Deep learning : introduction aux réseaux de neurones profonds

Pascal Maryniak

Sommaire de l’exposé

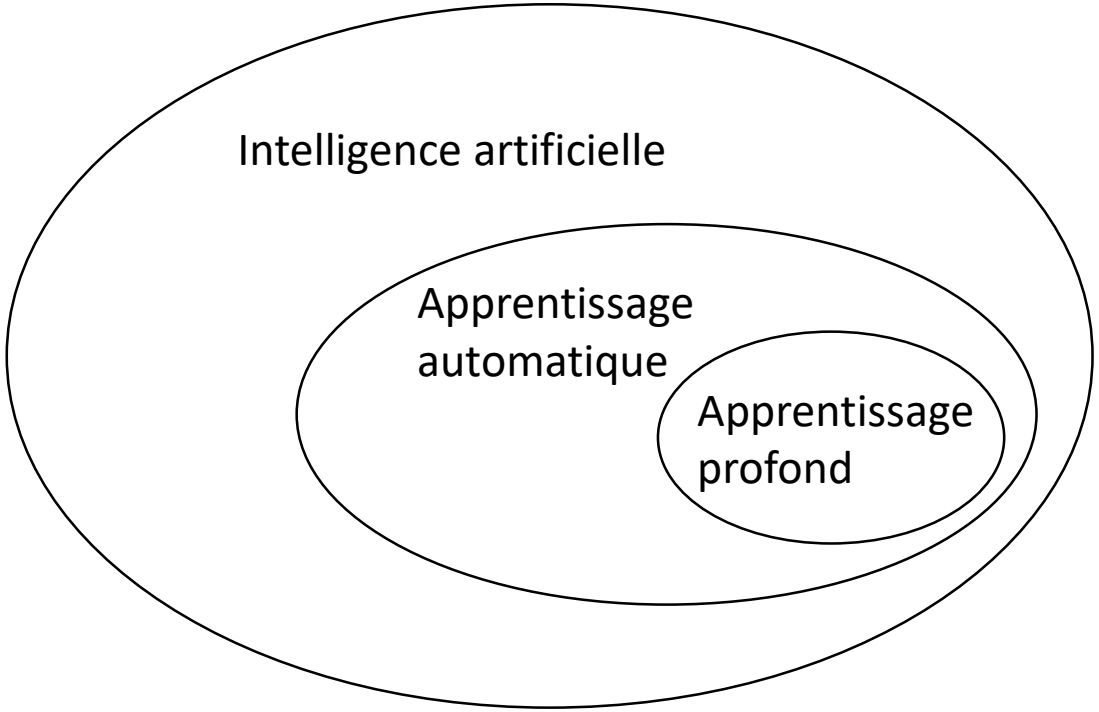
Qu’est ce que l’apprentissage profond ?

Structure et fonctionnement des réseaux de neurones

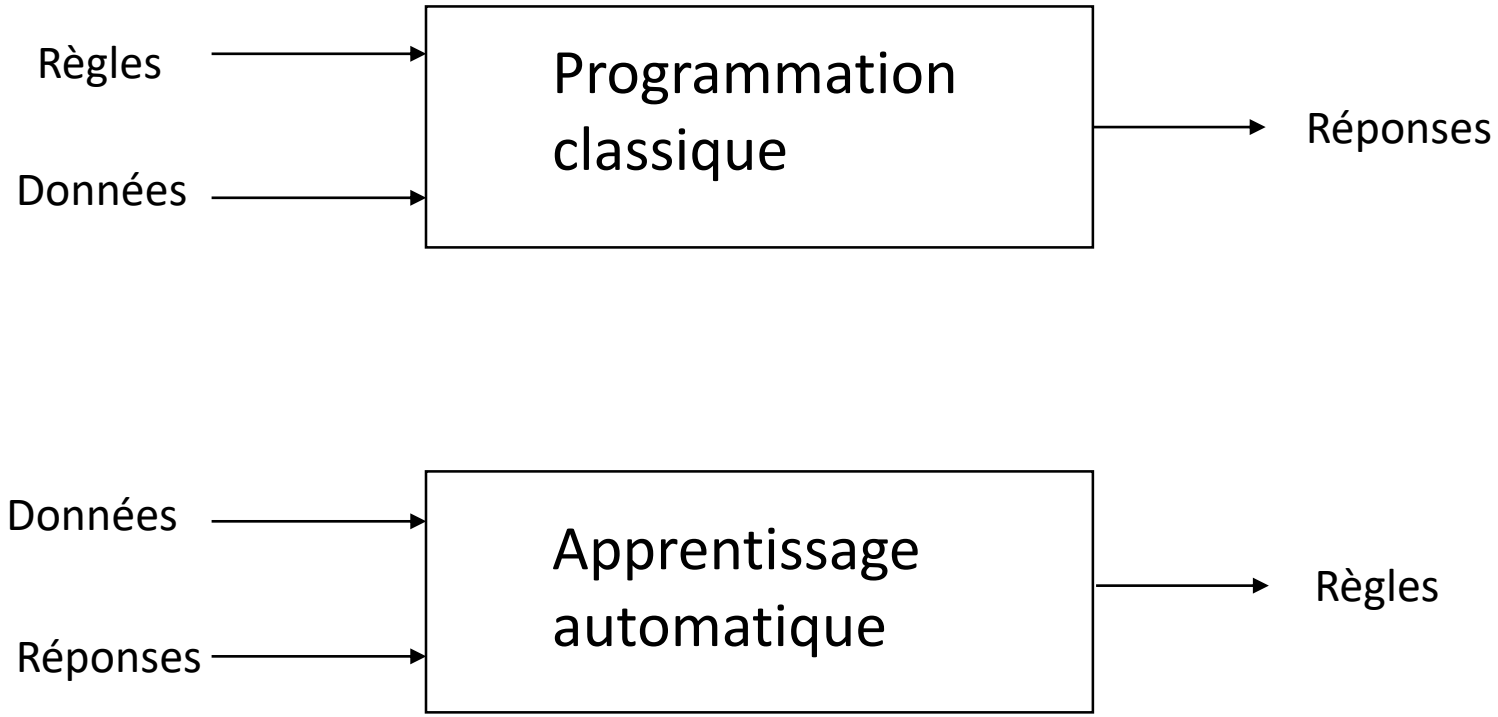
L’entraînement des réseaux de neurones profonds

Introduction aux modèles de langage

Intelligence artificielle, apprentissage automatique et apprentissage profond



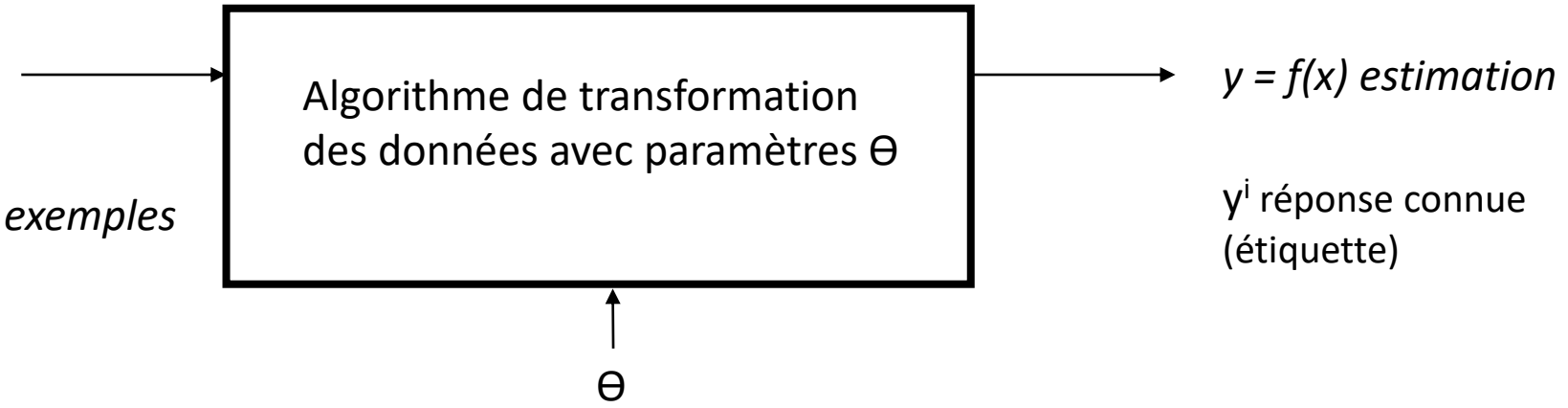
Apprentissage automatique versus programmation classique



Algorithmes d'apprentissage

x

Quel est cette race de chien ? Y = « berger australien »



Apprentissage : optimise les paramètres Θ de l'algorithme pour faire peu d'erreur sur les exemples

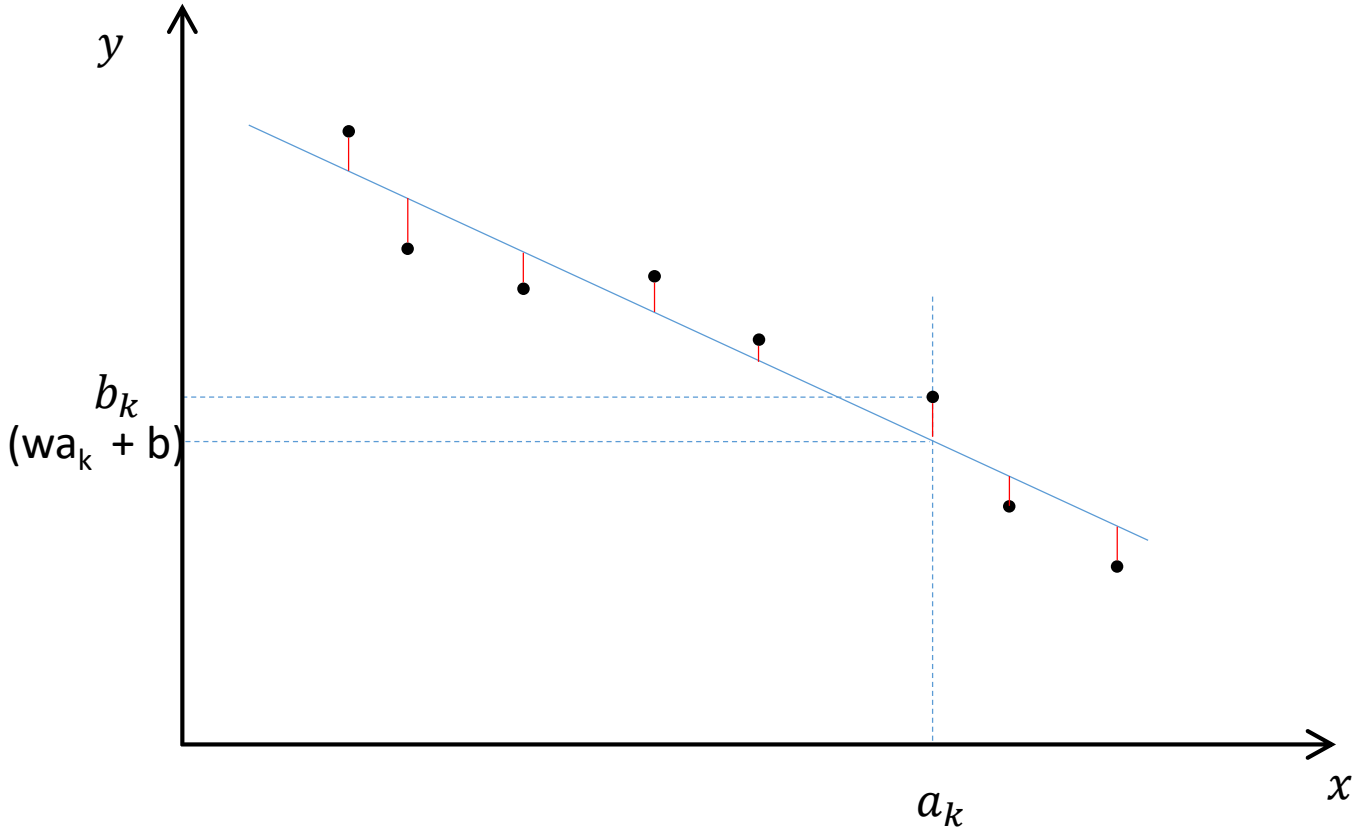
X^i : $f(X^i) \sim y^i$

Généralisation : pour des x inconnus, si f(x) est également très proche de la valeur attendue y

Exemple de la régression linéaire

- x est un scalaire représentant par exemple l'altitude et la réponse y la température
- On a un échantillon de K mesures : b_k température mesurée à l'altitude a_k
- Pour trouver la température y à une altitude quelconque x, on va faire passer une droite au plus près de l'ensemble des points de coordonnées (a_k,b_k)
- Pour ce faire, si l'équation de la droite recherché est y = w*x + b, le problème revient à trouver les paramètres w et b qui minimisent l'erreur quadratique moyenne L « loss » commise sur l'échantillon connu a_k :

