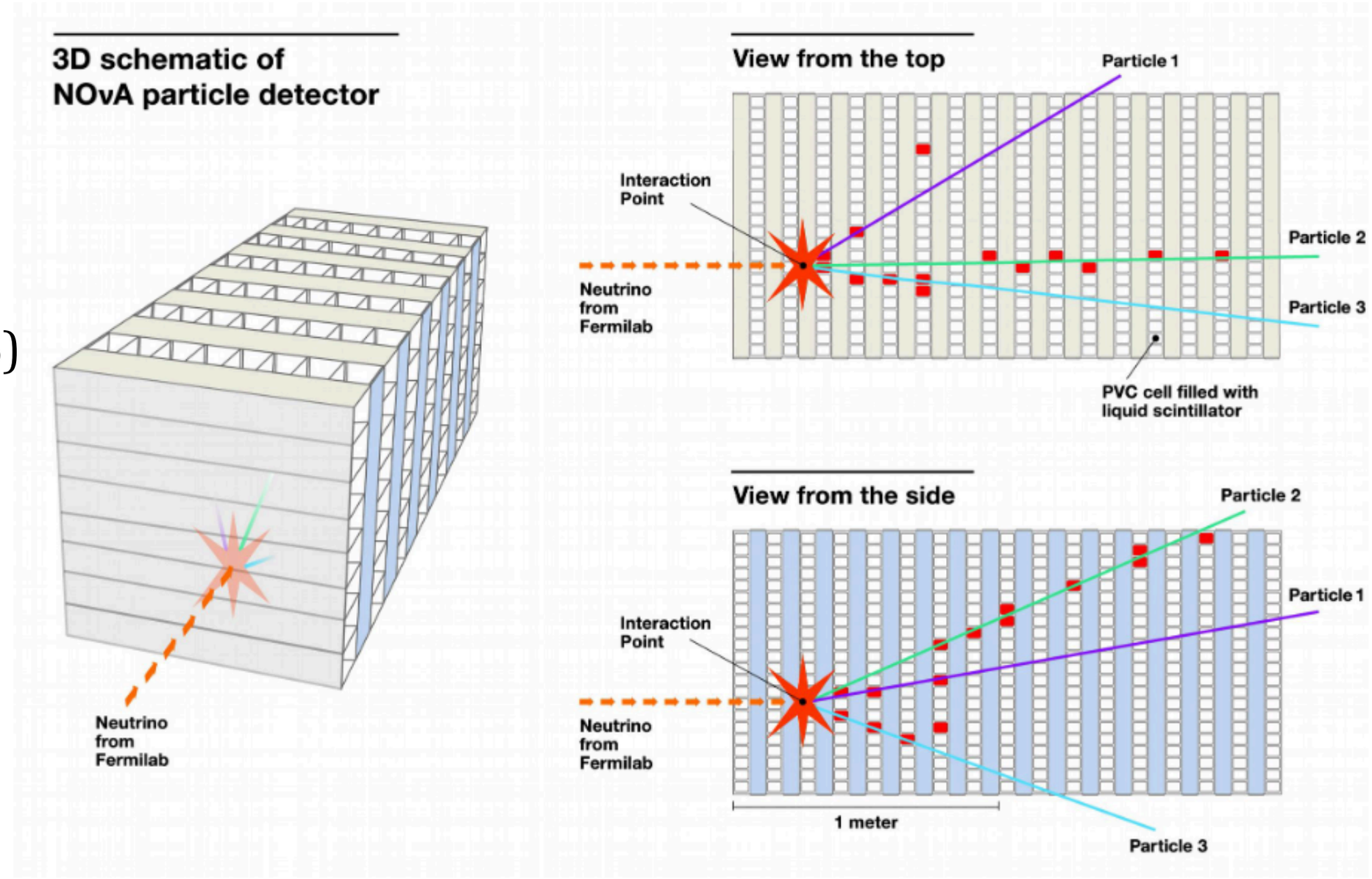
AlexNet (2012)

GoogleNet (2014)

Classification des événements dans l'experience Nova

Nova :

- Détecteur de neutrinos
- Basé à **Fermilab** (US)
- Étudie l'oscillation des **neutrinos**
- Mesure la désintégration de neutrino dans son détecteur

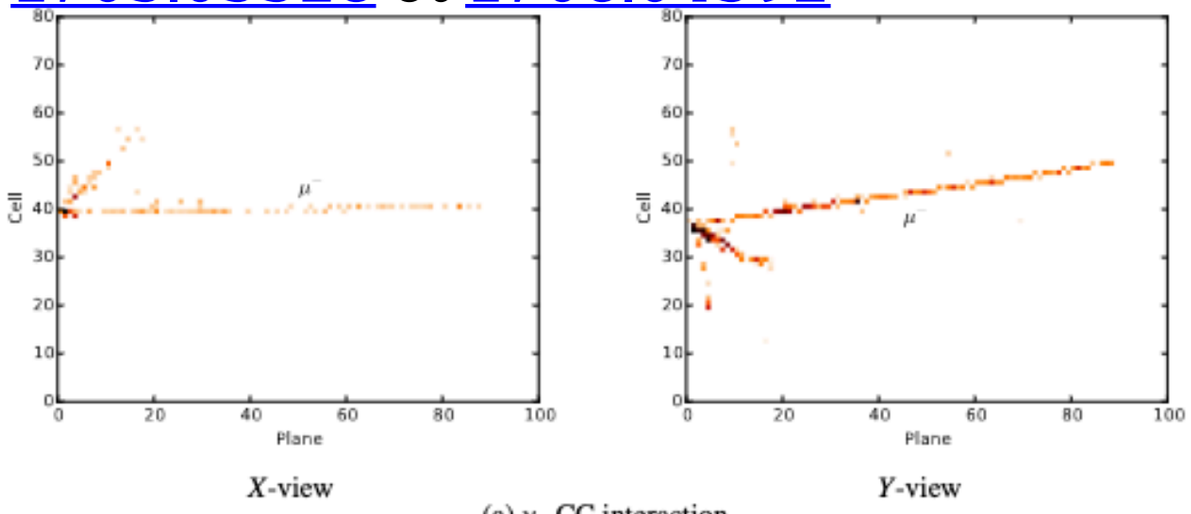


[A Convolutional Neural Network Neutrino Event Classifier]

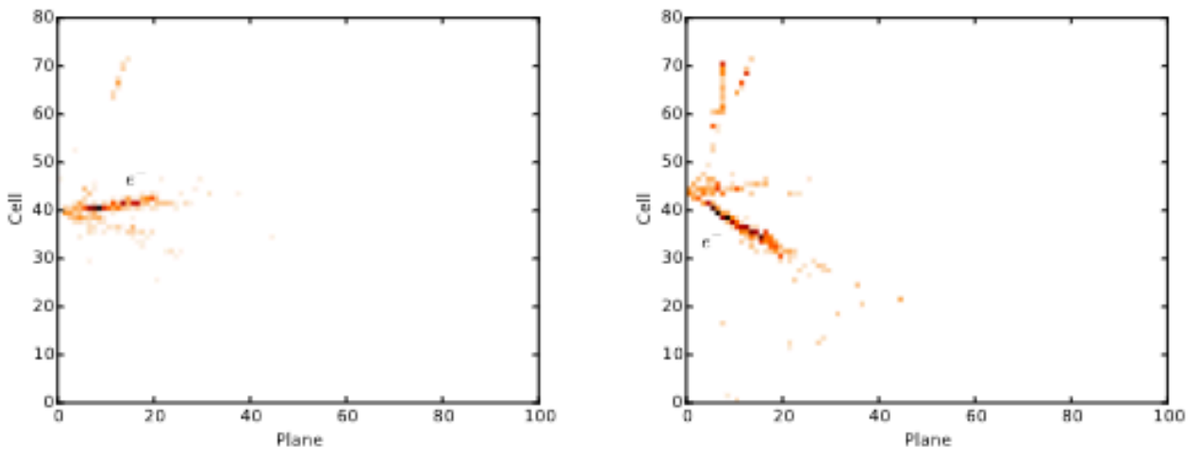
Classification des événements dans l'experience Nova

Utilisation de CNN pour la **classification des événements** :

- Inspiré par GoogleNet
- Difficulté : 2 images par input
- Utilisé dans des papiers avec des résultats de physique : [1703.03328](1703.03328) et [1706.04592](1706.04592)



(a) ν_μ CC interaction.



