Principe et Applications du Machine Learning (2)

Corentin Allaire

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

A 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
  - L'auto-encoder
  - Réseaux à base de graphe
- Application à la reconstruction d'objet en physique des hautes énergies

A quoi s'attendre

Machine Learning :

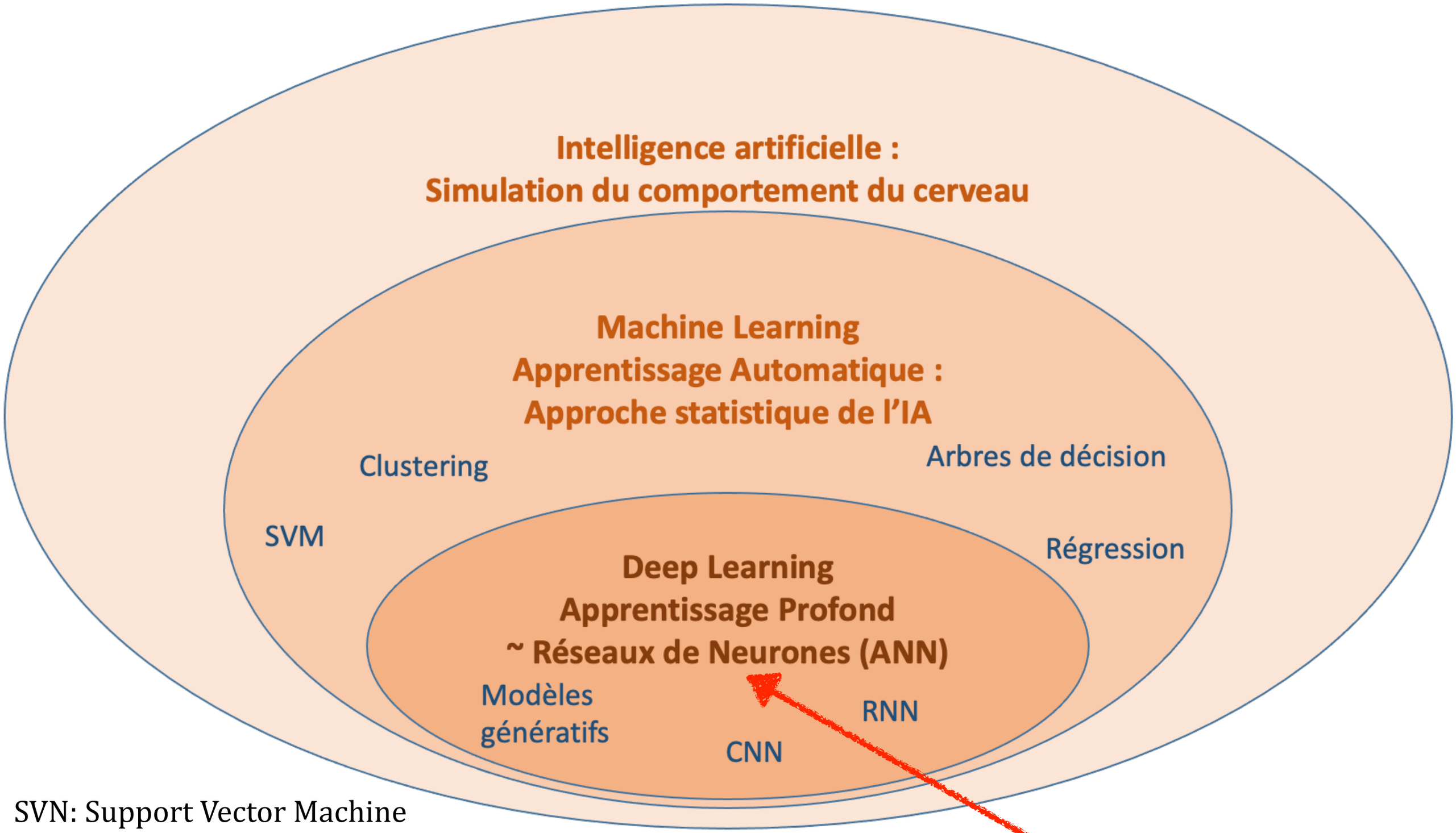
- Concepts fondamentaux de « L'IA »
- Les différentes techniques de Machine Learning :
  - Supervisé
  - Non-supervisé
- Application à la recherche du boson de Higgs

Deep Learning :

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

Aujourd'hui

Une question de nomenclature



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

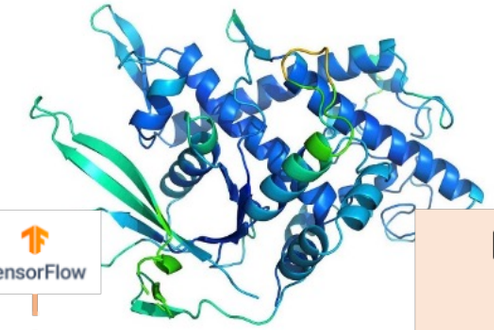
Une brève histoire du Deep Learning

Perceptron

Deep Blue

Expert systems

AlexNet

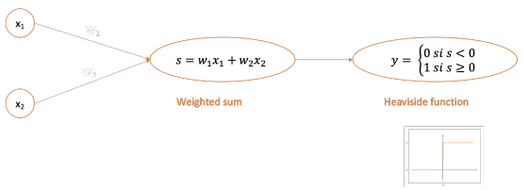


Diffusion models

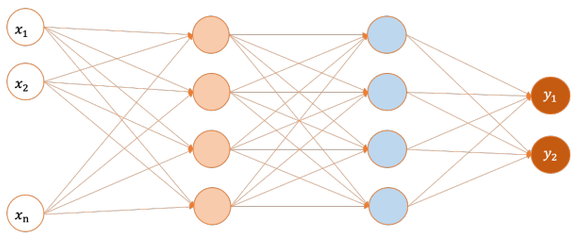
AlphaFold2

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

Formal neuron



Multi-Layer Perceptron
Gradient Back Propagation

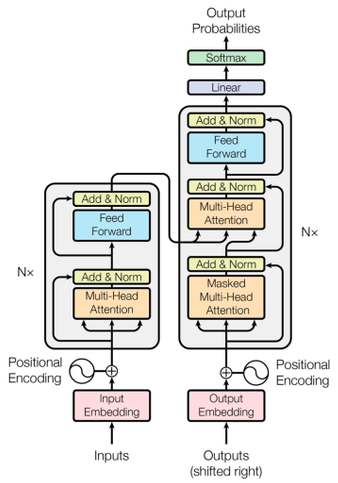


- GoogleBrain
- Watson
- Siri

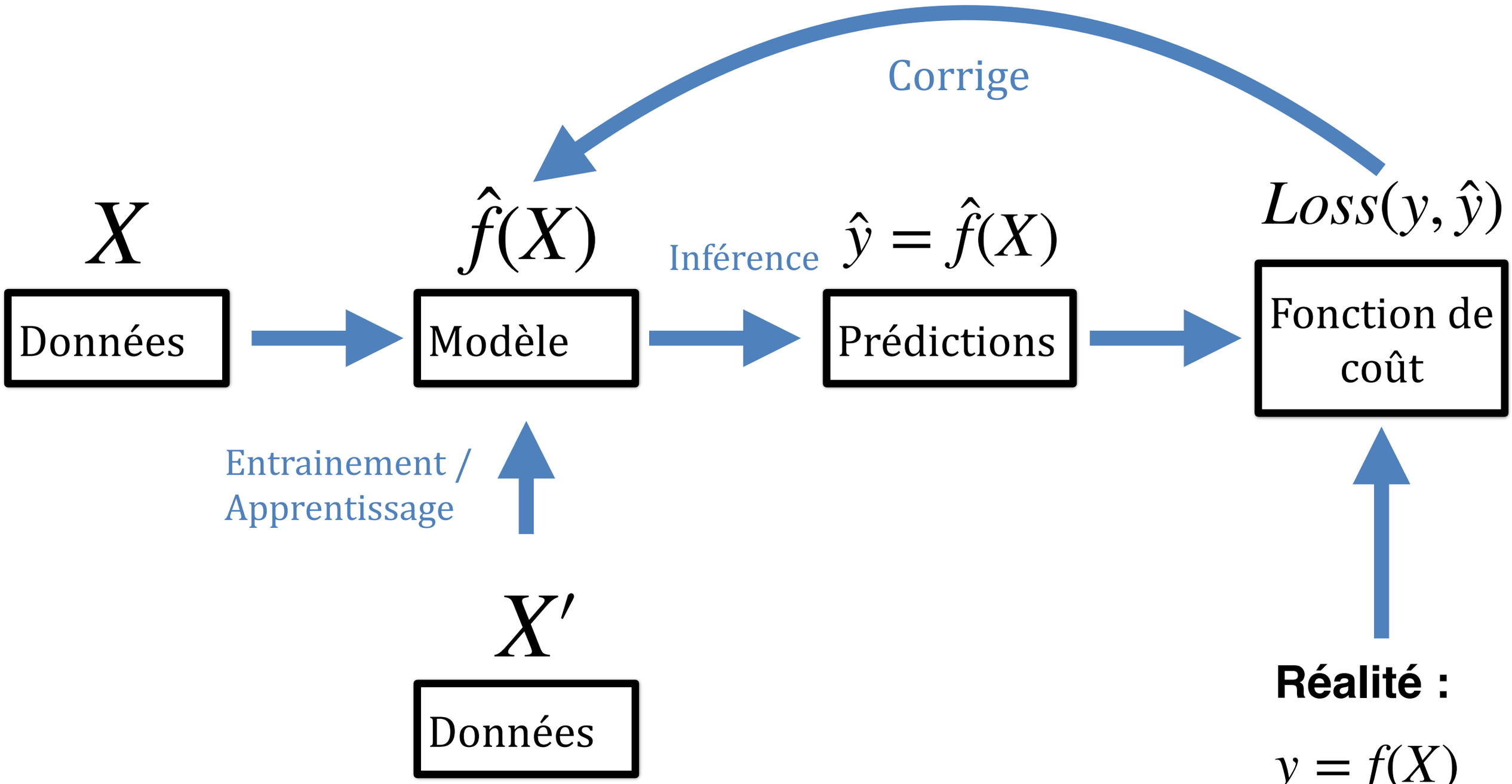
AlphaGo (DeepMind)

Transformers

GPT3

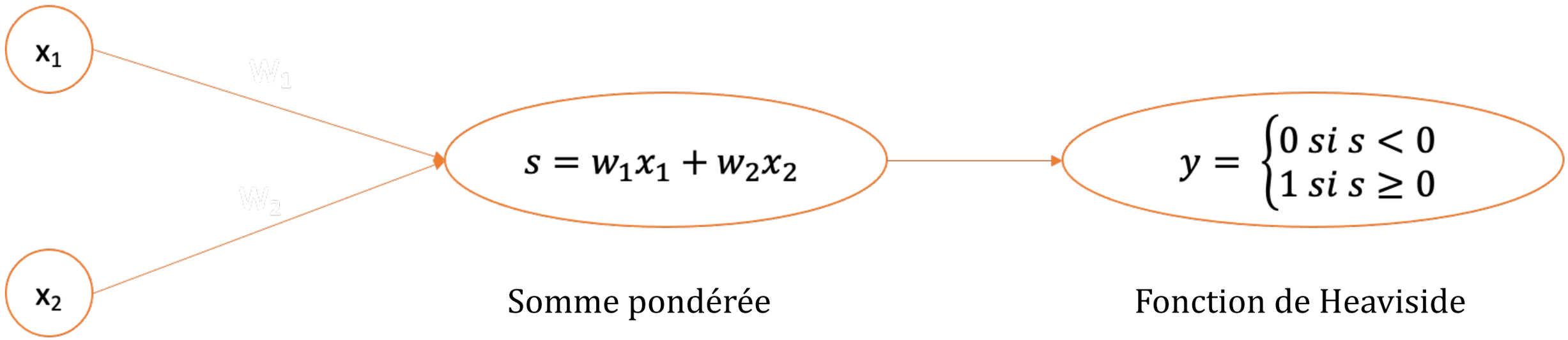


Entrainement supervisé

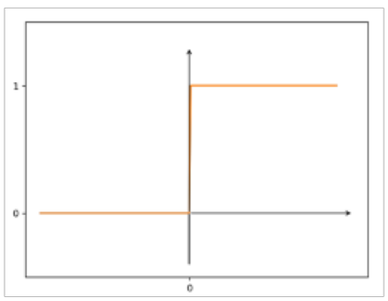


Le neurone formel

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

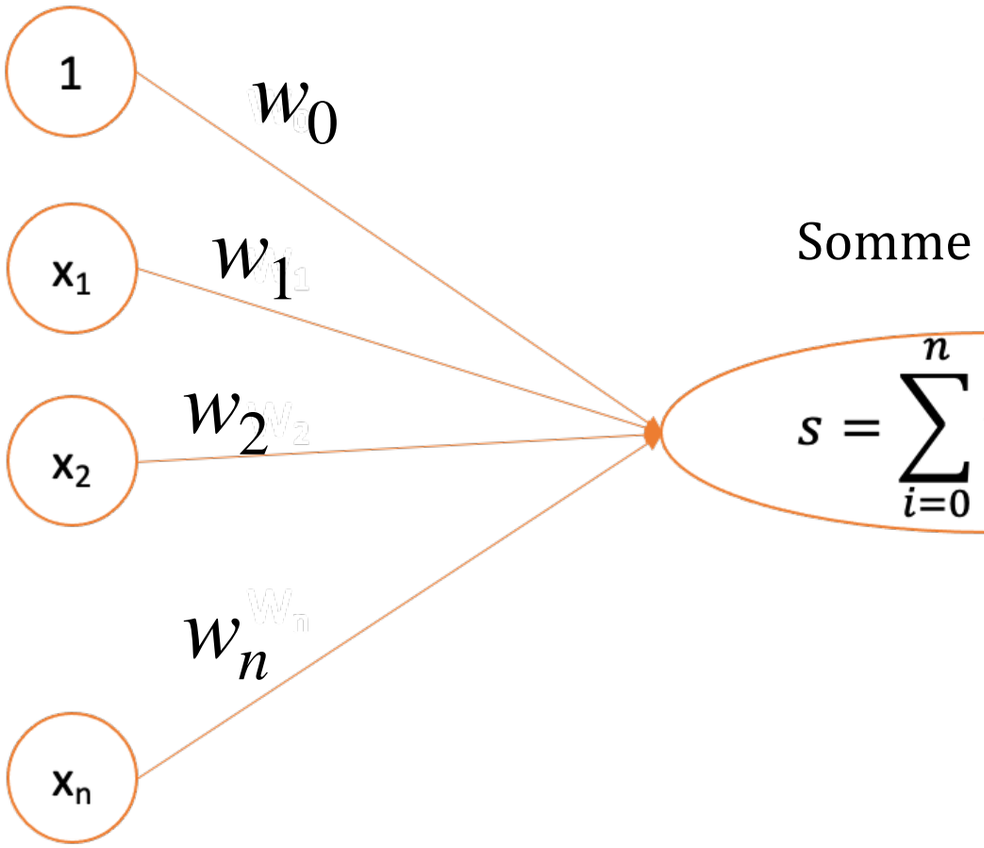


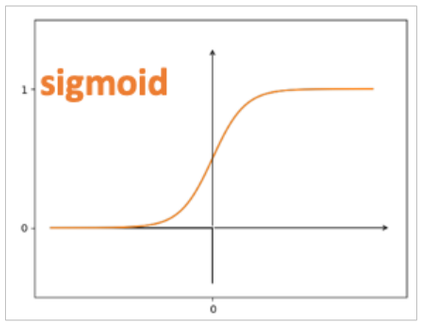
1943 : McCulloch & Pitts

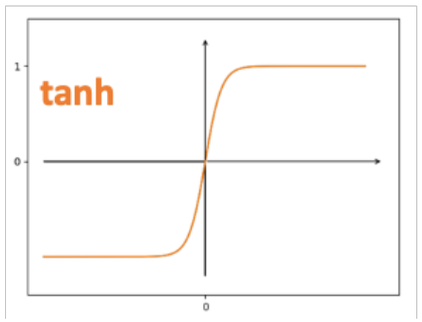


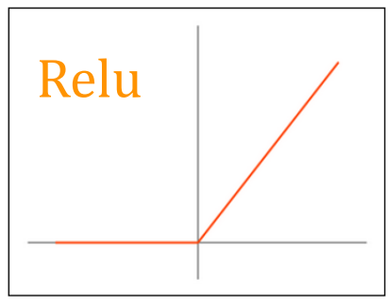
- Retourne uniquement des entrée **booléennes** (0,1)
- Assemblage de neurone : **Turing complets**

Le neurone formel : généralisation



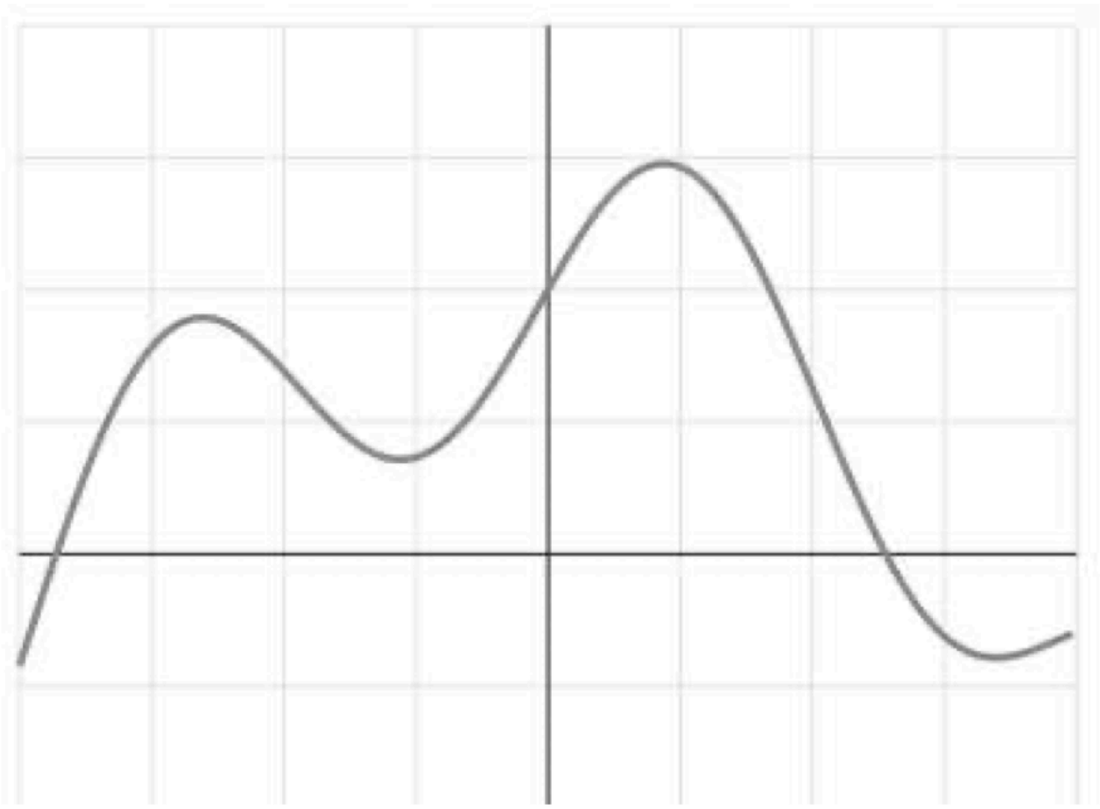


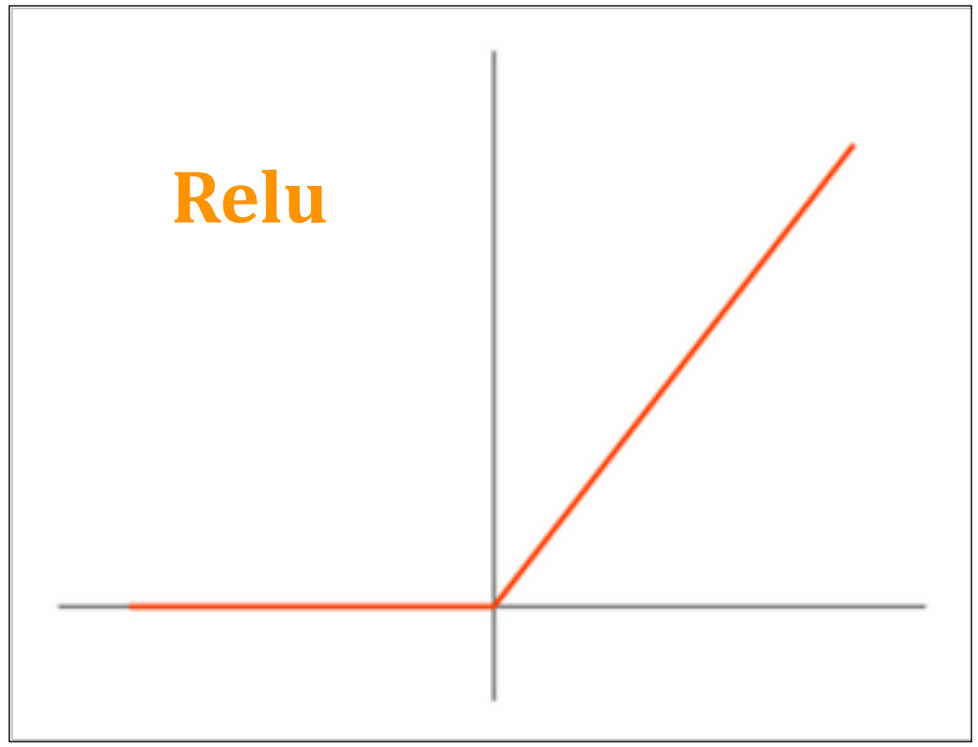




- Nombre d'entrées arbitraire
- Addition d'une constante par neurone : w_0
- **Fonction d'activation** appliquée au résultat du neurone : différents intervalles de valeur possible

Le neurone formel : approximateur universel

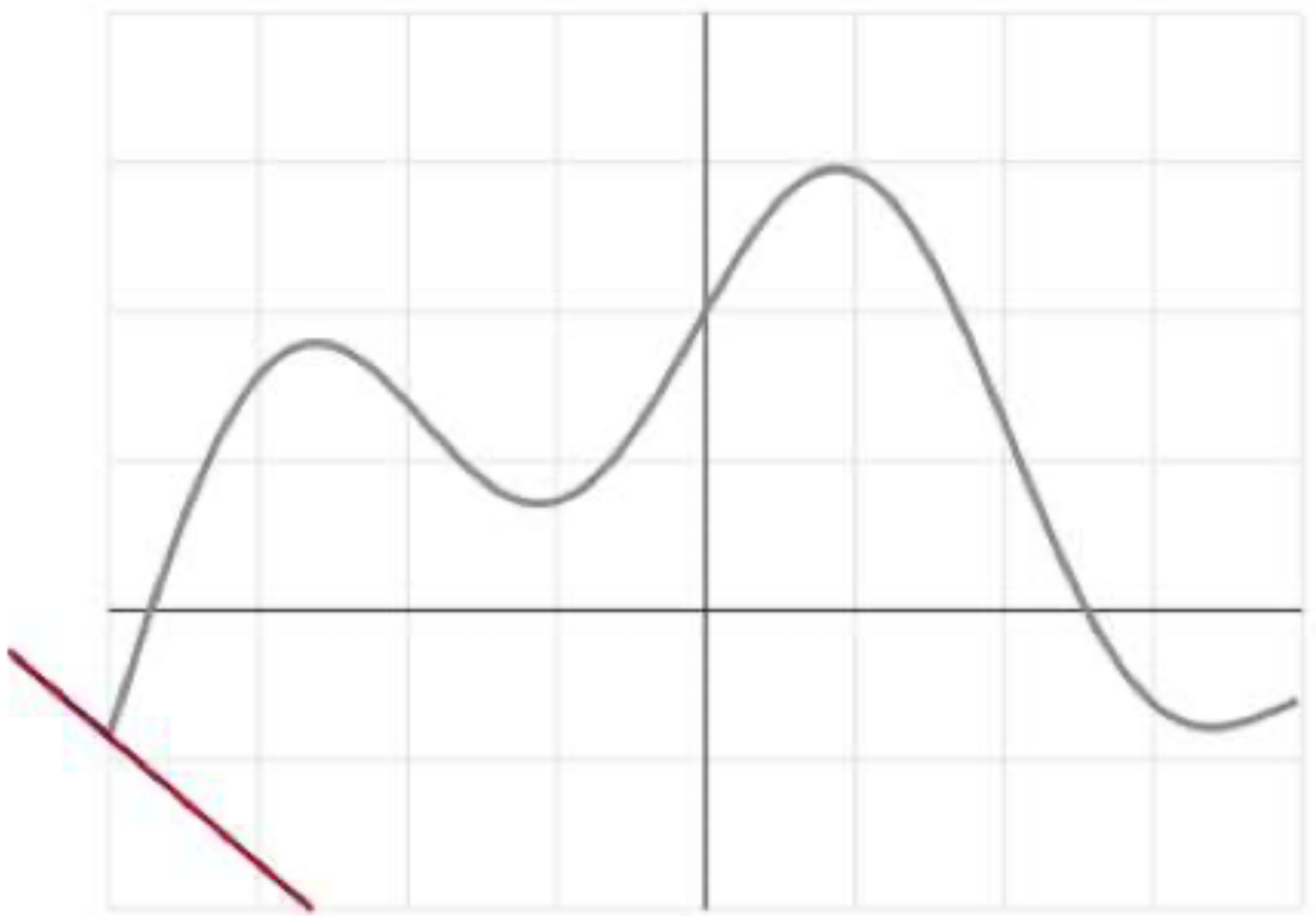




Rectified Linear Unit

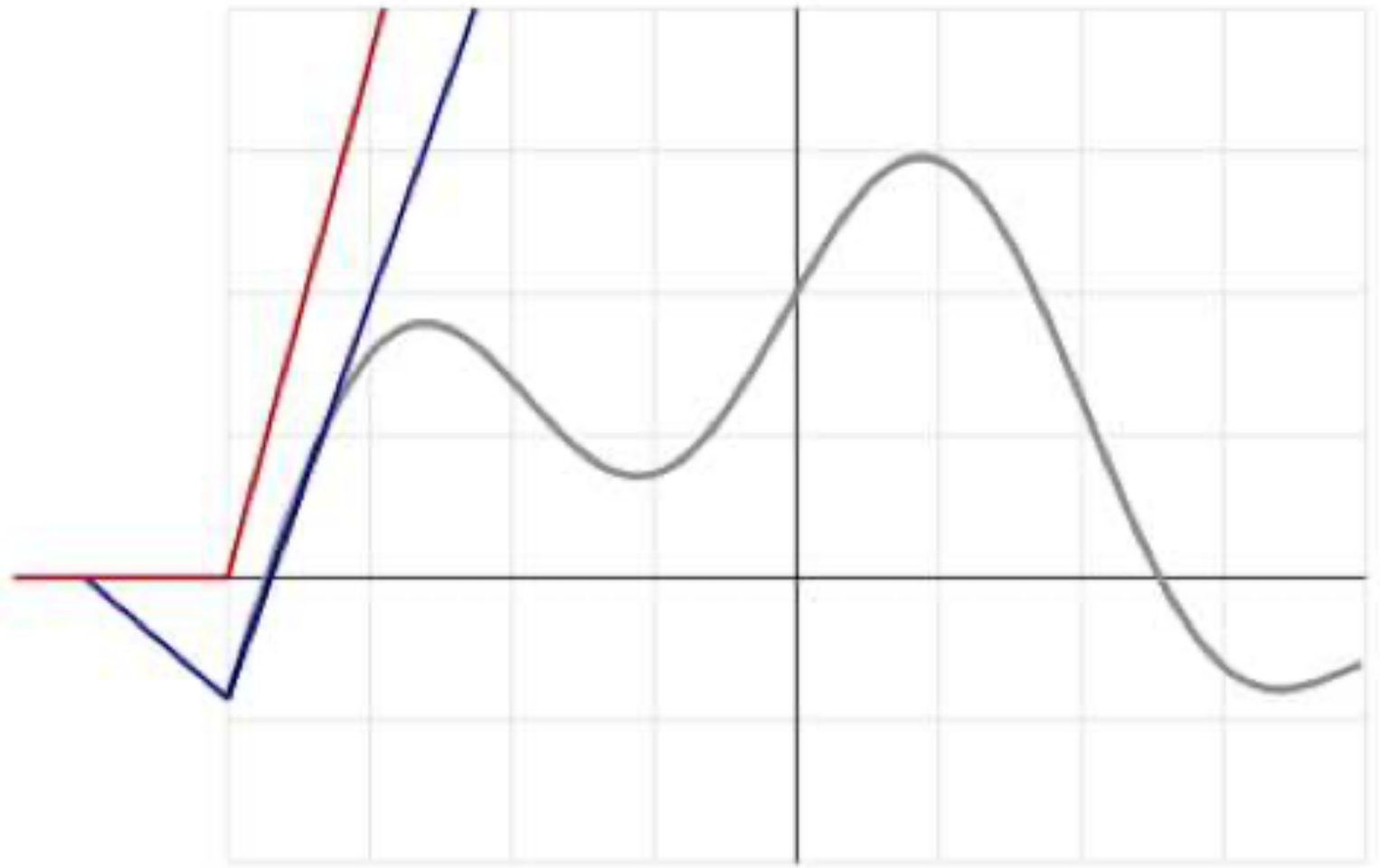
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones sont peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



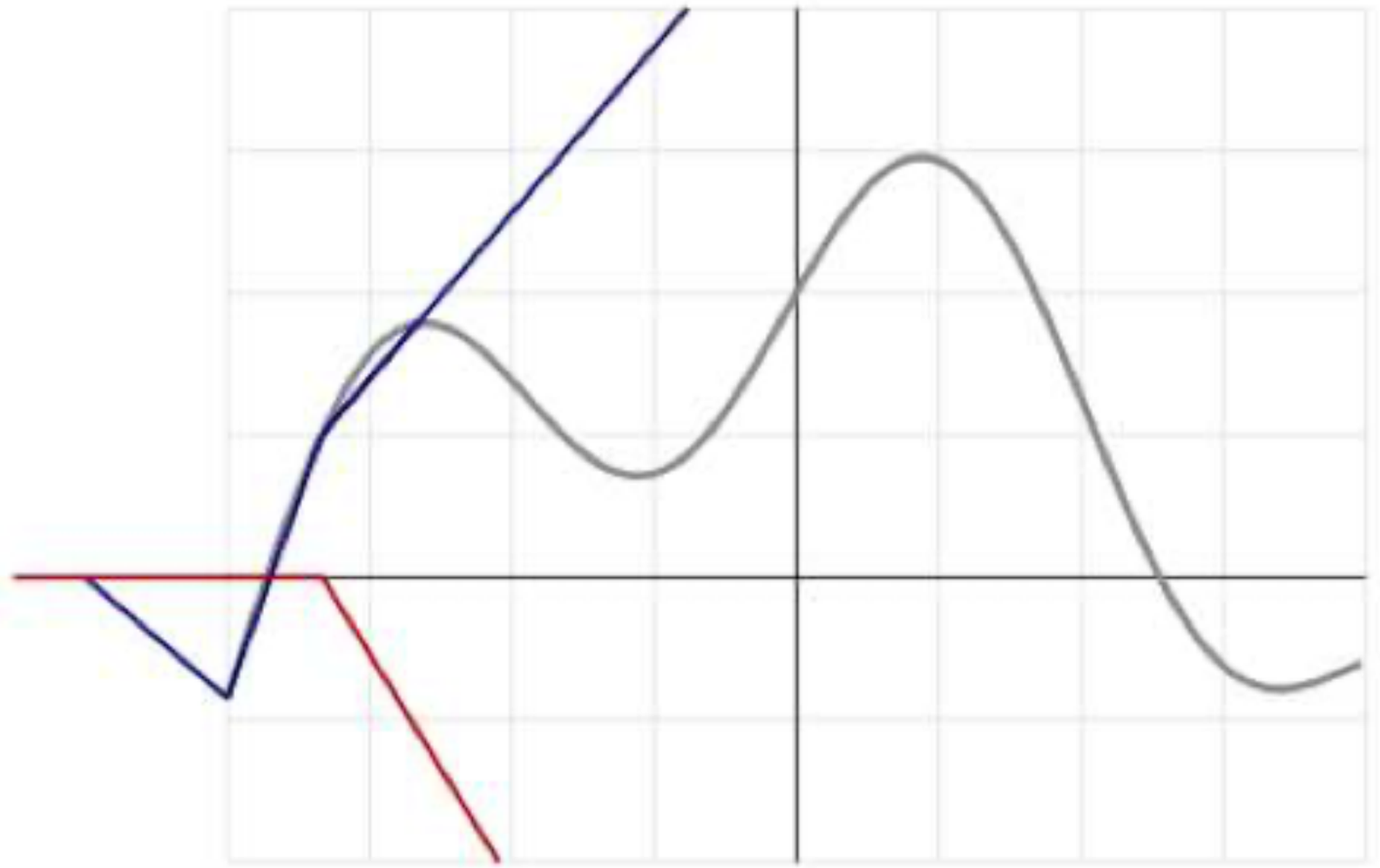
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones sont peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