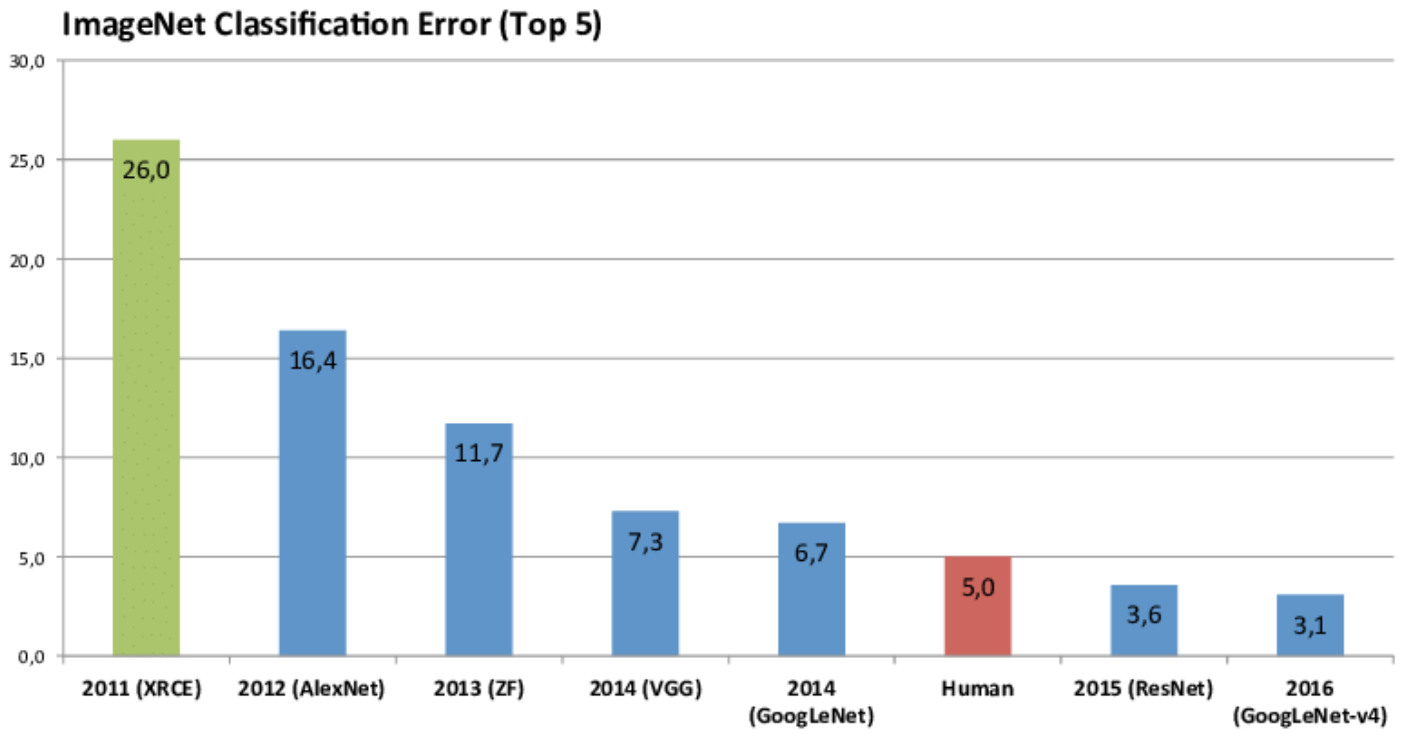
$$L(w, b) = \sum [b_k - (wa_k + b)]^2$$



L’explosion des réseaux de neurones

- Années 1950 : apparition des idées fondamentales
- Premier succès issus des Laboratoires Bell en 1989 : Yann Le Cun développe un réseau de neurones à convolution entraîné par descente de gradient et rétropropagation
- 1994 : mise sur le marché d’un système de lecture automatique des chèques et des codes postaux (LeNet, comme Le Cun)
- 2010 : lancement de la compétition annuelle ImageNet de classification d’images à grande échelle (ILSVRC : ImageNet large Scale Visual Recognition Challenge)
  - 1,4 millions d’images en couleur de haute résolution, en 1000 classes ou catégories différente (dont par exemple 200 différentes races de chien)
- 2011 : premiers résultats avec des approches classiques de vision par jusqu'à 74 % d’exactitude
- 2012 : les réseaux de neurones pulvérisent les approches classiques, une équipe dirigée par Alex Krizhevsky et conseillée par Geoffrey Hinton parvient à atteindre avec un réseau de neurones une précision de 83,6 %
- 2015 : les réseaux de neurones à convolution surpassent la capacité humaine, le gagnant atteignant une exactitude de 96,4 % (être humain : 95 %)

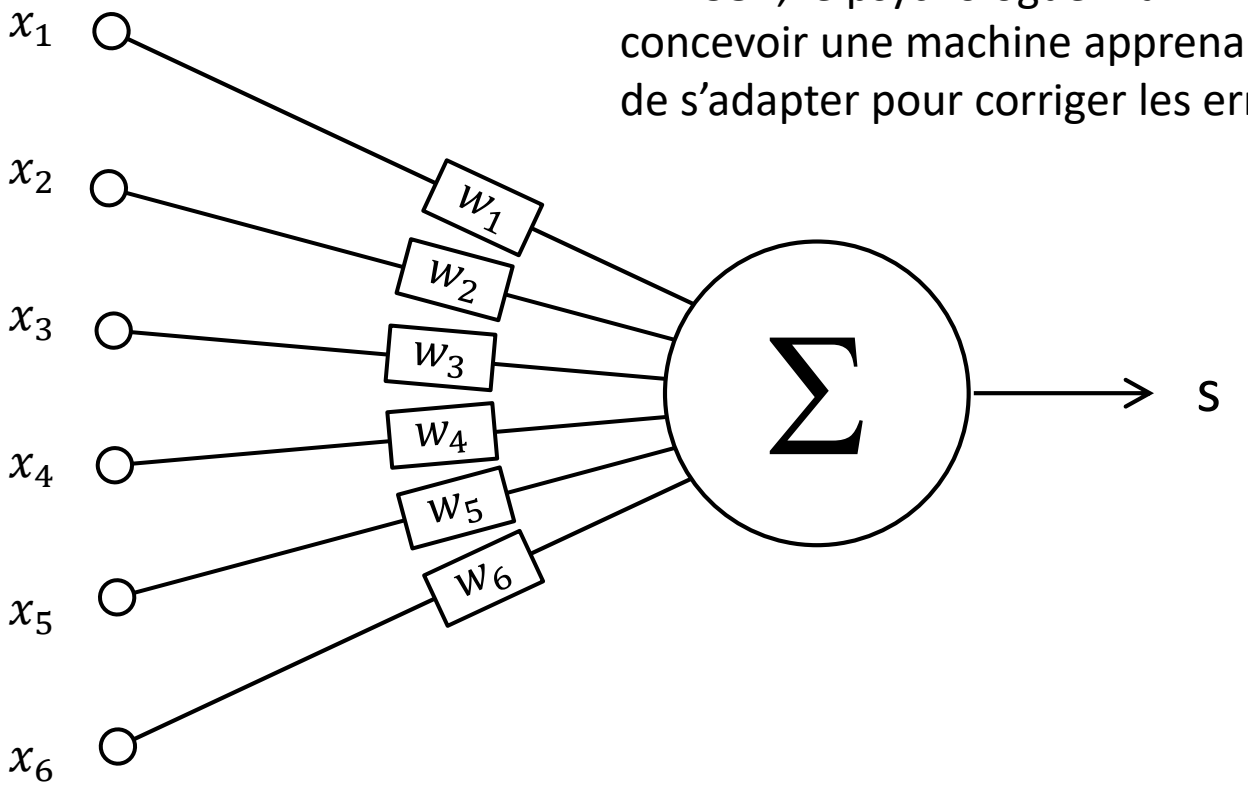
Historique des gagnants de la compétition ILSVRC



Le neurone informatique

En 1943, deux cybernéticiens et neuroscientifiques américains, Warren McCulloch et Walter Pitts proposent un modèle très simplifié inspiré du neurone biologique

En 1957, le psychologue Frank Rosenblatt reprend cette idée pour concevoir une machine apprenante, appelée le Perceptron, capable de s'adapter pour corriger les erreurs.



1 si $\sum_k w_i \cdot x_i \geq (-b)$

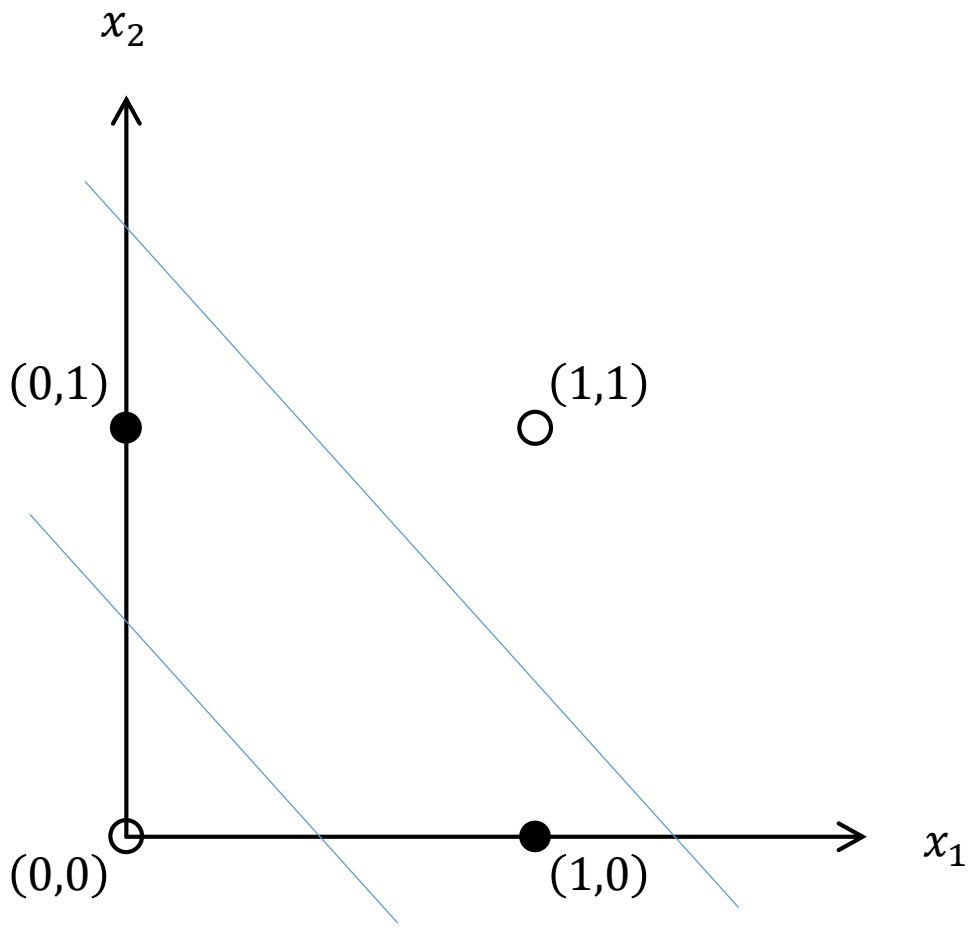
0 (ou -1) sinon

Les limitations du perceptron

Le perceptron repose sur un modèle linéaire qui s'avère incapable de reconnaître certains types de formes.

En effet, dans le cas d'un neurone binaire, la sortie vaut 1 si $\sum_k w_i \cdot x_i b \geq 0$ et 0 sinon si bien que le perceptron sépare l'espace des entrées en deux moitiés dont la frontière est l'hyperplan d'équation $\sum_k w_i \cdot x_i + b = 0$

Ainsi, la fonction logique OU exclusif des deux entrées logiques x_1 et x_2 n'est pas linéairement séparable. Elle prend la valeur 1 pour (0,1) et (1,0) (points noirs) et la valeur 0 pour (0,0) et (1,1) (points blancs). Il est impossible de séparer par une droite les points blancs des points noirs.



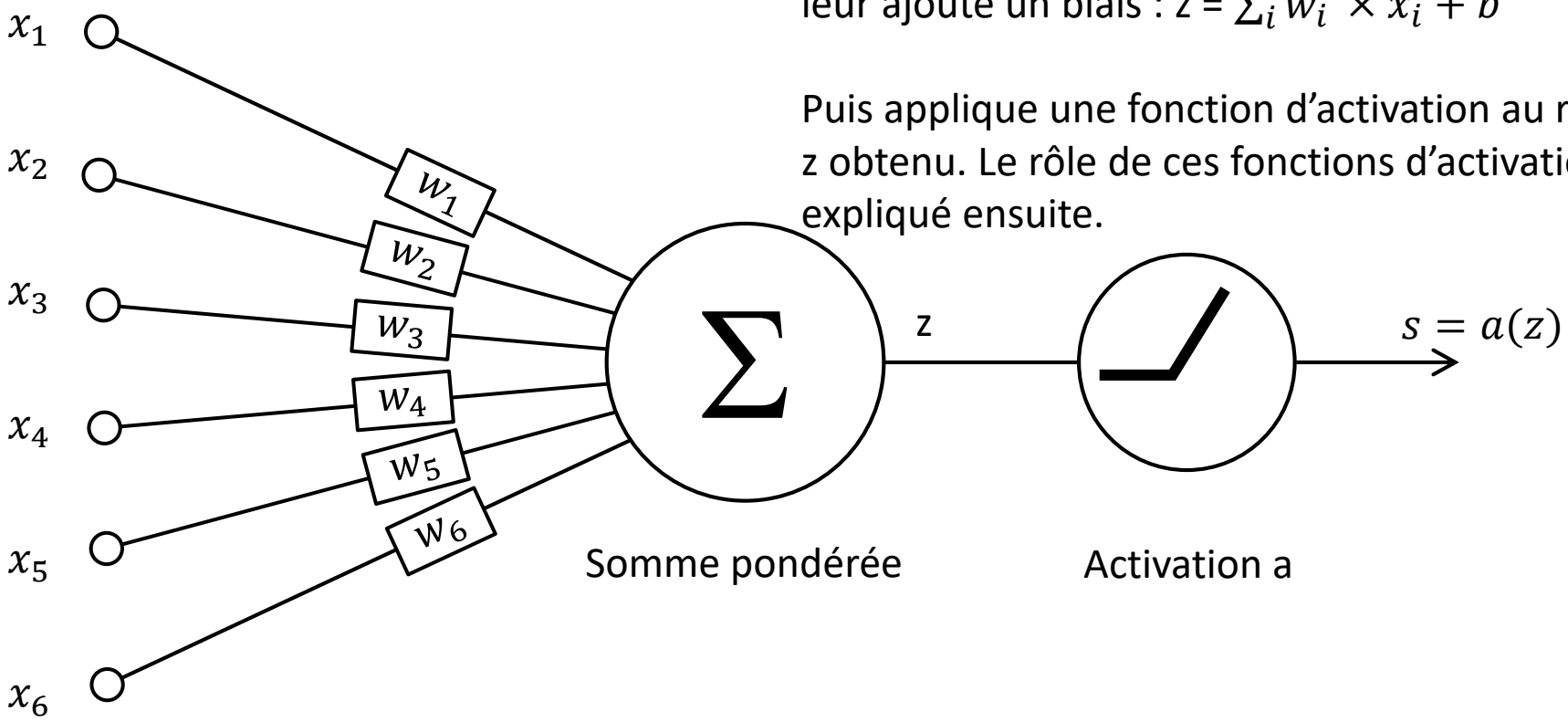
Source : «Quand la machine apprend » - Yann Le Cun