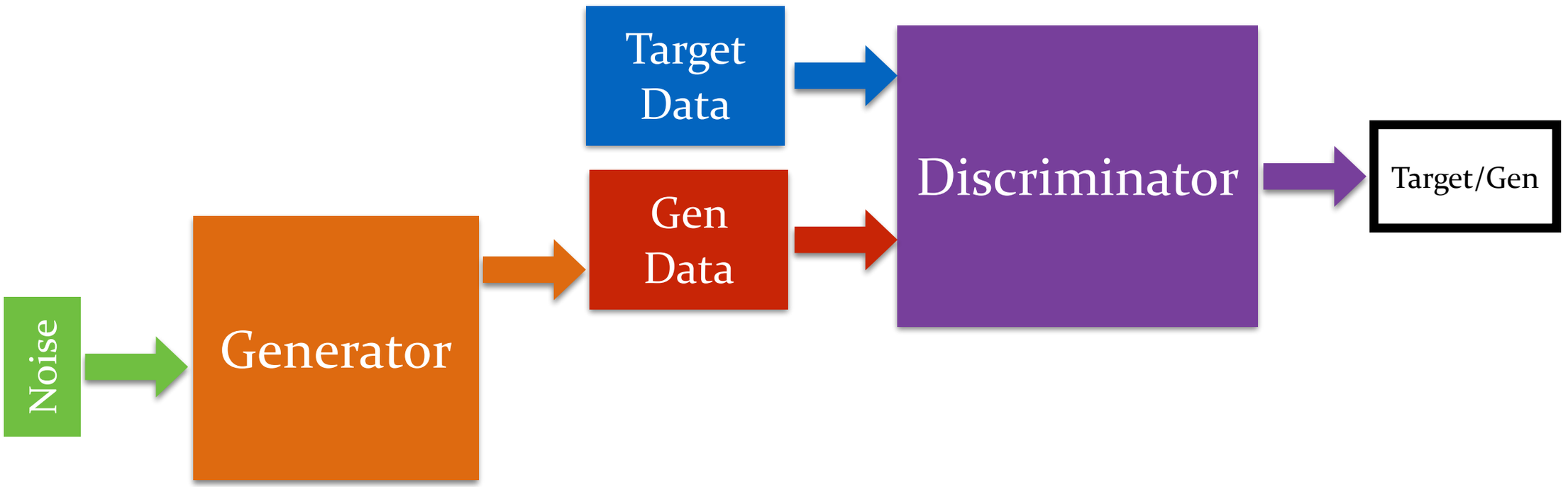
(b) ν_e CC interaction.

[A Convolutional Neural Network Neutrino Event Classifier]

Les Réseaux Antagonistes Génératifs (GAN)

- Technique **générative** : crée des données/images à partir de bruit

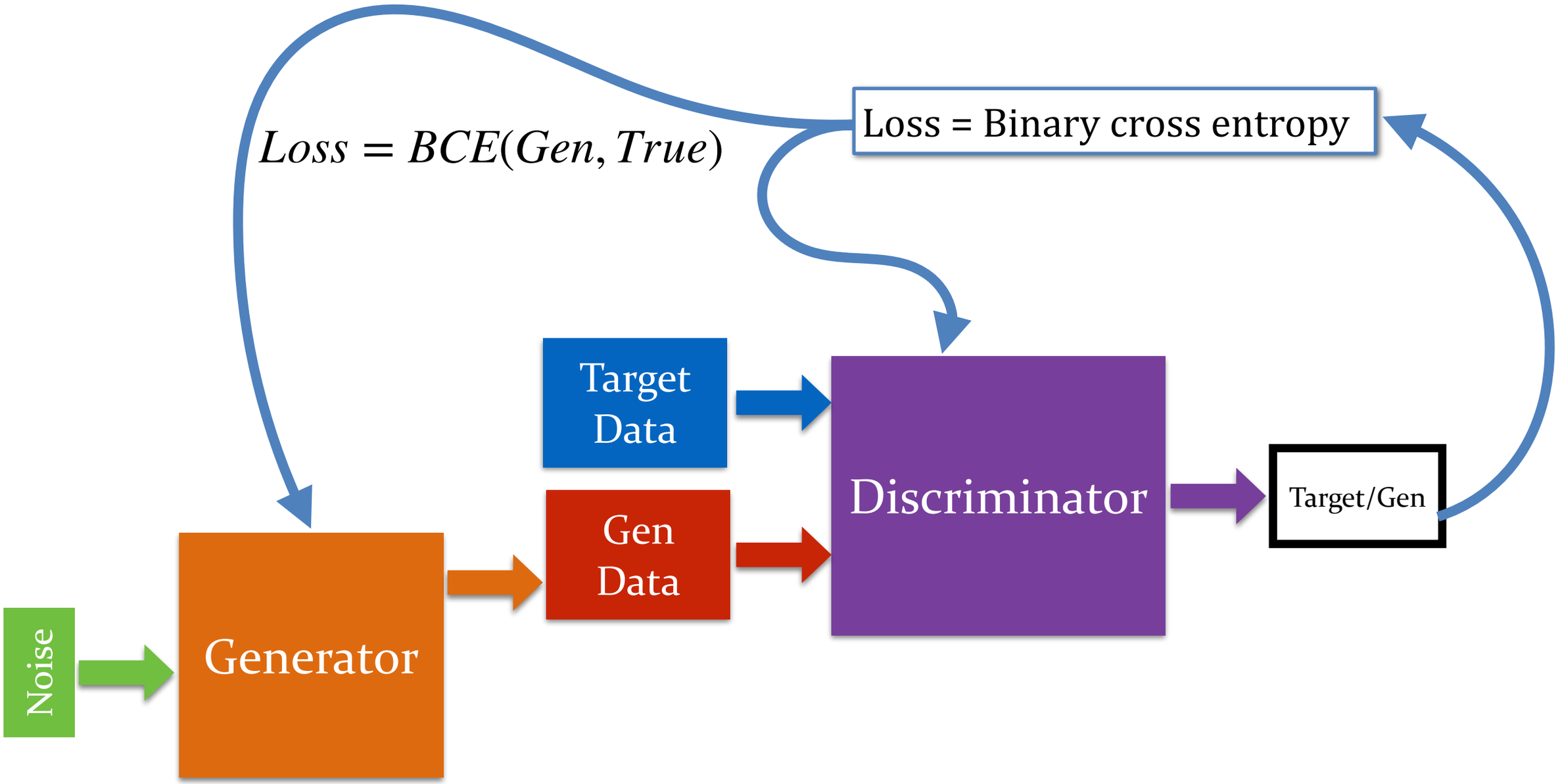
- Utilise deux réseaux de neurones :
  - **Générateur** : génère des données à partir de bruit
  - **Discriminateur** : sépare le bruit des données cibles



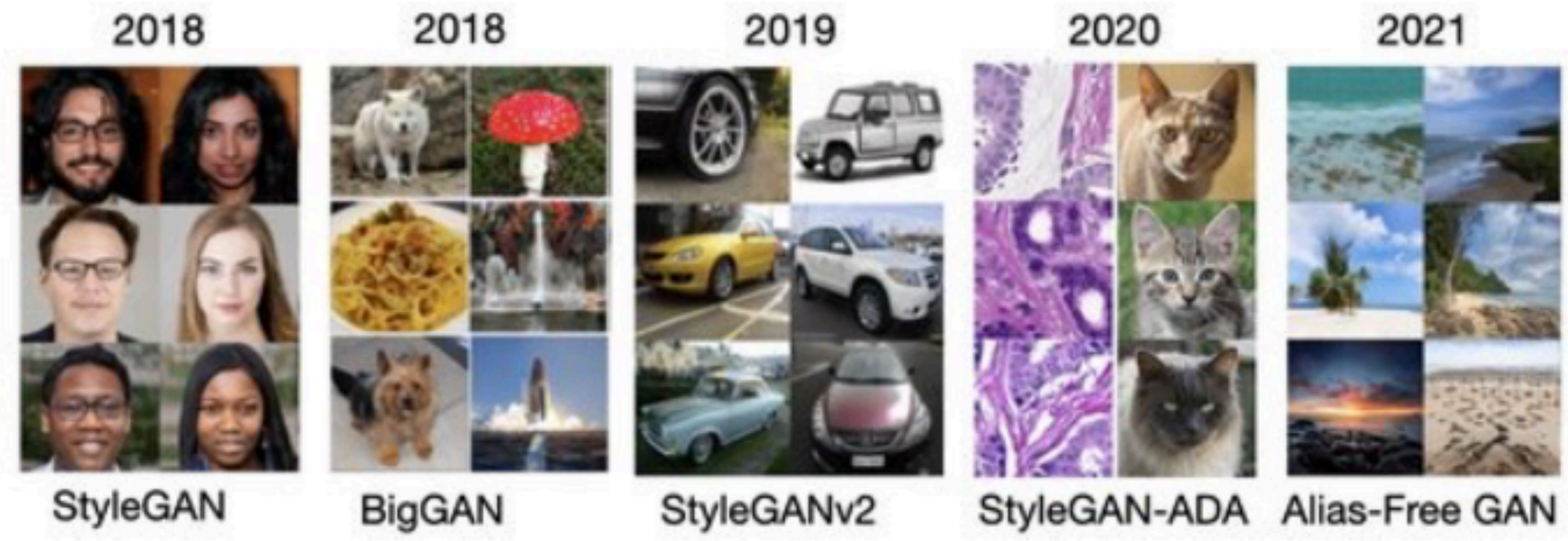
Les Réseaux Antagonistes Génératifs (GAN)

Même fonction de coût pour le discriminateur et le générateur :