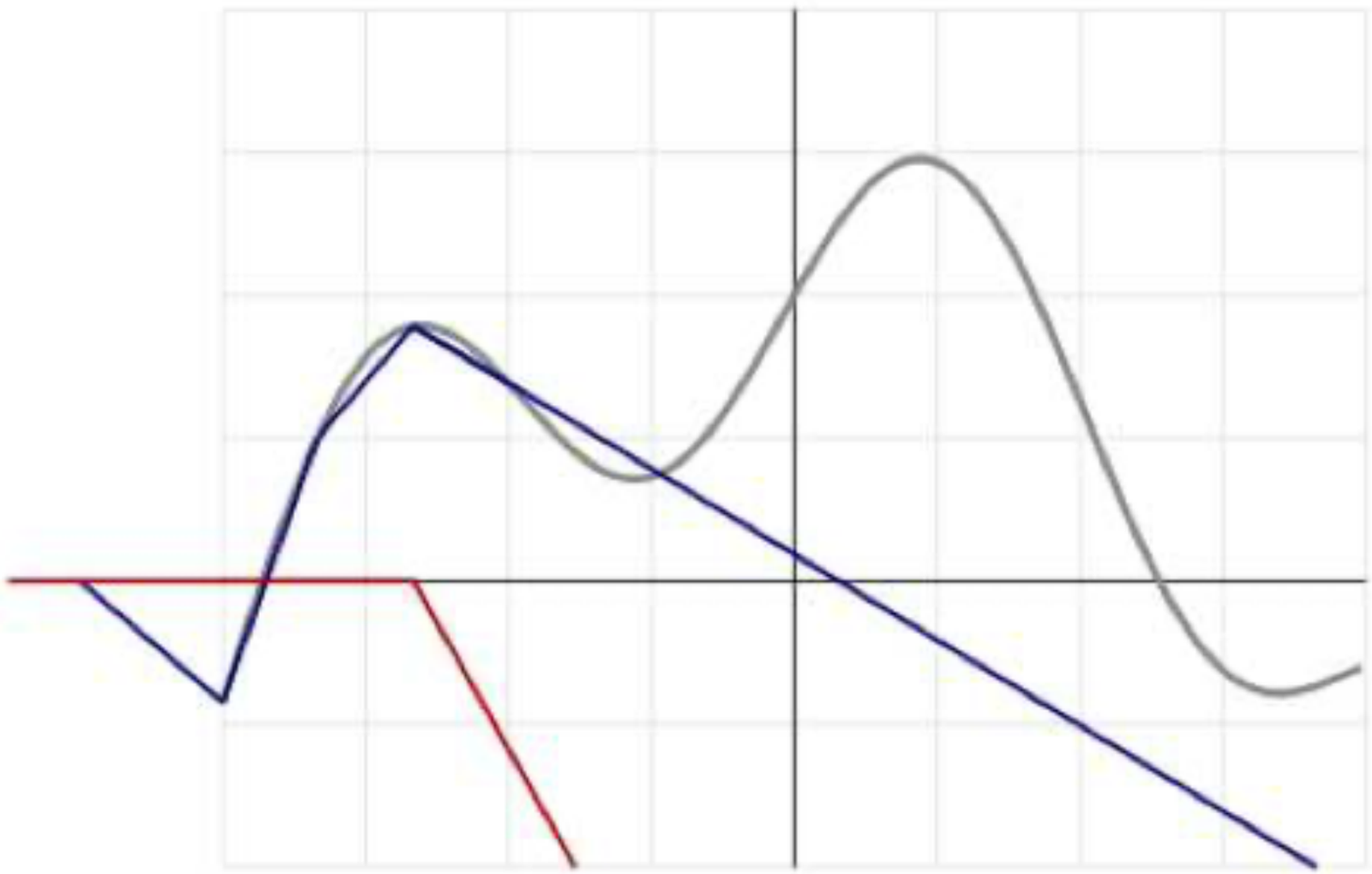
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones sont peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



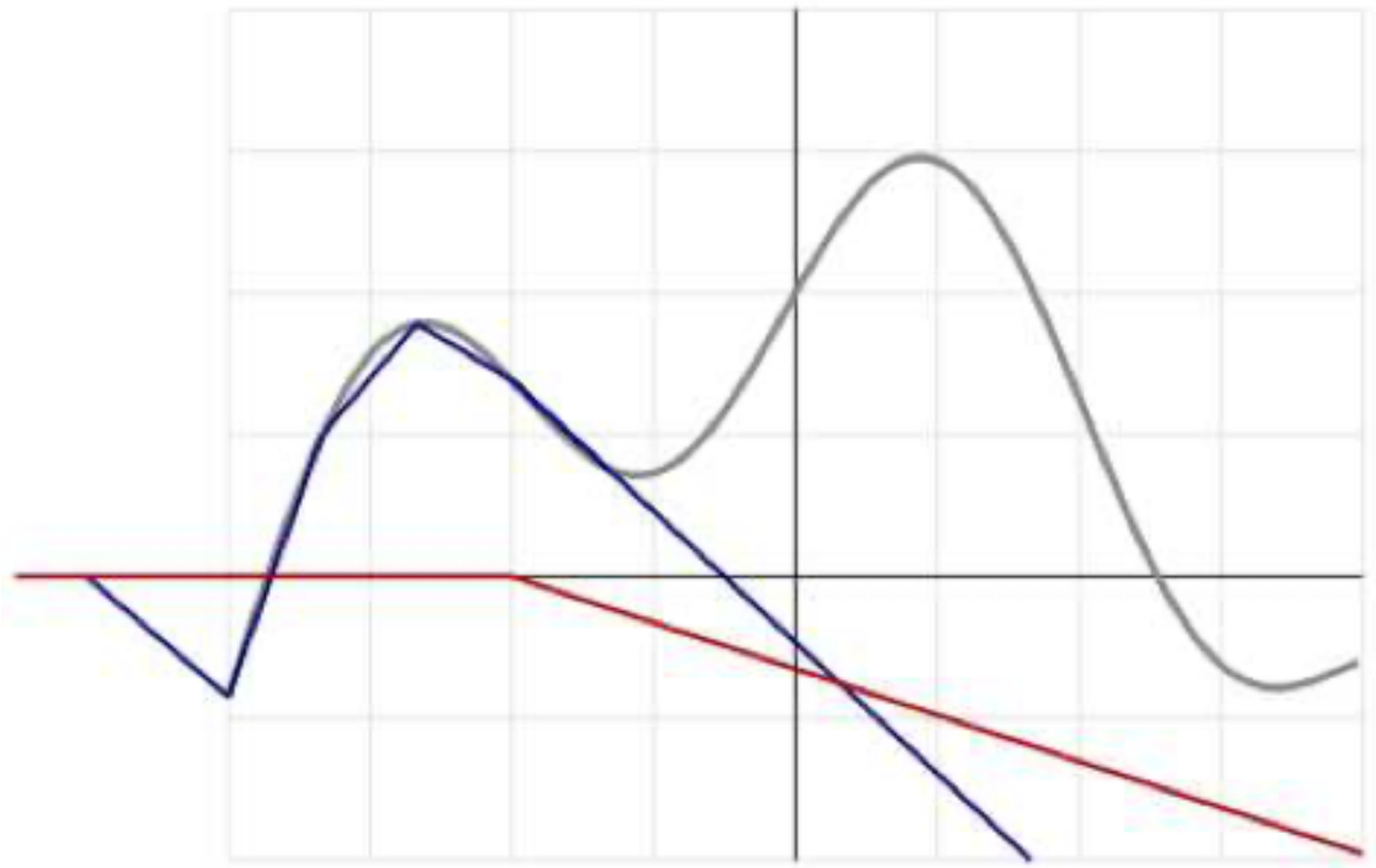
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones sont peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



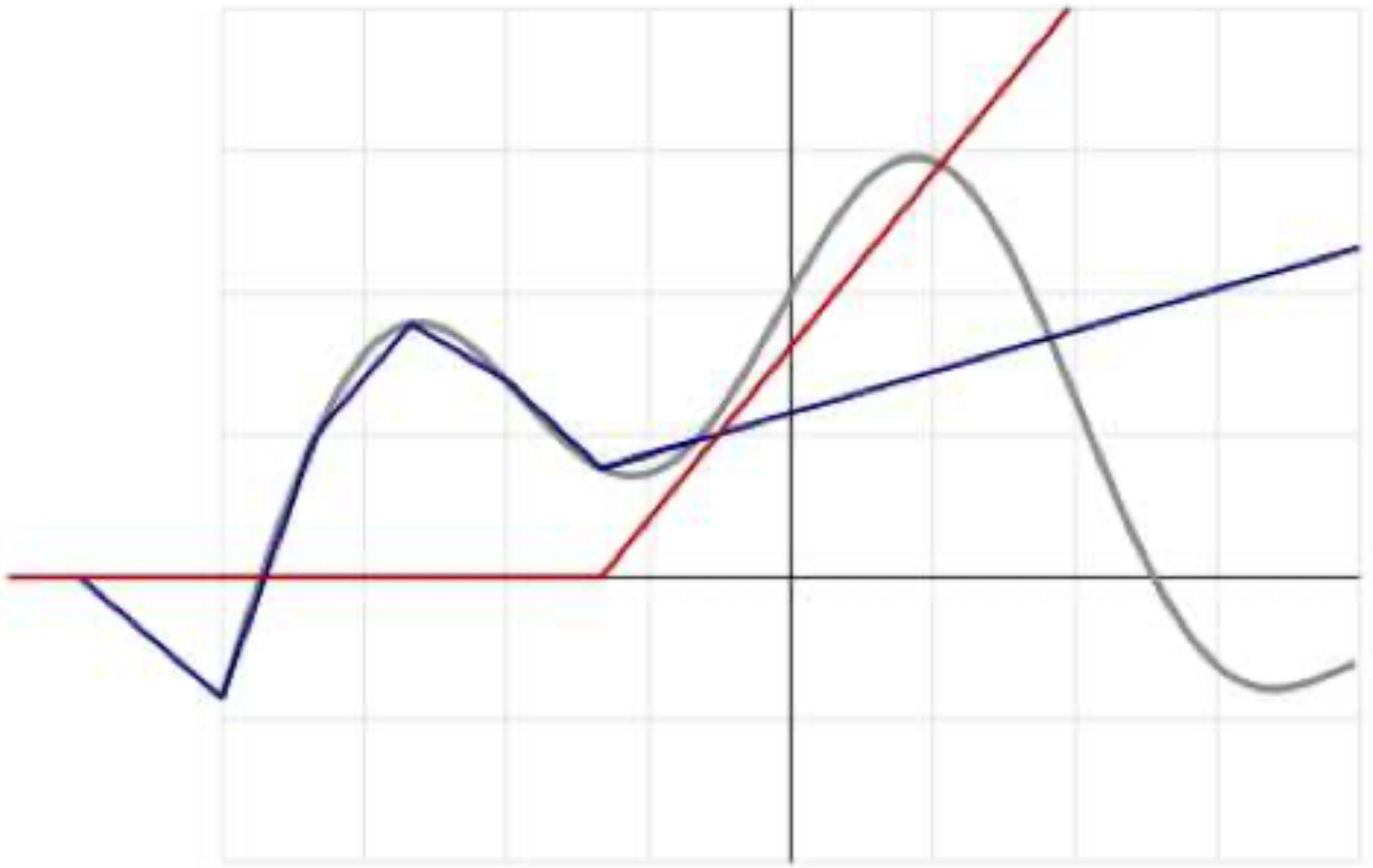
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones sont peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



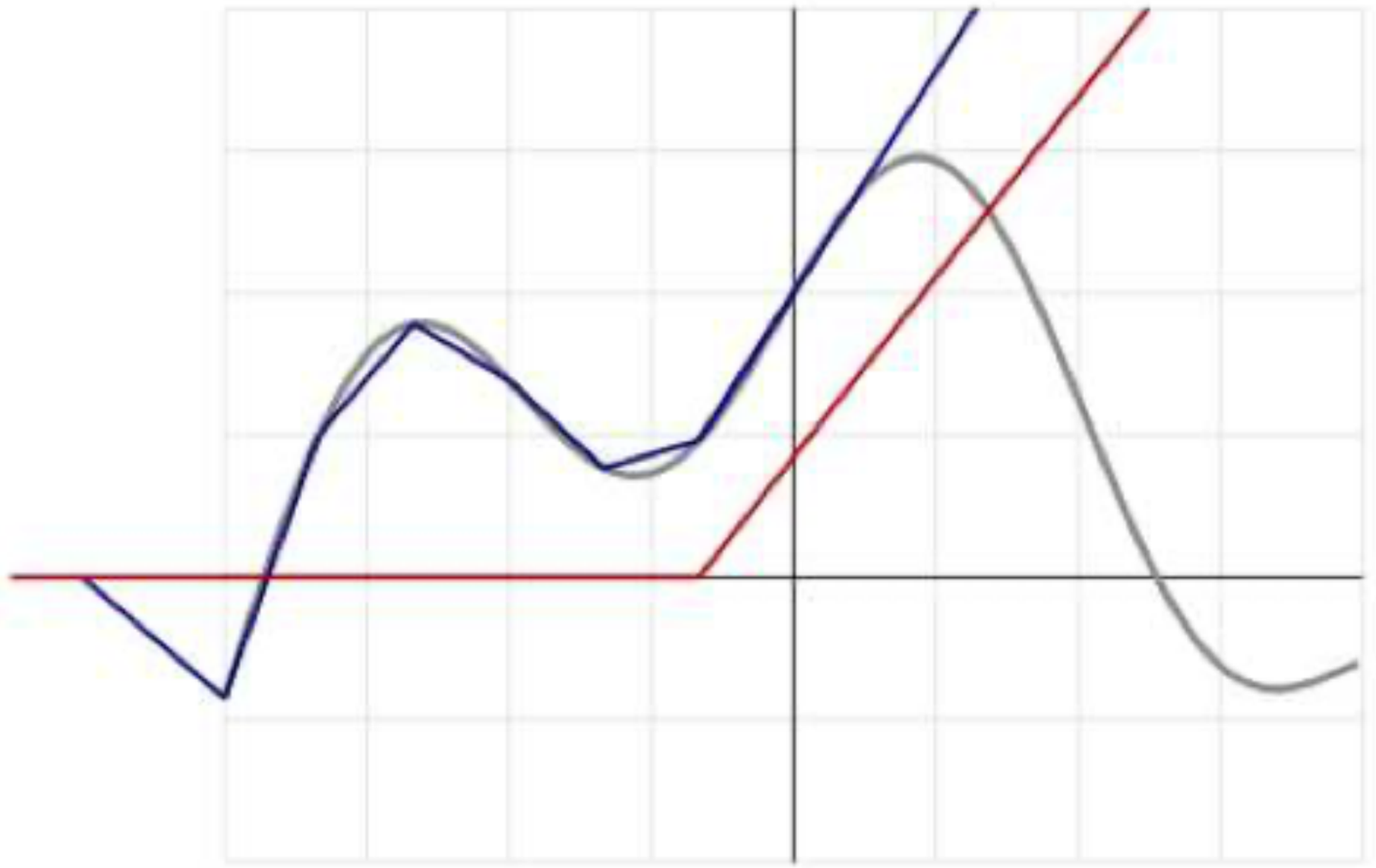
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones sont peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



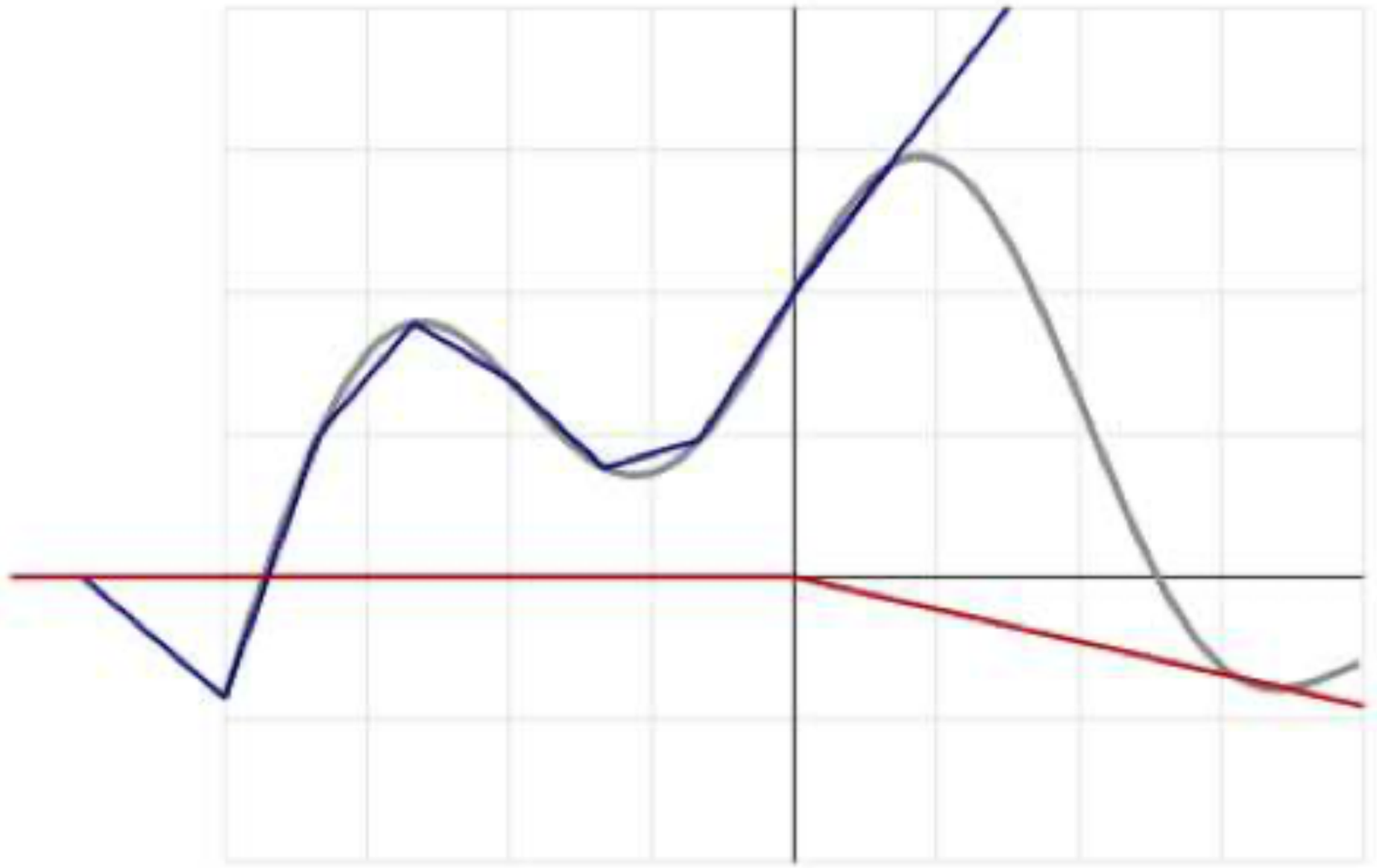
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones sont peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



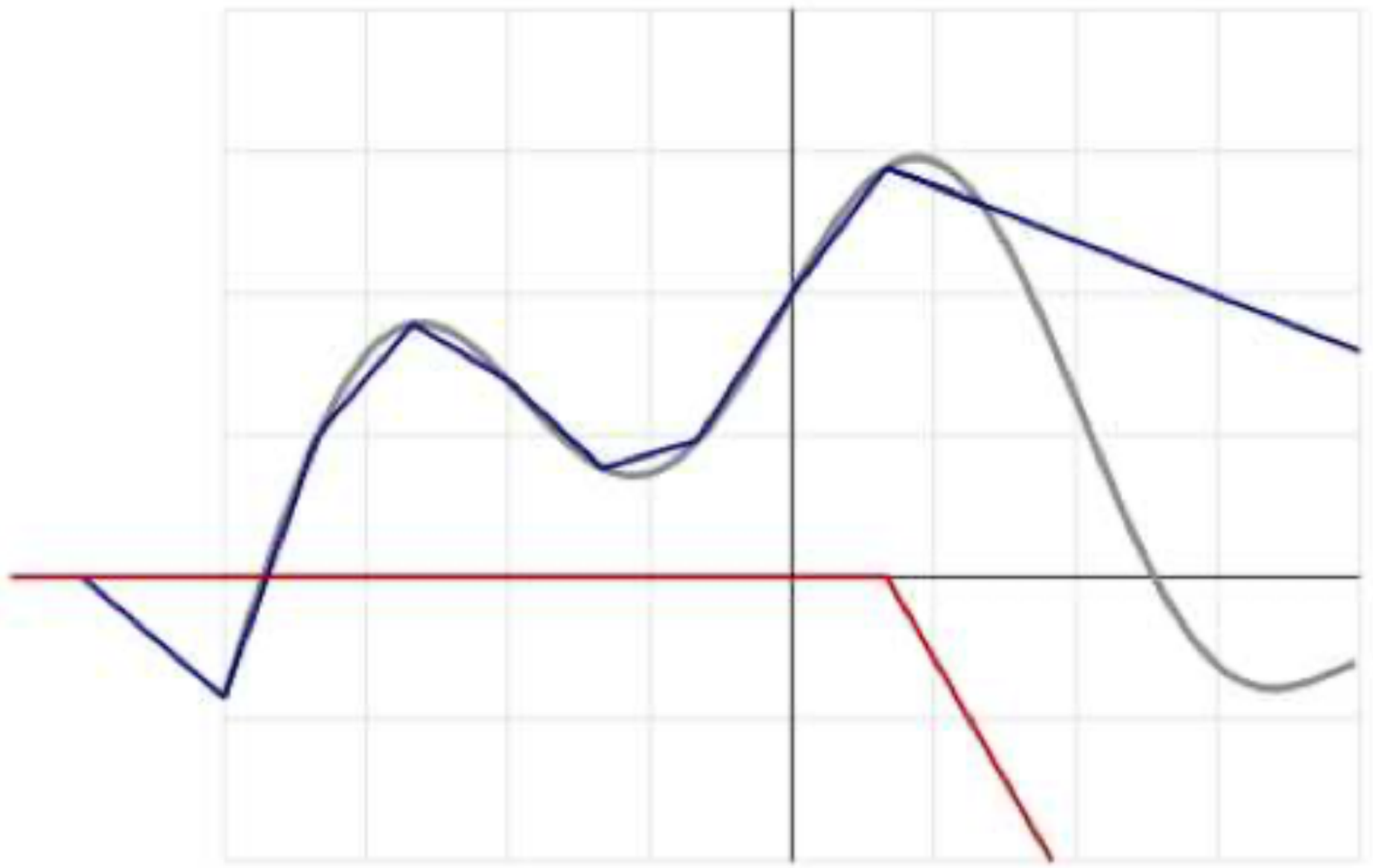
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones sont peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



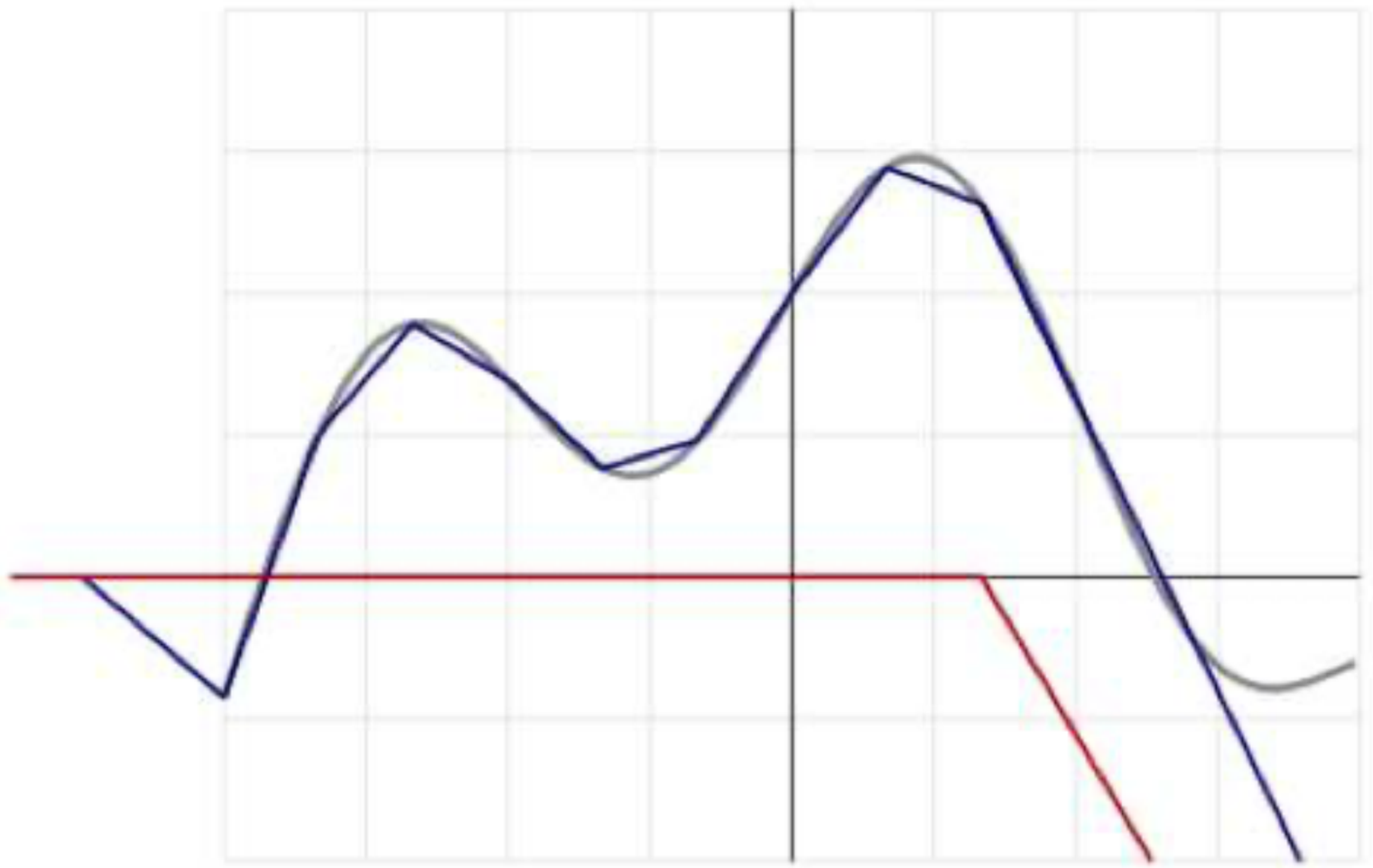
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones sont peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



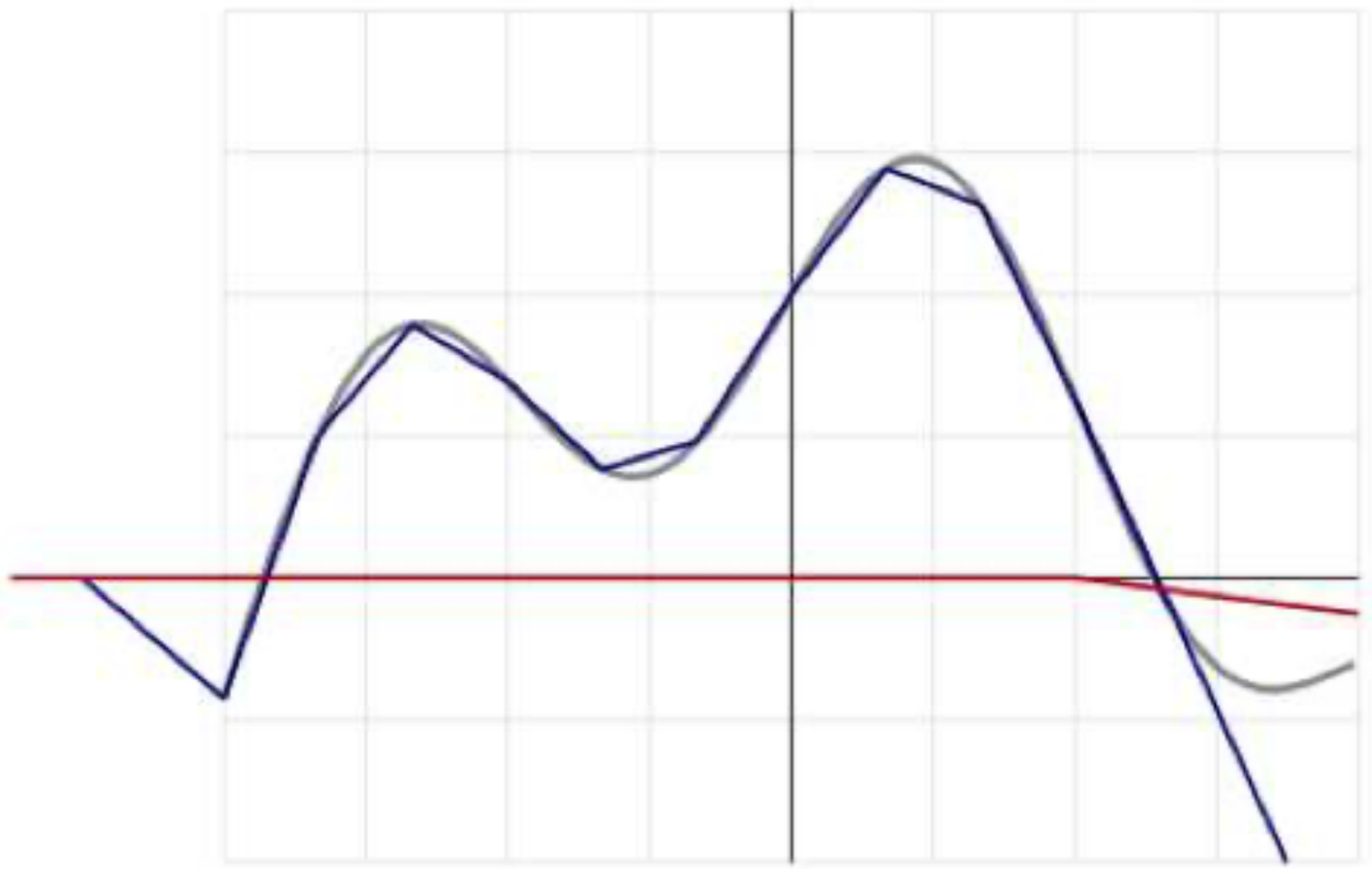
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones sont peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



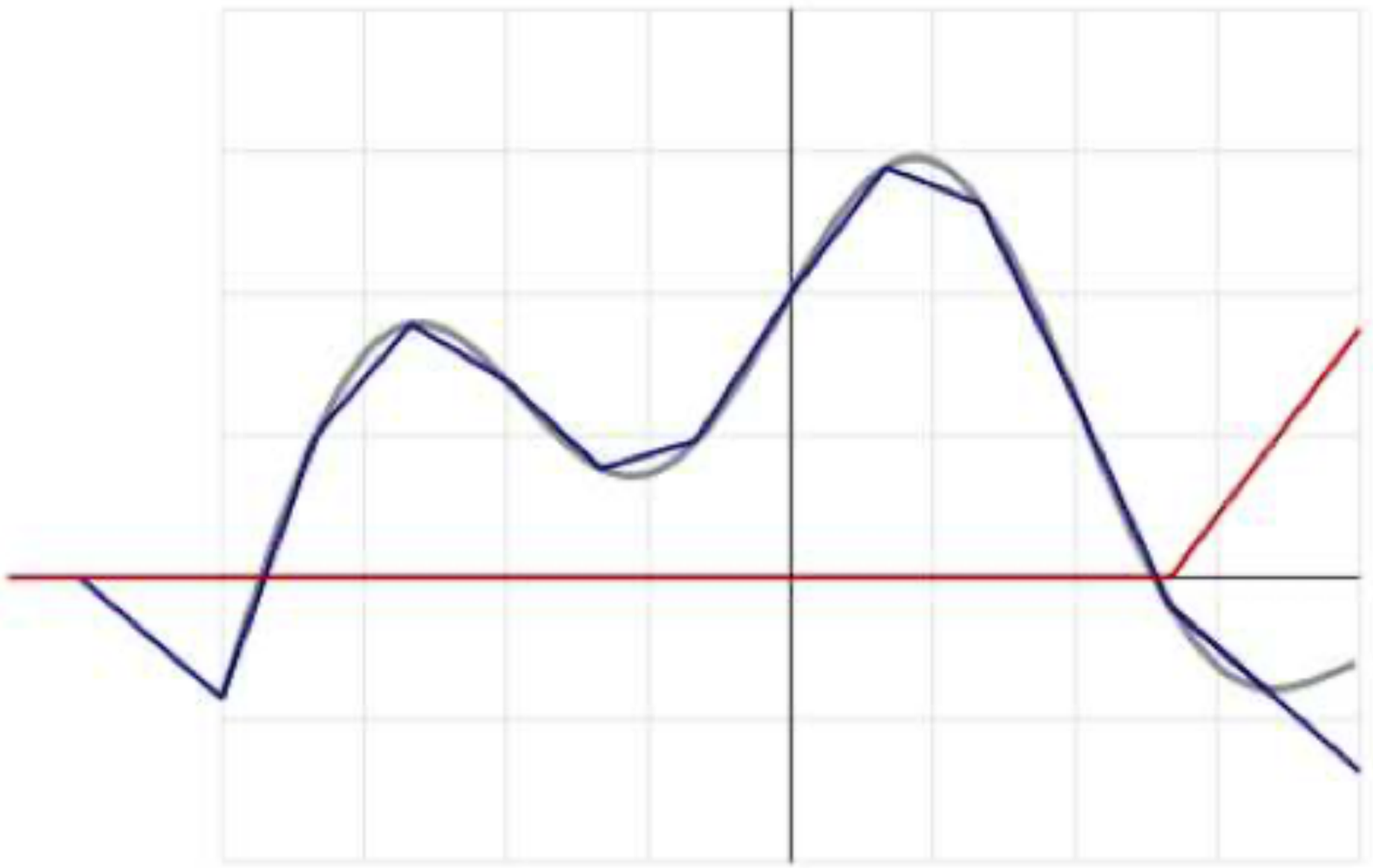
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones sont peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