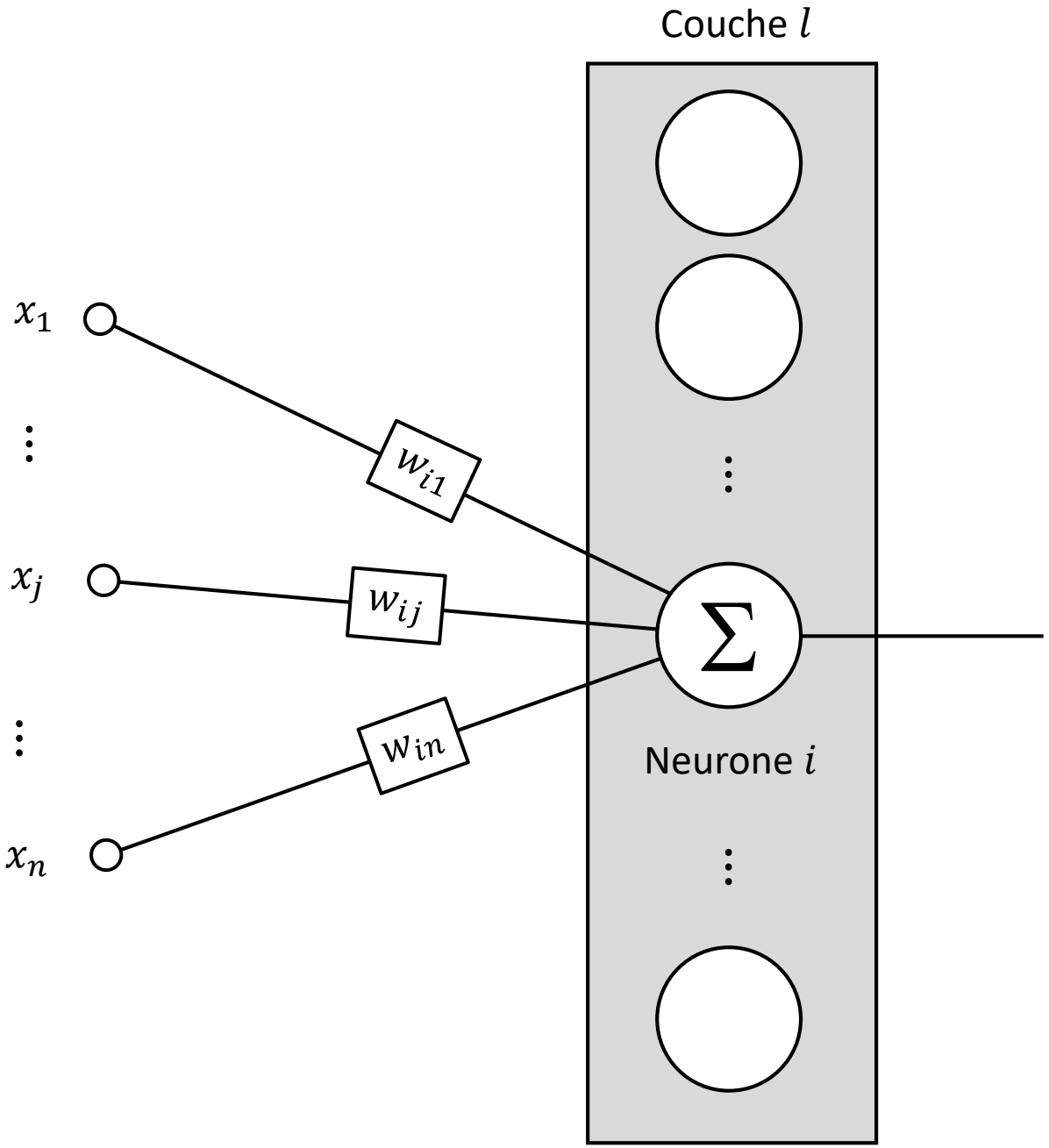
Le neurone informatique moderne

Le modèle de neurone utilisé dans les réseaux modernes reste très proche du modèle de celui de McCulloch et r Pitts

Il effectue la somme pondérée de ses entrées et leur ajoute un biais : z = $\sum_i w_i \times x_i + b$

Puis applique une fonction d'activation au résultat z obtenu. Le rôle de ces fonctions d'activation sera expliqué ensuite.

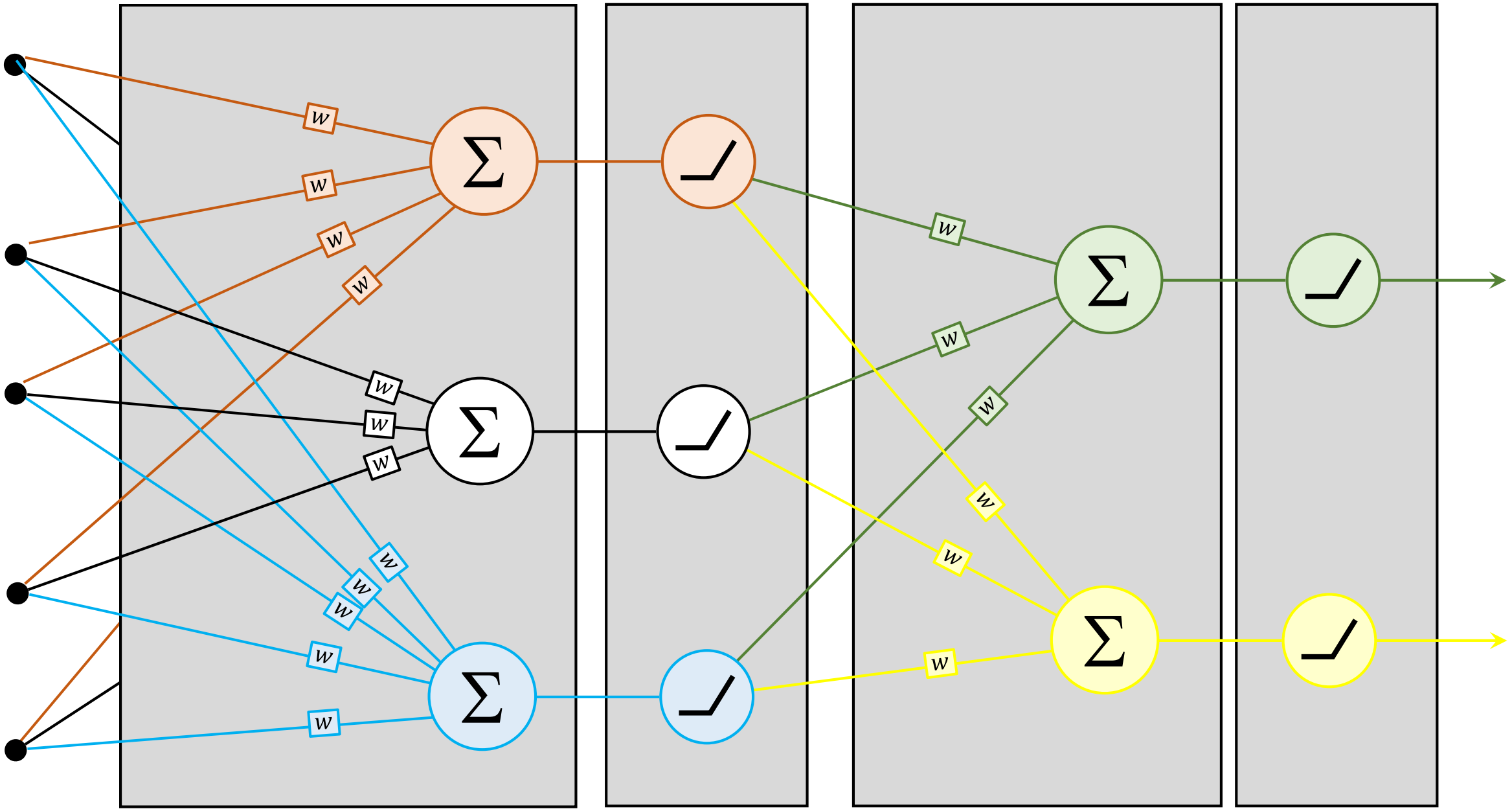




$$z_i = \sum_j w_{ij}{}^l \, z_j + b_i$$

Sous forme vectorielle,
S = W x Z + b

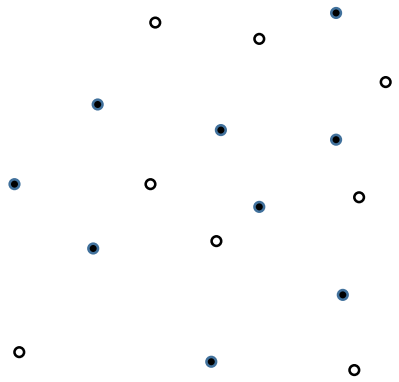
Du neurone au réseau de neurones profonds



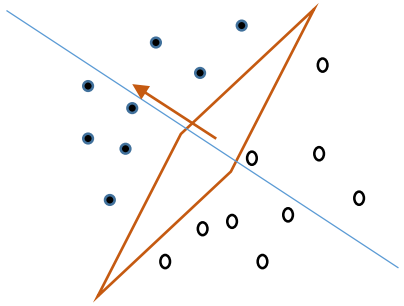
La stratégie de linéarisation (source : cours de Stéphane Mallat du 23 janvier 2019 au Collège de France)

Trouver un changement de variables qui permette de séparer les données par un plan

$x = (x_1,\ldots,x_N)$ ⟶ $\Psi(x) = (v_1,\ldots,v_N)$



Ψ ⟶



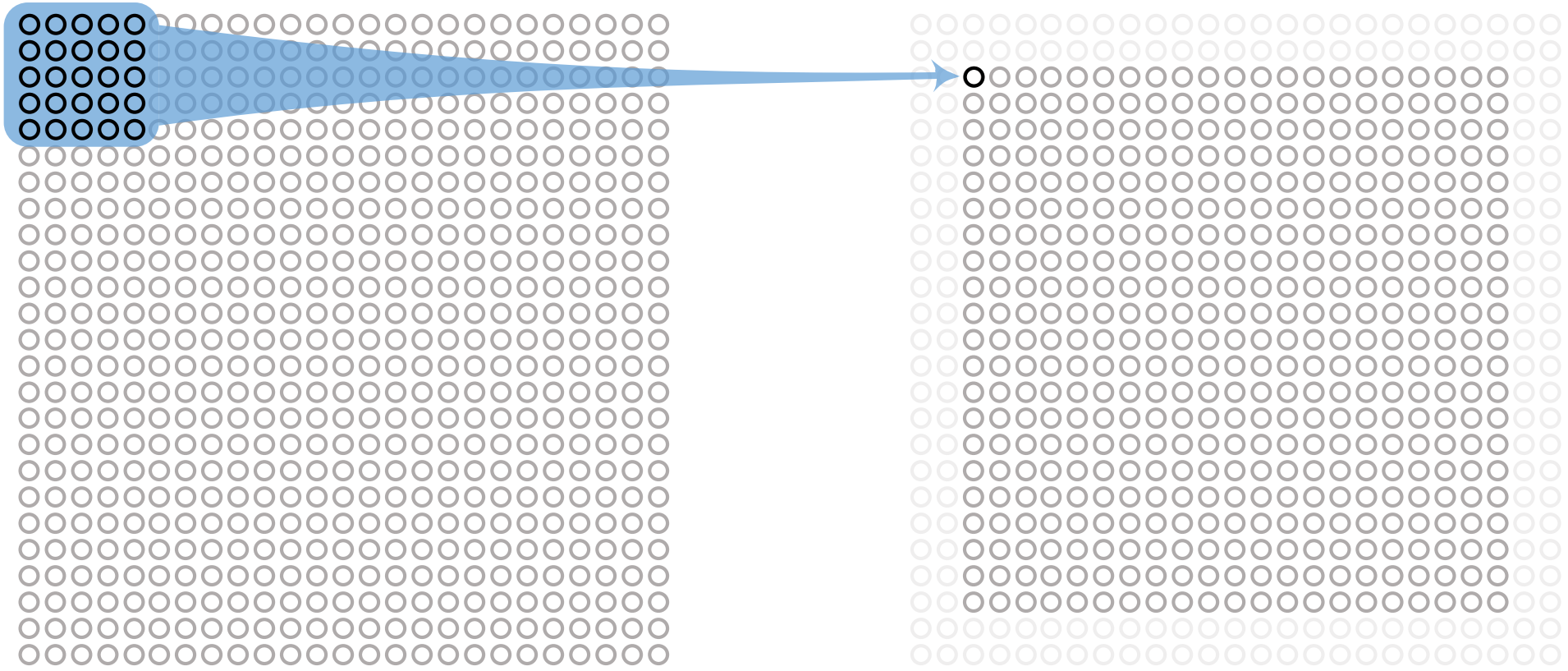
Classification : Trouver le bon plan qui va séparer les variables, donc la direction du vecteur w qui est orthogonal au plan de séparation des données.

Apprendre la fonction Ψ au moyen des réseaux de neurones.