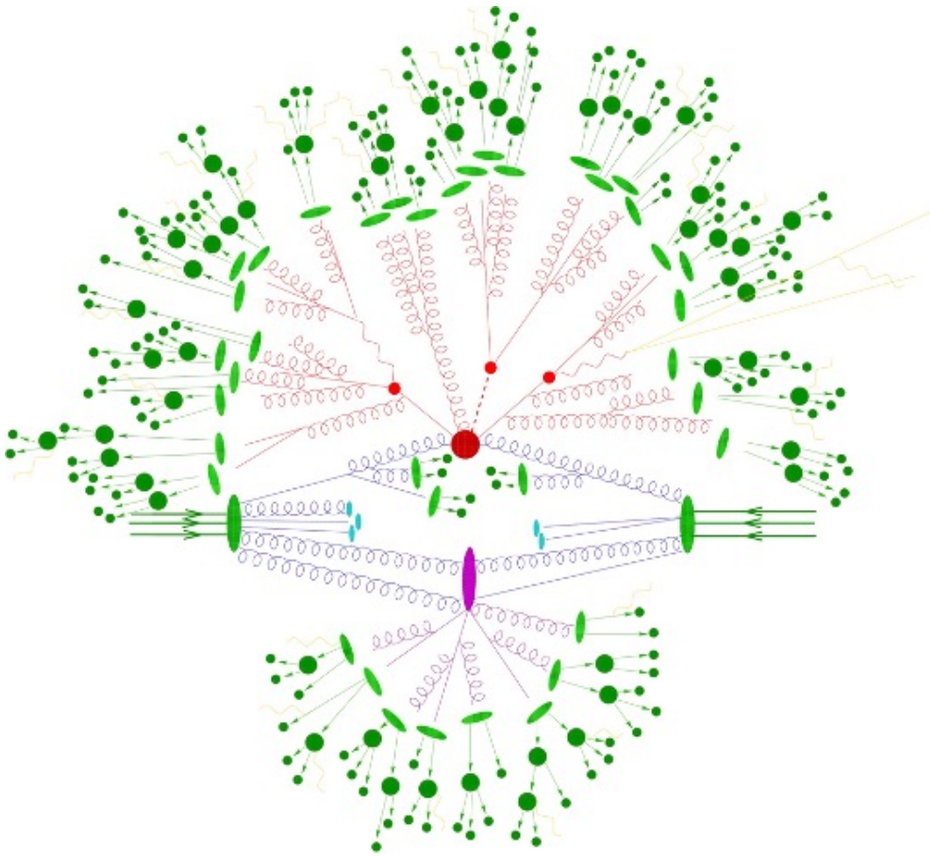
- Le discriminateur essaie de **séparer** les vraies entrées des entrées simulées
- Le générateur essaie de **tromper** le discriminateur



Les Réseaux Antagonistes Génératifs (GAN)



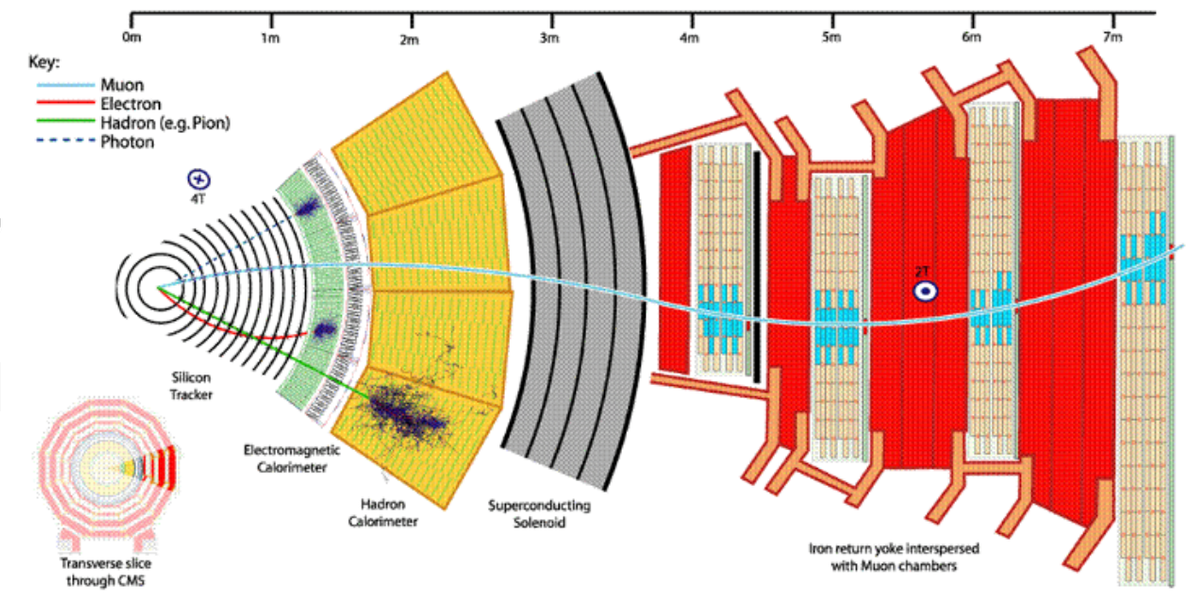
Simulation des événements dans ATLAS/CMS



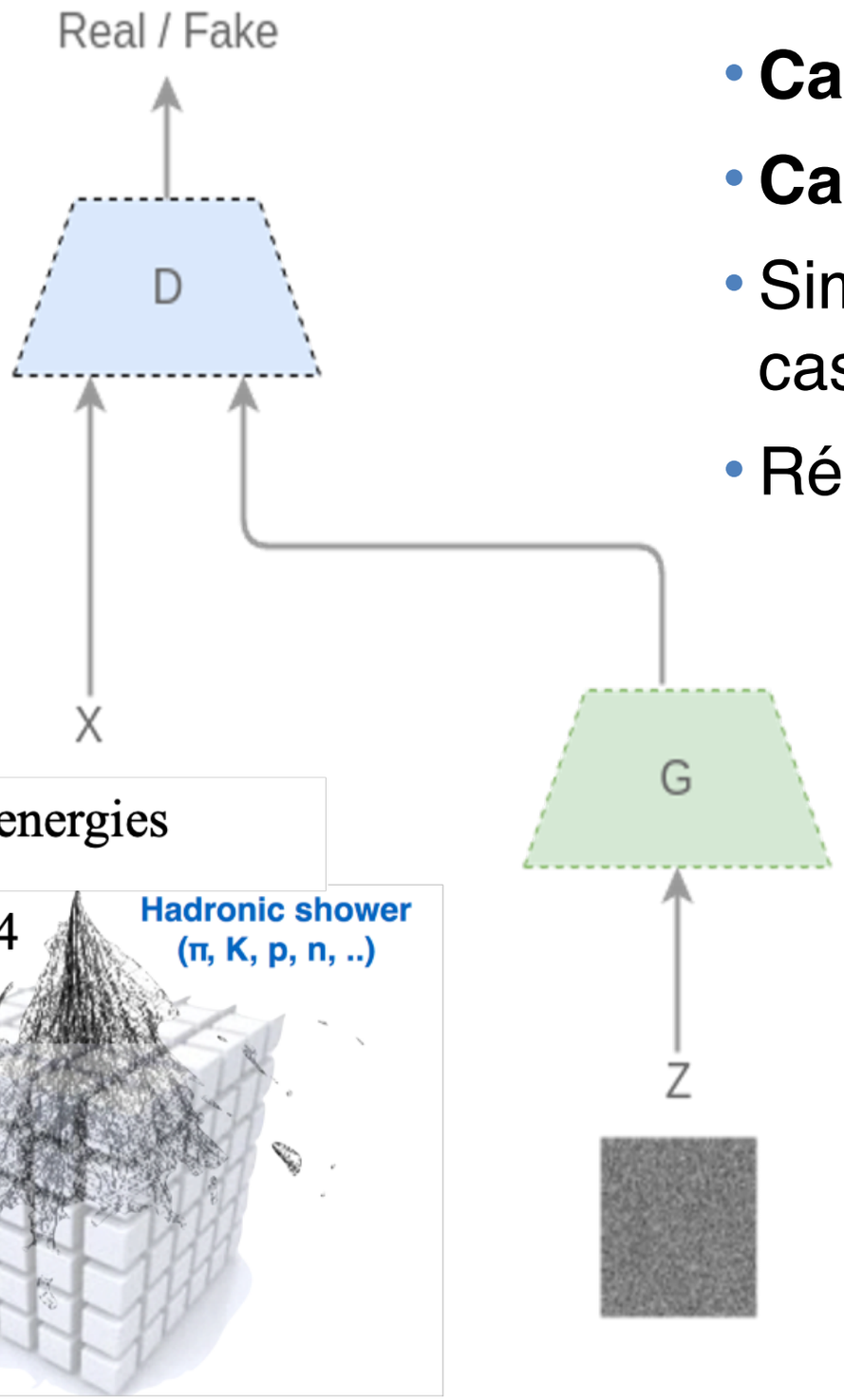
Simulateurs :