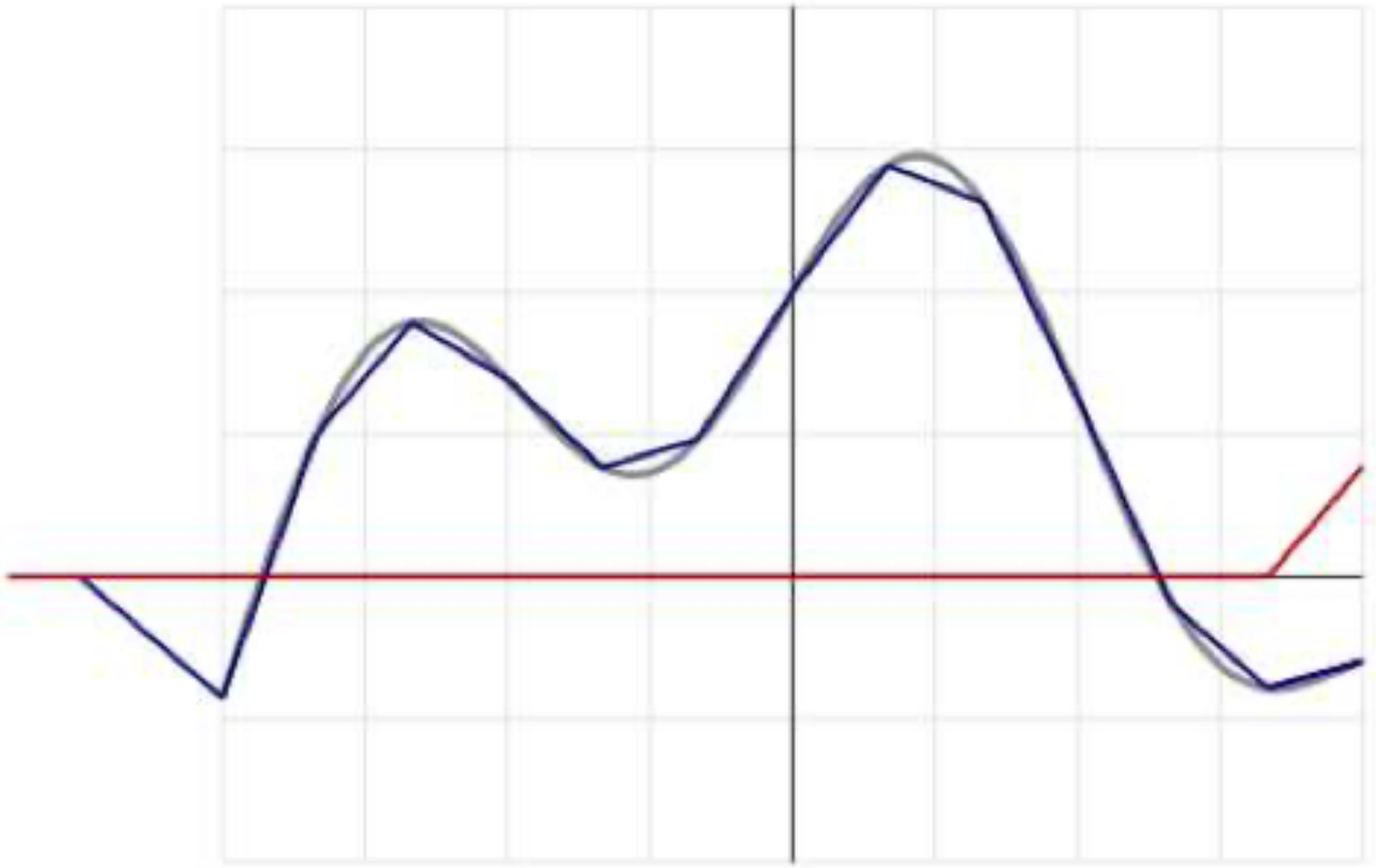
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones sont peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



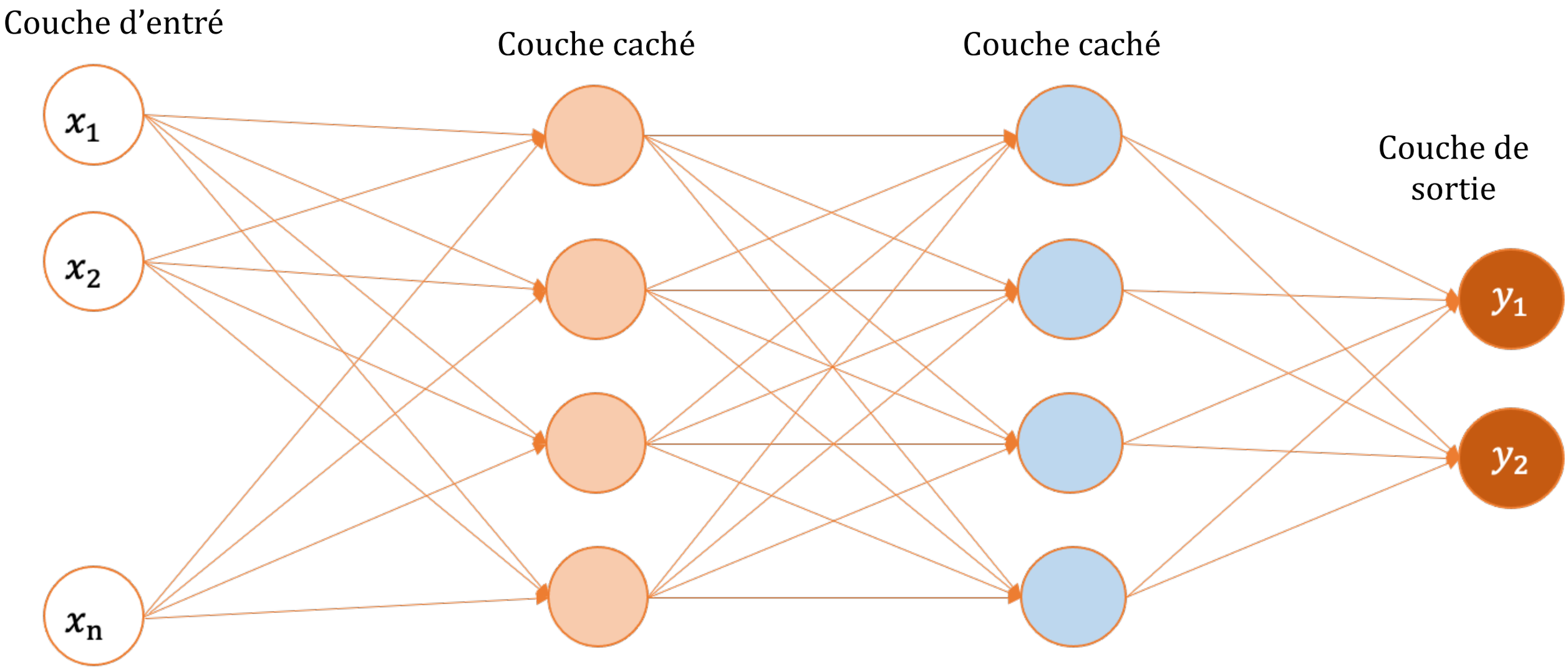
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones sont peuvent naturellement remplir ce rôle

Le neurone formel : approximateur universel



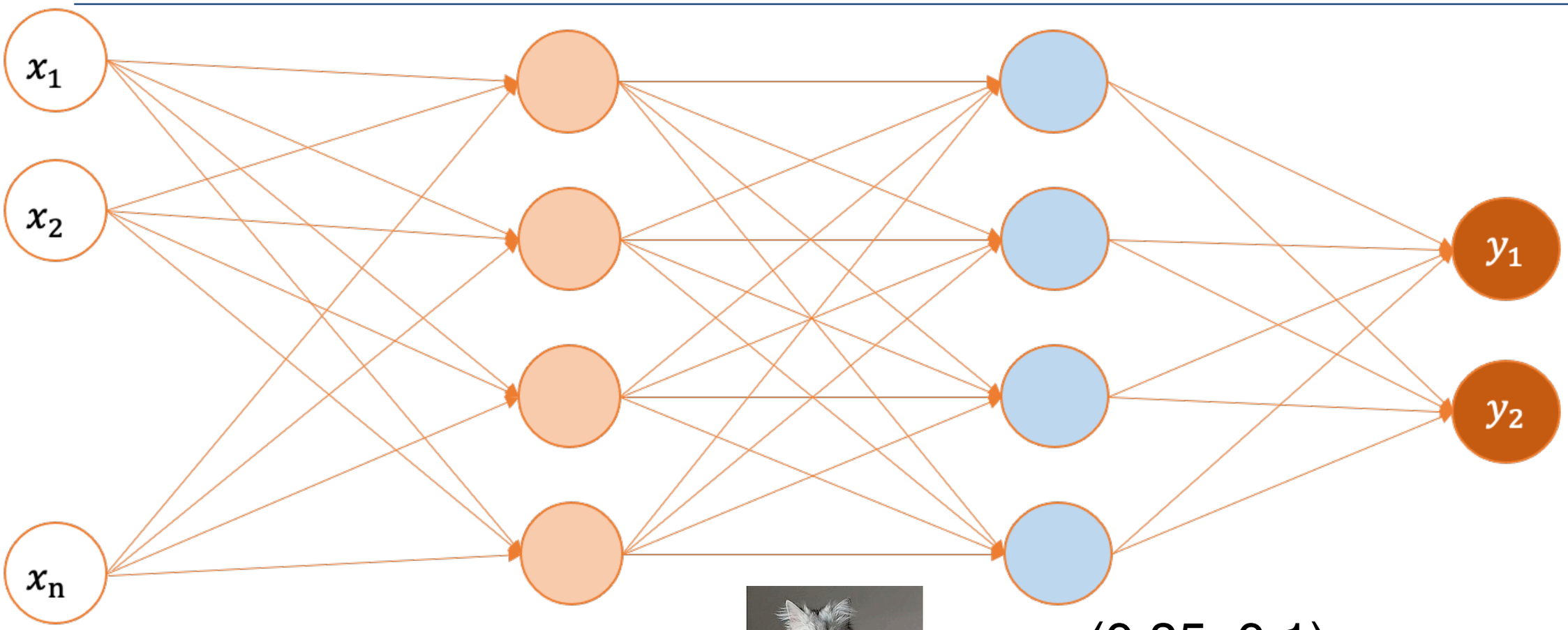
- Toute fonction mathématique réelle peut être **approximée** comme une **somme de fonction Relu** translaté
- Les réseaux de neurones sont peuvent naturellement remplir ce rôle

Perceptron multicouche (MLP)



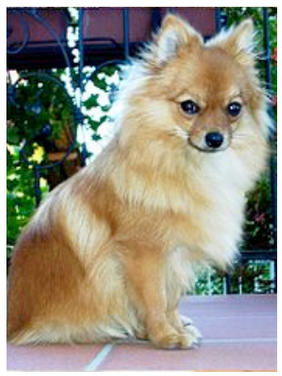
- Neurones assemblés ensemble pour **approximer** une fonction inconnue
- Taille des couches cachées et profondeur (nombre de couches) au choix du développeur

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

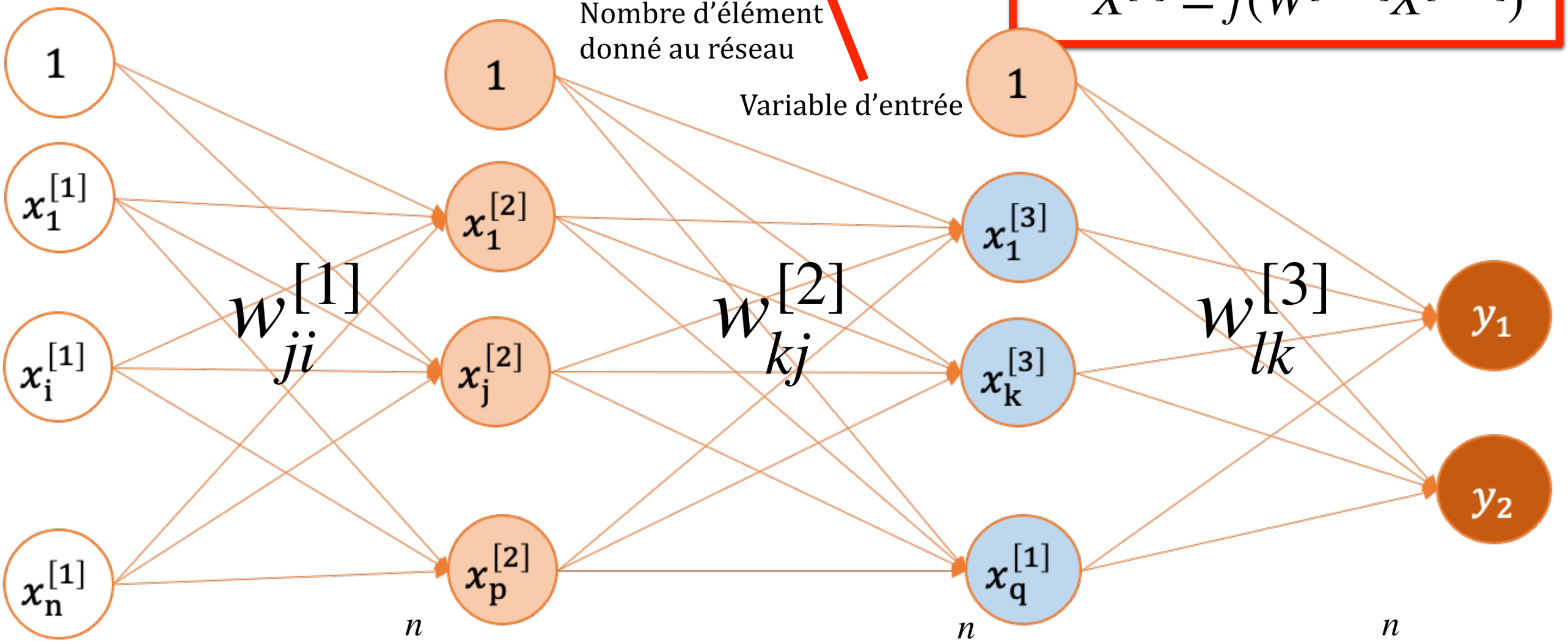
$$X^{[0]} : N \times n$$

Nombre d'élément donné au réseau

Variable d'entrée

D'un point de vue matricielle :

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



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

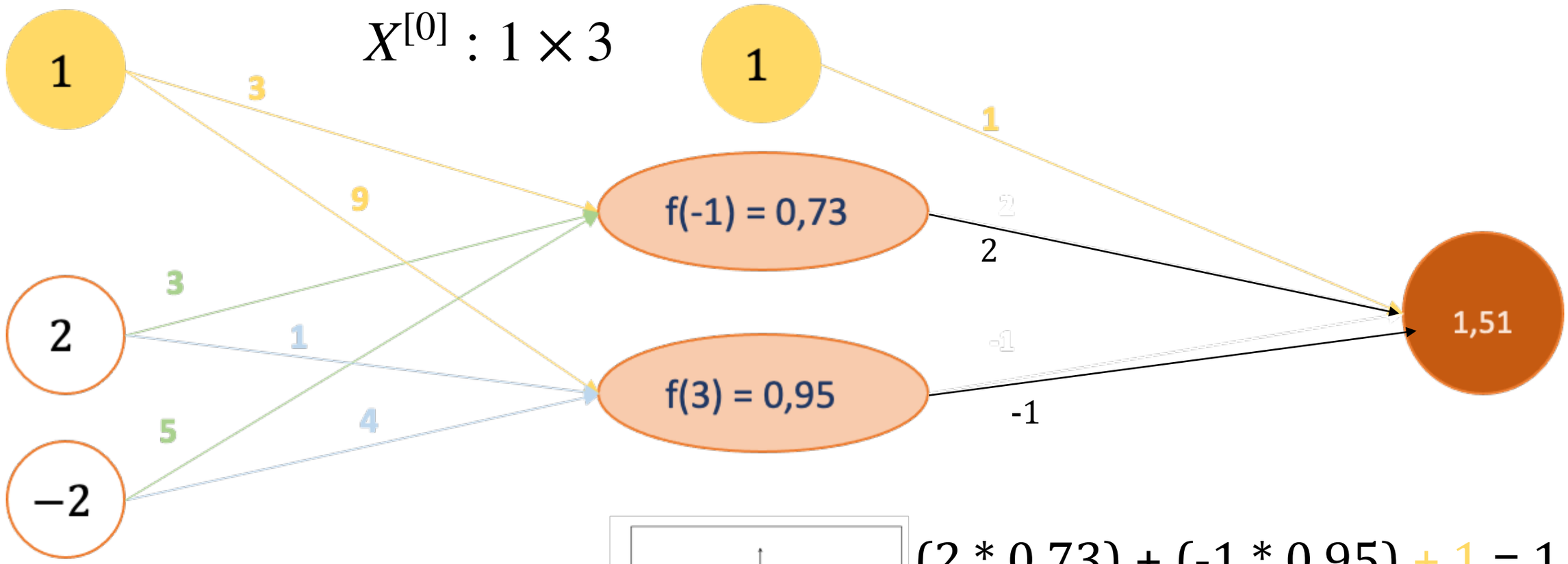
$$x_j^{[2]} = f(s_j^{[2]})$$

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

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

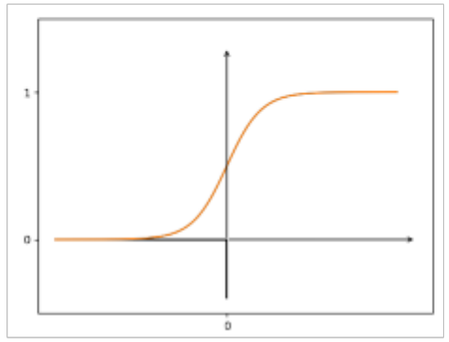
$$y_l = \sum_{k=0}^{n} w_{lk}^{[3]} x_k^{[3]}$$

Perceptron multicouche : Example



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

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



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

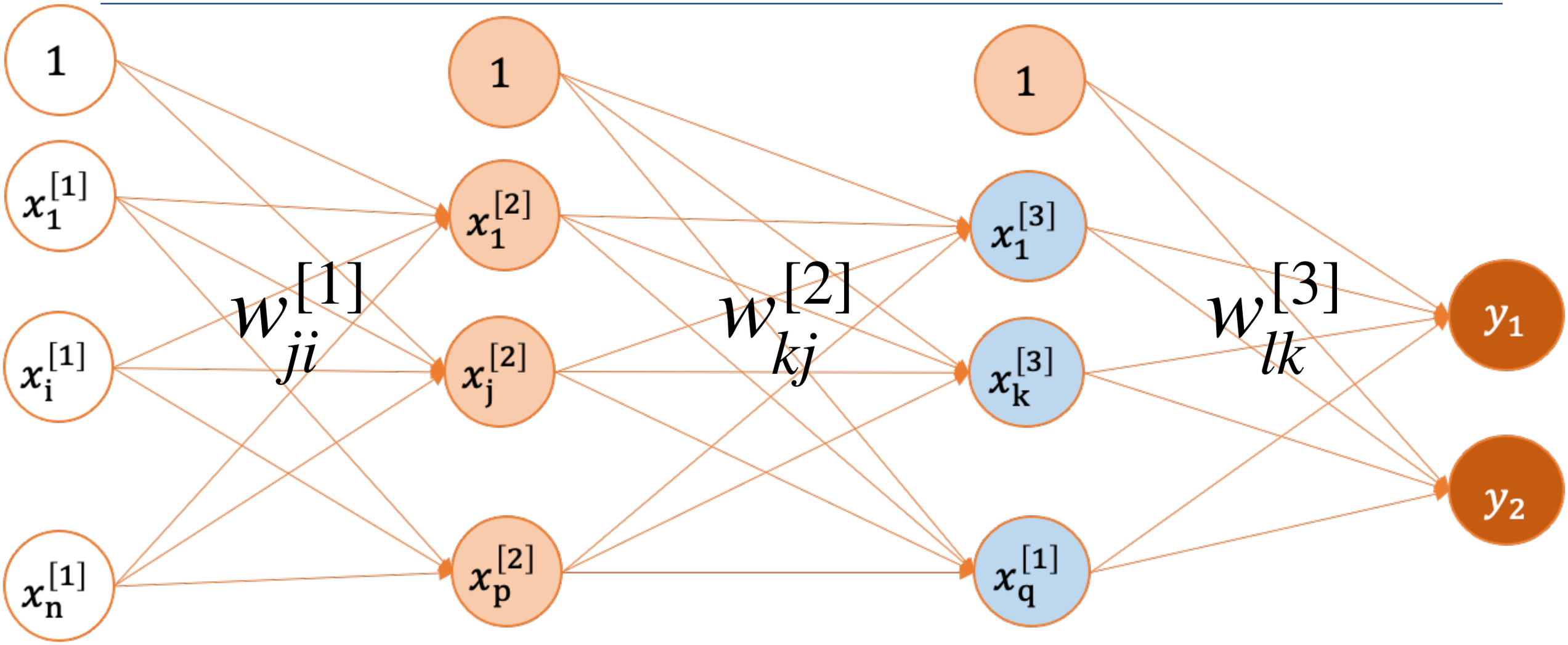
En notation matriciel :

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

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

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

Rétropropagation du gradient

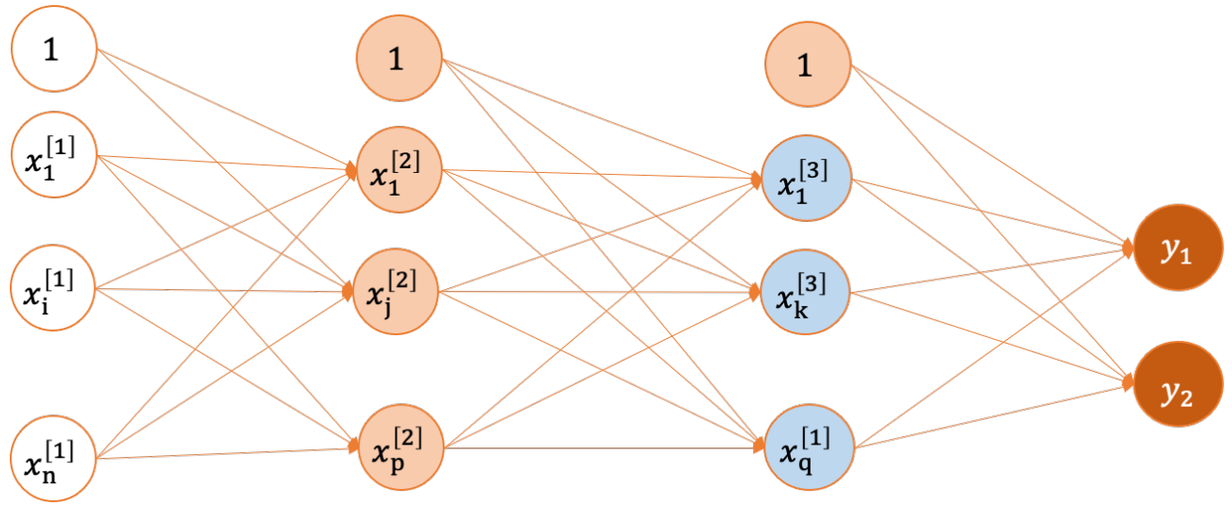


- Nombre de **paramètres** : $(n+1) \times p + (p+1) \times q + (q+1) \times 2$
- Si n = p = q =10 ➜ **242** paramètre à optimiser
- Comment effectuer cette **optimisation** de manière efficace ?

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)

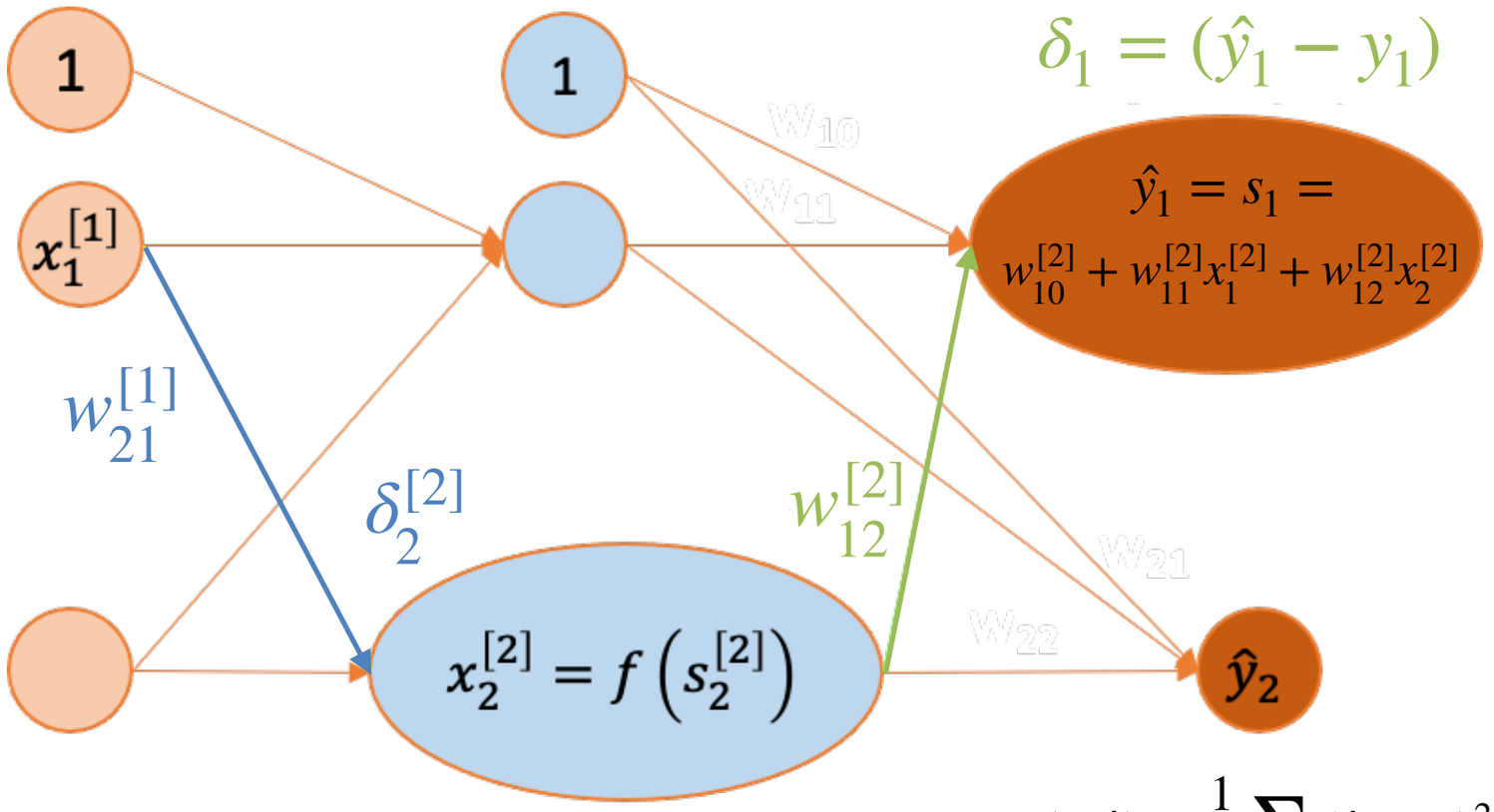
Rétropropagation du gradient

Dernière (P) couche :

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

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

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



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

Couche P-1 :

$x_1^{[1]}$

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

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

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 :

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

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 !

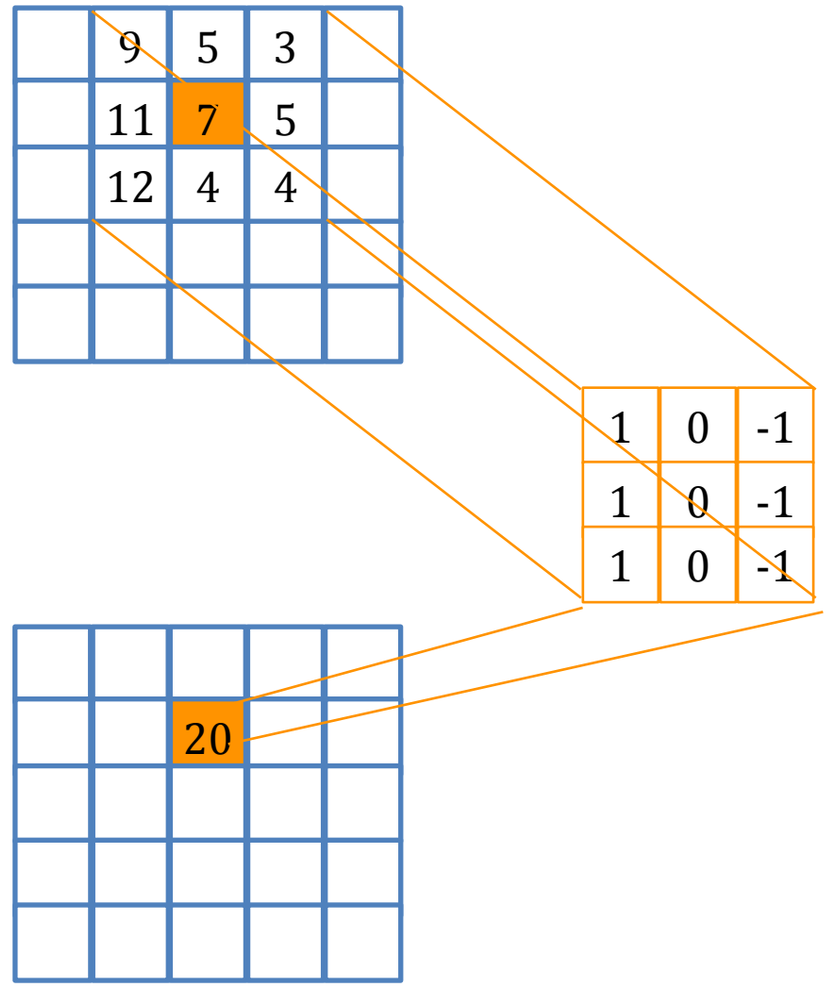
Les réseaux de convolution (CNN)

MLP : Peu adapté aux images

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

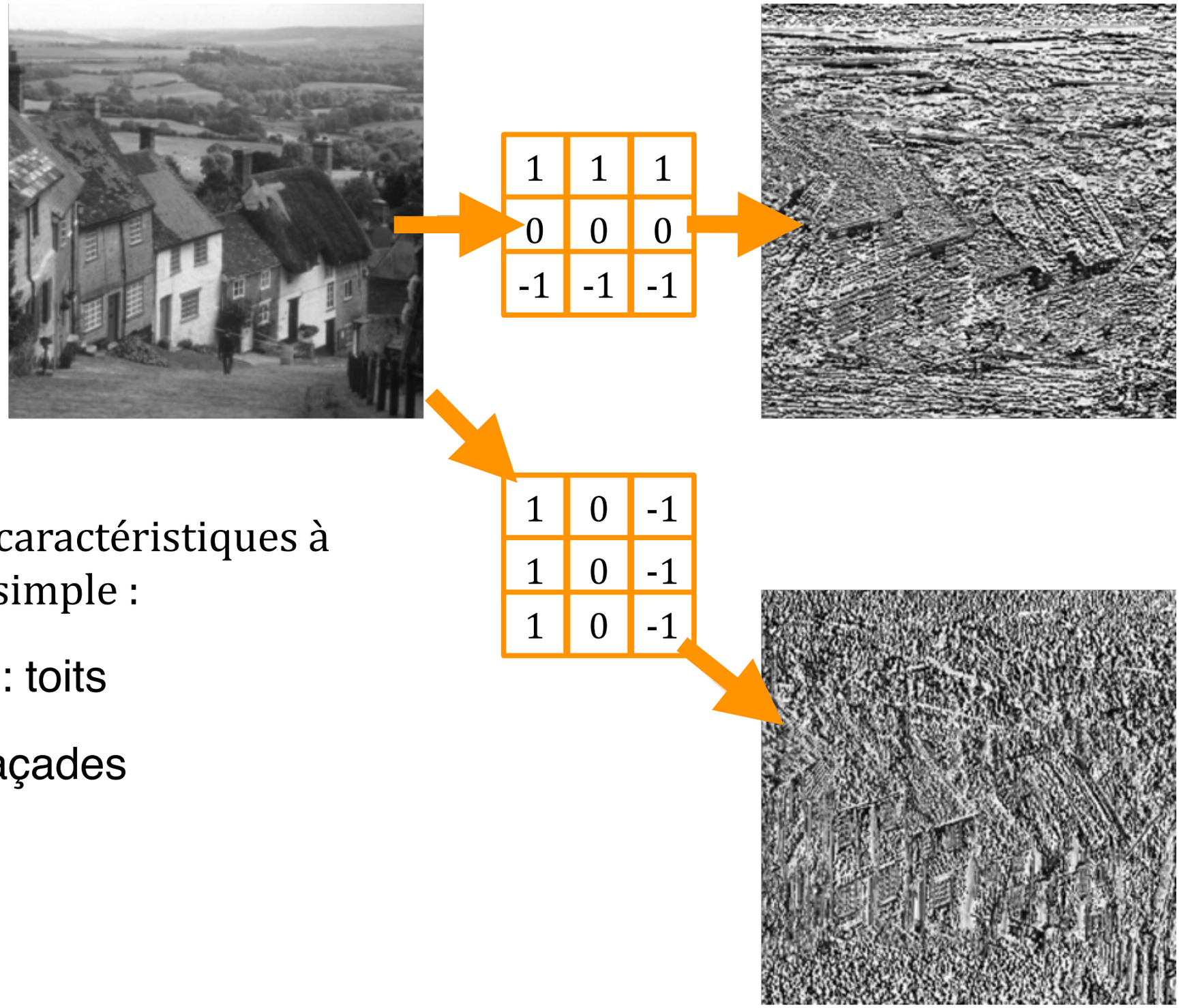
Utilisation de filtre de convolution :

- 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 :



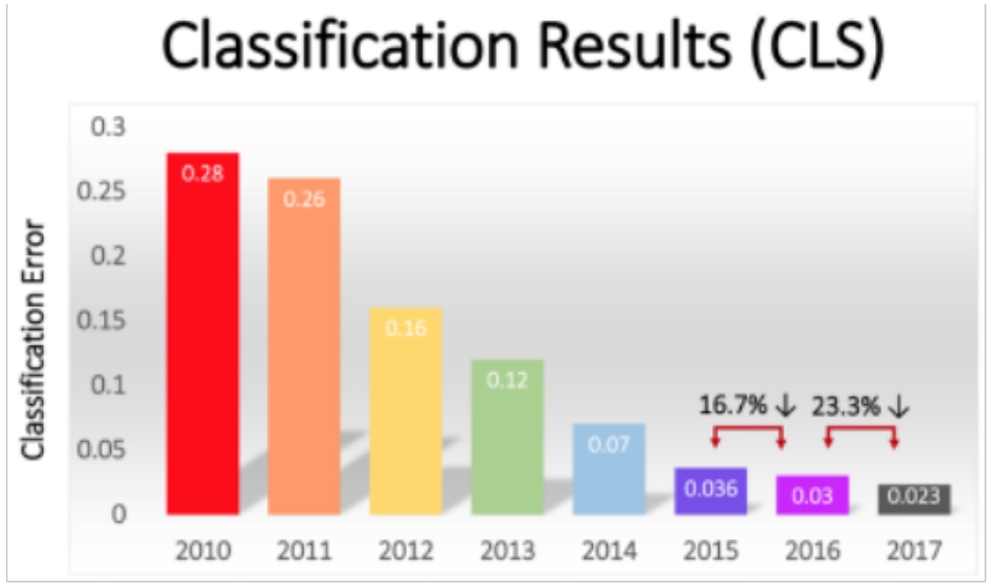
Extraction des caractéristiques à partir de filtre simple :