L’opération de convolution

Image en entrée

Carte de caractéristiques en sortie



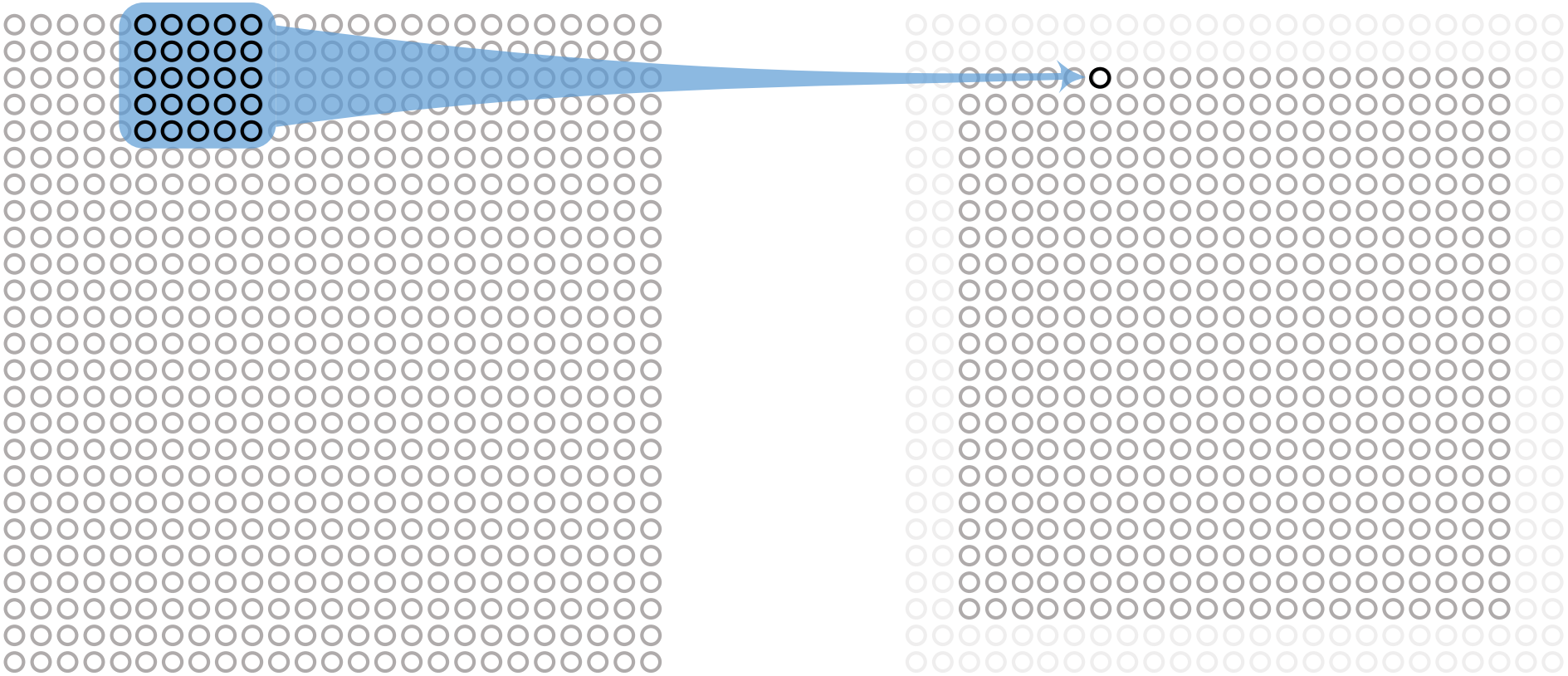
25 × 25

21 × 21

L’opération de convolution

neurones d’entrée

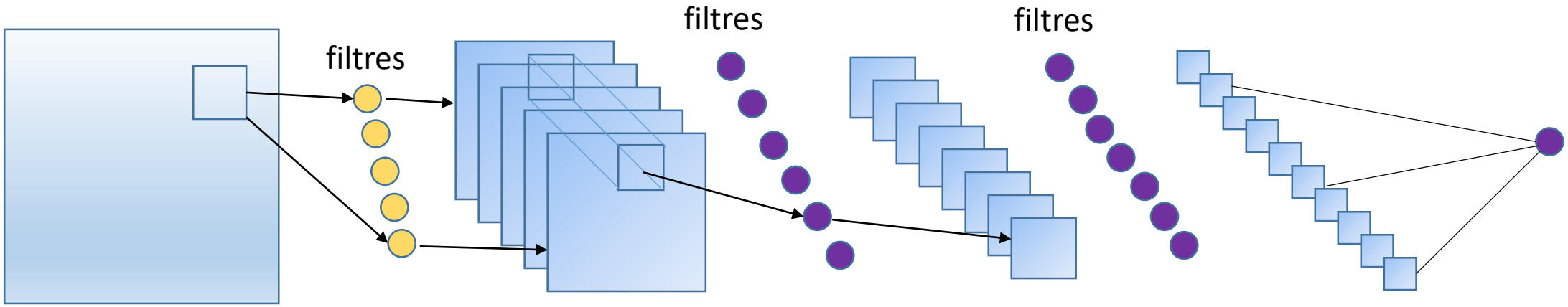
première couche cachée



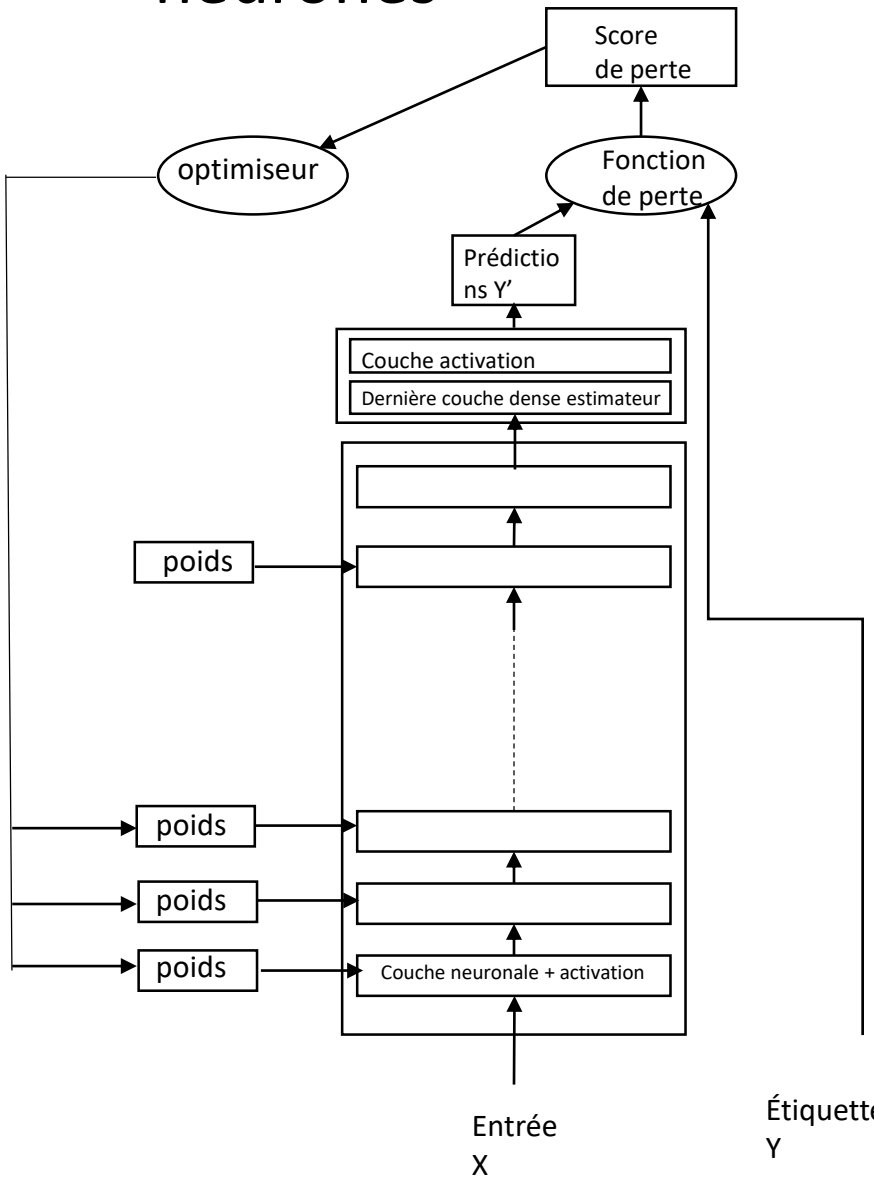
25 × 25

21 × 21

L’architecture d’un réseau de neurones convolutif

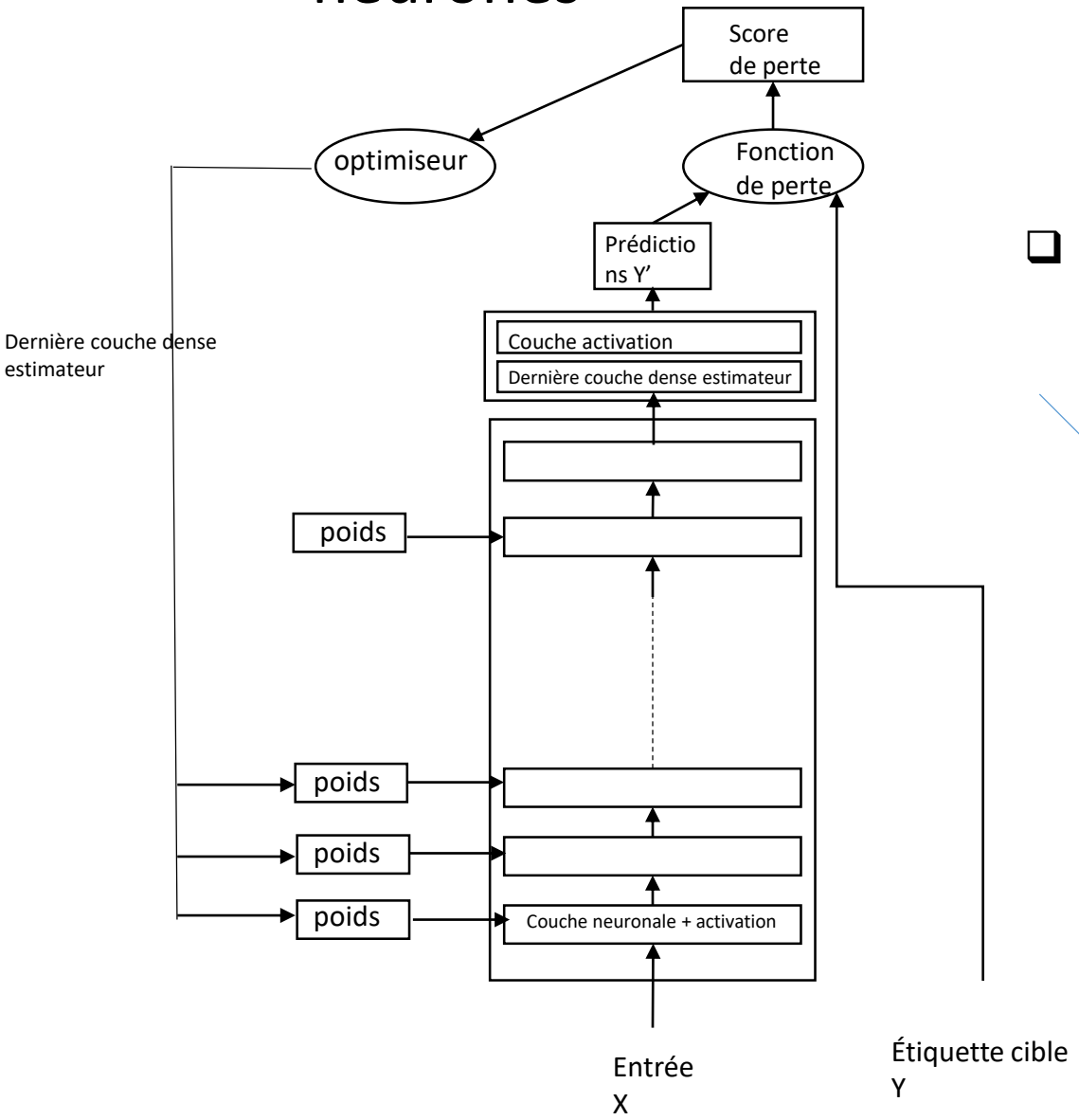


Architecture générale et fonctionnement d'un réseau de neurones



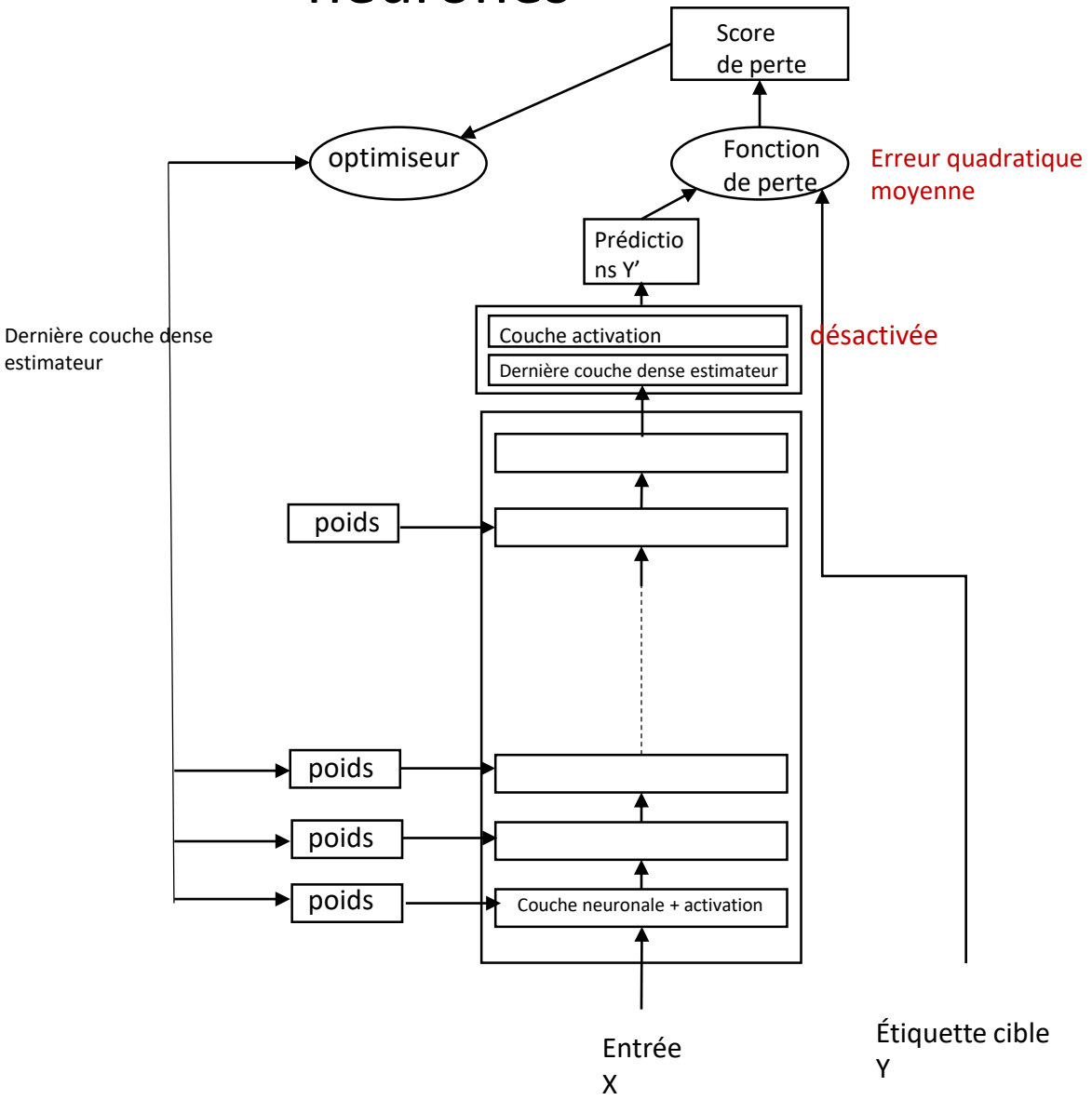
- L'architecture d'un réseau de neurones comporte un empilement de nombreuses couches adapté au problème à résoudre
- Le réseau alterne des couches linéaires (où chaque sortie est une somme pondérée des entrées) et des couches non-linéaires possédant le même nombre d'entrées et de sortie et qui consistent le plus souvent en la fonction ReLU (Rectified Linear Unit),prenant pour une entrée x la valeur maximale de x et de 0. Ces couches non-linéaires placées après chaque couche neuronale sont dites couches d'activation de la couche neuronale qui la précède.
- Le réseau se termine par une couche entièrement connectée (c'est-à-dire que toutes ses entrées sont connectées à tous ses neurones)
- La fonction de perte mesure l'erreur entre la valeur calculée par le réseau Y' pour l'entrée X et la valeur attendue Y (étiquette)
- L'optimiseur ajuste les poids des couches pour que le score de perte soit le plus petit possible.

Architecture générale et fonctionnement d'un réseau de neurones



- La sortie du réseau est adaptée en fonction de la tâche cible à réaliser, selon qu'il s'agisse :