- Combine une connaissance précise de la physique pour prendre en compte l'interaction particules/matière (**Geant4**)
- Simulation **extrêmement précise** de nos détecteurs

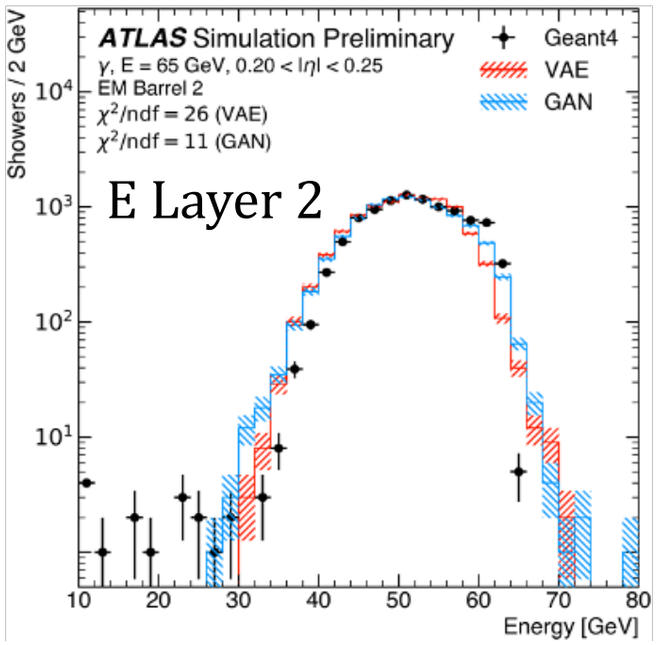
- Demande beaucoup de **ressources** :
  - Humaine (optimisation de la simulation, étalonnage…)
  - CPU (50% de nos ressources de calcul vont dans la simulation)

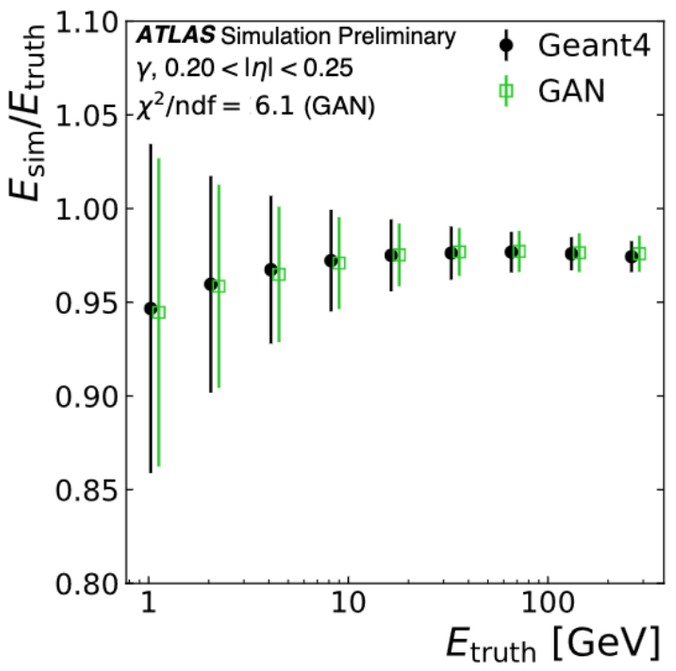


Simulation de calorimètre à l'aide de GAN



- **Calorimètre** : détecte l'énergie des particules
- **Cascades** de particules à l'intérieur
- Simulation du dépôt d'énergie de ces cascades avec un **GAN**
- Résultat réaliste obtenu **100x** plus rapidement





[Deep generative models for fast shower simulation in ATLAS]

Conclusion

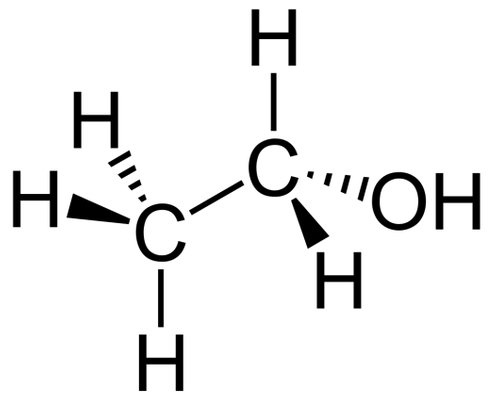
- Deep learning = Machine learning avec des réseaux de neurones
- Capable d'apprendre très efficacement de nos données
- Très grande variété d'algorithmes de machine learning ➡ adapter nos algorithmes à nos problèmes
- De plus en plus omniprésente en physique pour les tâches de reconstruction d'objets, de simulation…

Pour aller plus loin

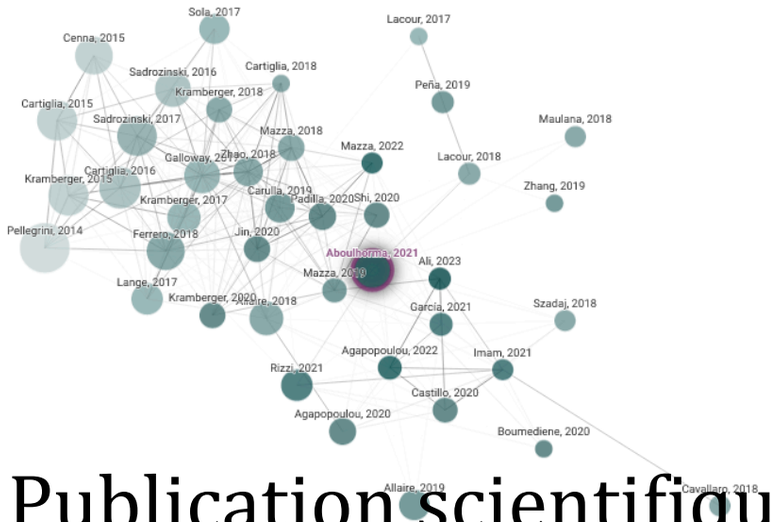
Graph Neural Network

Objectif : adapter la **structure** de notre réseau à la structure de nos données.

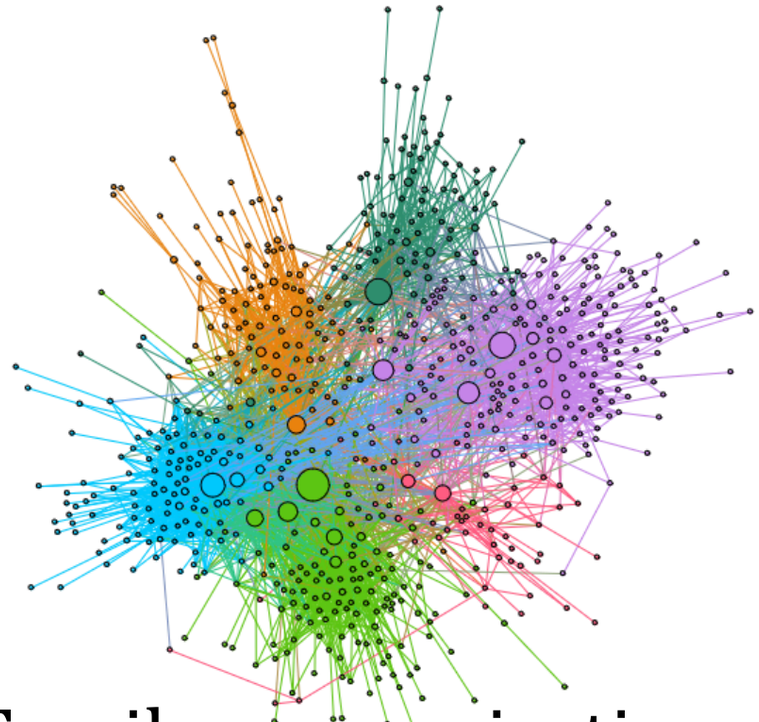
Grand nombre de structures représentable par des **graphes**



Molécule



Publication scientifique



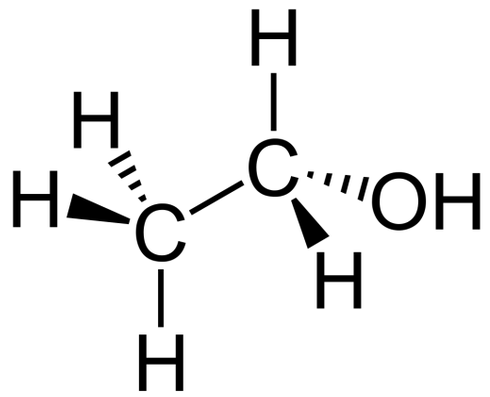
Email communication

Relation sociale

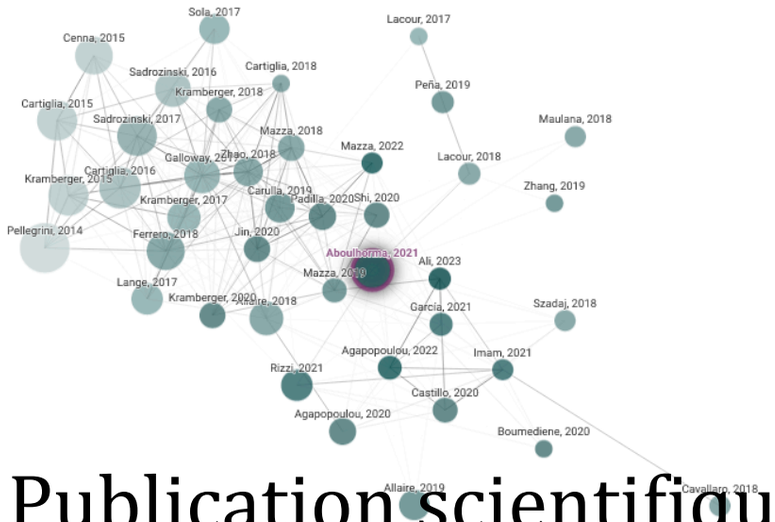
Graph Neural Network

Objectif : adapter la **structure** de notre réseau à la structure de nos données.