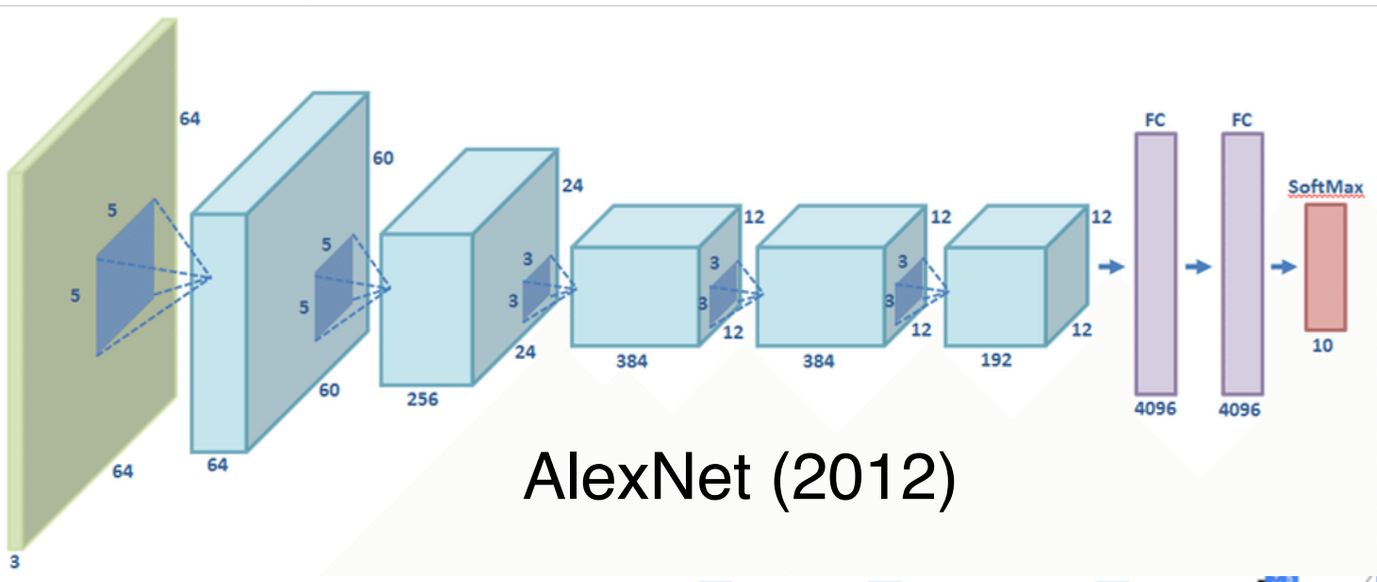
- Horizontale : toits
- Verticale : façades

AlexNet, 2012

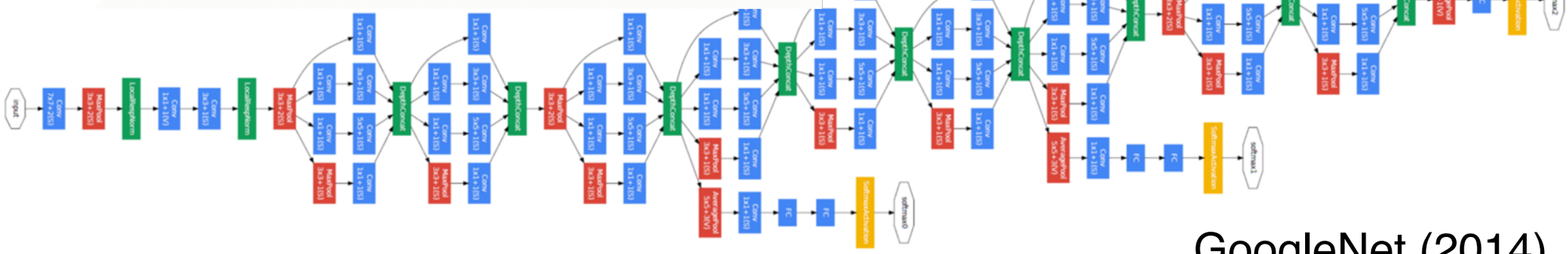
ImageNet Visual Recognition Chalenge :

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





AlexNet (2012)

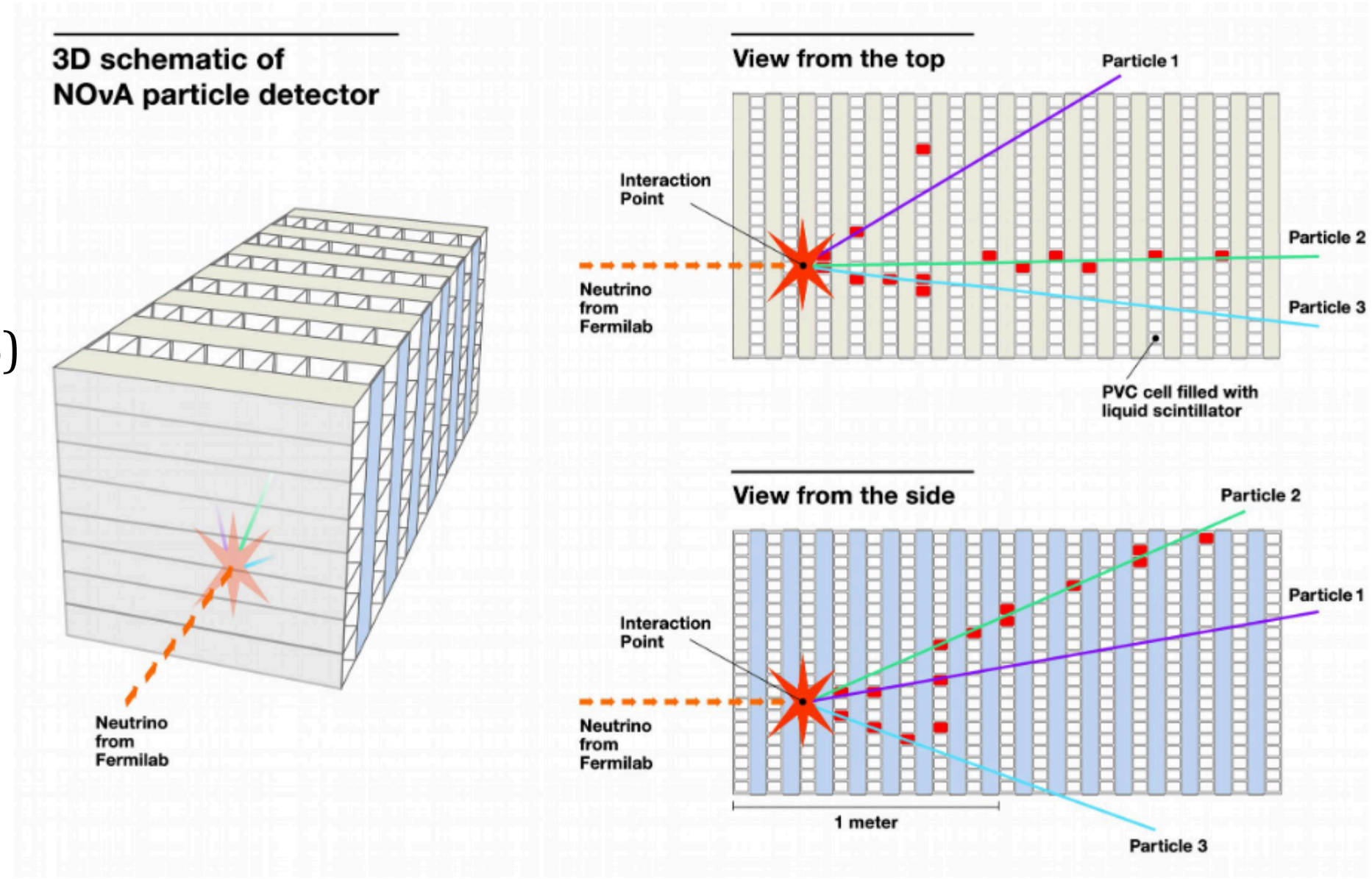


GoogleNet (2014)

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

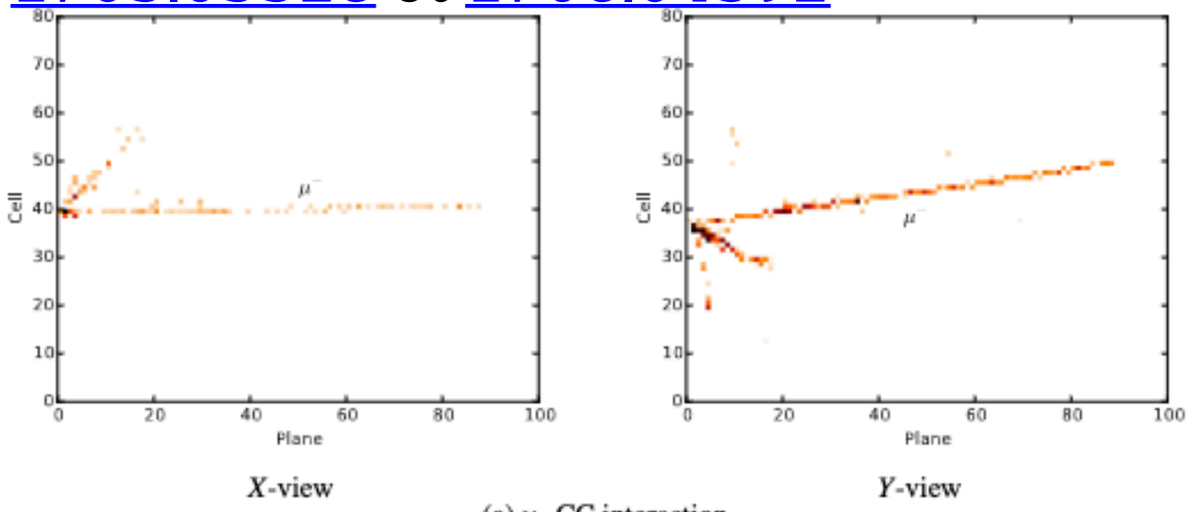


A Convolutional Neural Network Neutrino Event Classifier

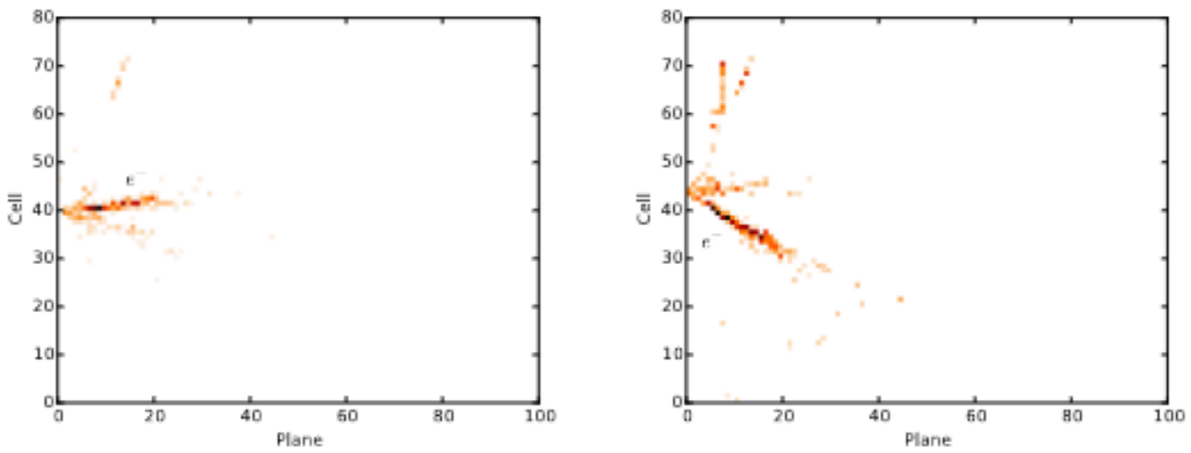
Classification des événements dans l'experience Nova

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.



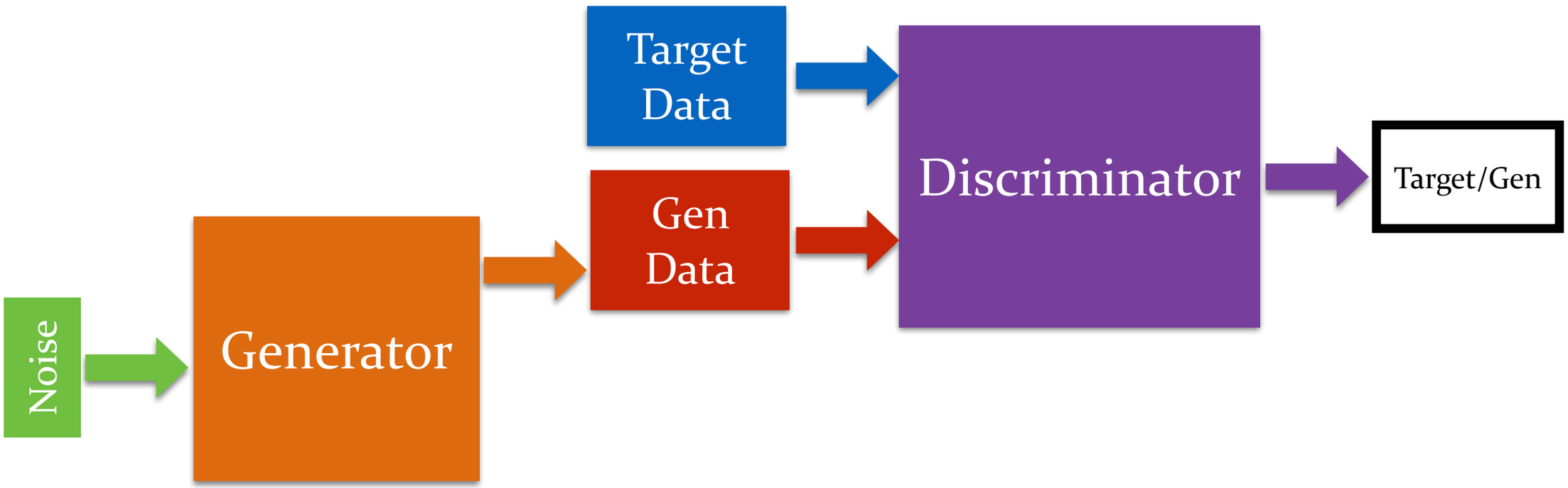
(b) ν_e CC interaction.

[A Convolutional Neural Network Neutrino Event Classifier]

Les Réseaux Antagonistes Génératifs (GAN)

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

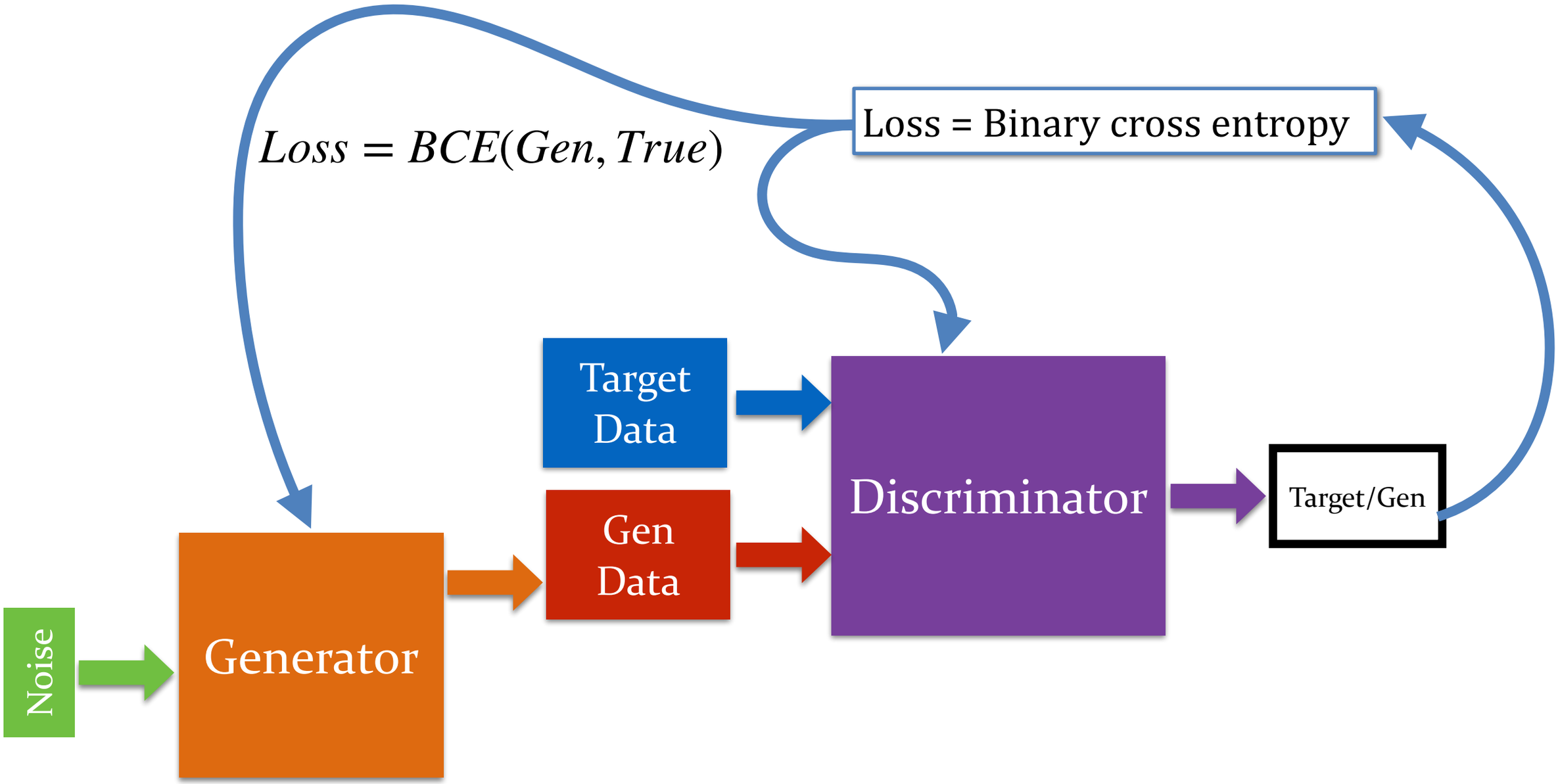
- Utilise deux réseaux de neurones :
  - **Générateur** : génère des données à partir de bruit
  - **Discriminateur** : sépare le bruit des données cibles



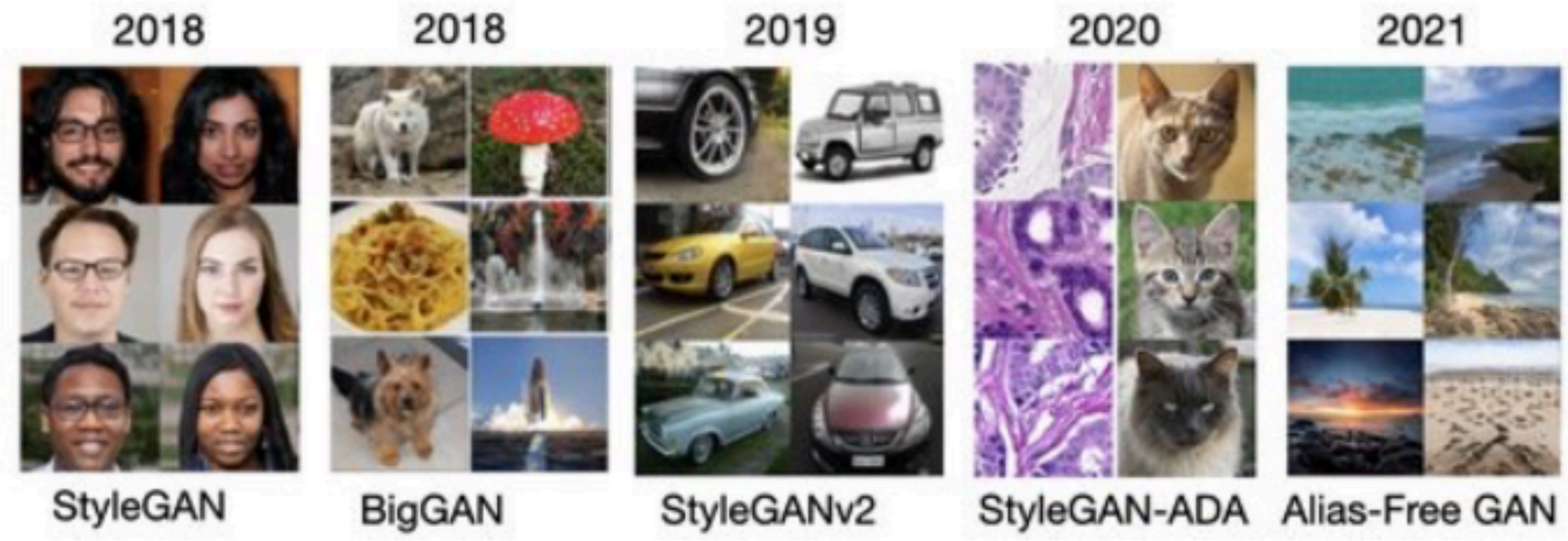
Les Réseaux Antagonistes Génératifs (GAN)

Même fonction de coût pour le discriminateur et le générateur :

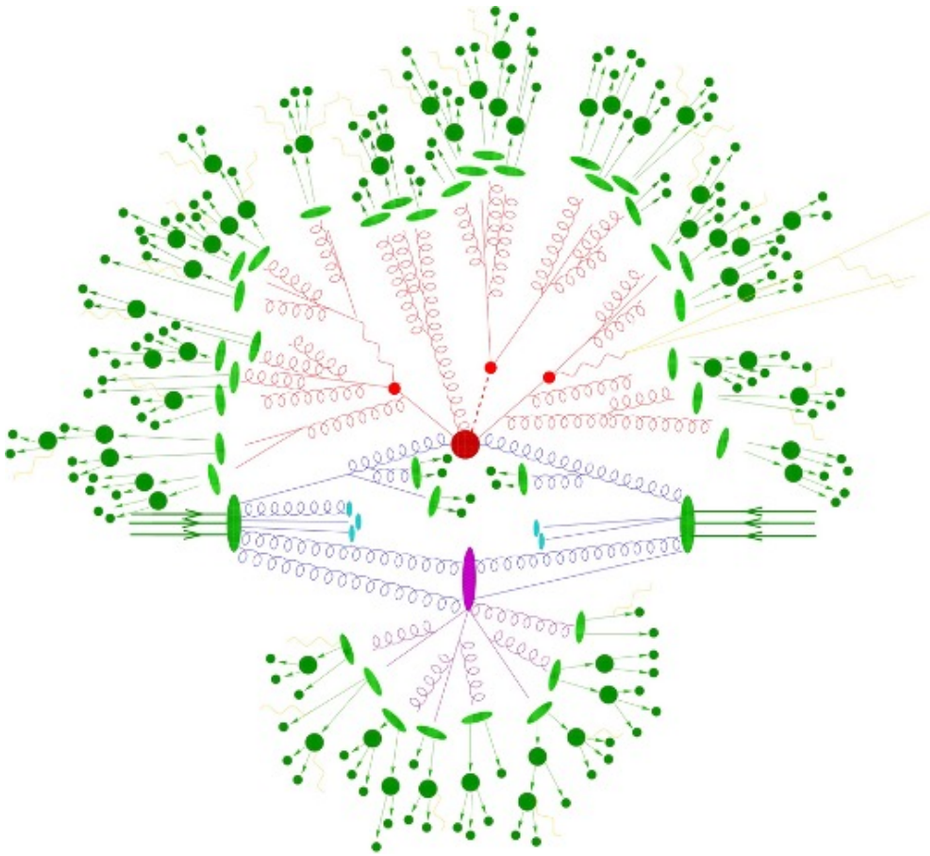
- Le discriminateur essaie de **séparer** les vraies entrées des entrées simulées
- Le générateur essaie de **tromper** le discriminateur



Les Réseaux Antagonistes Génératifs (GAN)



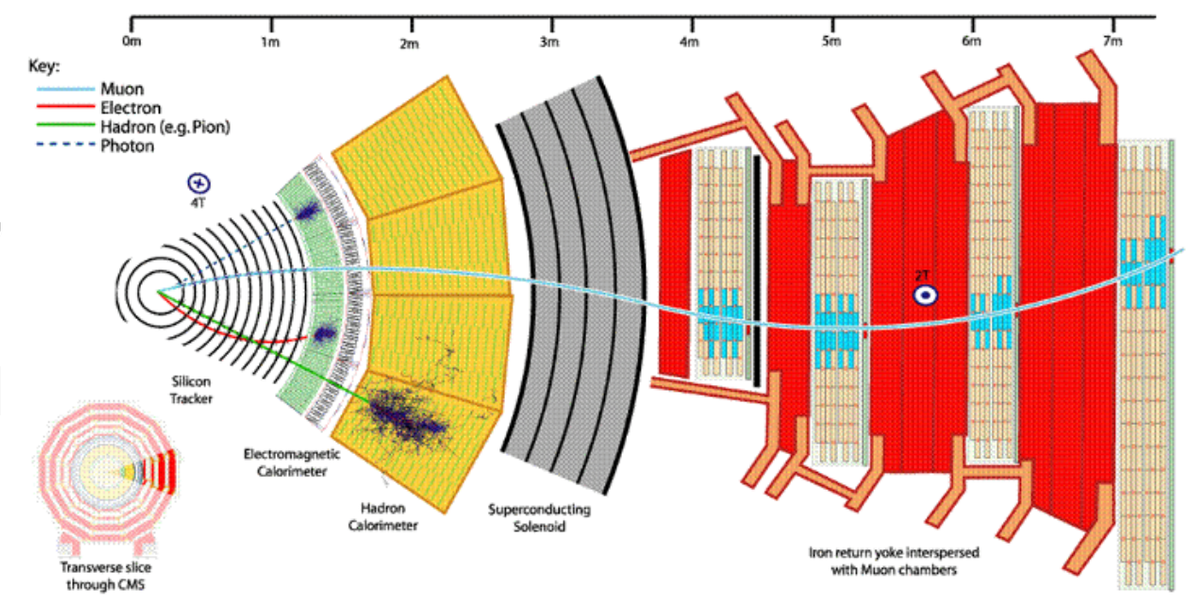
Simulation des événements dans ATLAS/CMS



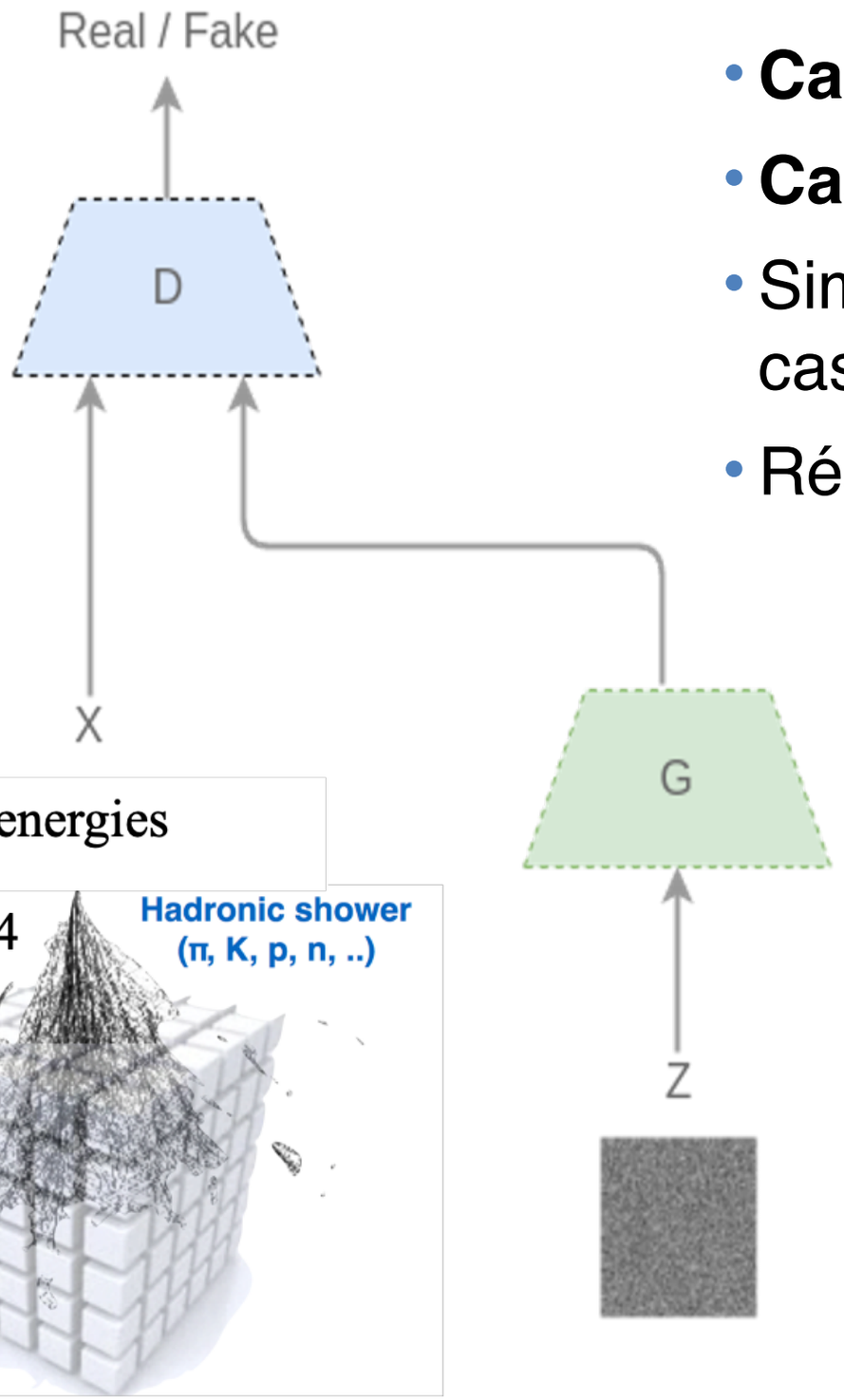
Simulateurs :

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

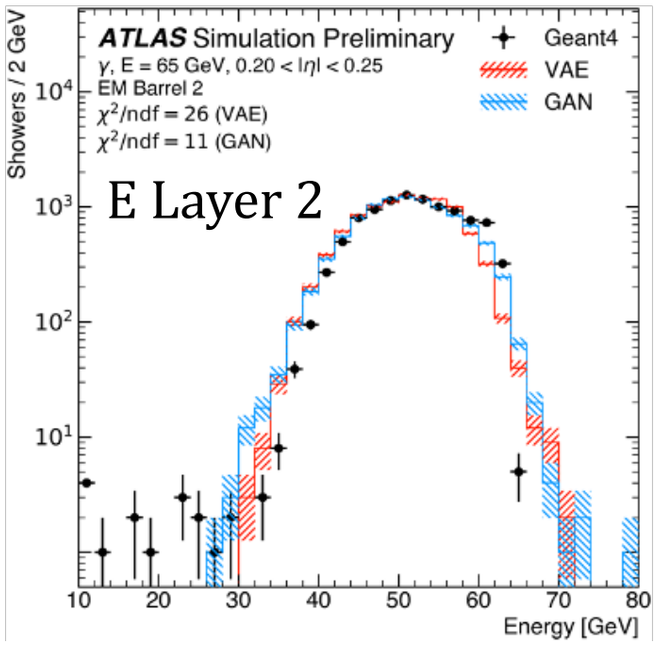
- Demande beaucoup de **ressources** :
  - Humaine (optimisation de la simulation, étalonnage…)
  - CPU (50% de nos ressources de calcul vont dans la simulation)

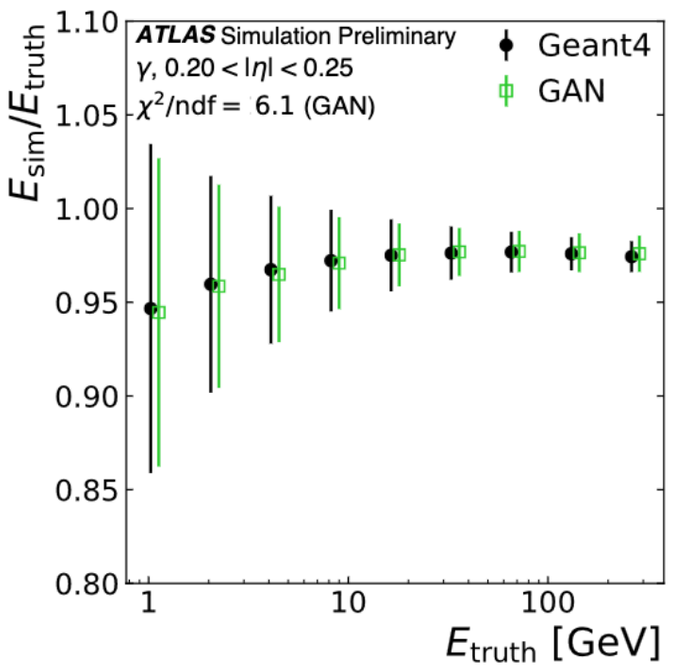


Simulation de calorimètre à l'aide de GAN



- **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



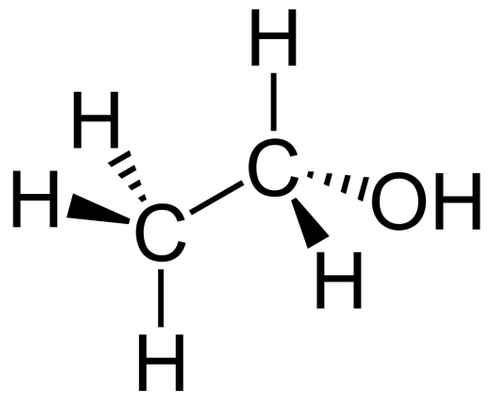


[Deep generative models for fast shower simulation in ATLAS]