  - D'une régression scalaire, où la cible est une valeur ou un vecteur de valeurs continues
  - D'une classification binaire où chaque exemple d'entrée doit être classé entre deux catégories exclusives (par exemple chien ou chat)
  - D'une classification en plusieurs classes consistant à classer chaque entrée dans une catégorie unique parmi un ensemble donné de plus de deux catégories

Architecture générale et fonctionnement d'un réseau de neurones



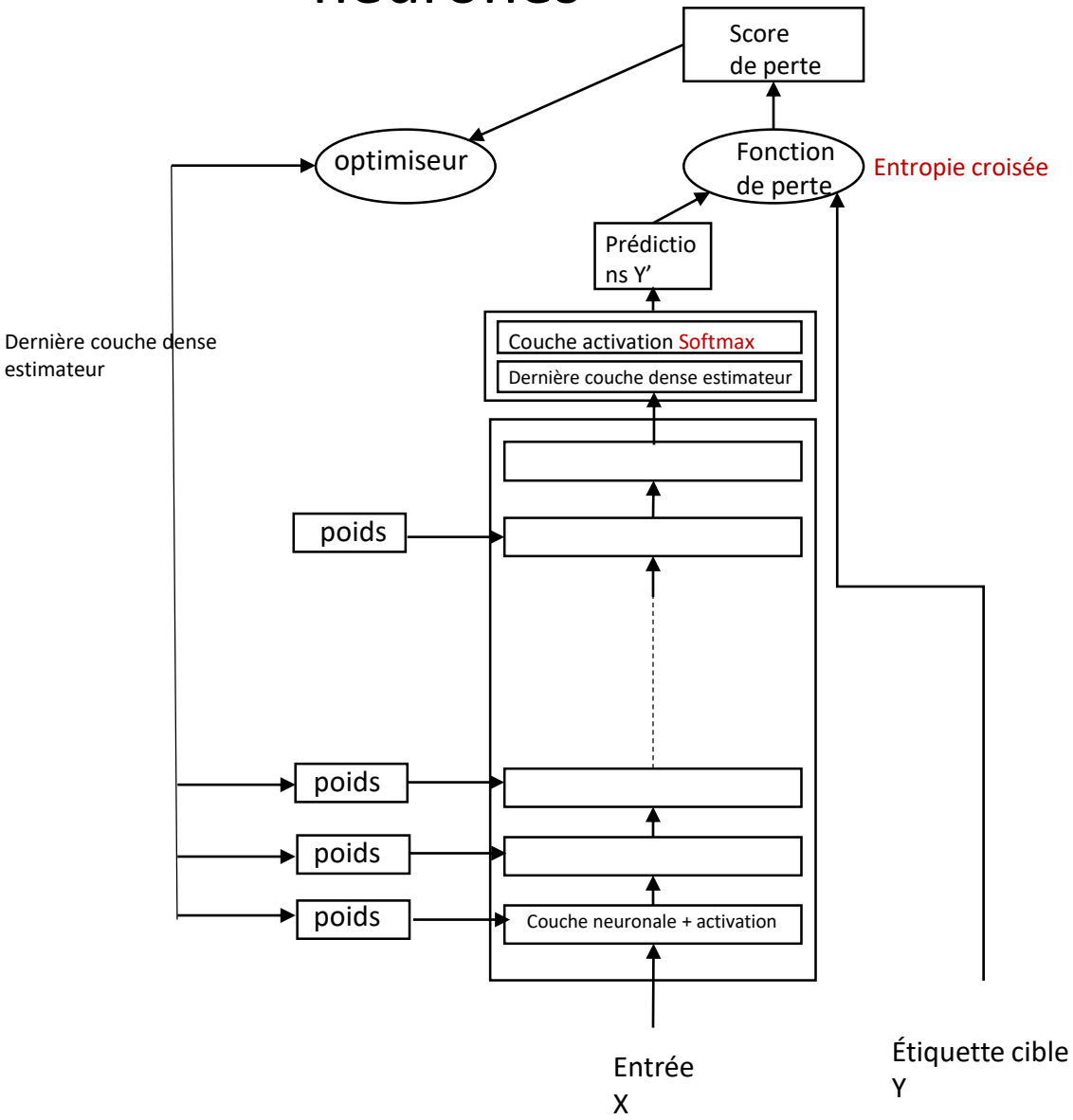
Régression scalaire ou vectorielle :

- Le réseau se termine par une couche dense.
- Aucune fonction d'activation associée., les différentes sorties fournissent directement les prédictions de valeurs attendues.
- La fonction de perte utilisée est l'erreur quadratique moyenne qui correspond à la moyenne des carrés des différences entre les prédictions et les cibles

$$\text{Score de perte} = \frac{1}{N}\sum_i [Y'^i - Y^i]^2$$

où Y'^i est la sortie du réseau pour l'entrée X^i et Y^i l'étiquette associée à cette entrée.

Architecture générale et fonctionnement d'un réseau de neurones

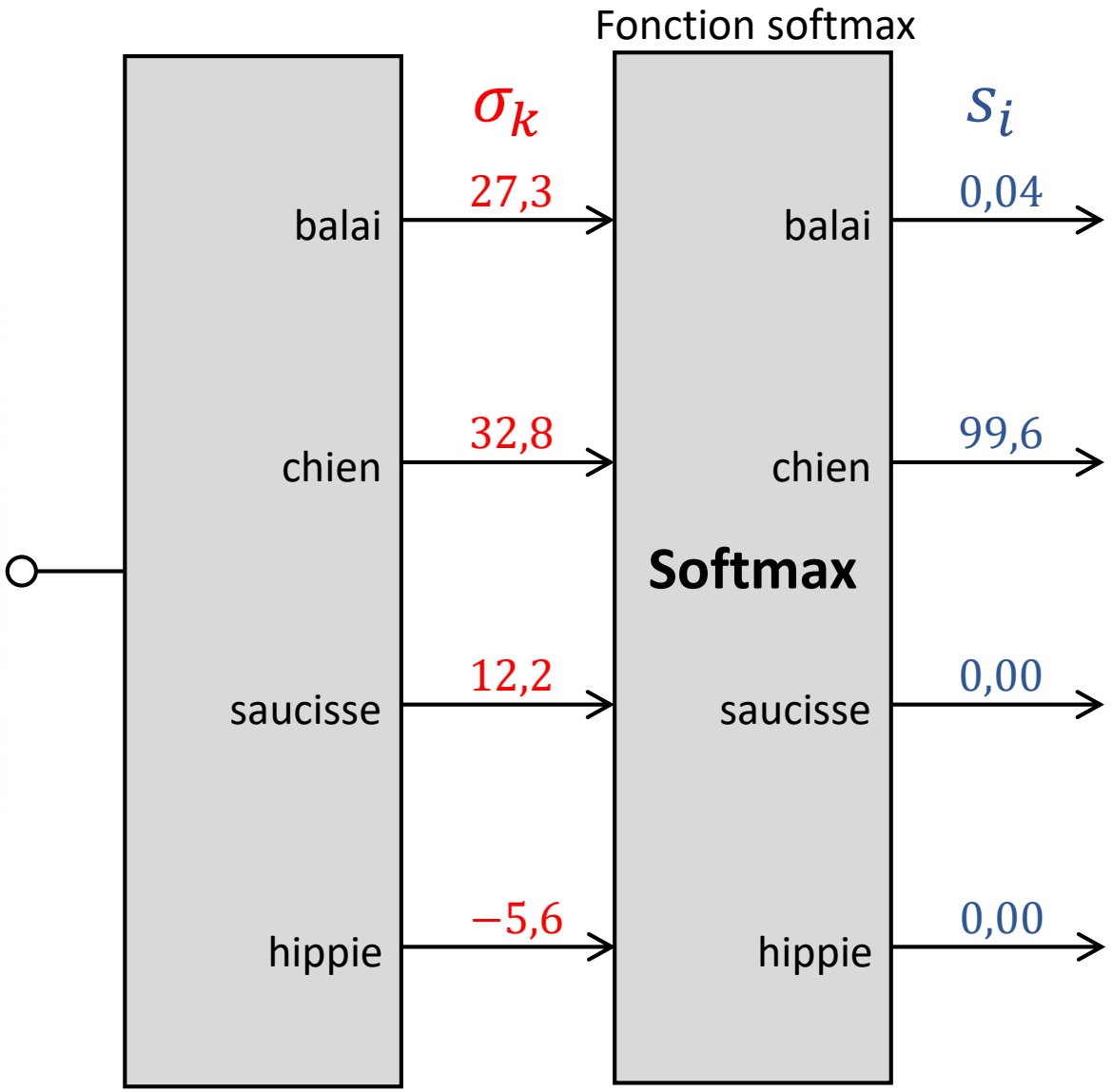


Classification en plusieurs classes

- Le réseau se termine par une couche dense avec un nombre de sorties égal au nombre de classes.
- Pour qu'une sortie représente la probabilité que l'entrée soit dans la classe qui lui correspond, la fonction d'activation sera la fonction Softmax qui produit une distribution de probabilités sur l'ensemble des classes comprises entre 0 et 1 et dont la somme vaut 1.
- La fonction de perte utilisée est l'entropie croisée entre les pseudo-distributions de probabilité Y' et Y