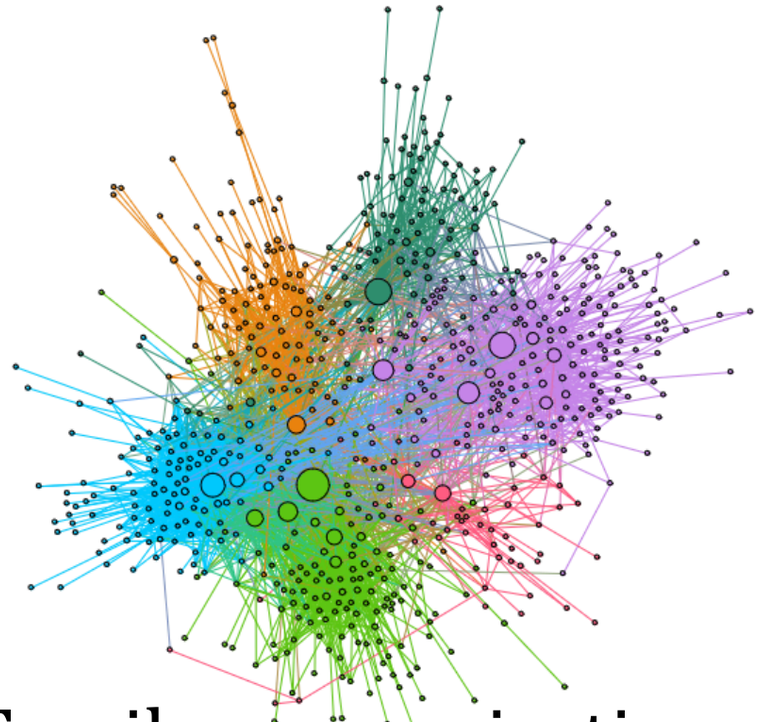
Grand nombre de structures représentable par des **graphes**



Molécule



Publication scientifique

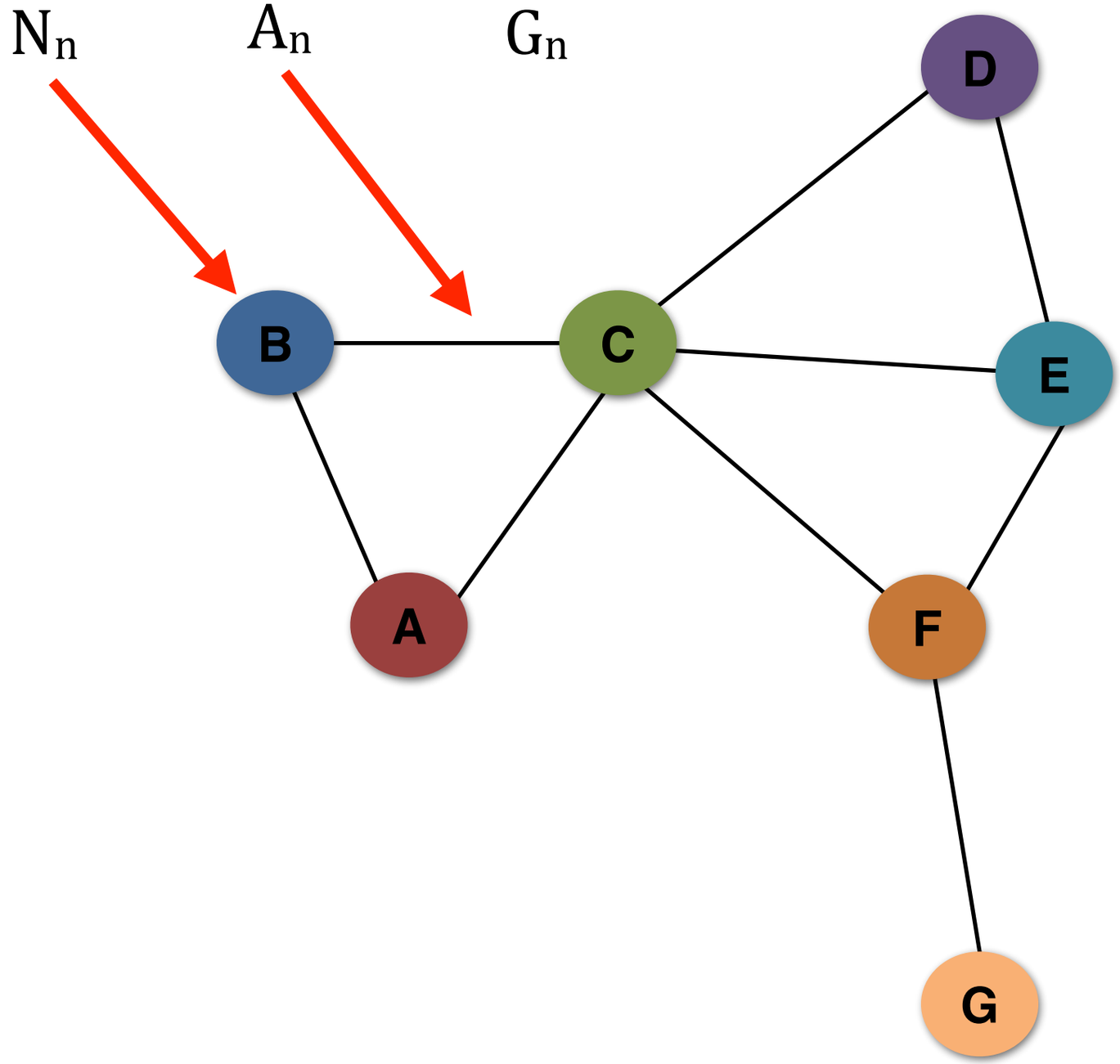


Email communication

Relation sociale

Graph Neural Network

Graphe : 3 type d'info : **Noeuds**, **arêtes** et **globale**



Graph Neural Network

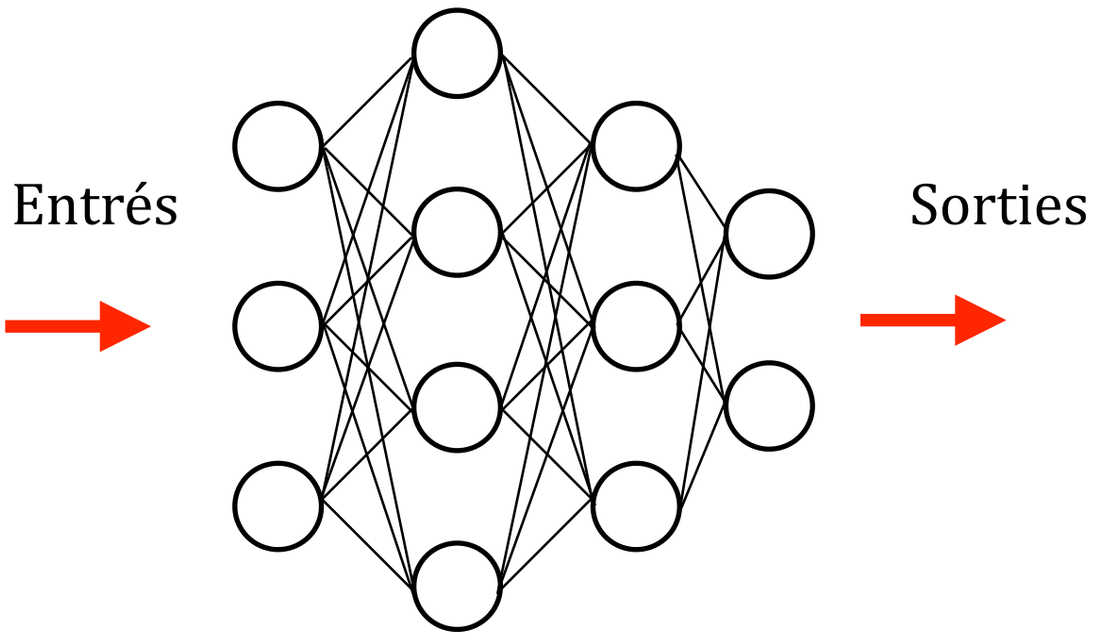
Graphe : 3 type d'info : **Noeuds**, **arêtes** et **globale**