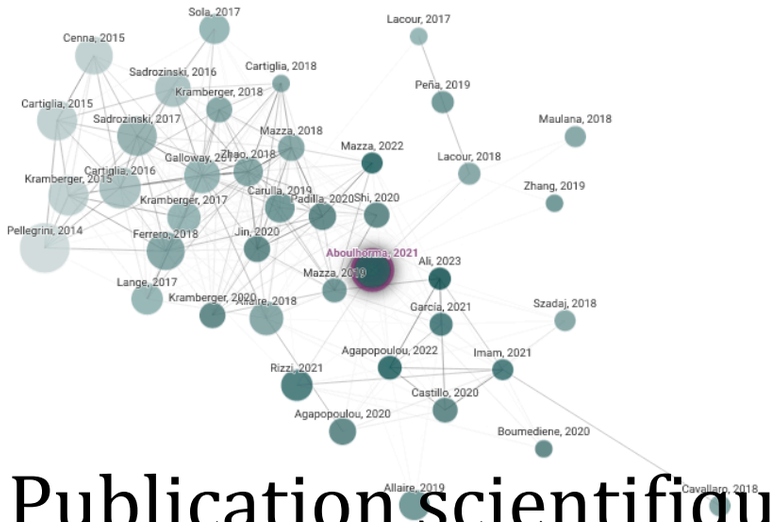
Graph Neural Network

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

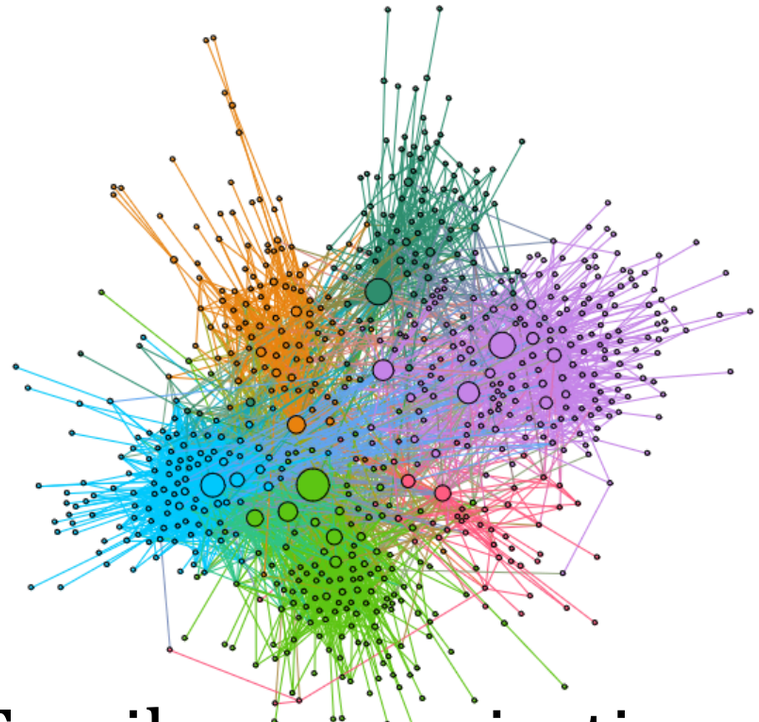
Grand nombre de structures représentable par des **graphes**



Molécule



Publication scientifique

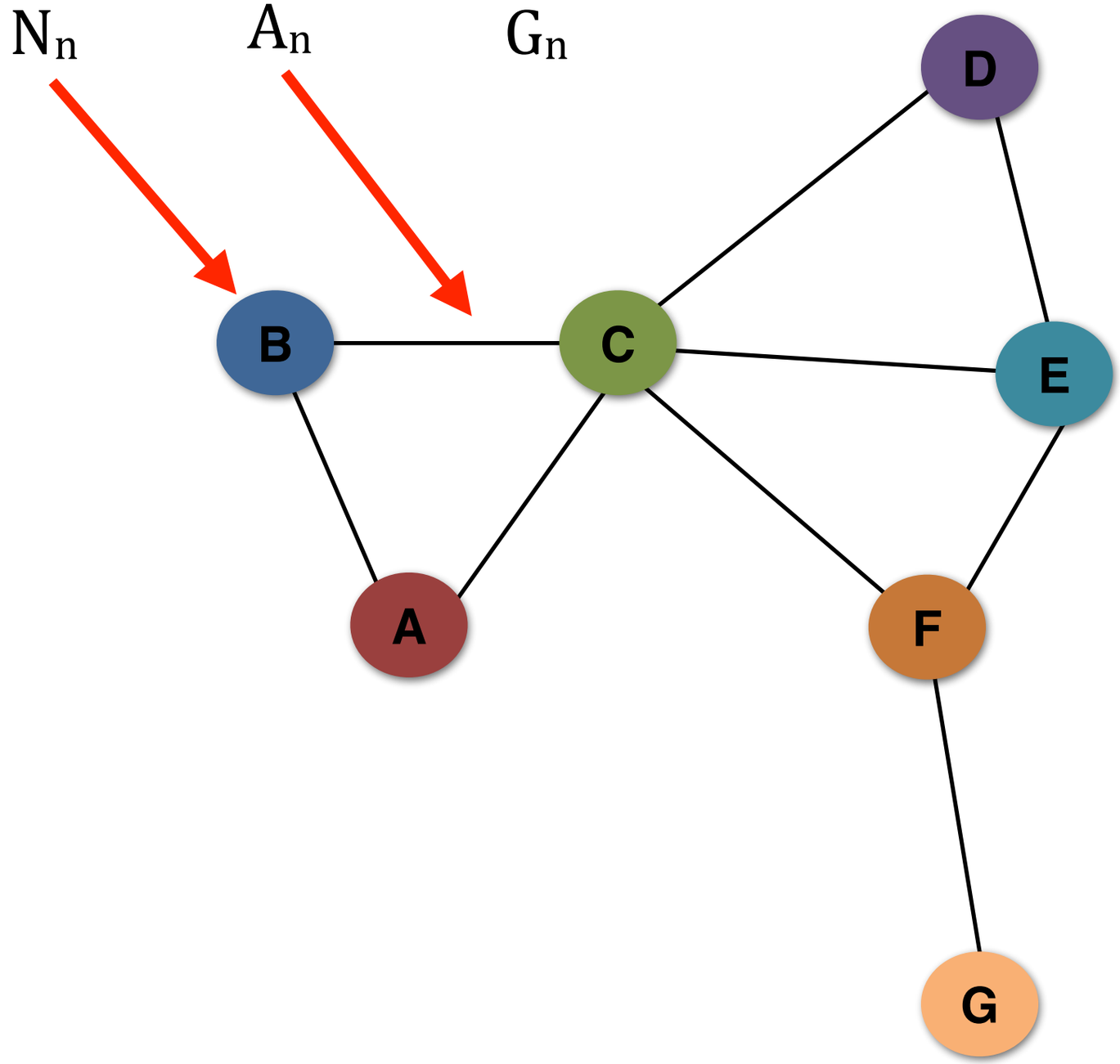


Email communication

Relation sociale

Graph Neural Network

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



Graph Neural Network

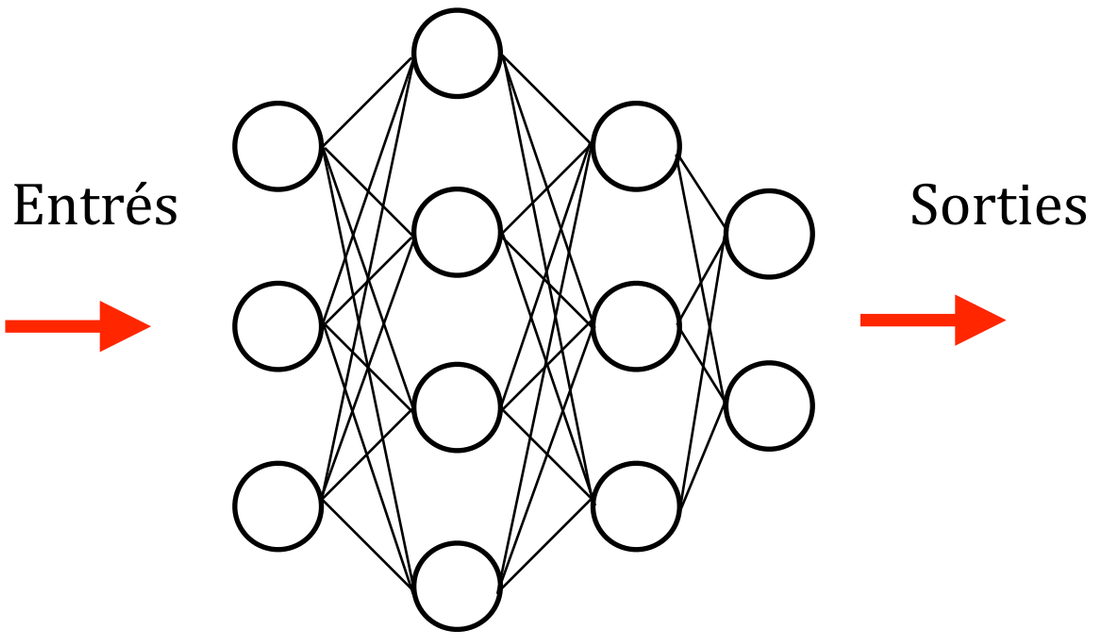
Graphe : 3 type d'info : **Noeuds**, **arêtes** et **globale**

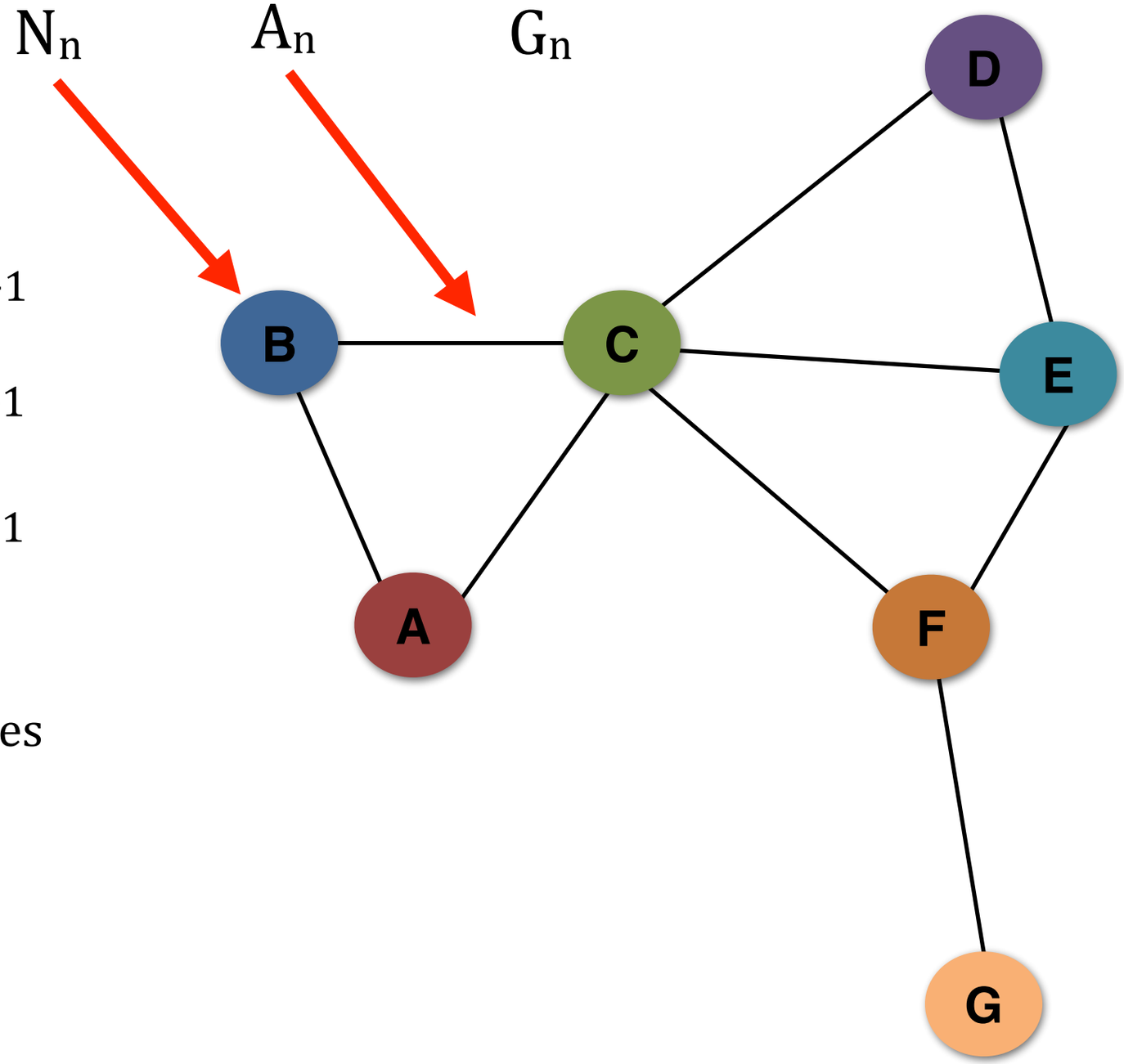
Application d'un NN à ces variables :

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

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

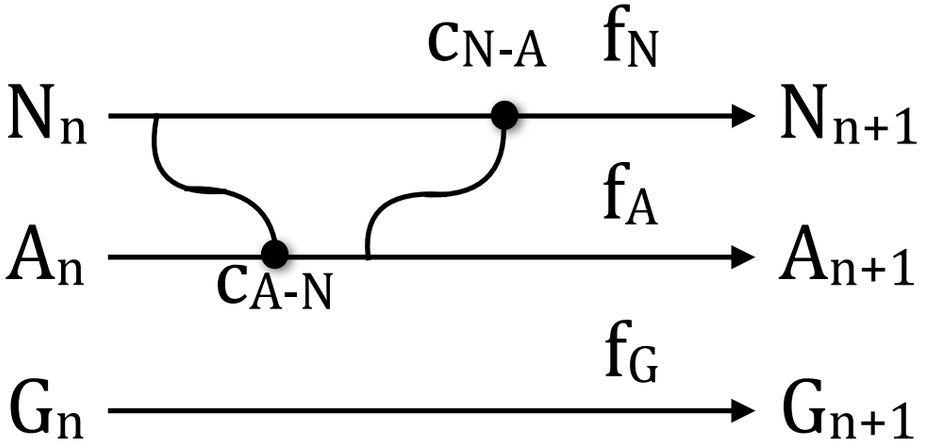
$$G_n \xrightarrow{f_G} G_{n+1}$$

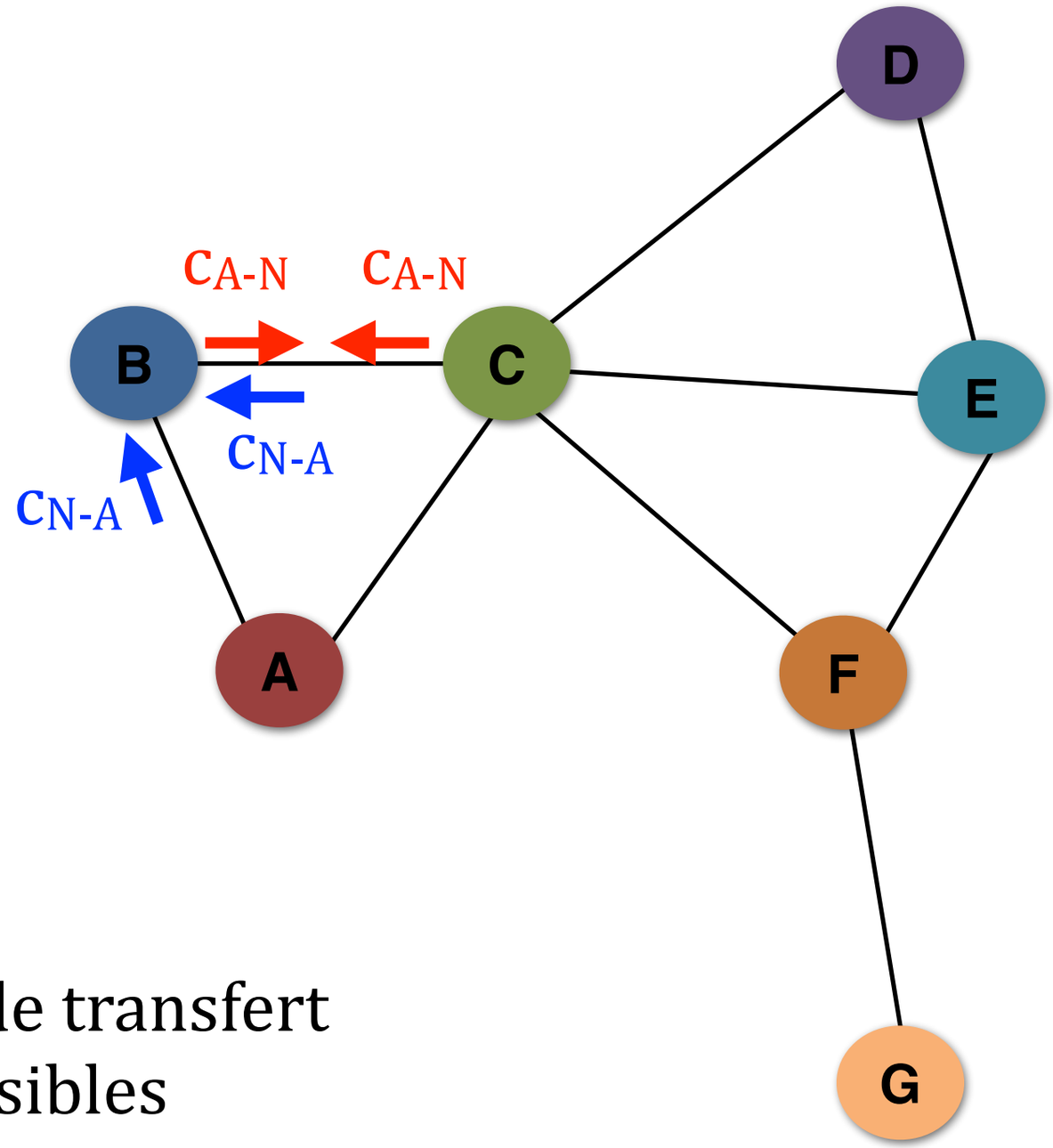




Graph Neural Network : Message Passing

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





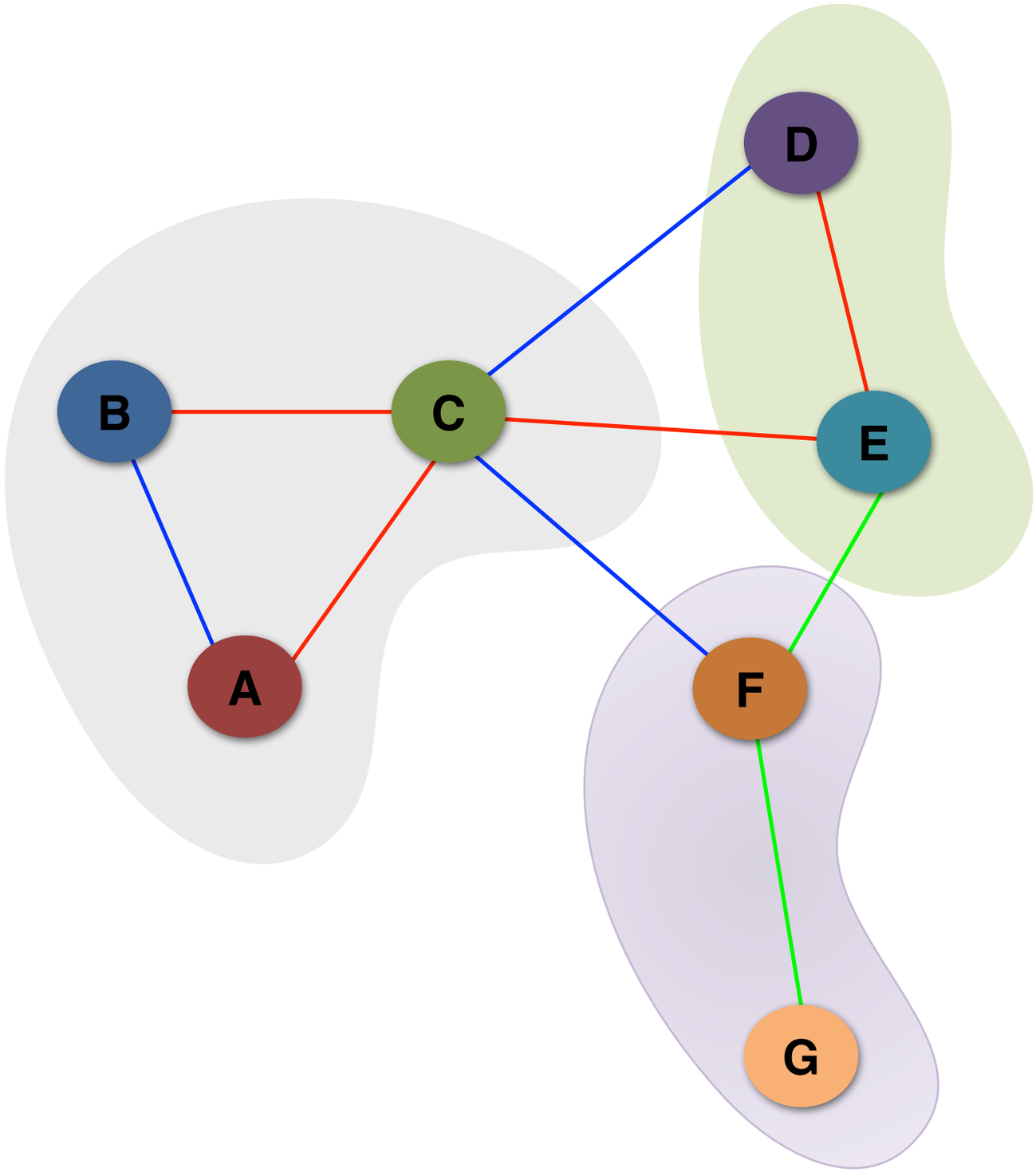
Différentes méthodes de transfert d'information sont possibles

Graph Neural Network : Classification

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

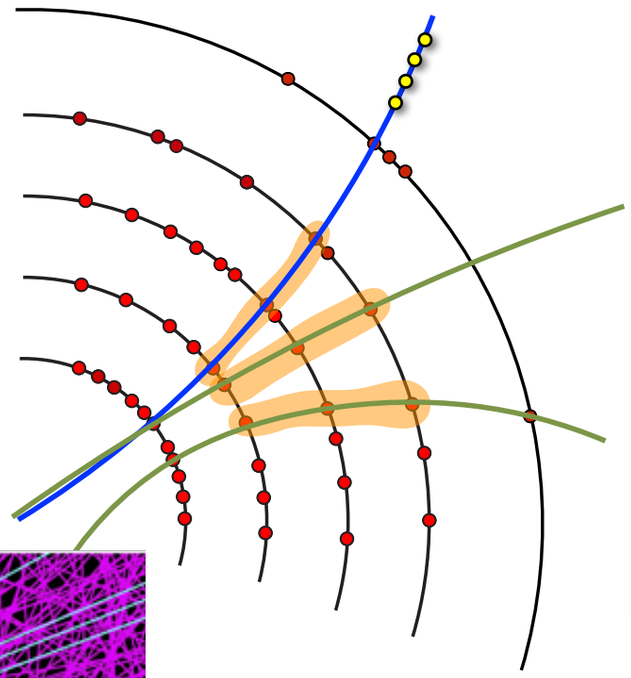
Exemples *(publications)* :

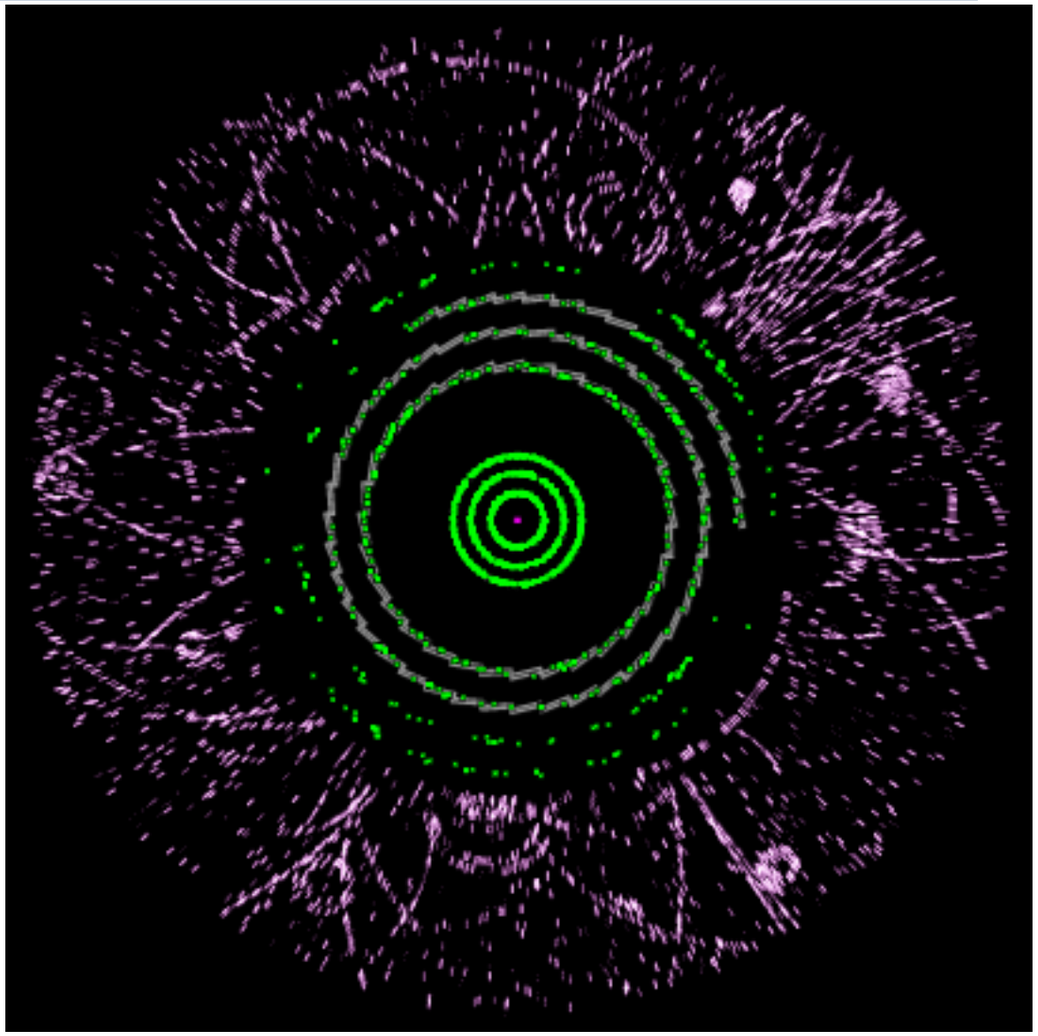
- Noeuds : Quelle catégorie de noeuds ?
  (laboratoires)
- Arêtes : Quel lien entre noeuds ?
  (relations professionnelles)
- Graph : Quel type de graph/ structure ?
  (stratégie de publication)

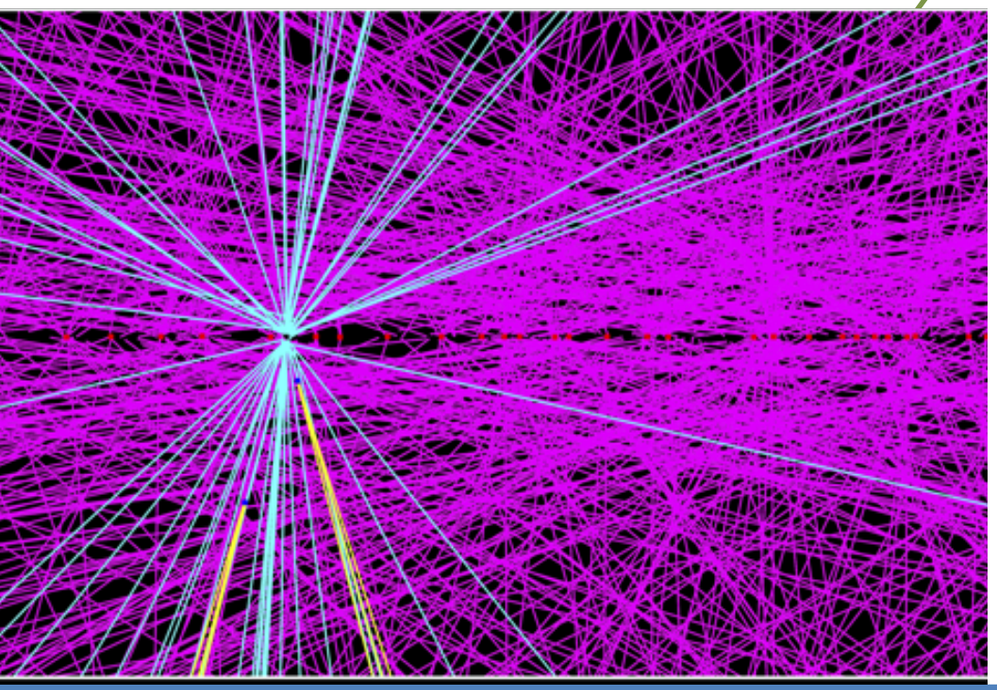


Trajectographie des particules chargées

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



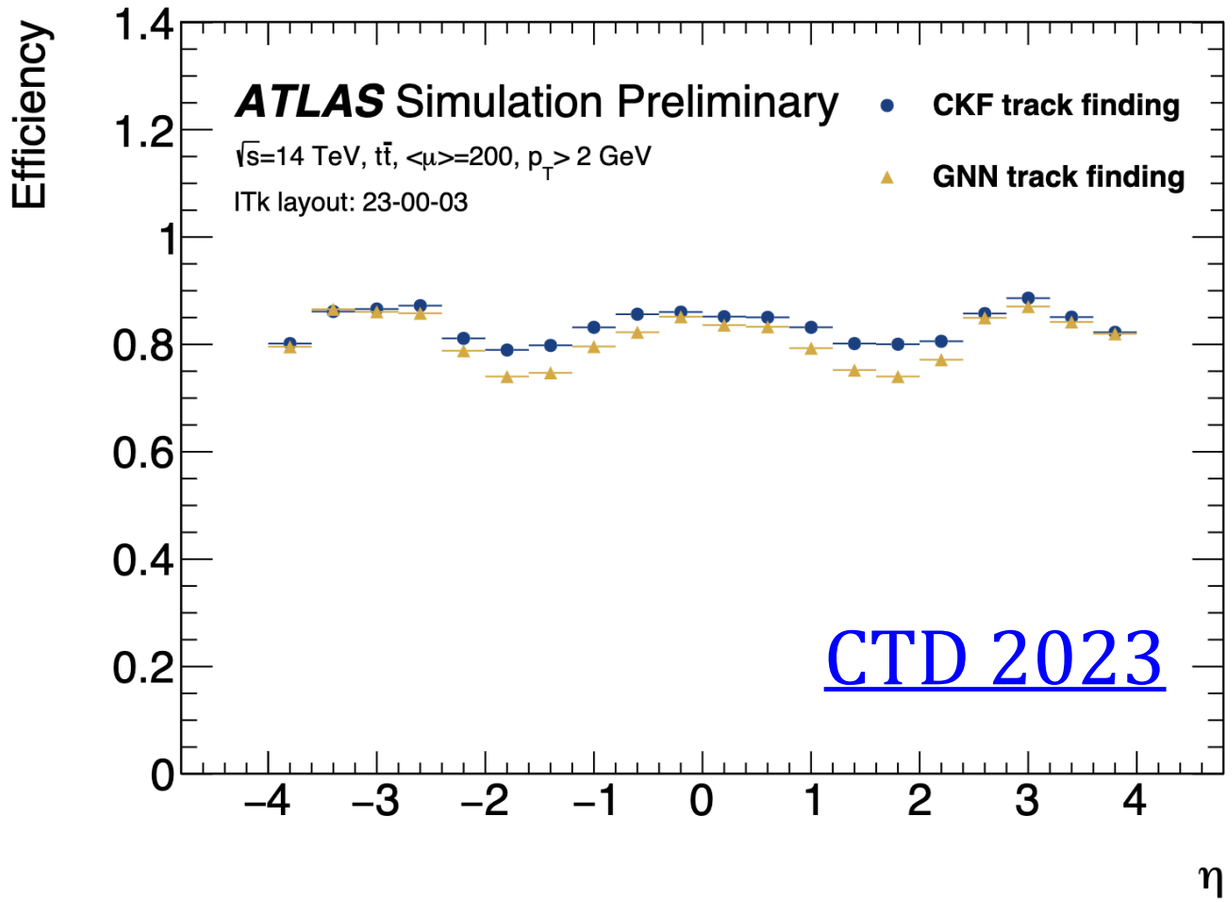


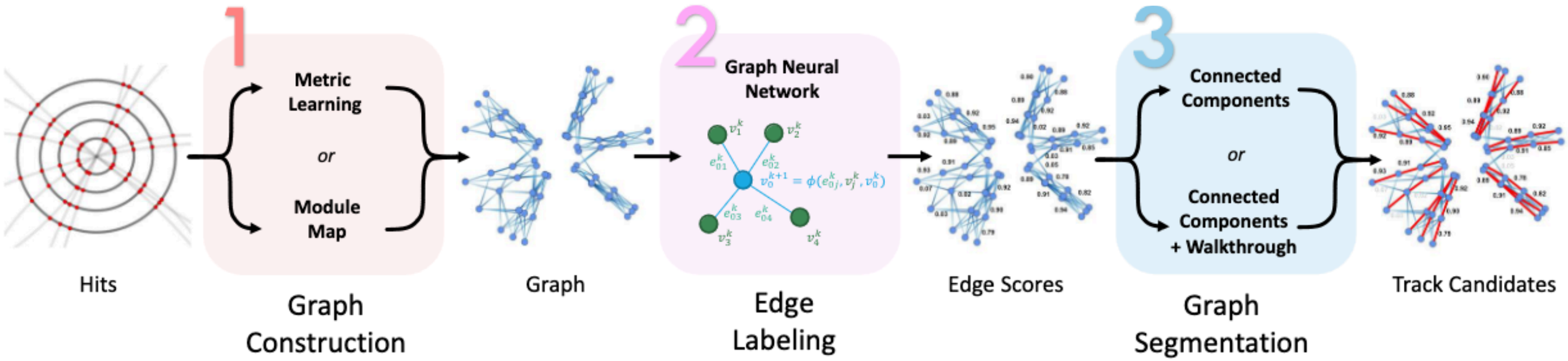


- Très complexe : beaucoup de **combinatoire** (~ 100 000 coups par collision)
- Algorithme très complexe : **Filtres de Kalman**
- Très fort coût CPU

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





[Physics Performance of the ATLAS GNN4ITk Track Reconstruction Chain]

Conclusion

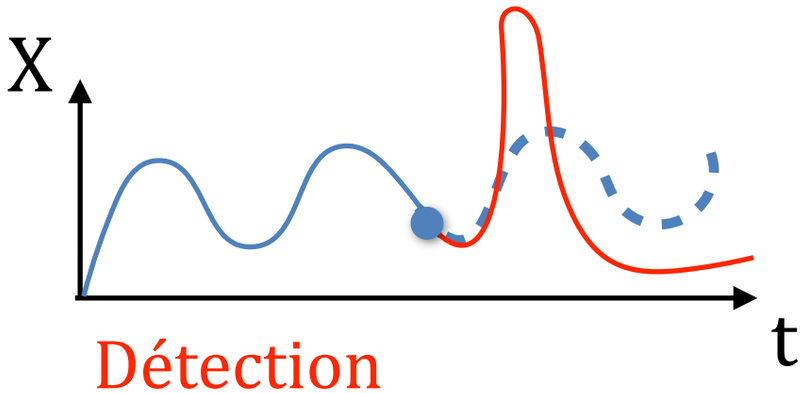
- Deep learning = Machine learning avec des réseaux de neurones
- Capable d'apprendre très efficacement de nos données
- Très grande variété d'algorithmes de machine learning ➡ adapter nos algorithmes à nos problèmes
- De plus en plus omniprésente en physique pour les tâches de reconstruction d'objets,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

Pour aller plus loin

Réseau récurent

Objectif : apprentissage à partir de séquence ou de série temporelle
Domaine : Traitement du signal, compréhension du language…

Exemple de séquence :



Détection d'anomalie

Traduction Automatique

Le film était très bien même si il était un peu long.