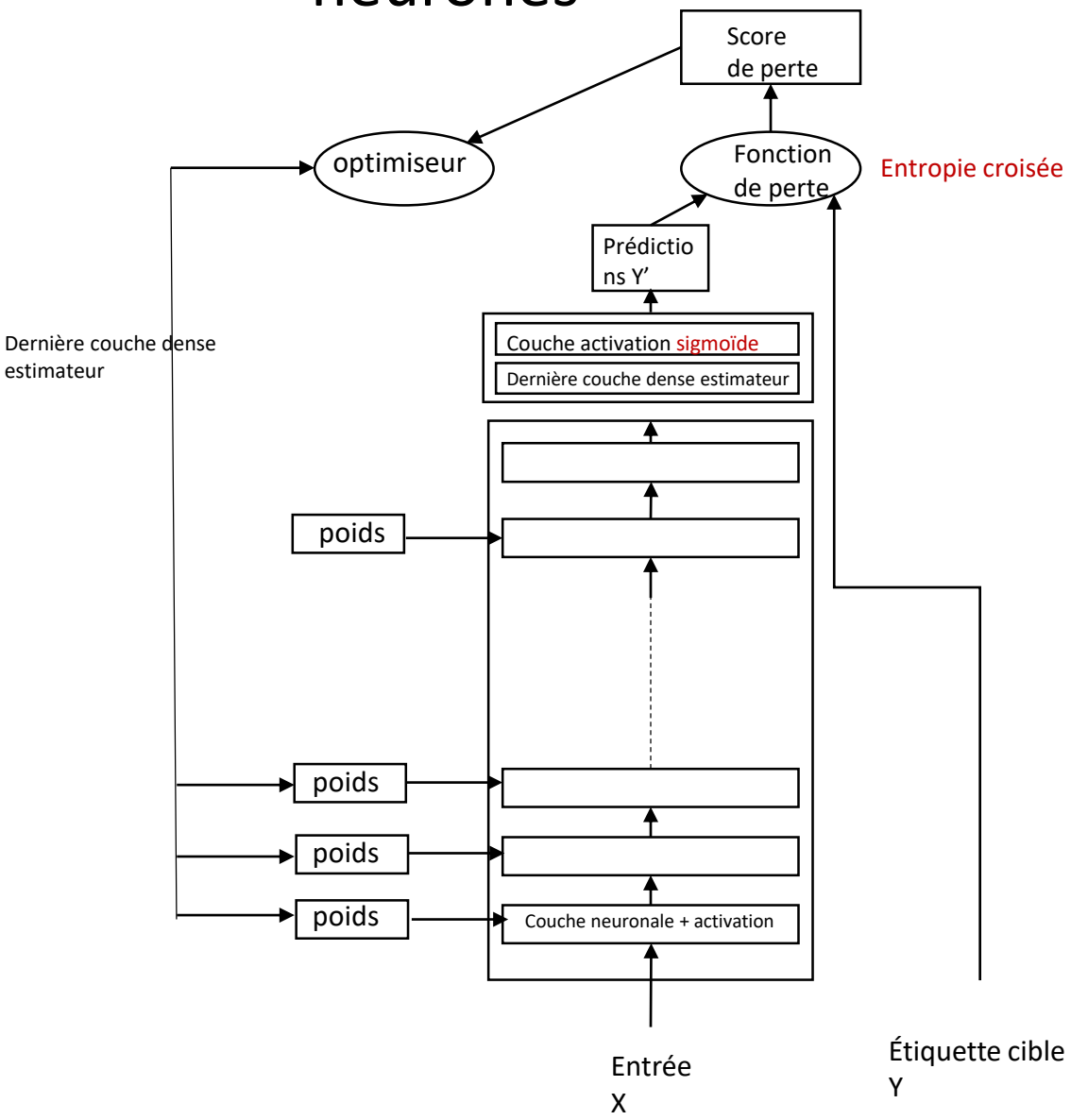
$$L(Y',Y) = -\sum_1^N (Y(k) * \log Y'(k)$$

Classification en plusieurs classes : la fonction Softmax



$$s_i = \frac{e^{\sigma_i}}{\sum_k e^{\sigma_k}}$$

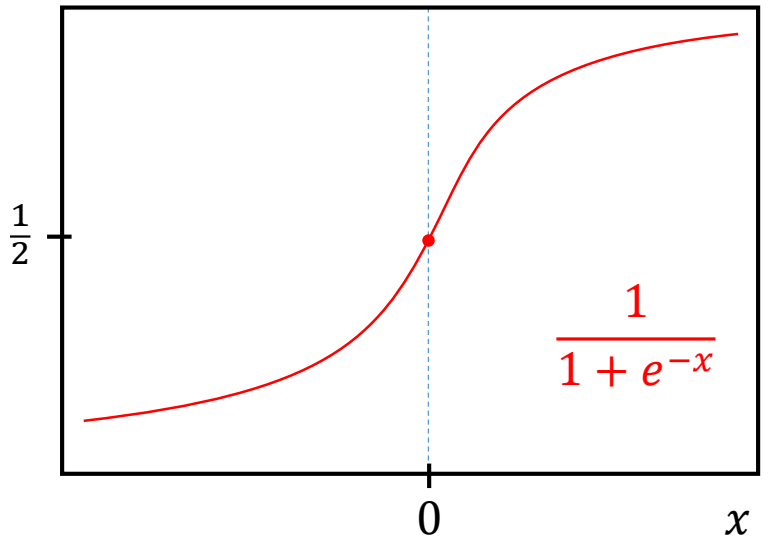
Architecture générale et fonctionnement d'un réseau de neurones



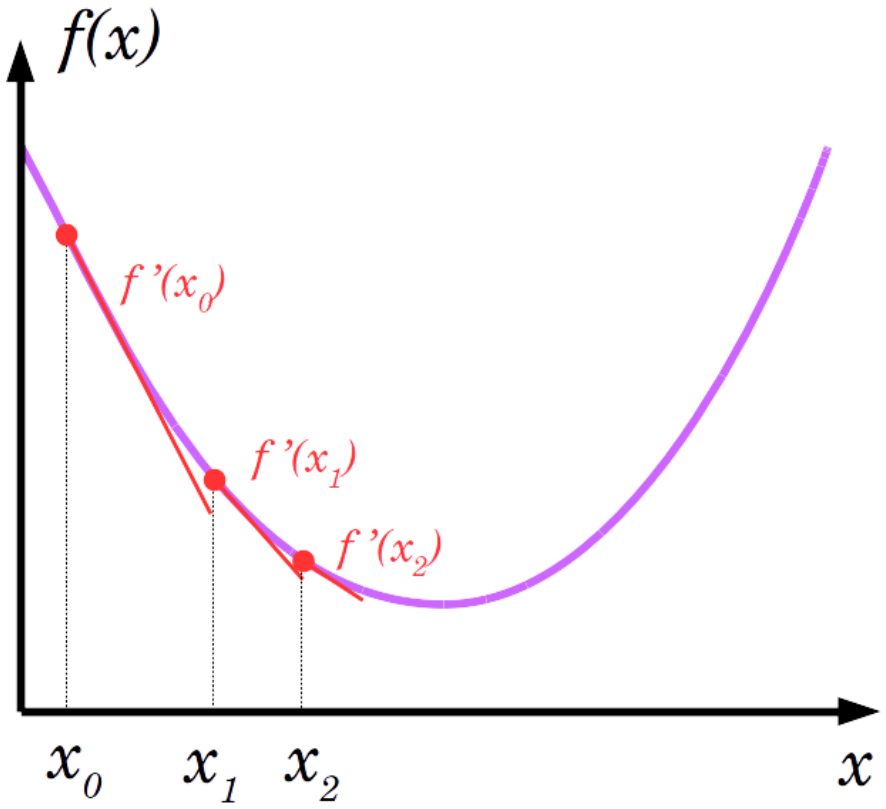
Classification binaire

- Le réseau se termine par une couche dense à une seule sortie soit un seul neurone.
- Cette couche utilisera la fonction d'activation sigmoïde qui « écrase » les valeurs dans l'intervalle [0,1] pour que la sortie finale soit interprétée comme une probabilité d'appartenance de l'entrée à la classe.
- La fonction de perte est la fonction entropie croisée

Fonction sigmoïde



Entraînement des réseaux de neurones : Une descente de gradient



On dispose d'un échantillon d'entrainement de n valeurs $(x^i)_{1\leq i\leq n}$ et des étiquettes associées $(y^i)_{1\leq i\leq n}$

Les sorties calculées pour cette échantillon sont : $(y'^i)_{1\leq i\leq n}$
Les paramètres du réseau sont regroupés dans une variable Θ

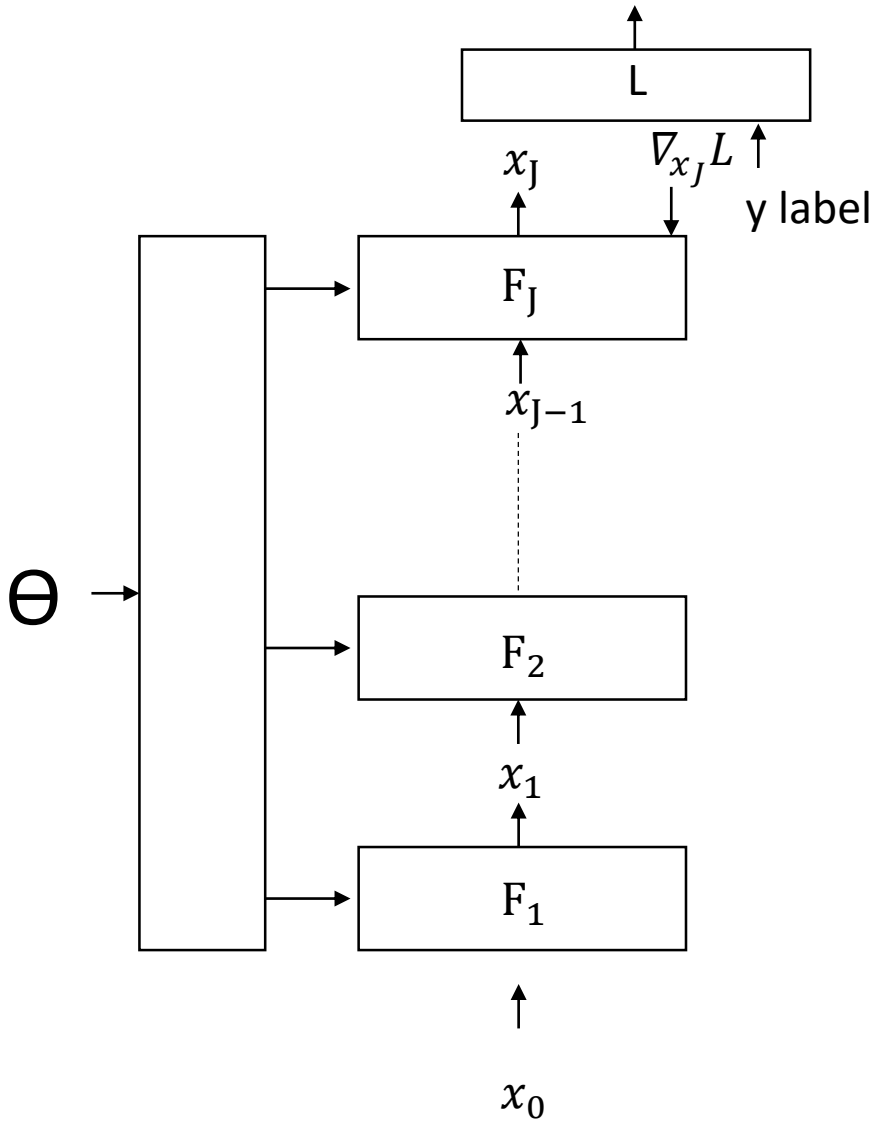
La fonction de perte (loss) s'exprime : $L(\Theta) = \frac{1}{n}\sum_1^n l(y'^i, y^i)$

Objectif de la descente de gradient : trouver le minimum de la fonction L(Θ)

- On part d'un point quelconque Θ_0
- A une étape n, le réseau est configuré avec la valeur Θ_n et pour faire baisser la « loss », on va bouger dans la direction du gradient : $\Theta_{n+1}=\Theta_n-\delta\nabla_\Theta(\Theta_n)$ pour se rapprocher au plus près du minimum Θ_m

Comme $\nabla_\Theta(\Theta_n) = \frac{1}{n}\sum_1^n \nabla_\Theta\, l(x_j^i, y^i)$, il suffit d'être capable de calculer la quantité : $\nabla_\Theta l(x_j^i, y^i)$ pour chaque i

La rétropropagation



Pour une entrée vectorielle quelconque x_0 du réseau, les sorties successives des différentes couches du réseau sont notées x_1,…, x_J, la sortie du réseau étant x_J

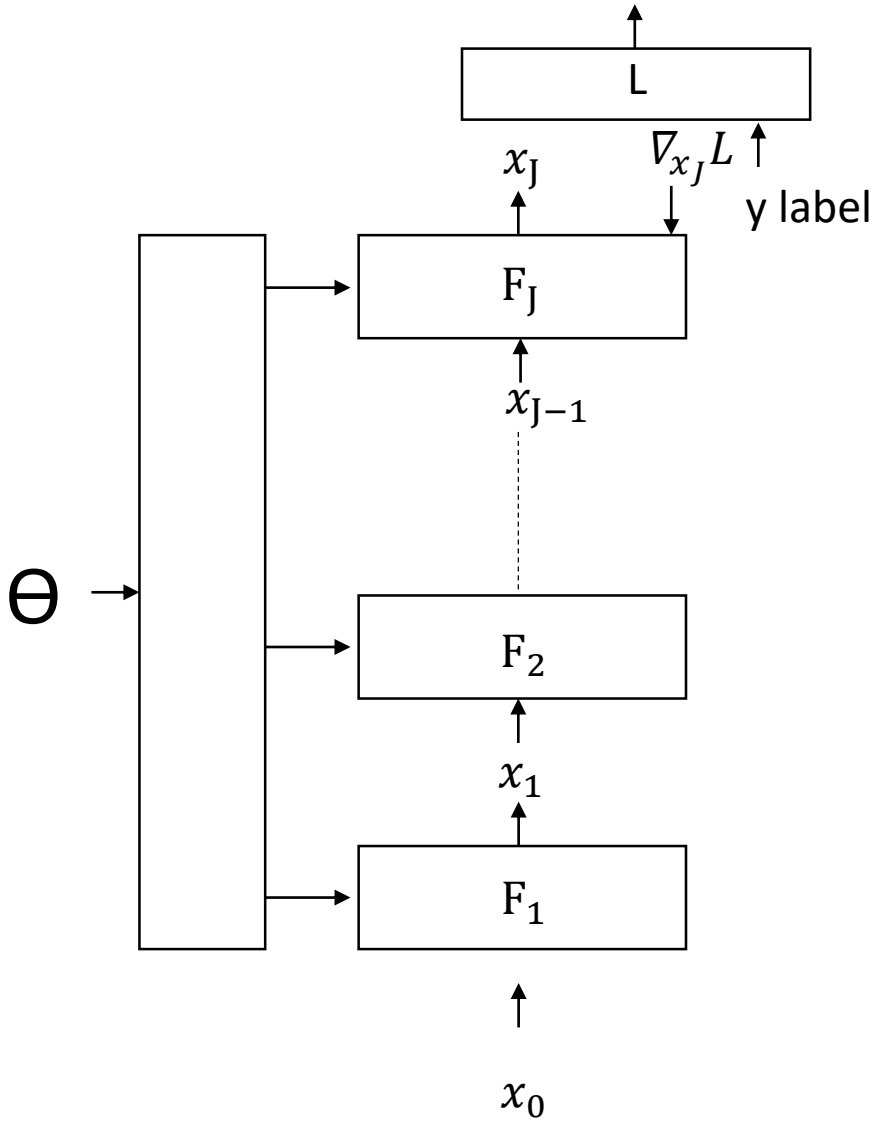
On dispose d'un échantillon d'entrainement de n valeurs $(x_0^i)_{1\leq i\leq n}$et des étiquettes associées $(y^i)_{1\leq i\leq n}$