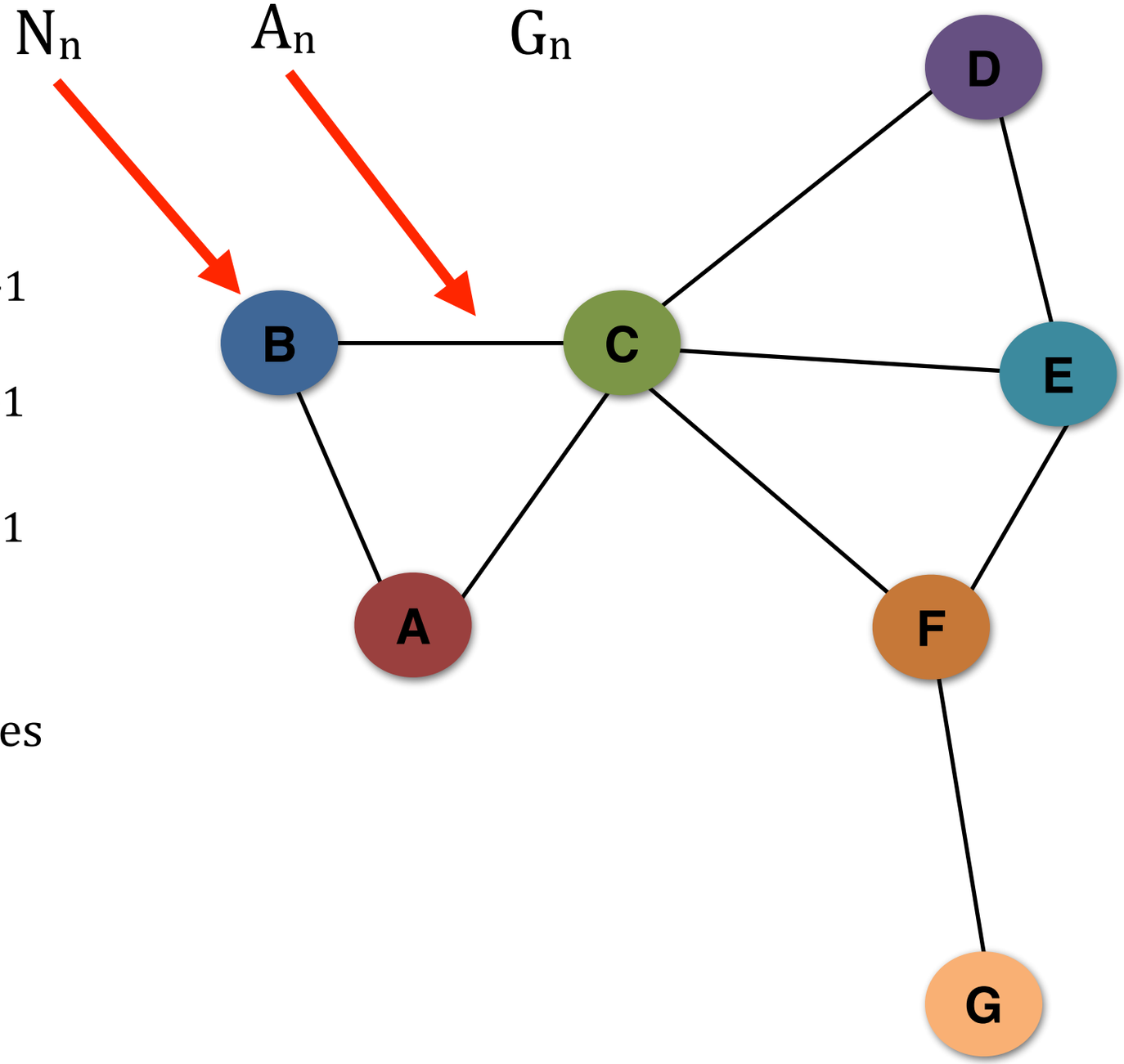
Application d'un NN à ces variables :

$$N_n \xrightarrow{f_N} N_{n+1}$$

$$A_n \xrightarrow{f_A} A_{n+1}$$

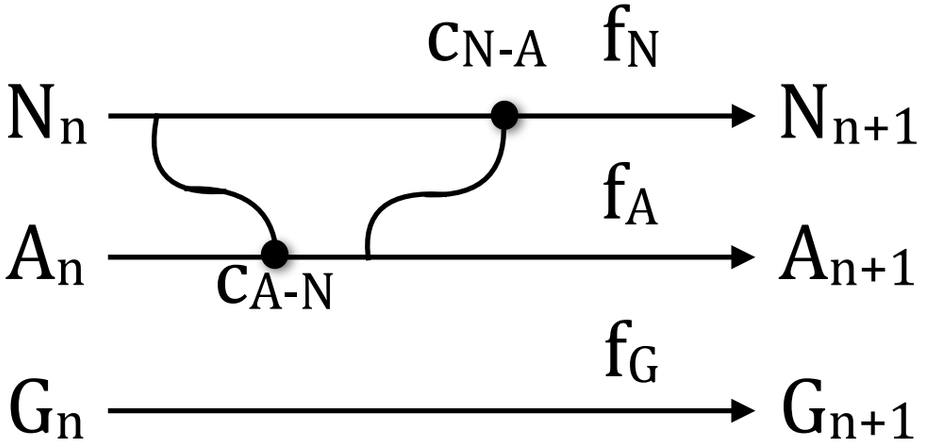
$$G_n \xrightarrow{f_G} G_{n+1}$$

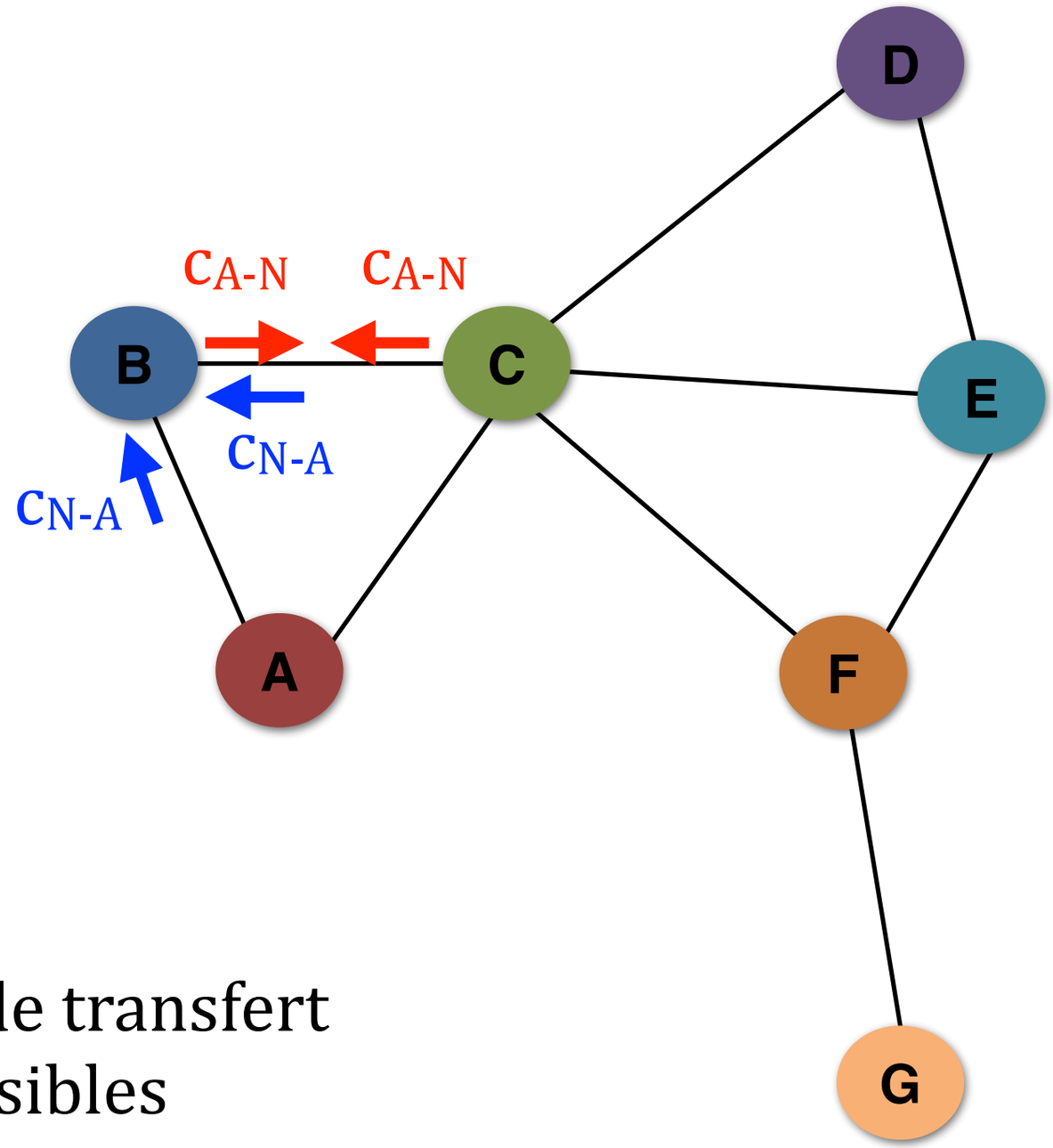




Graph Neural Network : Message Passing

Transfert d'information entre les éléments du graph avec une **fonction d'agrégation** :