The film was very good even if it was a little long.

Texte

Classification d'opinion

Analyse de vidéo

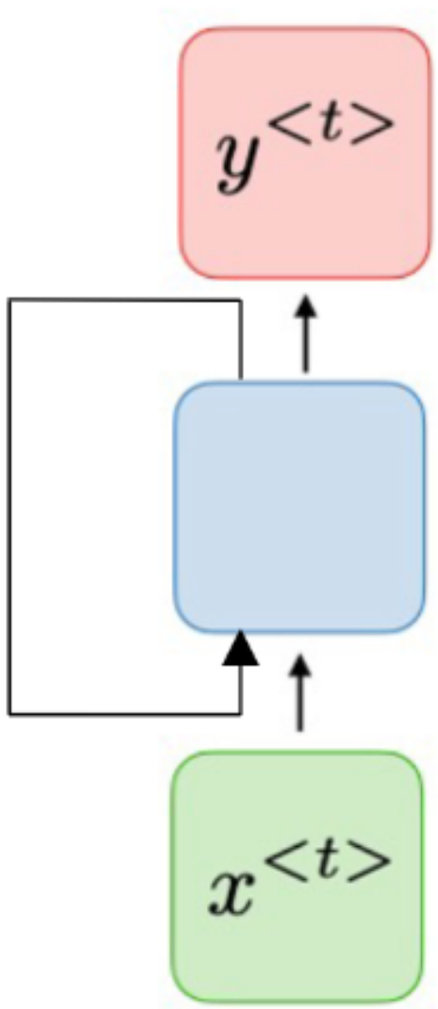
Course à pied

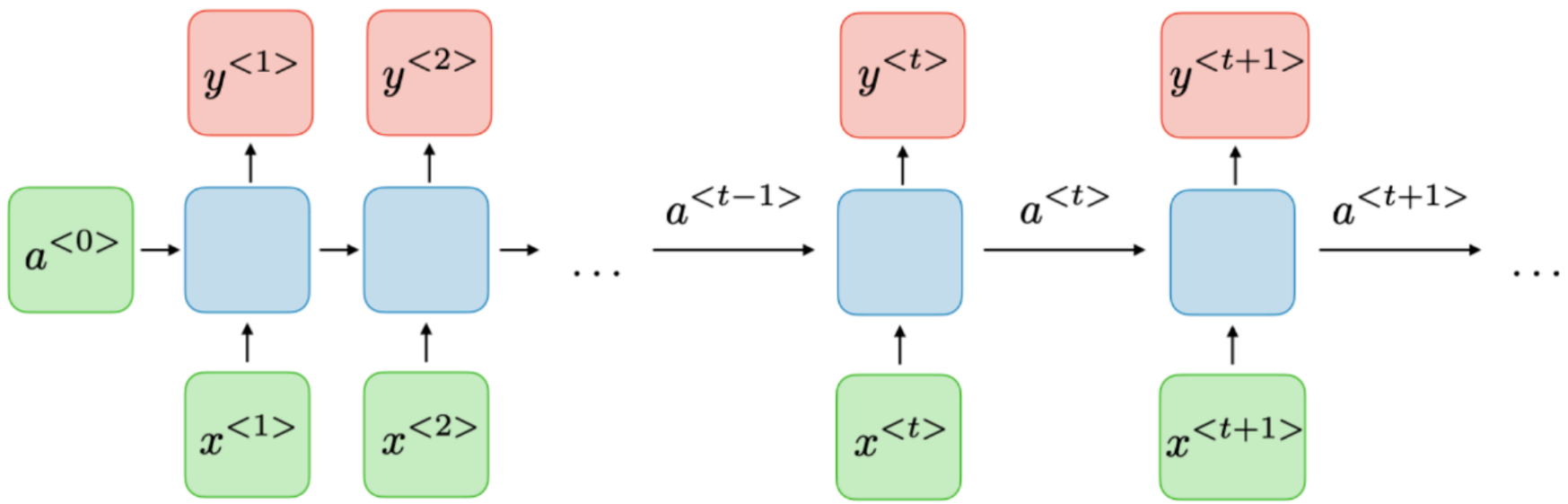
Texte

Reconnaissance de la parole

Réseau récurent

Les réseaux de neurone récurent (reculent neural network) sont des réseaux qui utilisent la sortie de l'inférence N pour l'inférence N+1

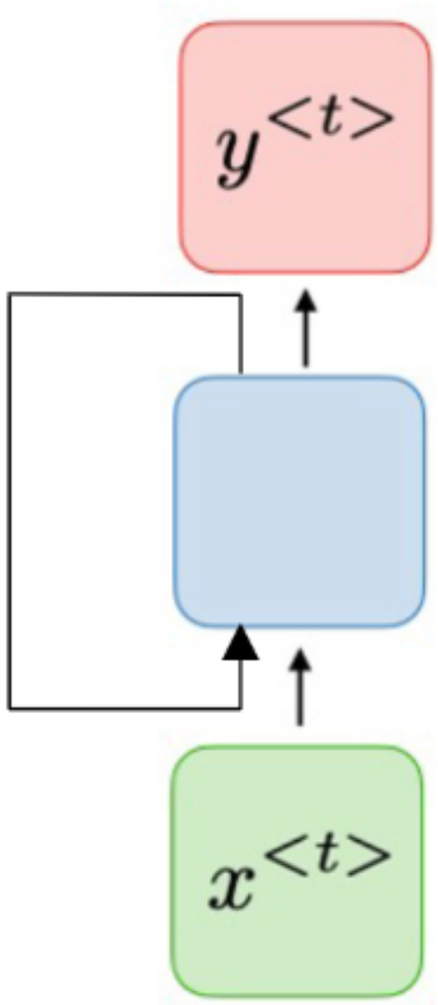




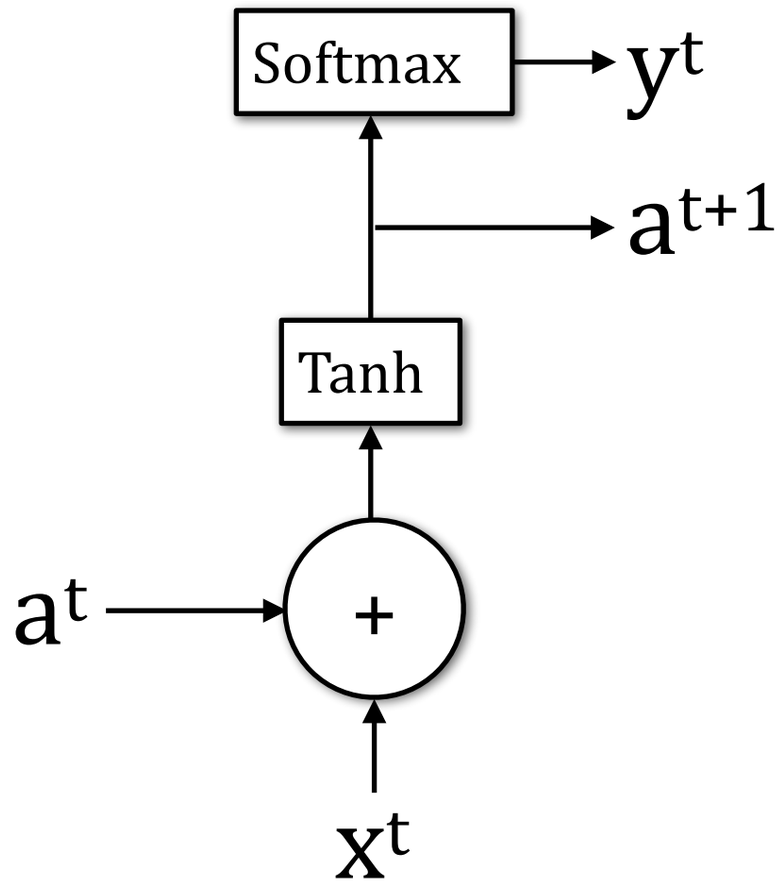
x : input au temps t
y : output au temps t
a : État propagé de t à t+1

Réseau récurent : Cell

Différentes architectures de RNN existe au niveau des cellules



Cas simple :

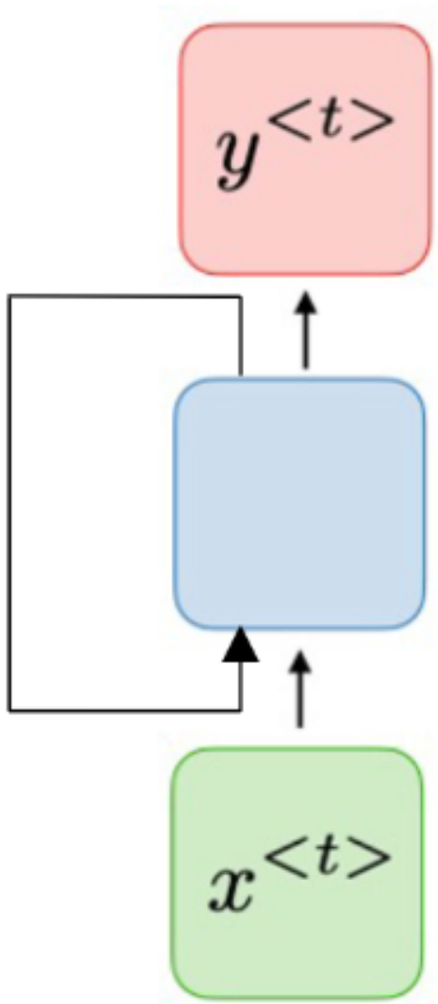


PB : évanouissement/ explosion du gradient

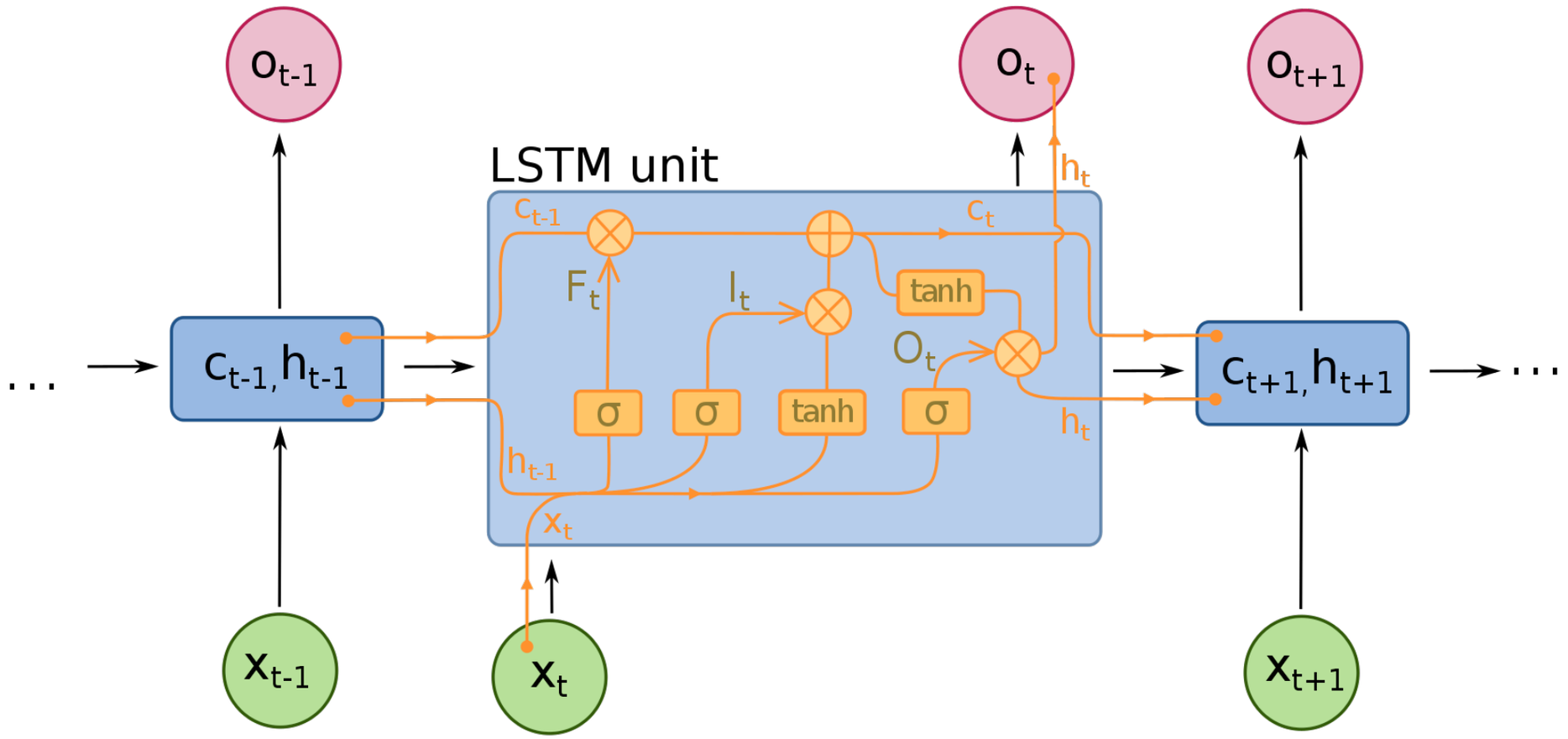
Propagation de gradient sur le long terme
-> multiplication de gradient
-> tend vers 0 ou l'infini

Réseau récurent : Cell

Différentes architectures de RNN existe au niveau des cellules

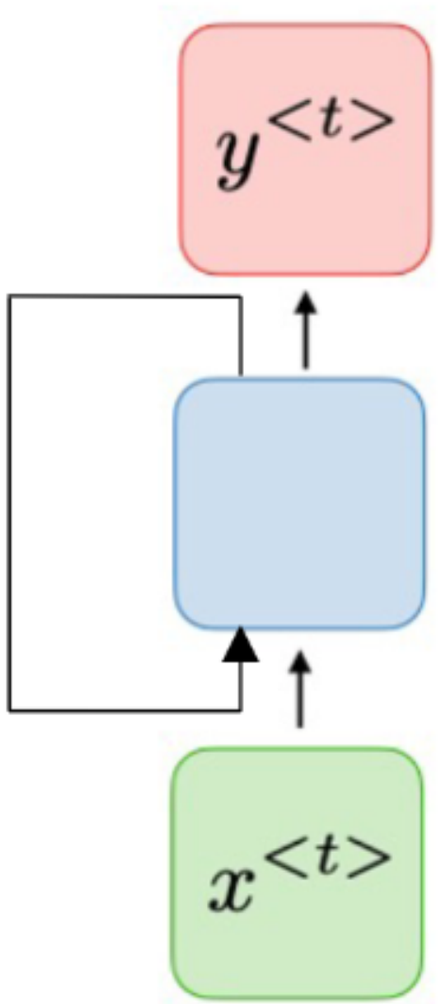


Long short-term memory :

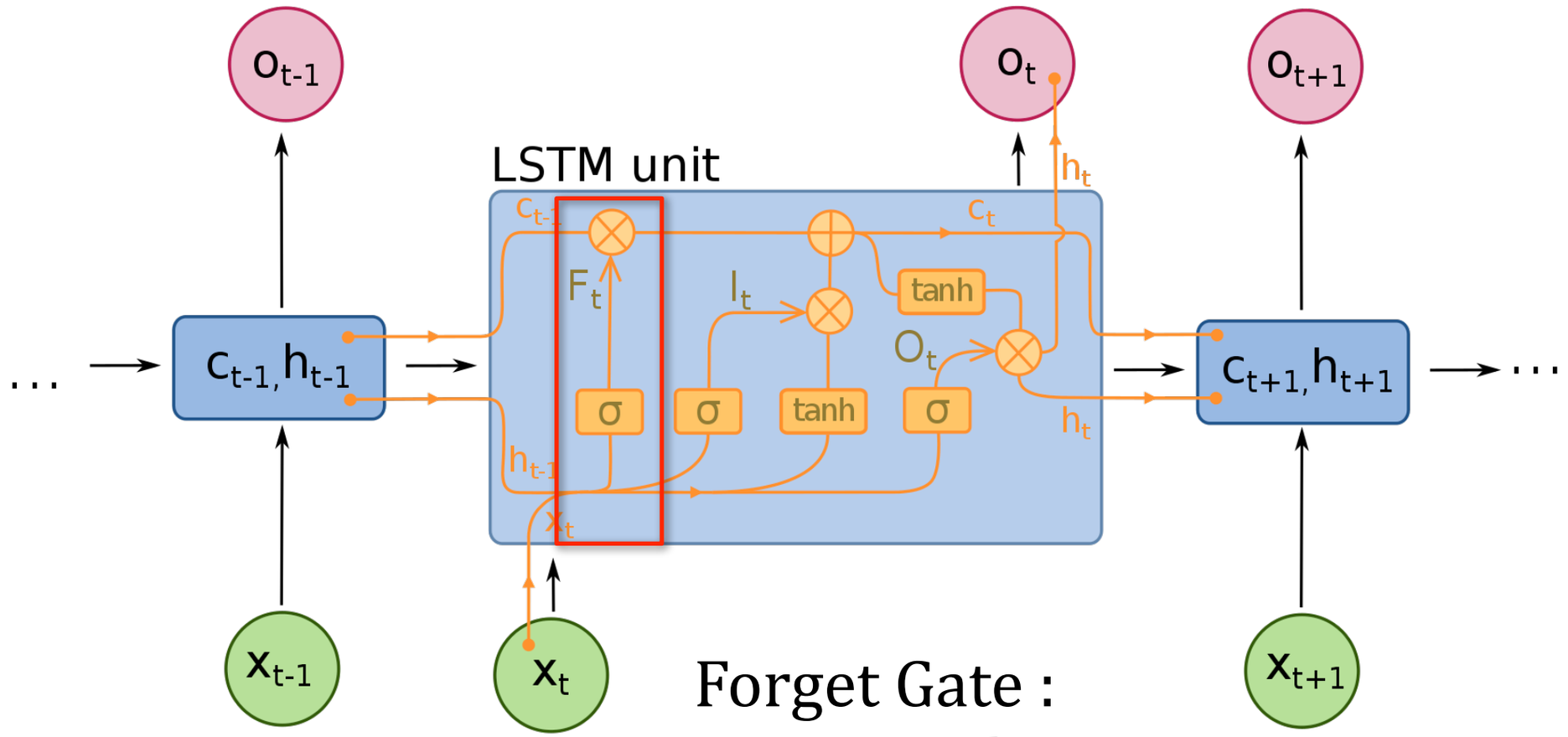


Réseau récurent : Cell

Différentes architectures de RNN existe au niveau des cellules



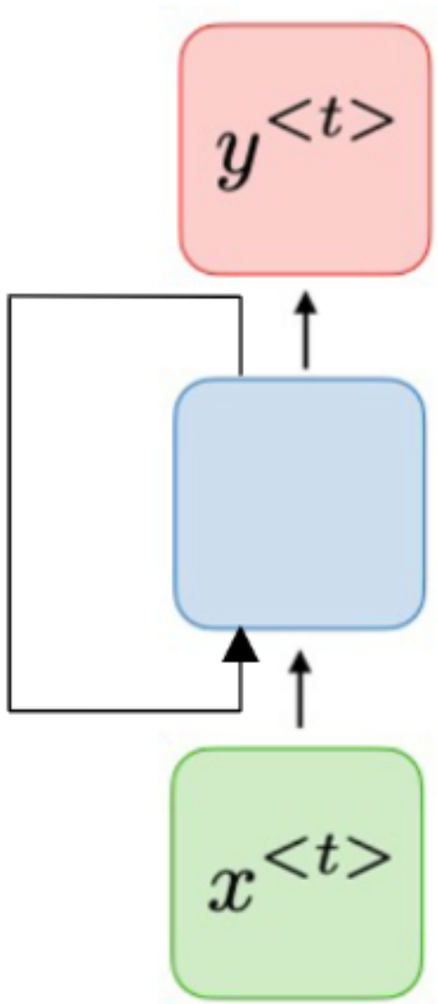
Long short-term memory :



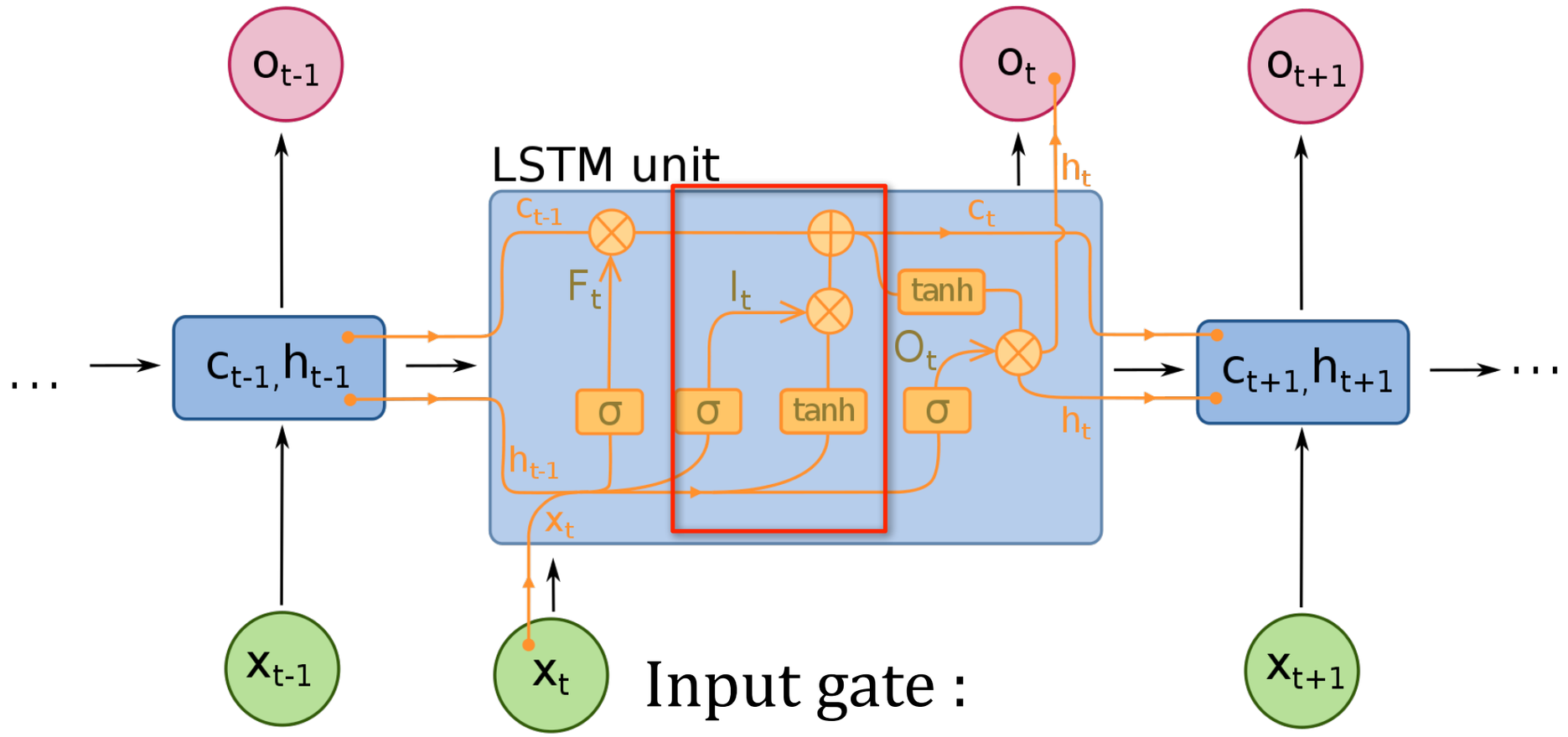
Forget Gate : supprime les informations non importantes

Réseau récurent : Cell

Différentes architectures de RNN existe au niveau des cellules



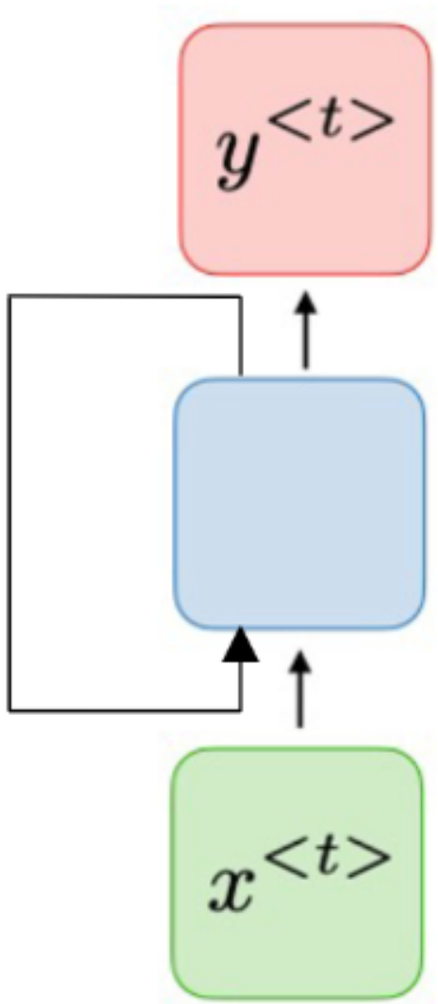
Long short-term memory :



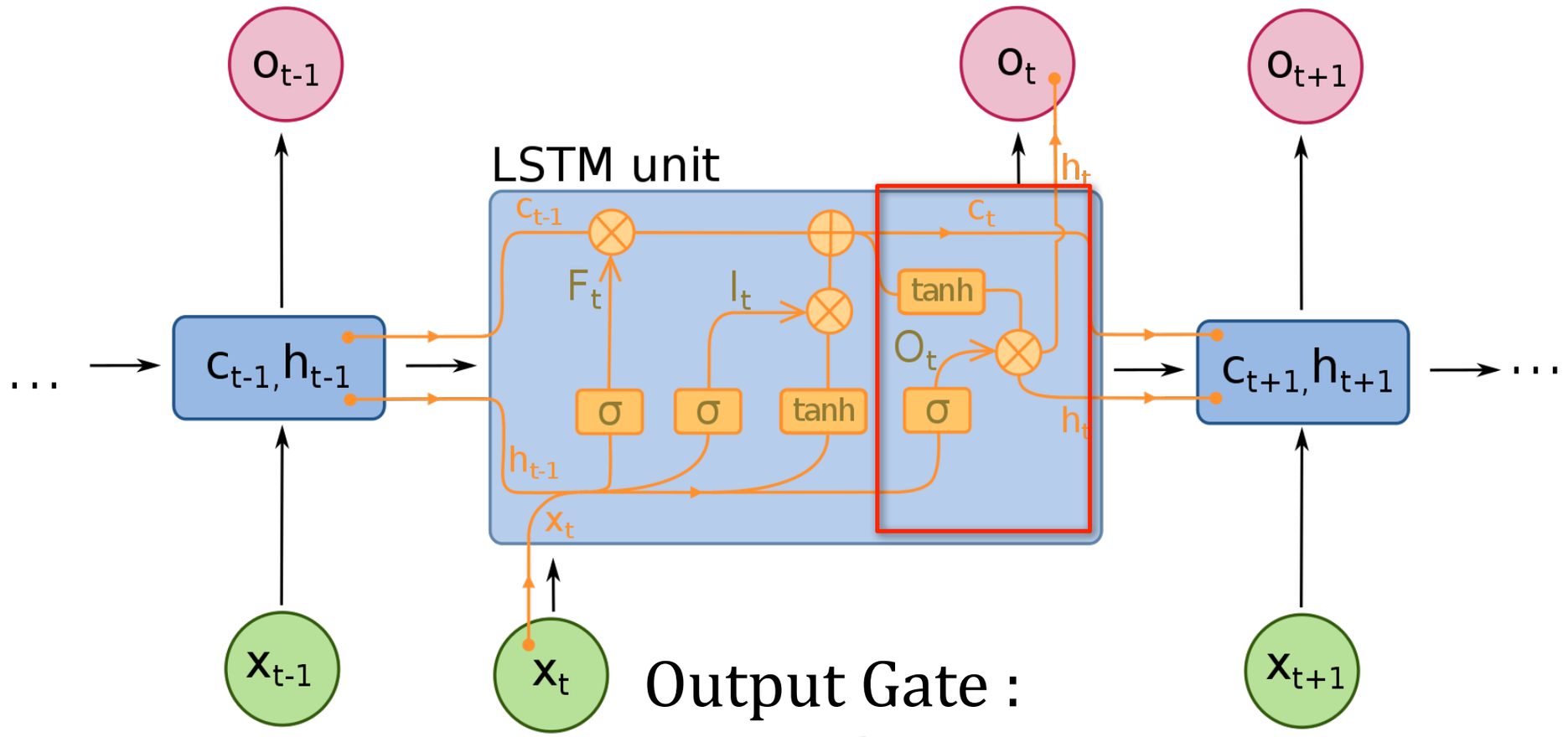
Input gate :
Passe uniquement les info jugées importantes

Réseau récurent : Cell

Différentes architectures de RNN existe au niveau des cellules



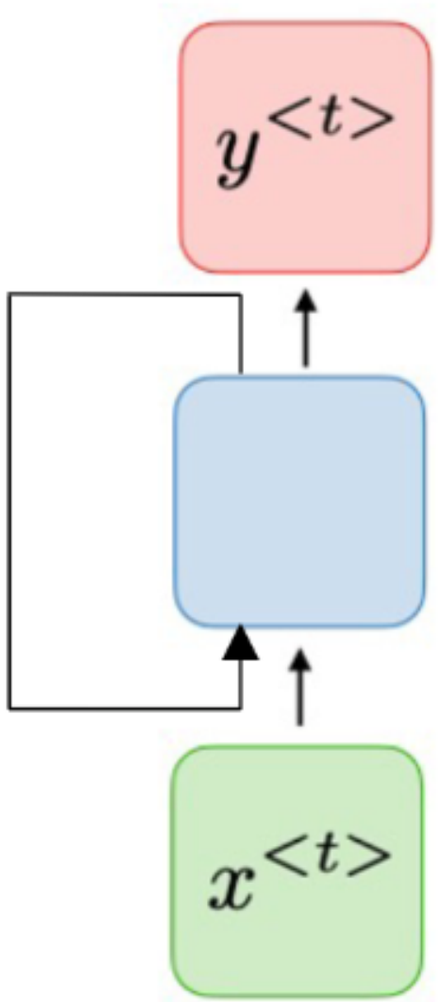
Long short-term memory :



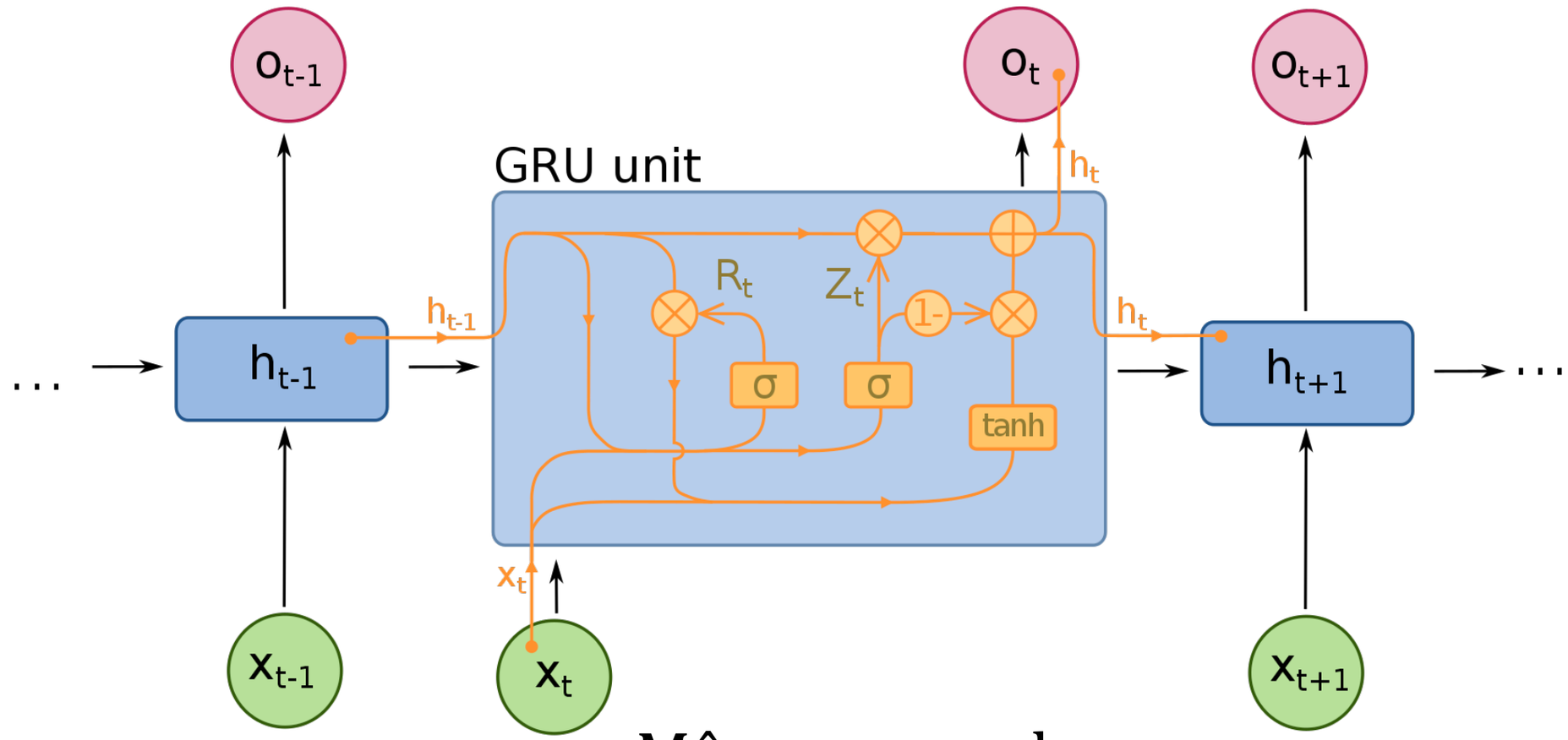
Output Gate :
Extrait les informations utiles