Les sorties calculées pour cette échantillon sont : $(x_j^i)_{1\leq i\leq n}$

La fonction de perte (loss) s'exprime : $L(\Theta) = \frac{1}{n}\sum_1^n l(x_j^i, y^i)$

Comme $\nabla_\Theta(\Theta_n) = \frac{1}{n}\sum_1^n \nabla_\Theta\, l(x_j^i, y^i)$, il suffit d'être capable de calculer la quantité : $\nabla_\Theta l(x_j^i, y^i)$ pour chaque i

La rétropropagation



Pour calculer cette quantité, $\nabla_\Theta l(x_J^i, y^i)$, qui est le gradient de la loss par rapport aux paramètres du réseau, on commence par calculer son gradient par rapport aux sorties du réseau, c'est-à-dire la quantité $\nabla_{x_J} l$. Dans la suite, l'indice i supérieur n'est plus nécessaire car on prend un seul exemple quelconque de l'ensemble d'apprentissage, x_0

<u>Cas de la régression linéaire</u> :

La fonction de perte est : $l(x_J,y) = \frac{1}{2}(x_J - y)^2$ donc $\frac{\partial l}{\partial x_J} = x_J - y$

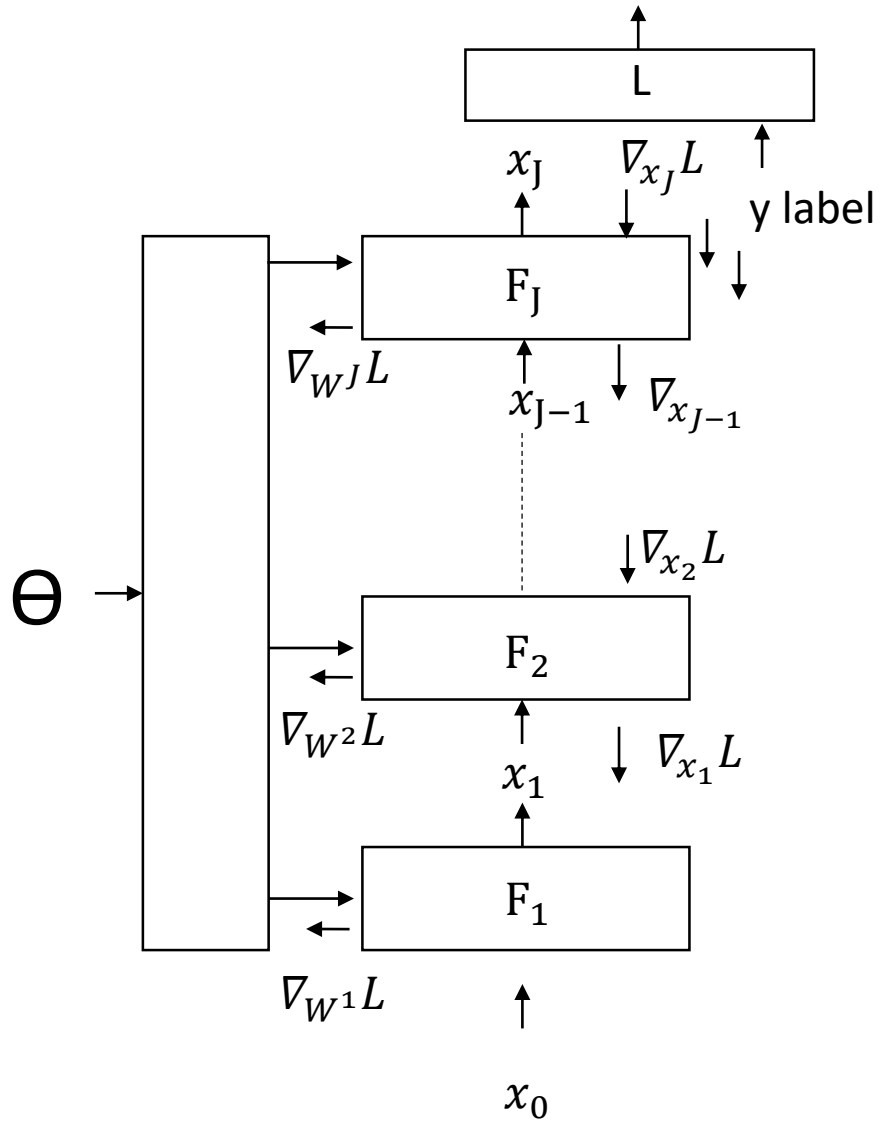
<u>Cas de la classification :</u>

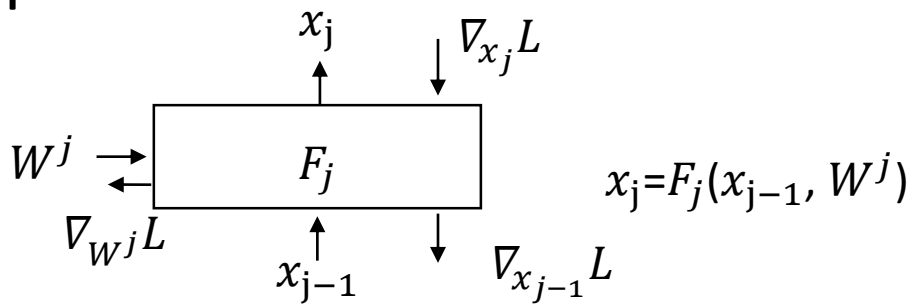
Dans le cas de la classification, la fonction Softmax est appliquée à la sortie x_J du réseau avant de calculer l'entropie croisée si bien que :

$l(x_J,y) = -\sum_1^N y(k) \times \log$ (Softmax (x_J(k)) où Y est un vecteur contenant des 0 partout sauf pour l'indice correspondant à la probabilité de la bonne classe, qui est la valeur 1.

On montre facilement que : $\nabla_{x_J} l$ = Softmax (x_J) - y

La rétropropagation





$x_j = F_j(x_{j-1}, W^j)$

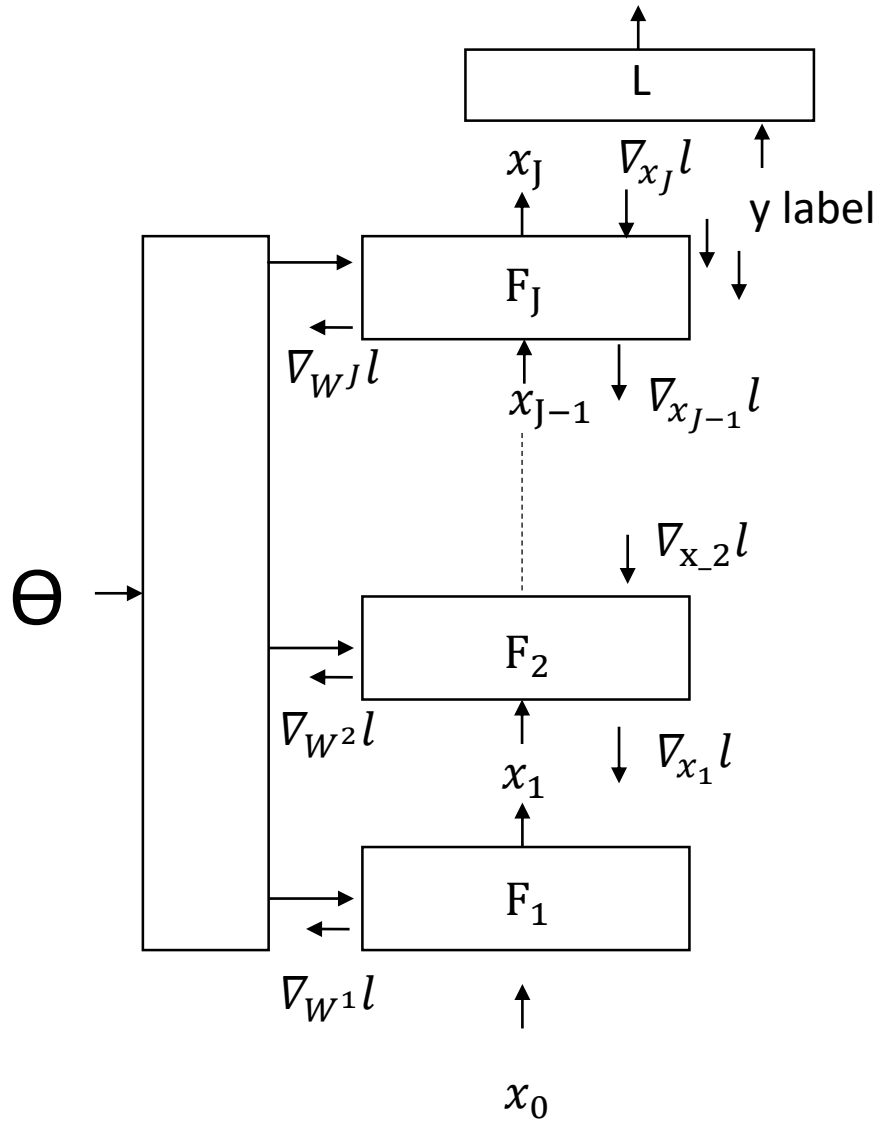
Puis on calcule les gradients de la perte par rapport aux sorties de chaque couche du réseau de proche en proche, en partant de la dernière couche, et en remontant ainsi en arrière jusqu'aux sorties de la première couche, d'où le terme de rétro propagation du gradient.

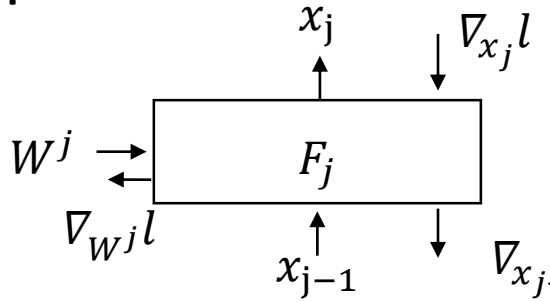
La méthode est la suivante : pour chaque couche j du réseau dont l'entrée est notée x_{j-1} et la sortie x_j , il suffit de considérer la fonction de perte par rapport aux variables x_{j-1} comme la composition de deux fonctions :

- La fonction F_j qui à x_{j-1} associe x_j
- La fonction qui à x_j associe la perte l

Si f et g sont des fonctions d'une seule variable, la dérivée de g o f est la fonction (g′o f) *f′

La rétropropagation





$x_j = F_j(x_{j-1}, W^j)$

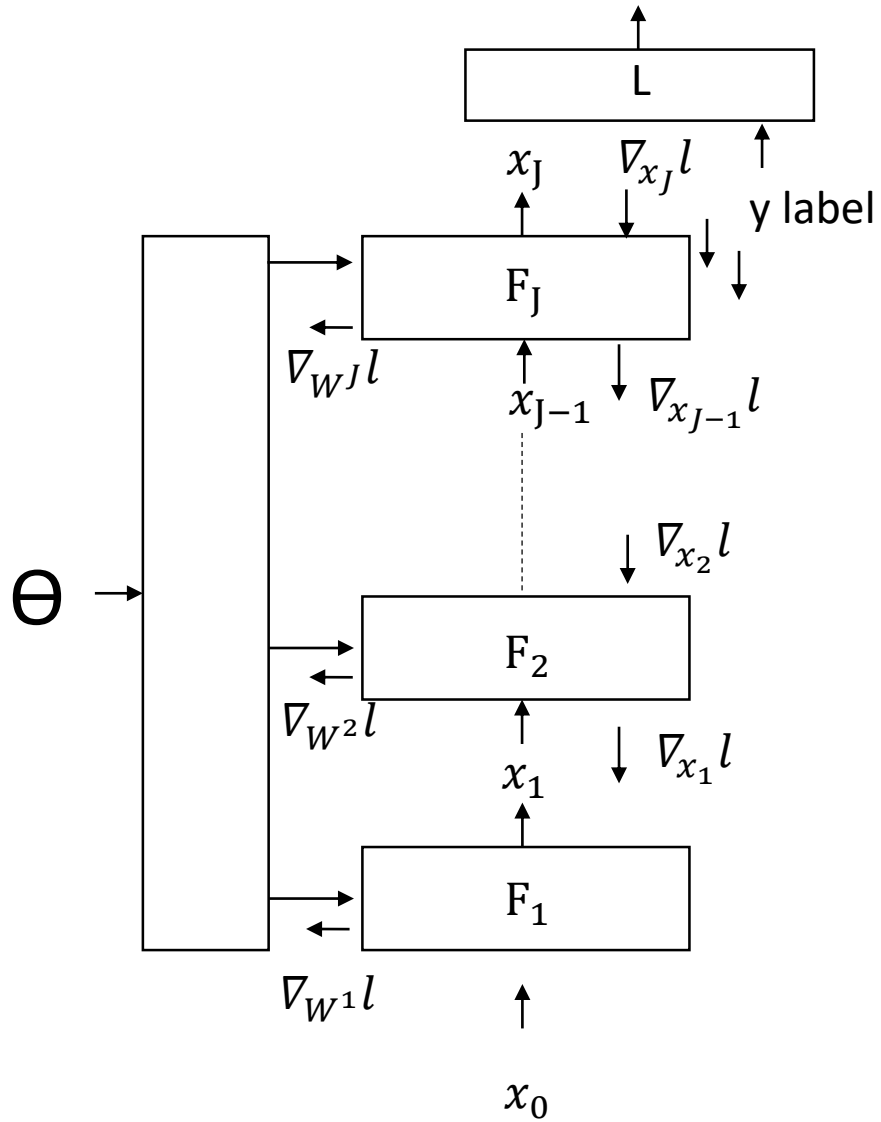
Ce résultat se généralise avec des fonctions à plusieurs variables $\nabla_{x_{j-1}} l = \frac{\partial F_j}{\partial x_{j-1}} * \nabla_{x_j} l$ où la notation $\frac{\partial F_j}{\partial x_{j-1}}$ désigne le jacobien de la fonction F_j c'est-à-dire la matrice dont chaque ligne d'indice k est la suite des dérivées partielles des composantes de F_j par rapport à la kième variable x_{j-1}(k)