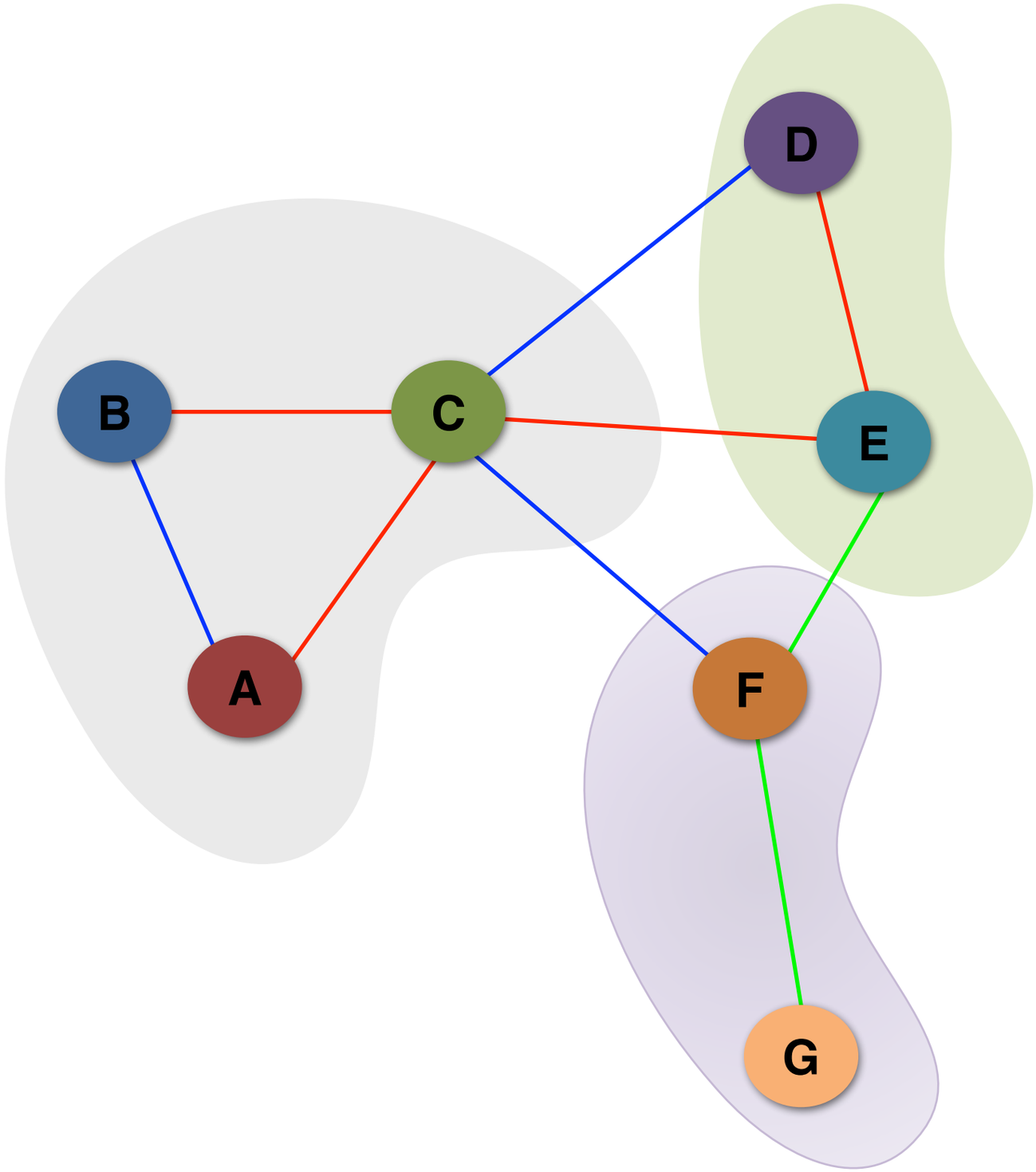
Différentes méthodes de transfert d'information sont possibles

Graph Neural Network : Classification

Utilisation d'un dernier NN pour la classification des Noeuds, des arêtes ou du graph

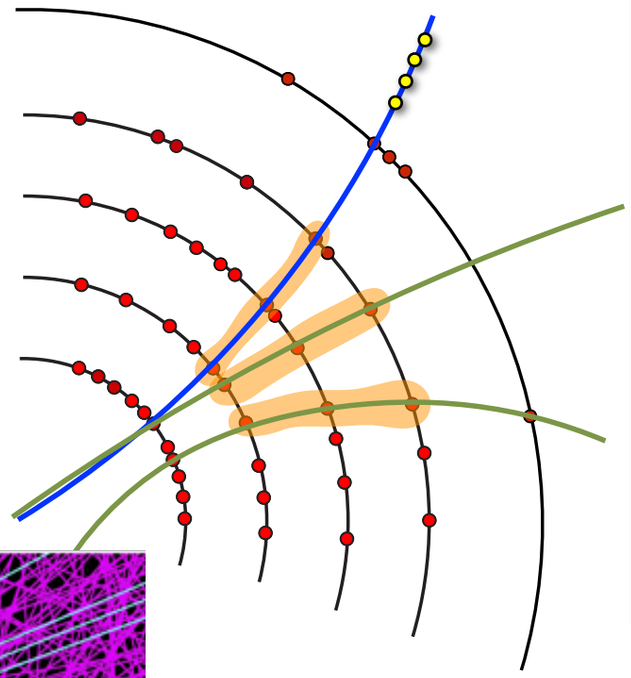
Exemples *(publications)* :

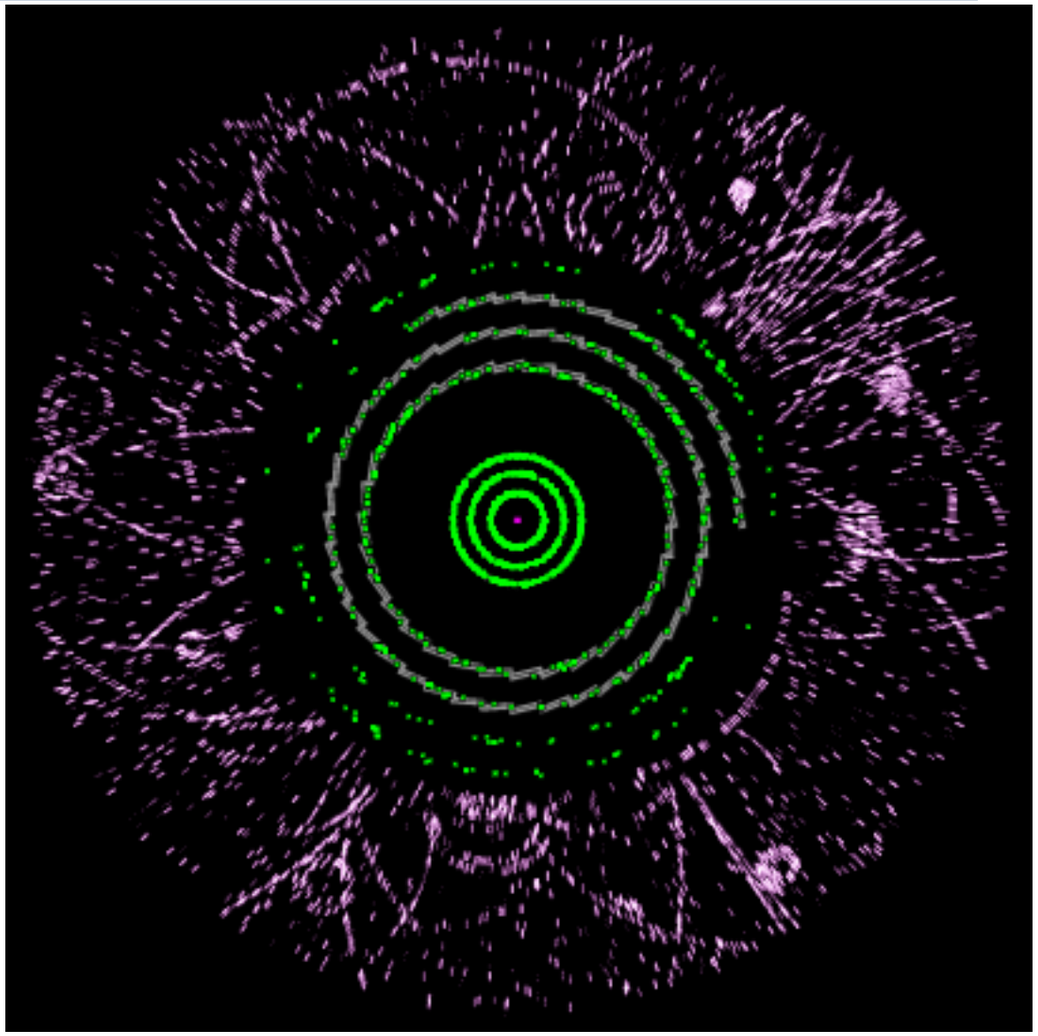
- Noeuds : Quelle catégorie de noeuds ?
  (laboratoires)
- Arêtes : Quel lien entre noeuds ?
  (relations professionnelles)
- Graph : Quel type de graph/ structure ?
  (stratégie de publication)