Réseau récurent : Cell

Différentes architectures de RNN existe au niveau des cellules



Gated recurrent unit :



Même approche sans Output Gate

https://arxiv.org/abs/1412.3555

Transformer

- Difficulté de memoire dans les RNN -> les réseaux oublient l'information initiale

- Solution : Transformer (https://arxiv.org/abs/1706.03762)

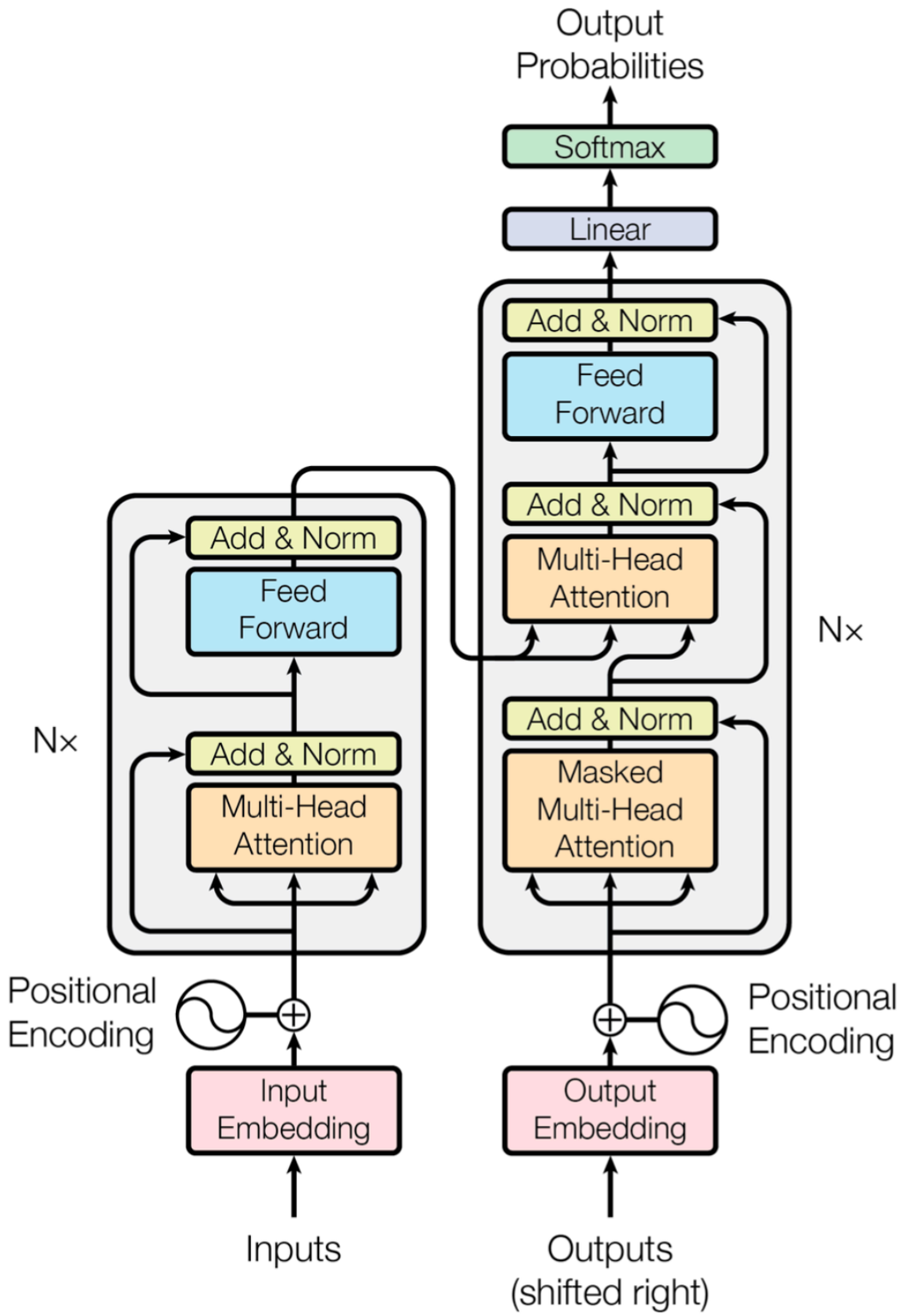
- Réseaux utilisant un mécanisme d'attention pour garder en mémoires les informations

- ENORME succès : application à la plupart de domaines capable de classification, d'analyse et de génération

- Base de tout les Large Language Model (LLM) dont chatGPT

Transformer

- Mécanisme d'attention : apprend les liens entres les différents éléments
- Encoding : apprend les liens entre les différent élément de l'input
- Decoding : combine ce qui à déjà été produit avec résultat de l'encoding pour produire un nouvel output

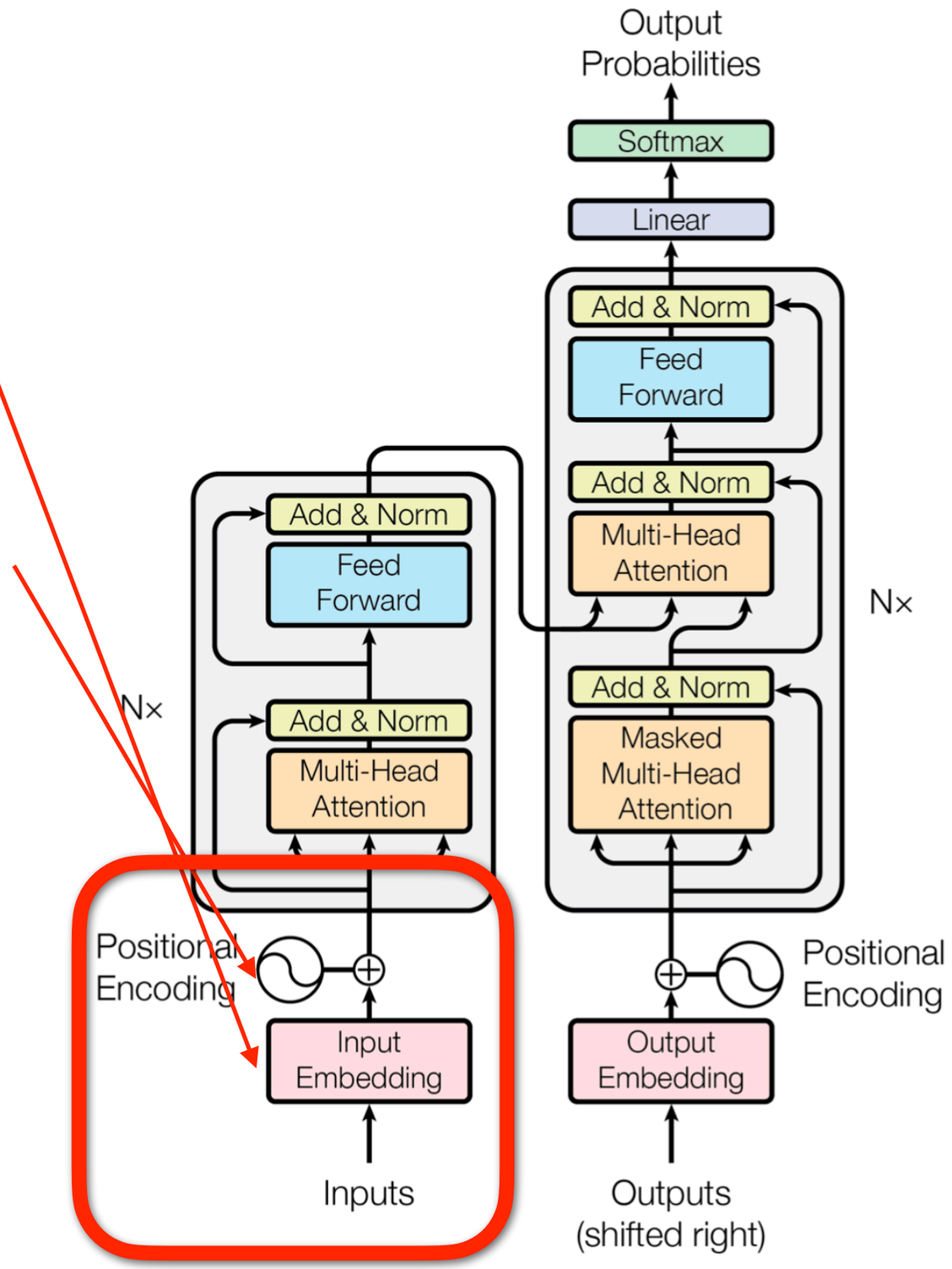


Transformer : Embedding

- Transforme chaque élément de l'input en vecteur taille d :

  $x_1, x_2, \ldots, x_N$

- Dans le cadre du language la position dans la phrase doit aussi être encodé -> modification du vecteur



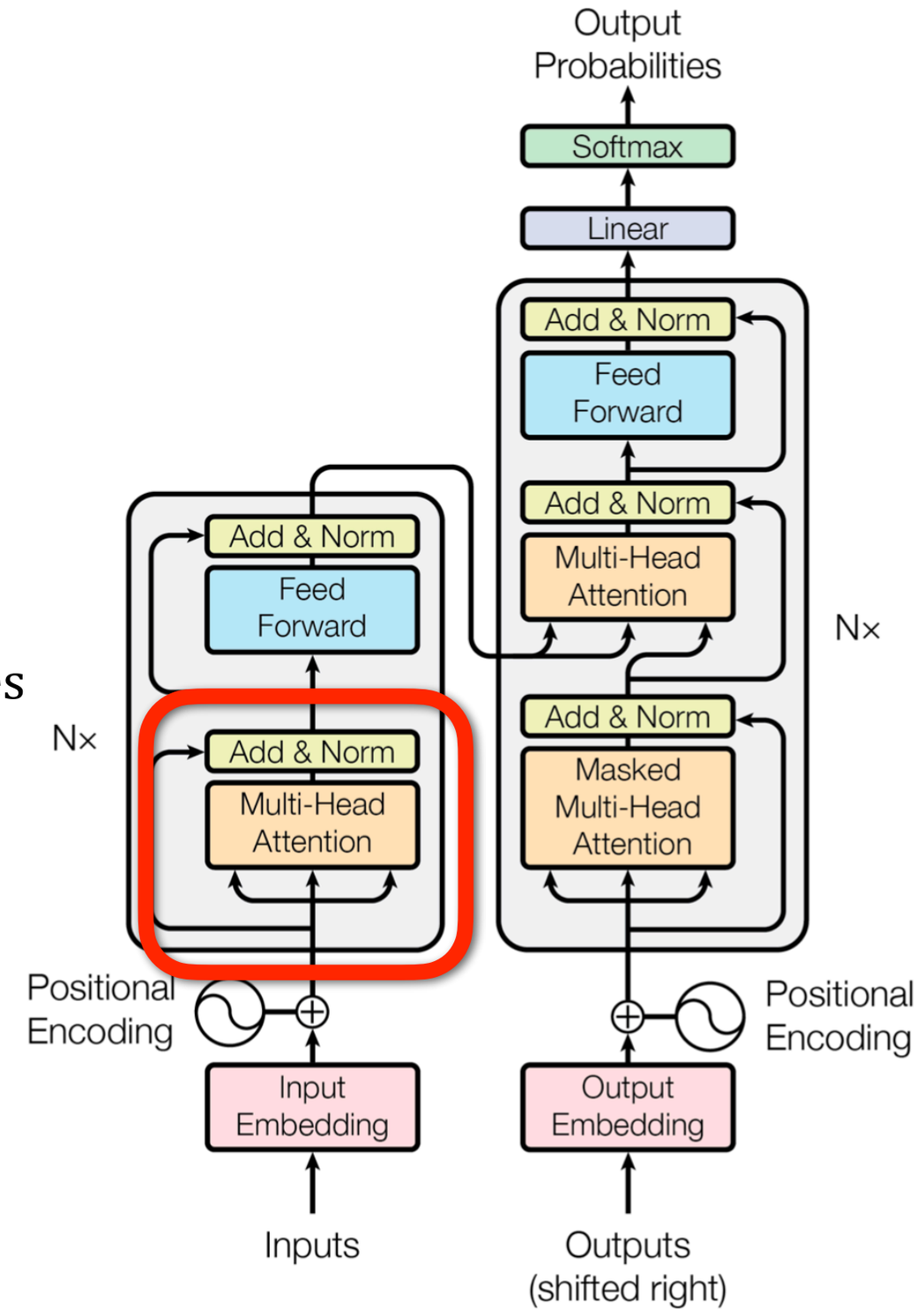
Transformer : Attention

- Objectif : apprendre les liens qui existent entre les inputs
- Transforme les input en 3 vecteurs de taille d_k par multiplication matricielle :

Embedding : x_i

Queries : $q_i = x_i \times W^Q$
Keys : $k_i = x_i \times W^K$
Values : $v_i = x_i \times W^V$

Paramètres appris



Transformer : Attention

Example pour q_2 :

$$score1 = q_2 \times k_1$$
$$score1 = score1/\sqrt{d}$$
$$score1 = SoftMax(score1) = 0.9$$
$$score1 = score1.v_1$$

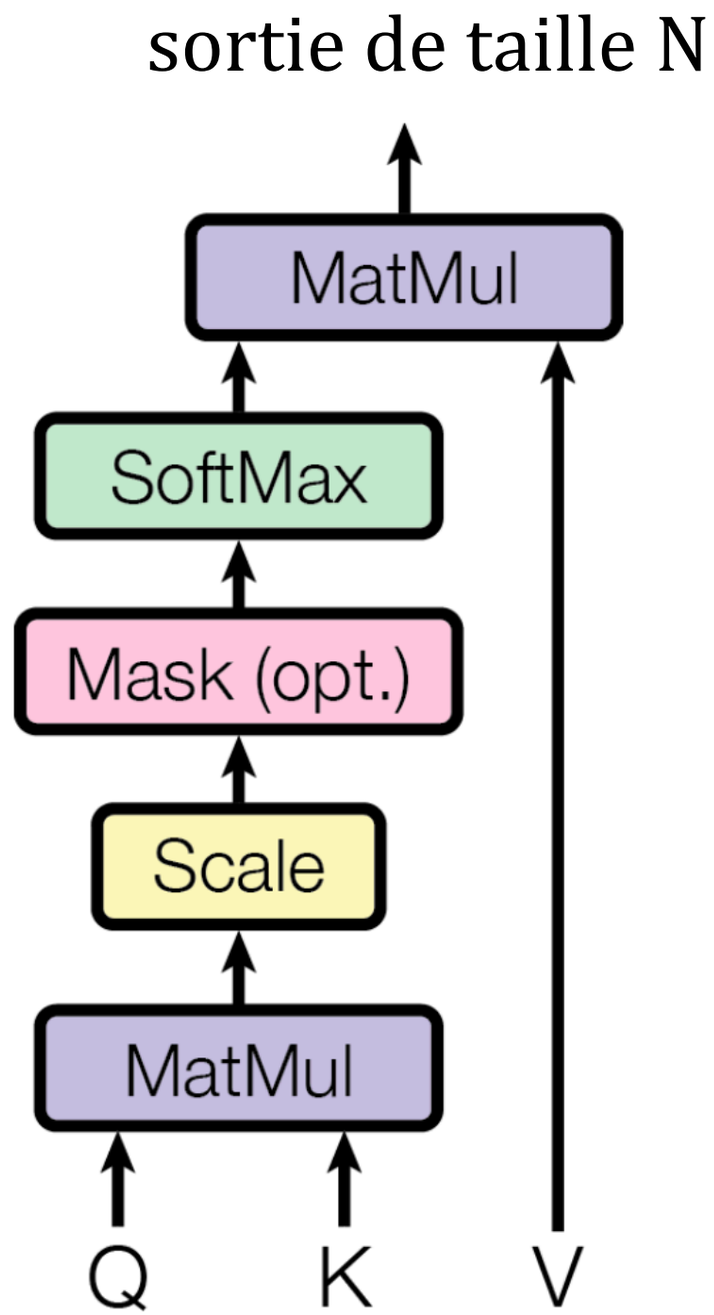
$$score2 = q_2 \times k_2$$
$$score2 = score2/\sqrt{d}$$
$$score2 = SoftMax(score2) = 0.1$$
$$score2 = score2.v_2$$

...

$$z_2 = \sum_{I=1}^{N} scoreI$$

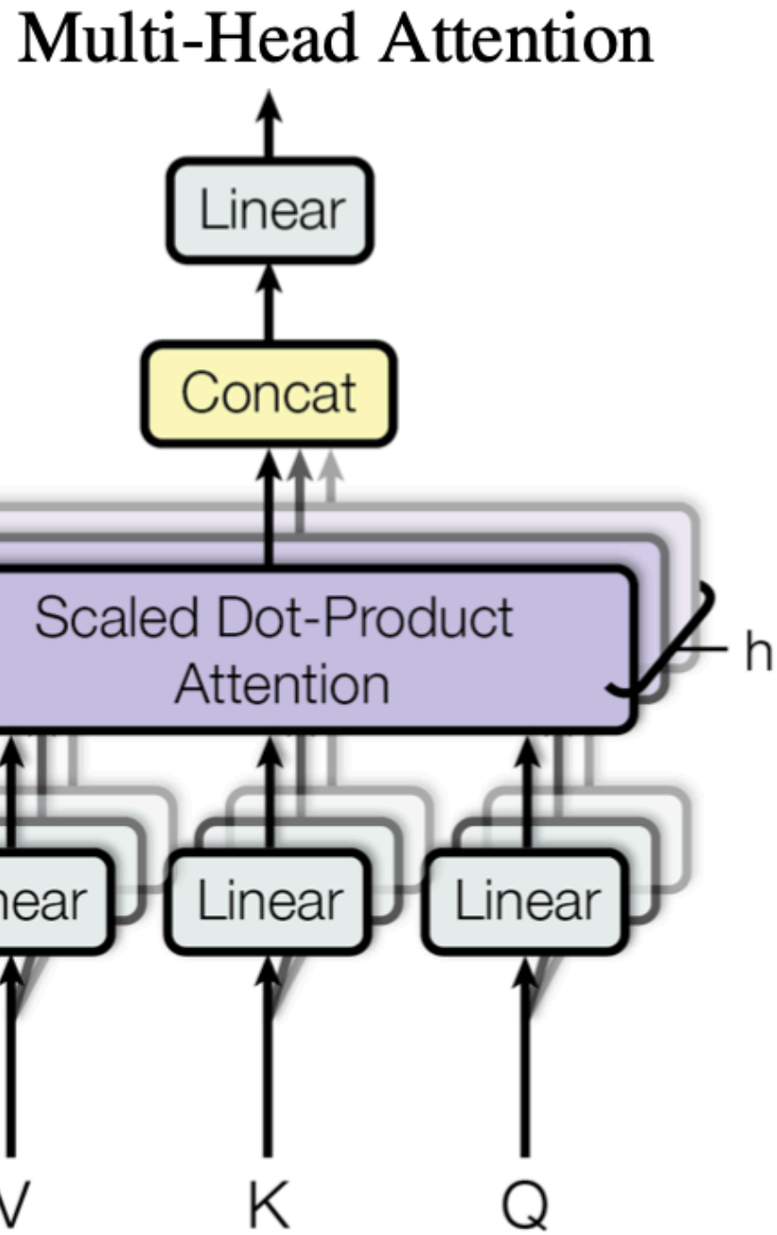
Fort lien

Faible lien



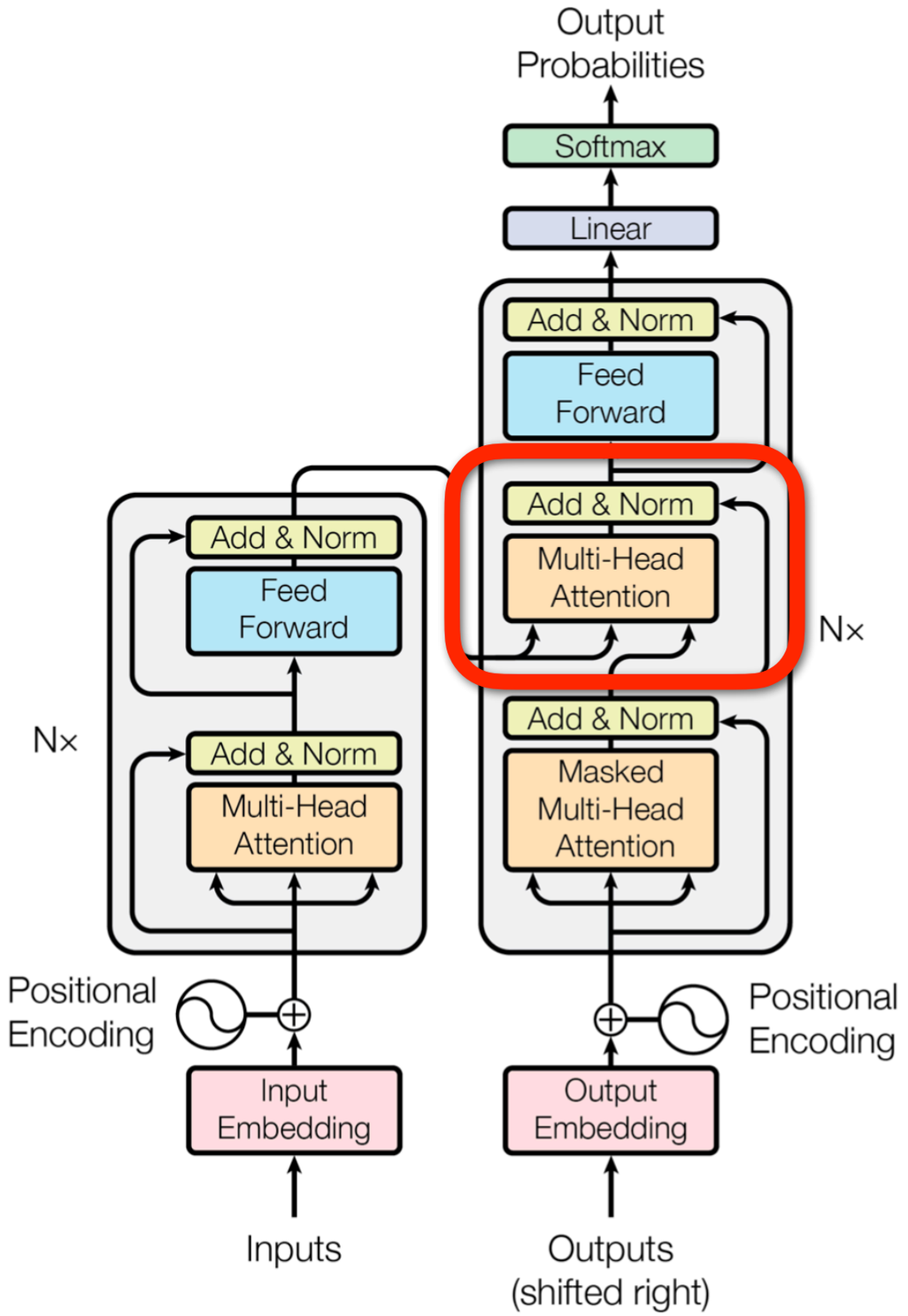
Transformer : Attention

- Opération effectuée h fois en parallèle tel que : $d = h \times d_k$
- Plus efficace et rapide
- Résultat combiné a-posteriori



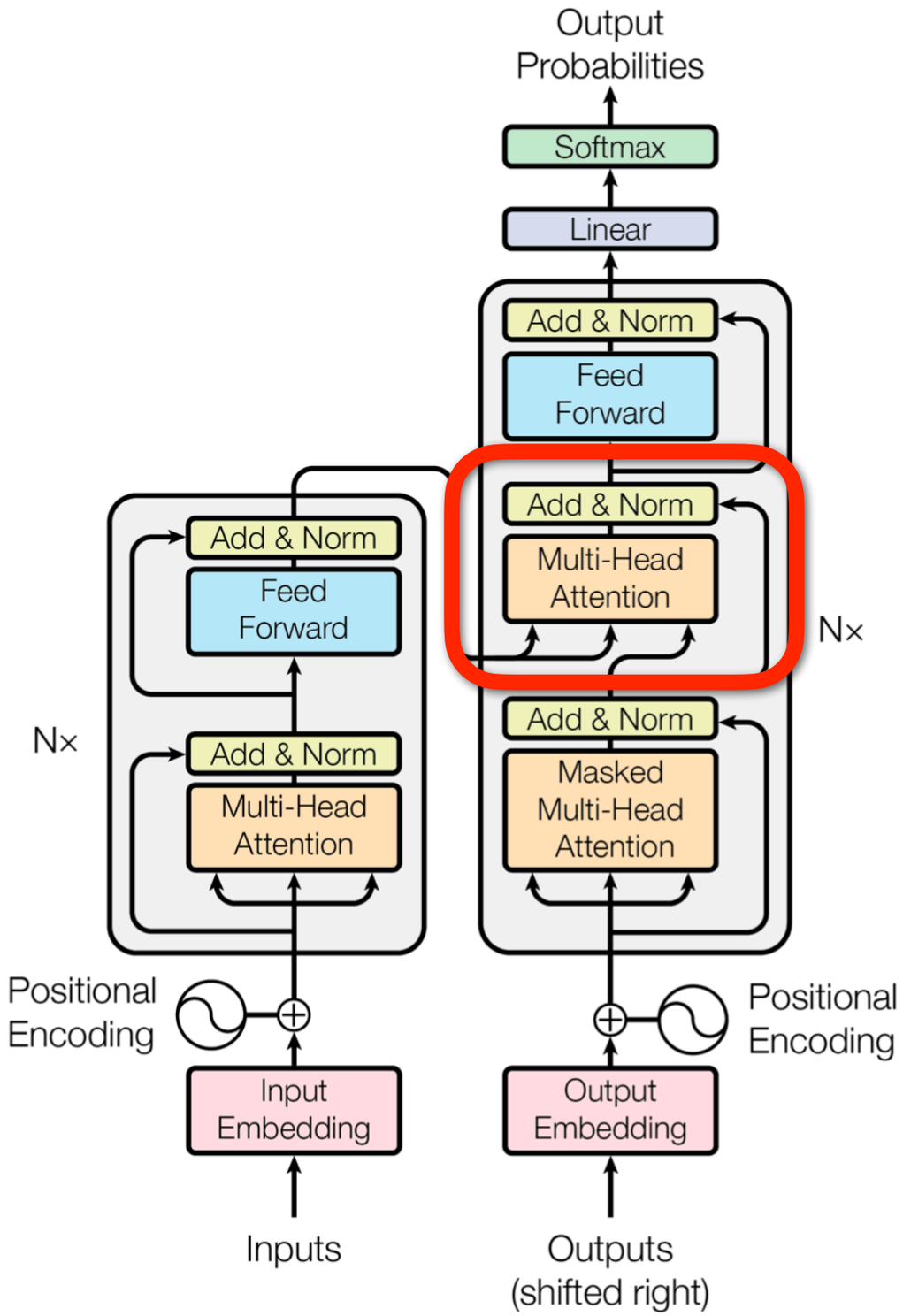
Transformer : Décodeur

- Après l'attention passage dans un NN
- Dans le décodeur les queries q' des output précédent sont comparé aux keys et values de l'encodeur
- L'opération ce répète jusqu'a ce que l'output corresponde au caractère de fin d'opération



Transformer : Exemple

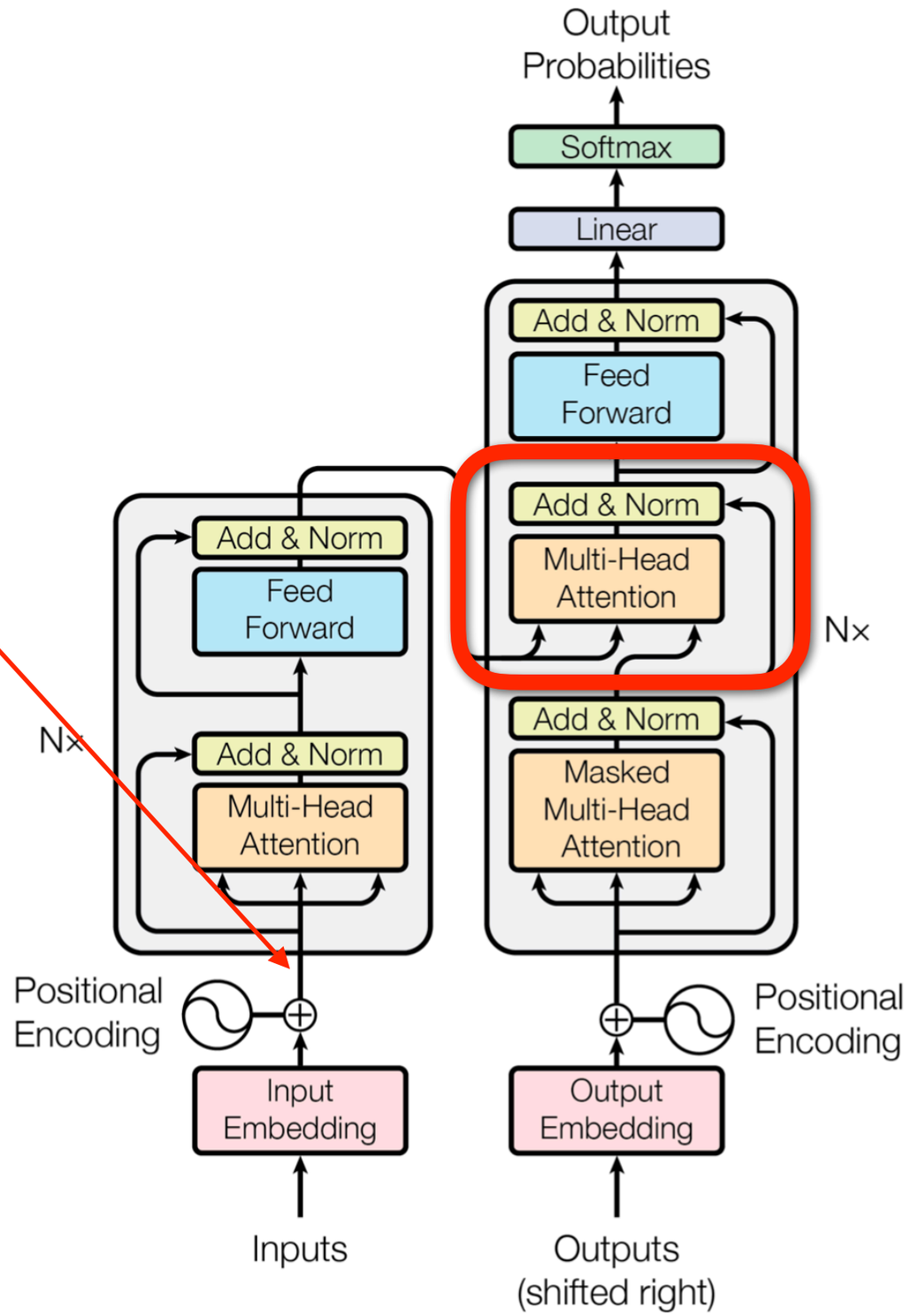
Traduction : « Je mange un kiwi »



Transformer : Exemple

Traduction : « Je mange un kiwi »

Embedding : [x_1, x_2, x_3, x_4]

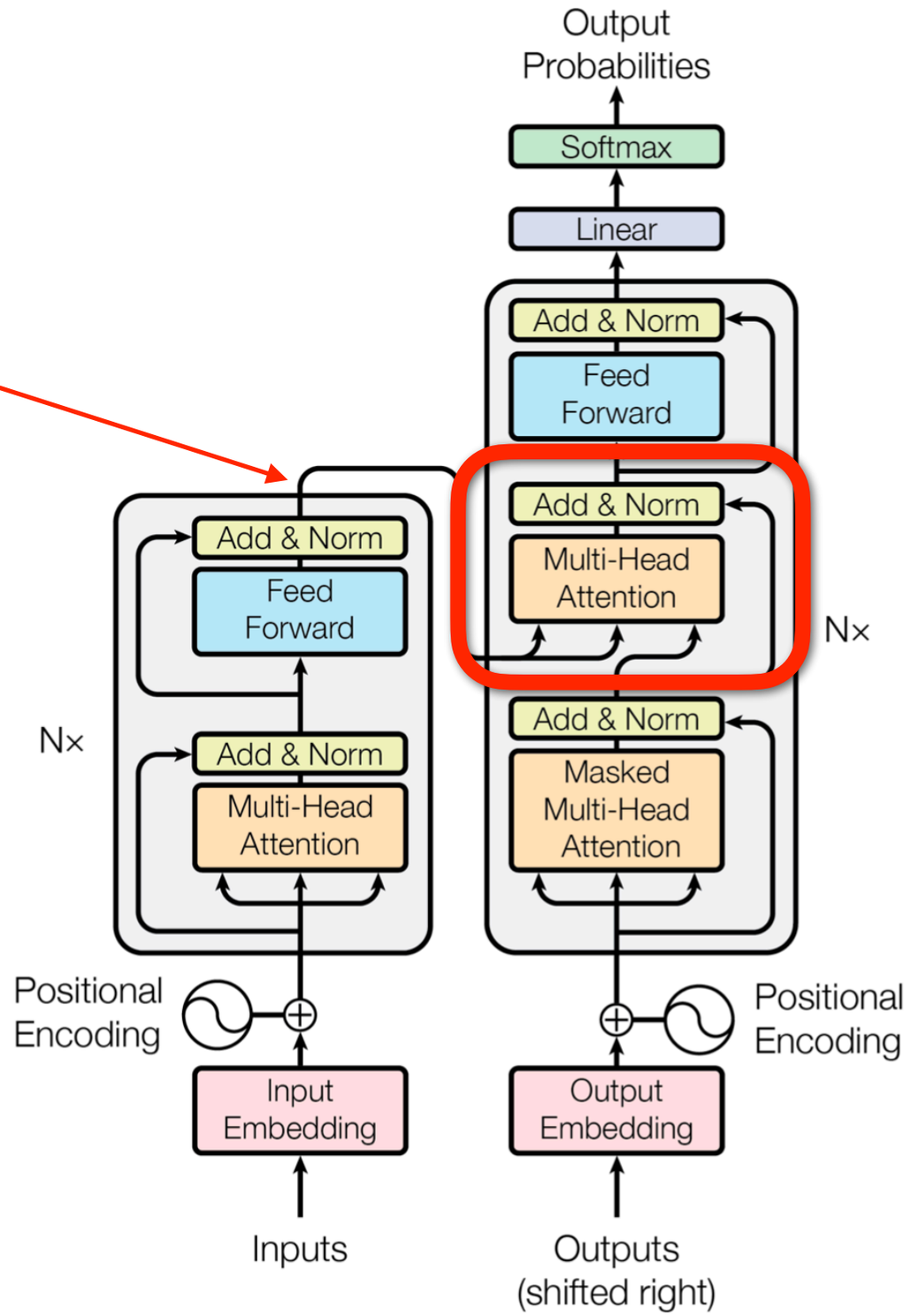


Transformer : Exemple

Traduction : « Je mange un kiwi »

Embedding : $[x_1, x_2, x_3, x_4]$

N boucle d'encodage : $[z_1, z_2, z_3, z_4]$



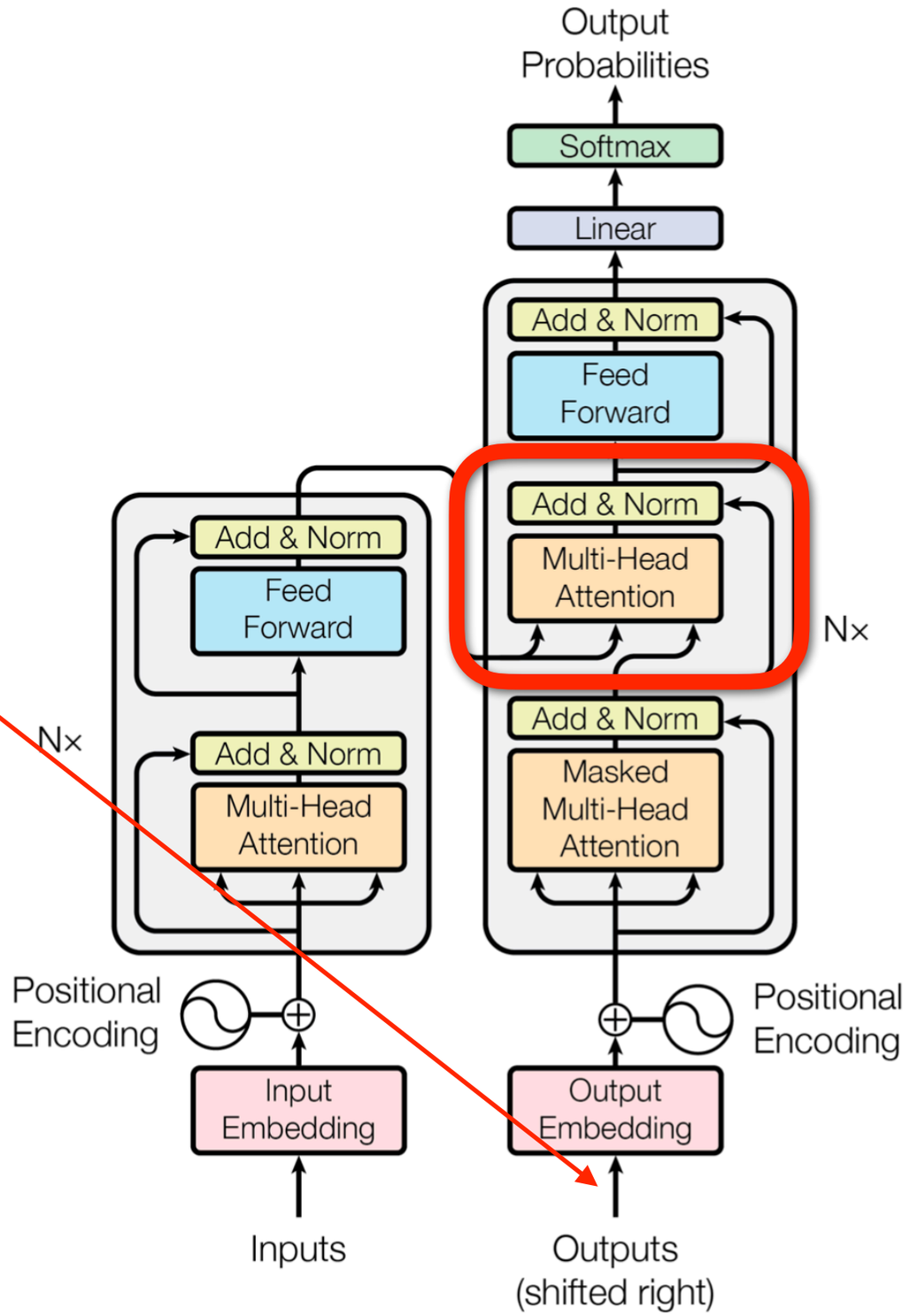
Transformer : Exemple

Traduction : « Je mange un kiwi »

Embedding : $[x_1, x_2, x_3, x_4]$

N boucle d'encodage : $[z_1, z_2, z_3, z_4]$

Décodage : Output = $\emptyset$ -> o_0



Transformer : Exemple

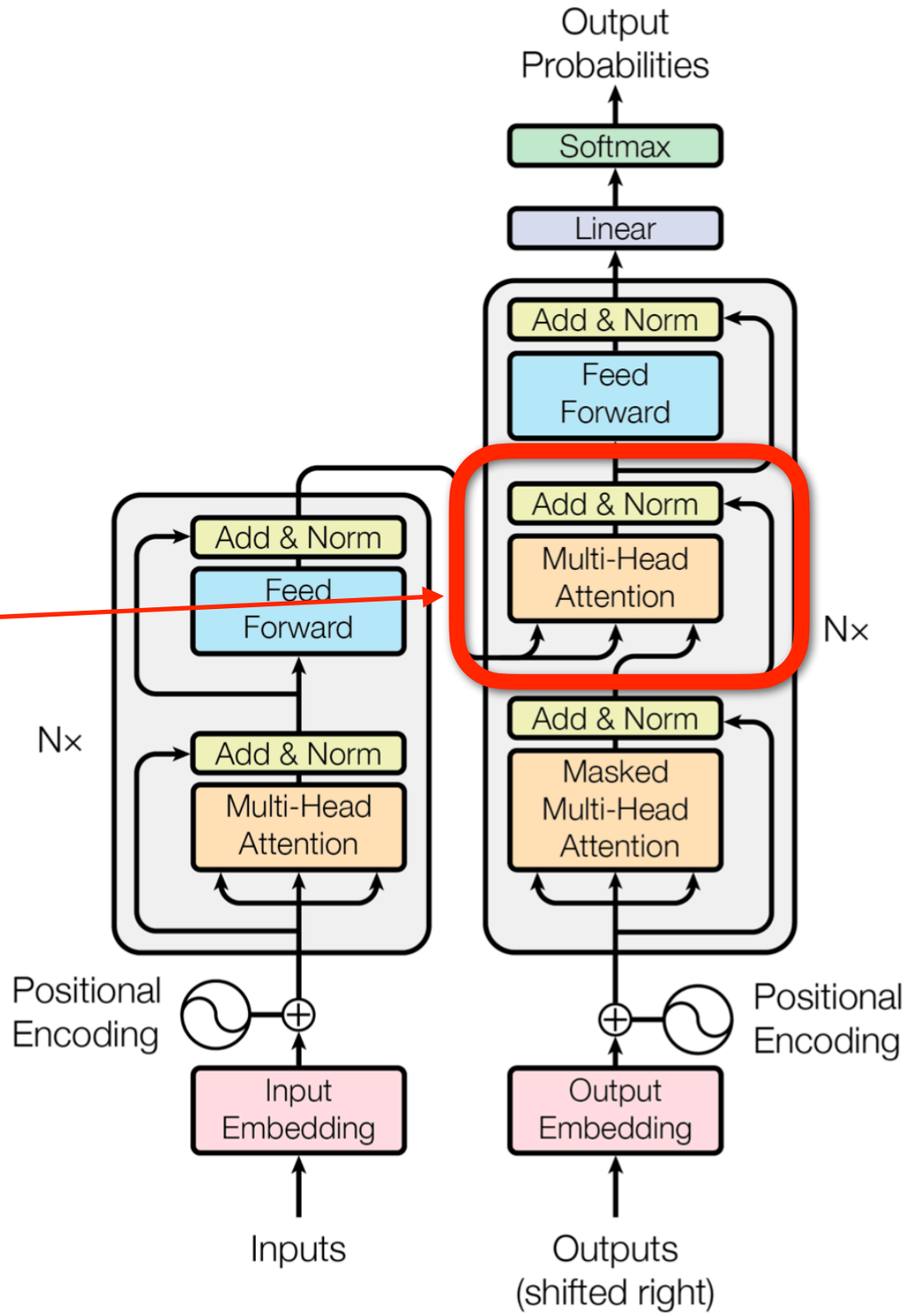
Traduction : « Je mange un kiwi »

Embedding : $[x_1, x_2, x_3, x_4]$

N boucle d'encodage : $[z_1, z_2, z_3, z_4]$

Décodage : Output = ∅ -> o_0

Décodage : combine $[z_1, z_2, z_3, z_4]$ et les o_i



Transformer : Exemple

Traduction : « Je mange un kiwi »

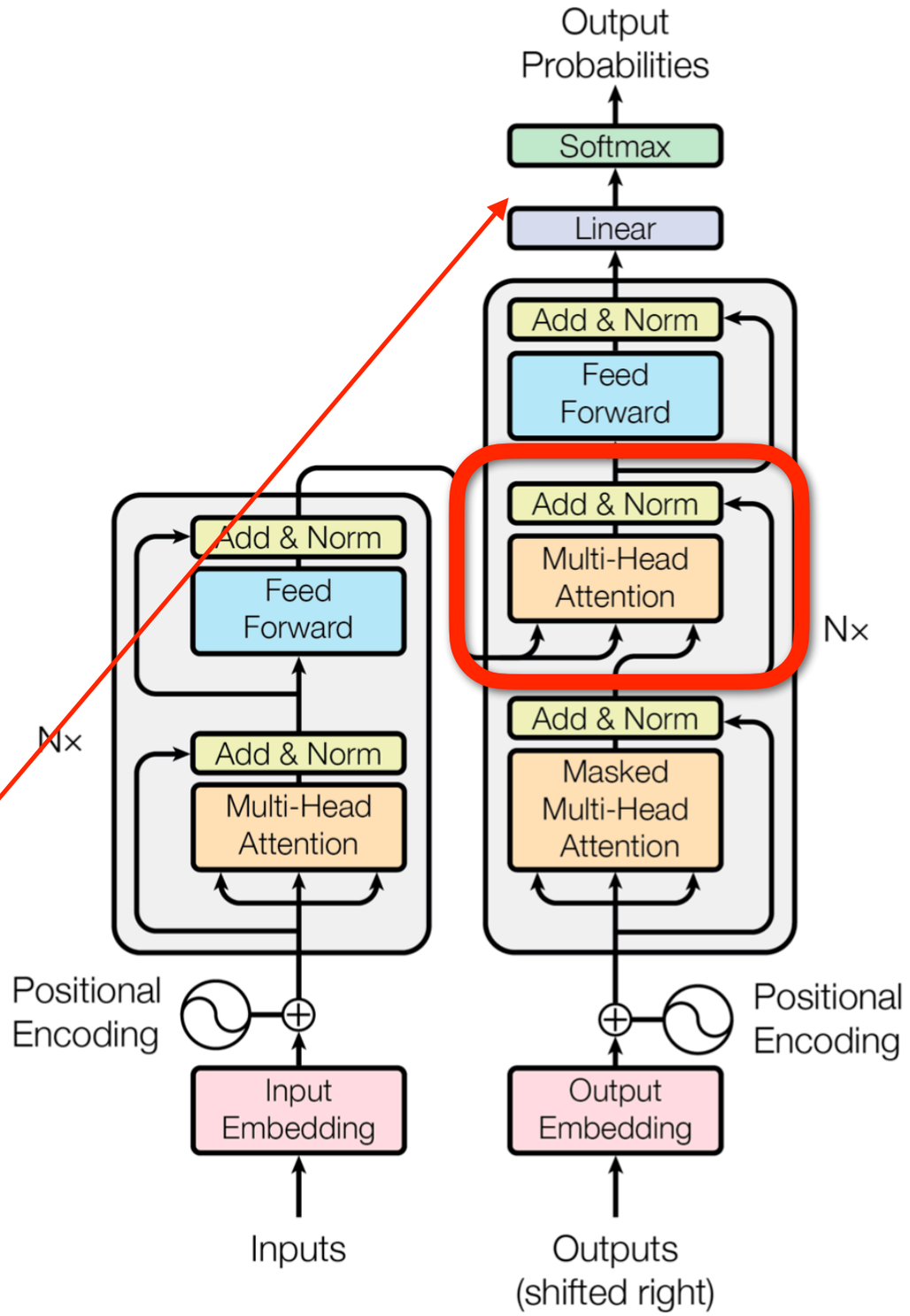
Embedding : $[x_1, x_2, x_3, x_4]$

N boucle d'encodage : $[z_1, z_2, z_3, z_4]$

Décodage : Output = ∅ -> o_0

Décodage : combine $[z_1, z_2, z_3, z_4]$ et les o_i

Output : o'_1 probabilité des différents mots -> Sélectionne le max : o_1 = I



Transformer : Exemple

Traduction : « Je mange un kiwi »

Embedding : $[x_1, x_2, x_3, x_4]$

N boucle d'encodage : $[z_1, z_2, z_3, z_4]$

Décodage : Output = ∅ -> o_0

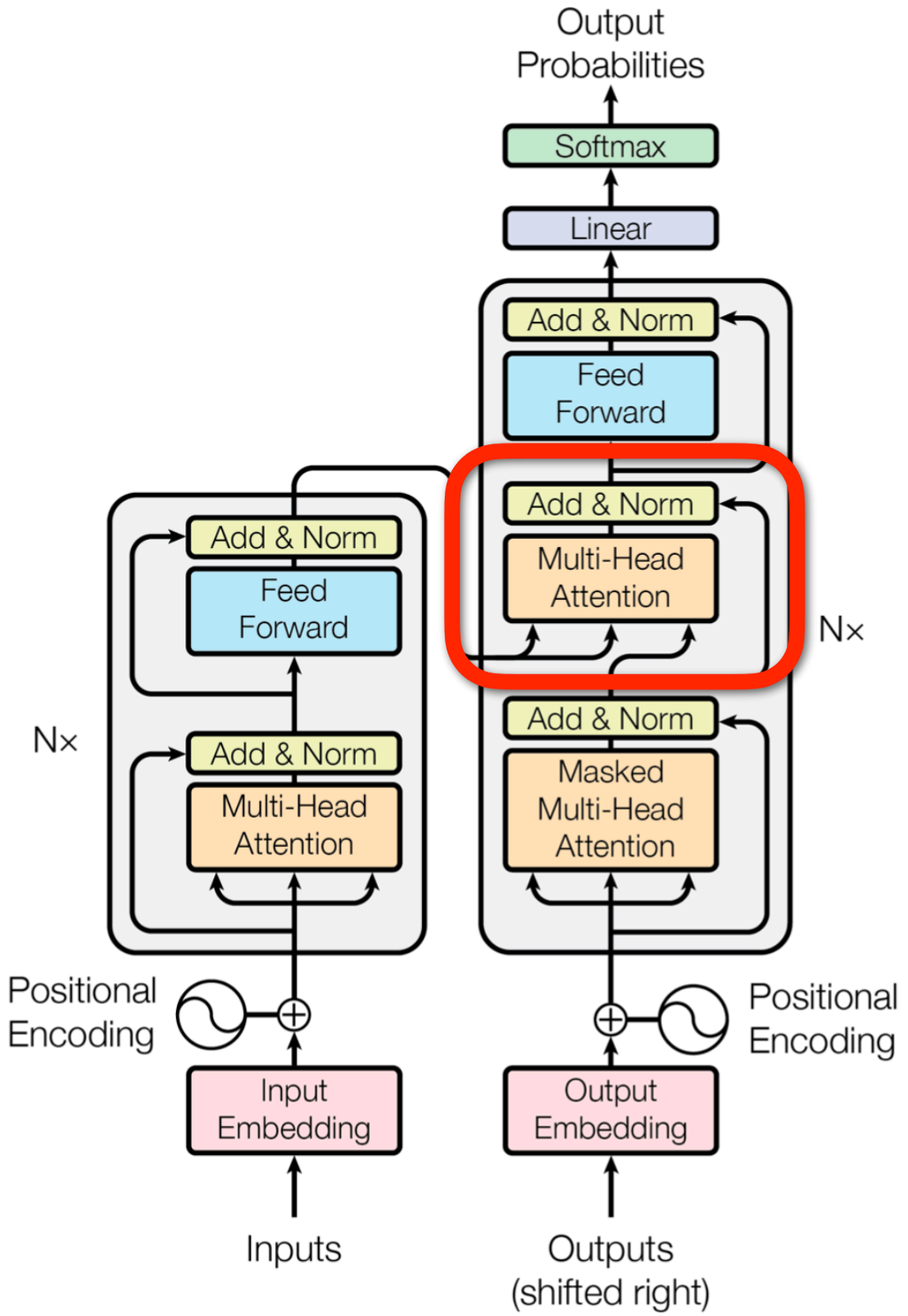
Décodage : combine $[z_1, z_2, z_3, z_4]$ et les o_i

Output : o'_1 probabilité des différents mots -> Sélectionne le max : o_1 = I

Continue le décodage : o_1-> o_2 = am $[o_1, o_2, o_3]$-> o_4 = a

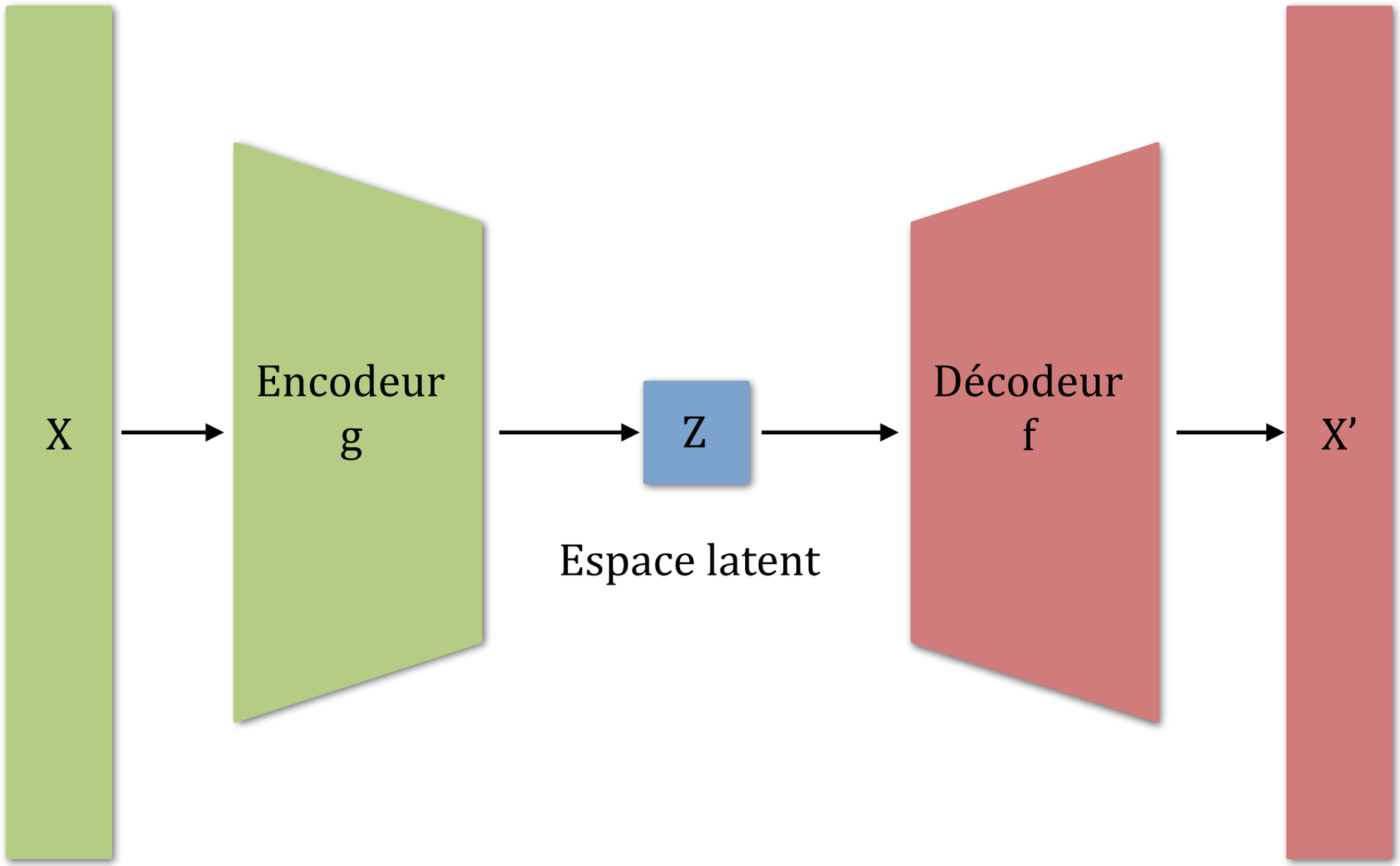
$[o_1, o_2]$-> o_3 = eating $[o_1, o_2, o_3, o_4]$-> o_5 = kiwi

$[o_1, o_2, o_3, o_4, o_5]$-> o_6 = <eos>



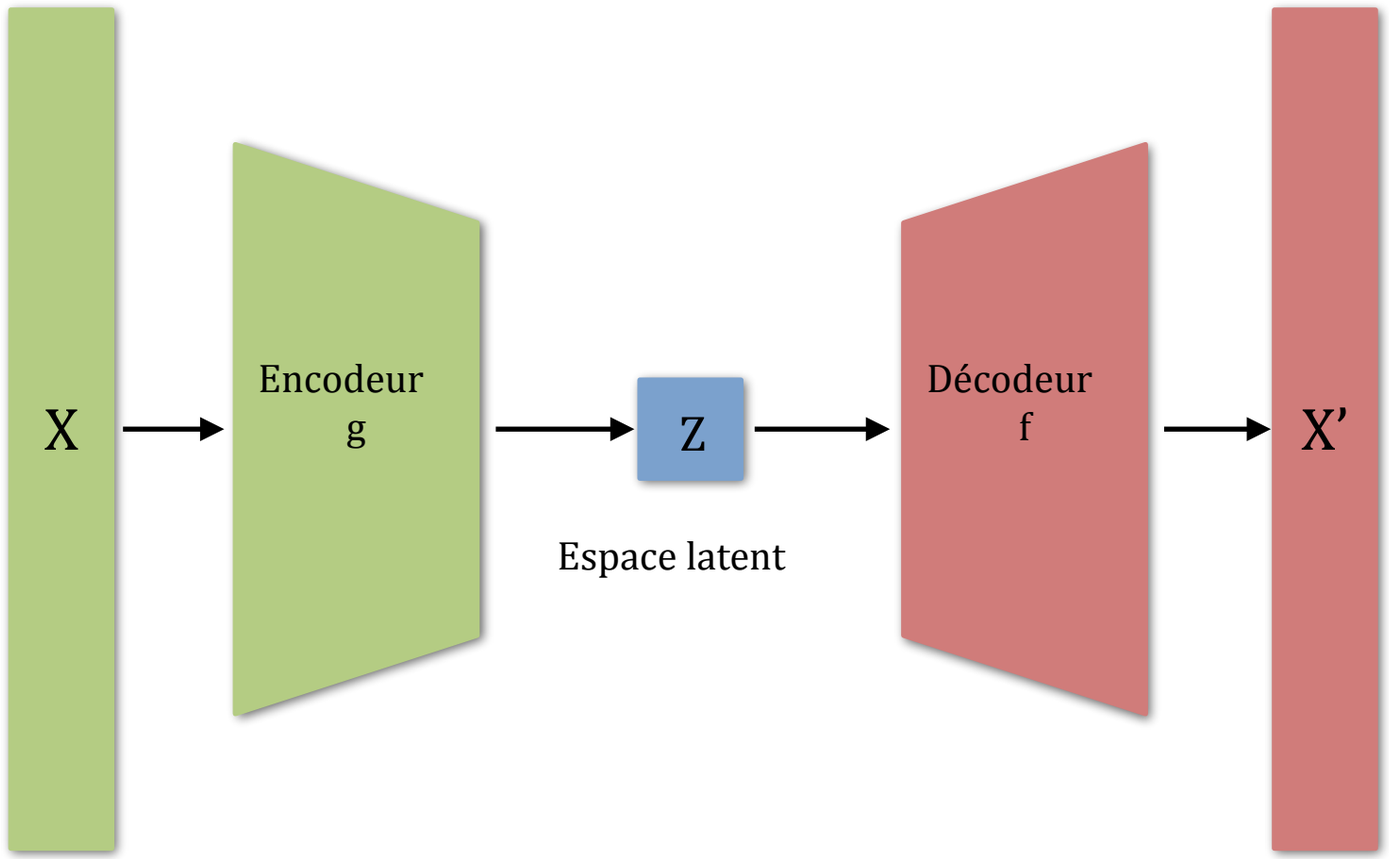
Auto Encoders

Application : Débruitage, réduction de dimension, détection d'anomalie



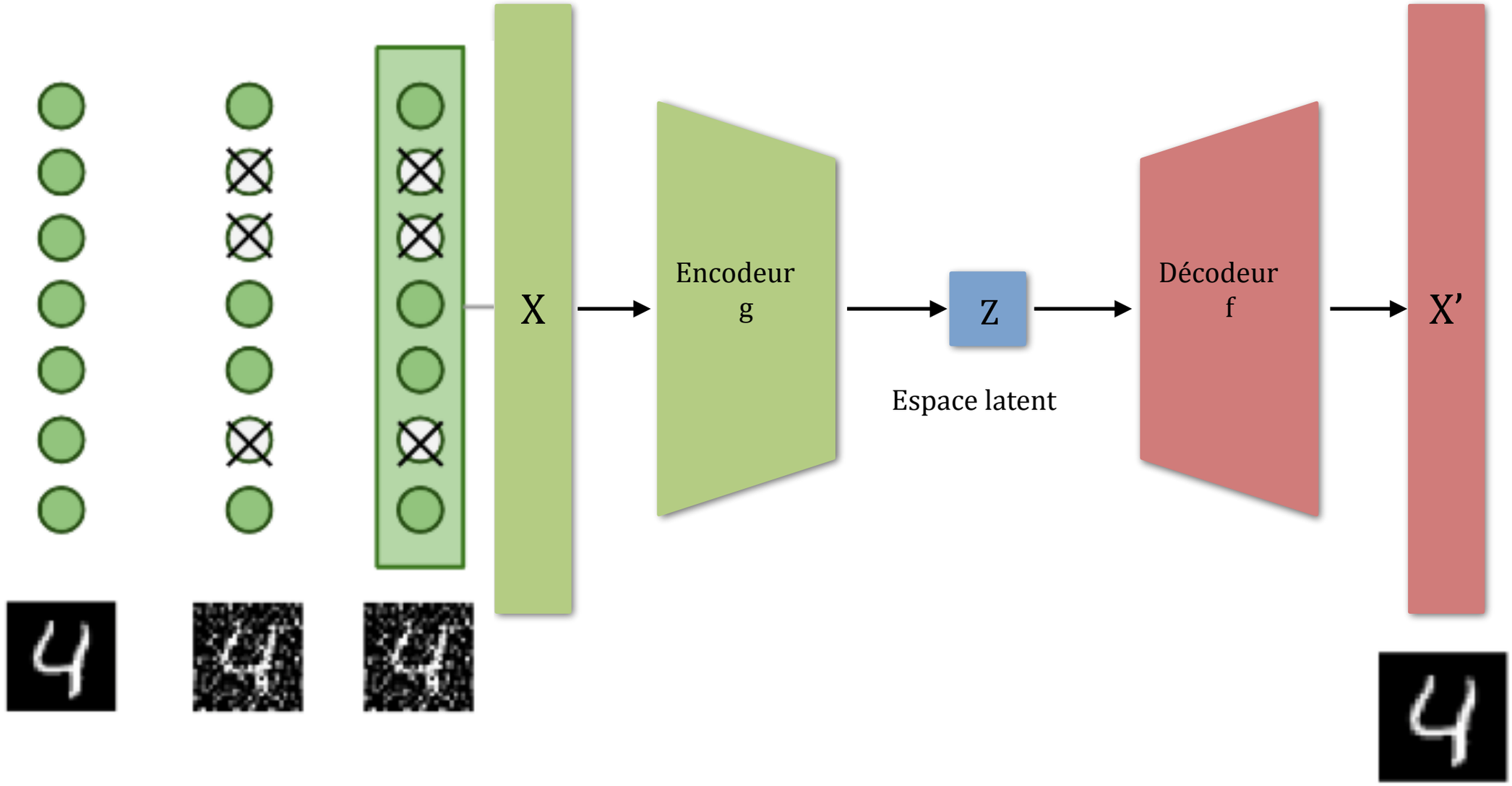
Auto Encoders

- Encodeur : réduit la taille de l'entrée en un petit espace latent
- Décodeur : reconstruit l'entrée à partir de l'espace latent
- Fonction de coût minimise la différence entre X' et X



Auto Encoders : debruitage

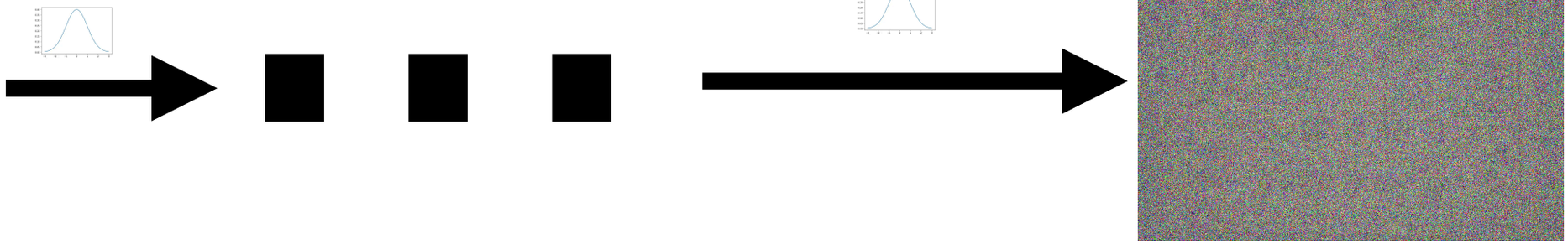
Même espace latent avec
des entré bruité



Diffusion models

Approche similaire au AE mais appliqué à la génération

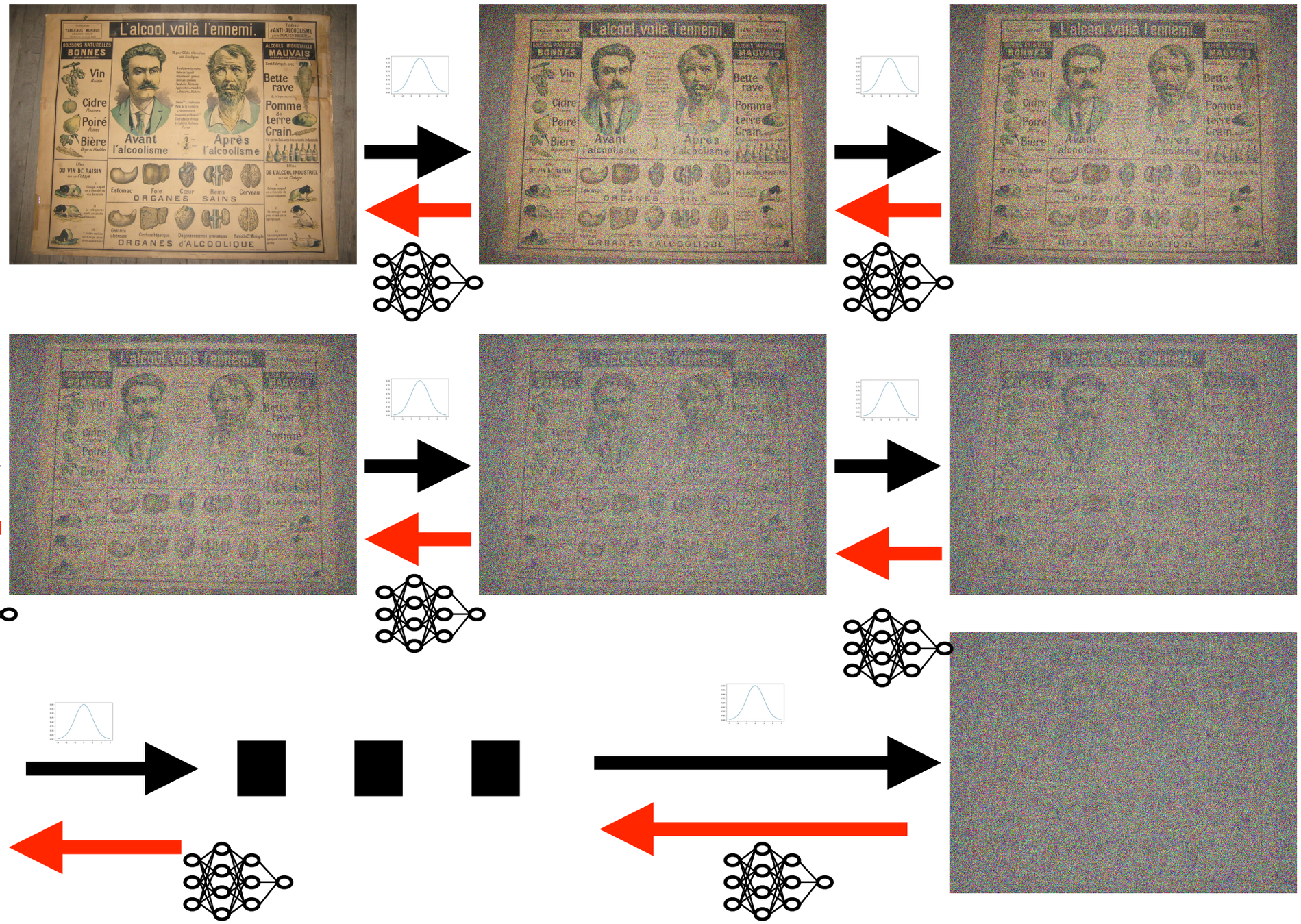
Ajout successif de petit bruit gaussien



Diffusion models

Approche similaire au AE mais appliqué à la génération

NN qui apprend à retirer le bruit gaussien



Diffusion models

Peut partir du bruit pour reconstruire des images

Possibilité d'ajouter des condition à la reconstruction