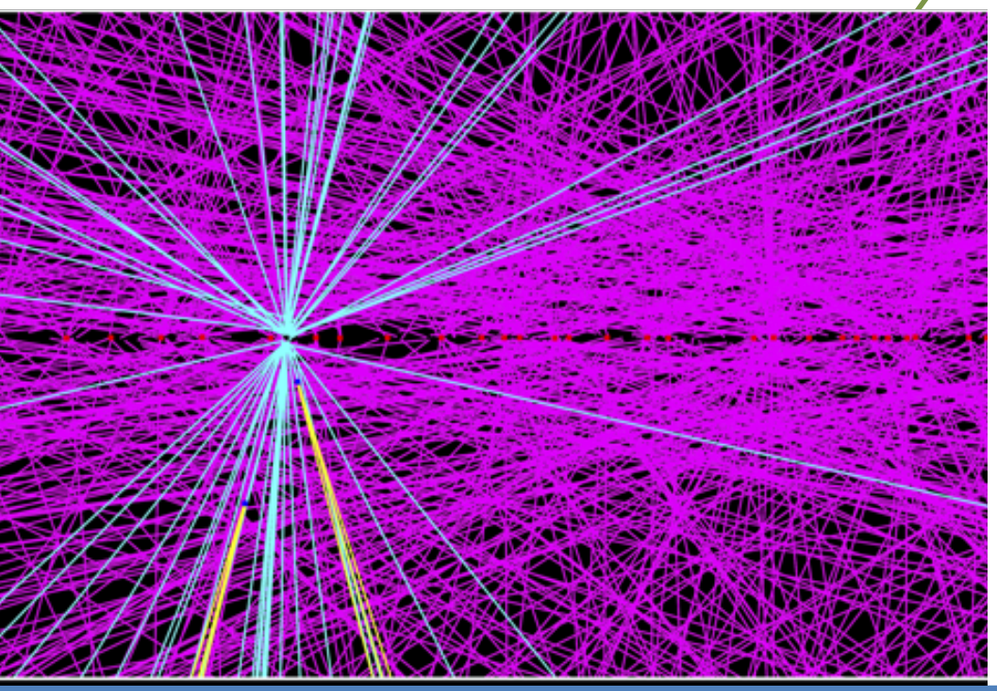


Trajectographie des particules chargées

- Particules chargées : **dépôt d'énergie** dans le détecteur (coups)
- **Trajectographie** : connecter ensemble ces coups pour reconstruire une trajectoire



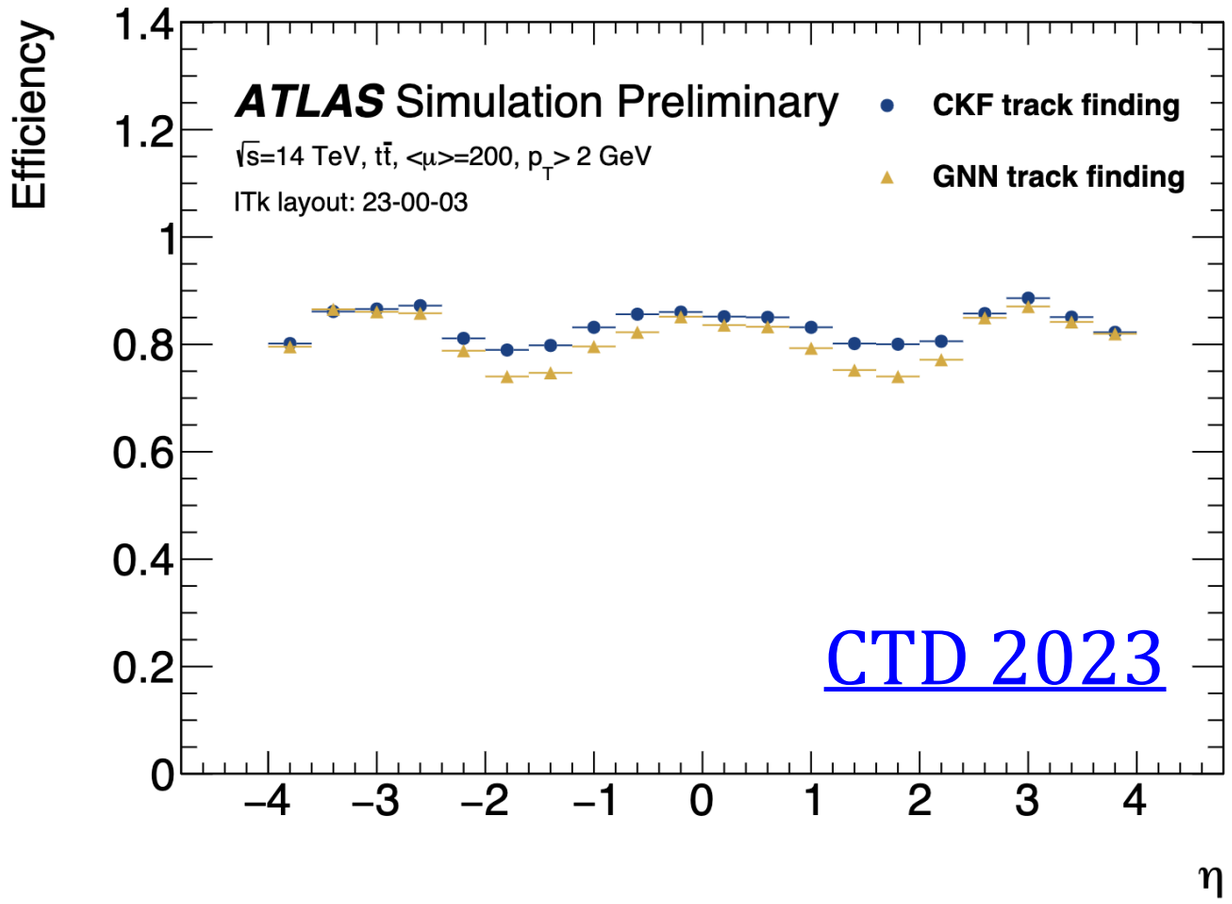




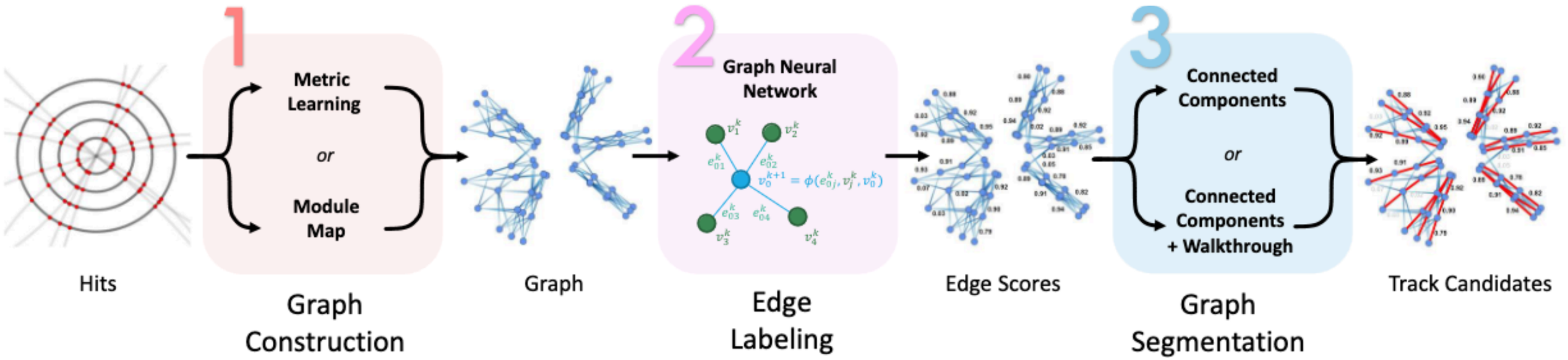
- Très complexe : beaucoup de **combinatoire** (~ 100 000 coups par collision)
- Algorithme très complexe : **Filtres de Kalman**
- Très fort coût CPU

GNN4ITk : Trajectographie avec un GNN

- Appliqué au futur **trajectographe** d'ATLAS (ITk)
- **Coups = Noeuds**
- Connexion entre coups = arrêtes
- **Classification** d'arête : bonnes arêtes ➡ trajectoire d'une particule
- Performance compatible avec les algorithmes classique



CTD 2023



Physics Performance of the ATLAS GNN4ITk Track Reconstruction Chain