Le résultat final recherché, à savoir le gradient de la perte par rapport aux poids de la couche j du réseau peut se calculer en considérant la fonction de perte comme la fonction composée des deux fonctions :

- La fonction F_j qui, à W^j associe la sortie x_j
- La fonction qui à x_j associe la perte l

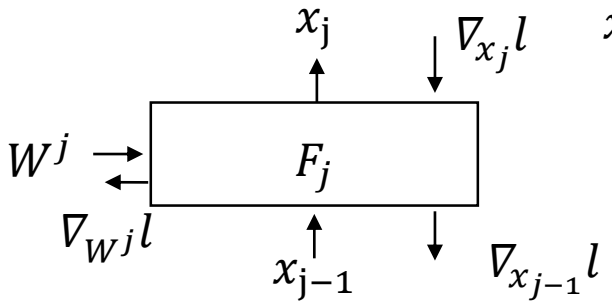
$$\nabla_{W^j} l = \frac{\partial F_j}{\partial W^j} * \nabla_{x_j} l$$

La rétropropagation



$$\nabla_{x_{j-1}} l = \frac{\partial F_j}{\partial x_{j-1}} * \nabla_{x_j} l$$

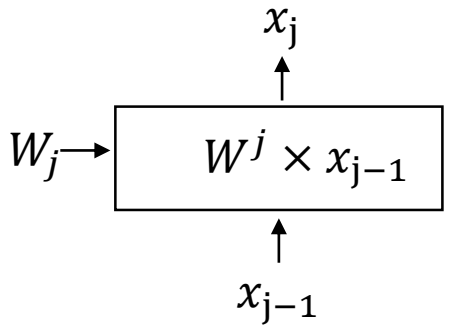
$$\nabla_{W^j} l = \frac{\partial F_j}{\partial W^j} * \nabla_{x_j} l$$



$x_j = F_j(x_{j-1}, W^j)$

Deux cas sont à considérer selon que la couche j est une couche neuronale ou une couche d'activation.

<u>Cas d'une couche neuronale :</u>



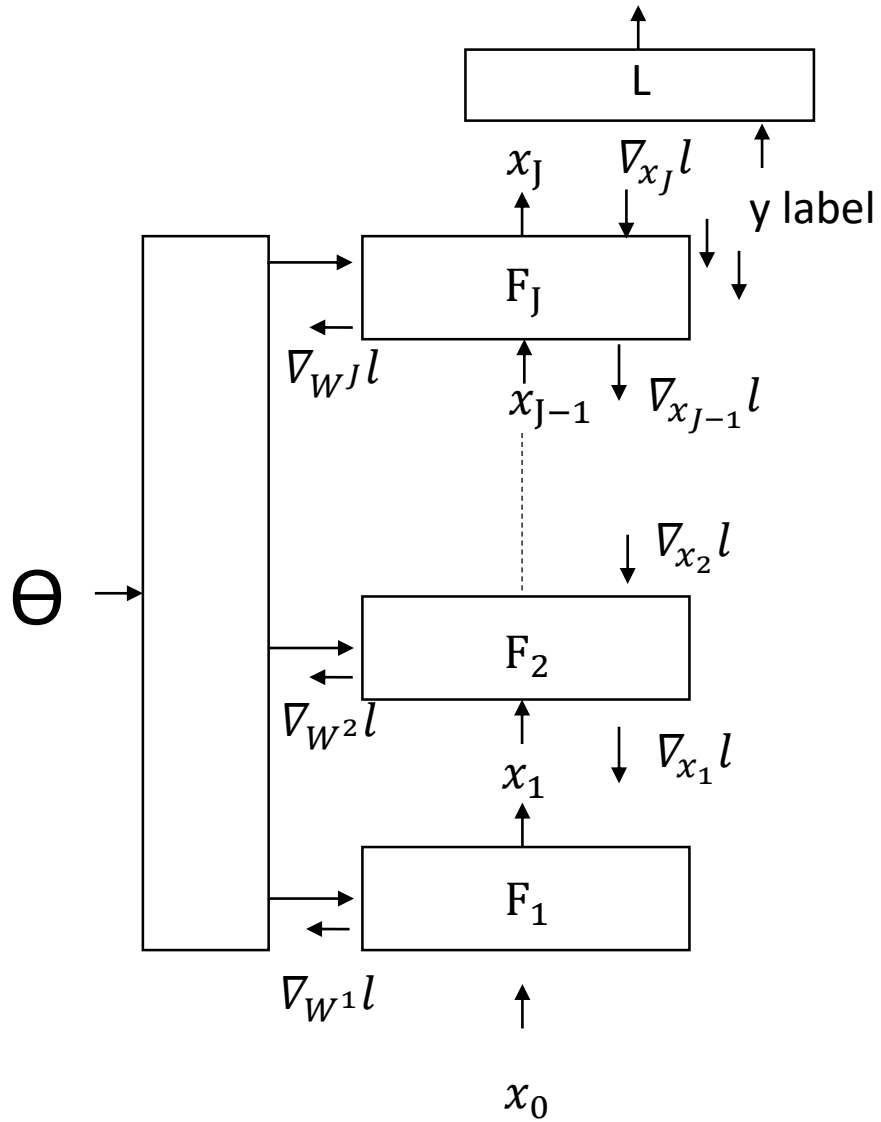
$x_j = F_j(x_{j-1}, W^j) = W^j * x_{j-1}$

$\frac{\partial F_j^1}{\partial x_{j-1}}$ = transp (W^j), transposée de la matrice W^j

En effet, on a, pour tout k
$x_j(k) = \sum_l W_{kl}^j * x_{j-1}(l)$ d'où :
$\frac{\partial x_j(k)}{\partial x_{j-1}(l)} = W_{kl}^j$

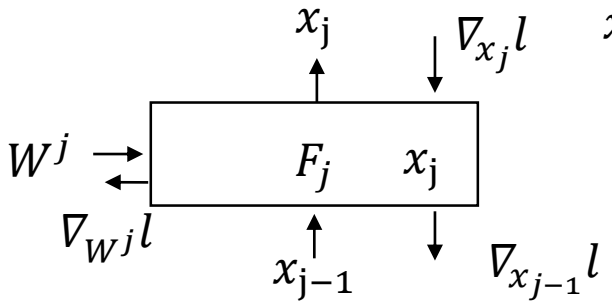
On a donc :
$\nabla_{x_{j-1}} l$ = transp (W^j) $\times \nabla_{x_j} l$

La rétropropagation



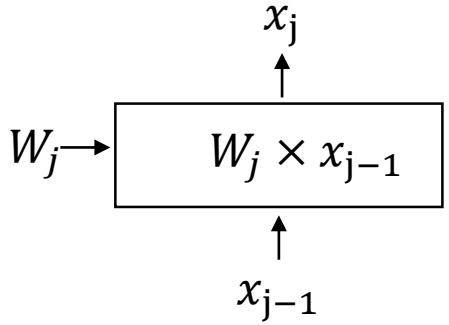
$$\nabla_{x_{j-1}} l = \frac{\partial F_j}{\partial x_{j-1}} * \nabla_{x_j} l$$

$$\nabla_{W^j} l = \frac{\partial F_j}{\partial W^j} * \nabla_{x_j} l$$



$x_j = F_j(x_{j-1}, W^j)$

Cas d'une couche neuronale :



$x_j = F_j(x_{j-1}, W^j) = W^j * x_{j-1}$

On a montré précédemment que :
$\nabla_{x_{j-1}} l$ = transp $(W^j) \times \nabla_{x_j} l$

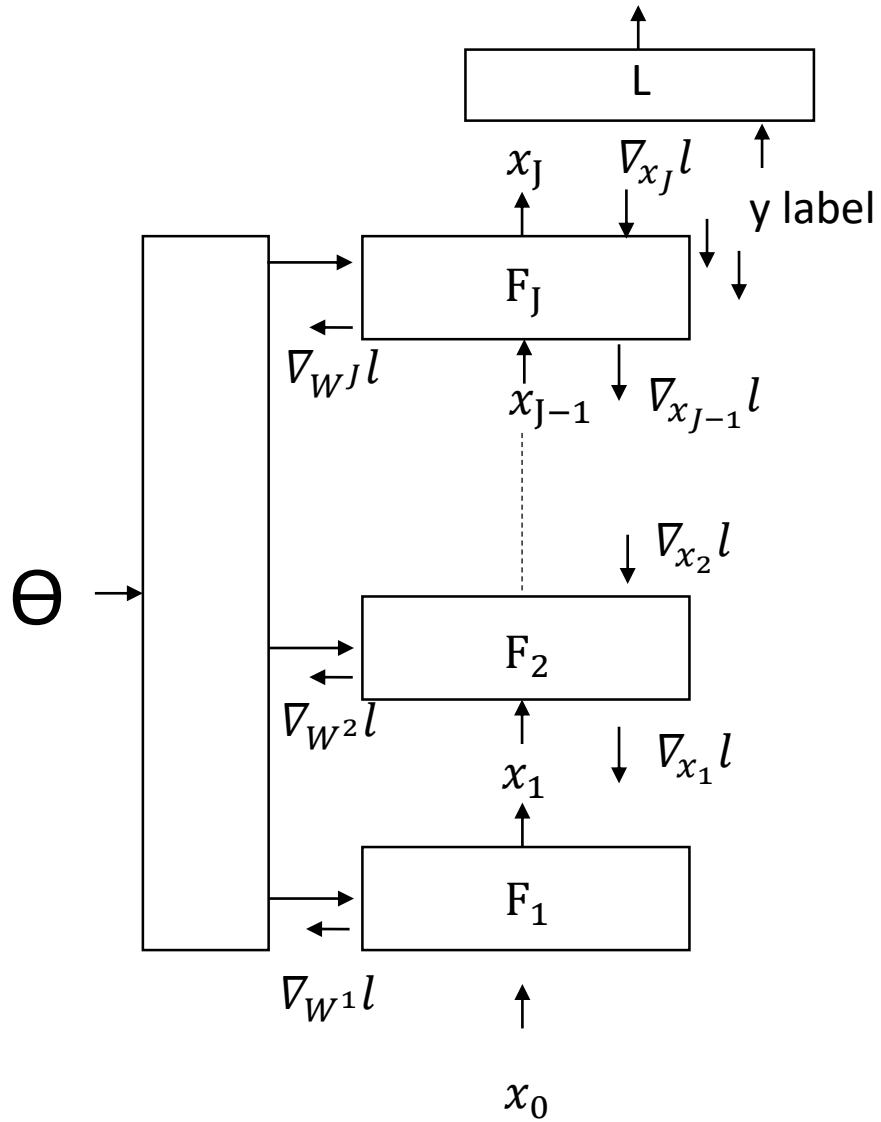
On peut montrer facilement que si $\nabla_{W^j} l$ désigne la matrice constituée des dérivées partielles $\left[\frac{\partial l}{\partial W^j_{mp}}\right]$,

$\nabla_{W^j} l = \nabla_{x_j} l$ * transp(x_{j-1})

ce qui peut s'écrire aussi :

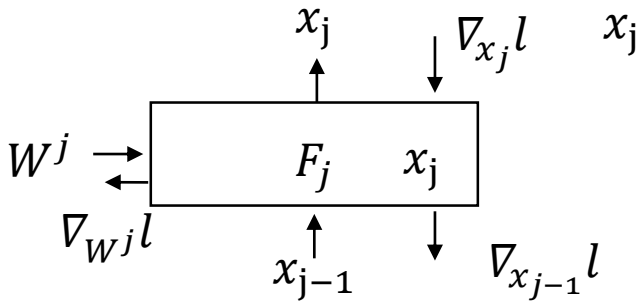
$\frac{\partial l}{\partial W^j_{mp}} = \frac{\partial l}{\partial x_j(m)} \times x_{j-1}(p)$

La rétropropagation

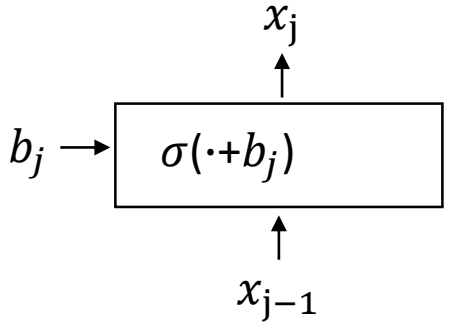


$$\nabla_{x_{j-1}} l = \frac{\partial F_j}{\partial x_{j-1}} * \nabla_{x_j} l$$

$$\nabla_{W^j} l = \frac{\partial F_j}{\partial W^j} * \nabla_{x_j} l$$



<u>Cas d'une couche d'activation</u> :



Le jacobien de la fonction $\sigma(\cdot + b_j)$ est la matrice diagonale [diag σ'] qui contient les dérivées de σ sur les différentes composantes du vecteur $x_{j-1} + b_j$

On a donc :

$\nabla_{x_{j-1}} l = [\text{diag } \sigma'] \times \nabla_{x_j} l$
et
$\nabla_{b_j} l = [\text{diag } \sigma'] \times \nabla_{x_j} l$

Mise en œuvre de la descente de gradient

Dans la pratique, la descente de gradient est effectuée en itérant plusieurs fois sur l'ensemble des données d'apprentissage.

Chaque itération sur l'ensemble des données est appelée une « epoch ».

Un choix aléatoire d'un vecteur de poids est effectué avant le lancement du processus.

Puis, à chaque itération ou epoch, l'ensemble des données est découpé de manière aléatoire en sous-ensembles appelés « mini-batch » de taille égale fixée par un paramètre en entrée du processus global.
Au sein d'une epoch, la descente de gradient est effectuée pas à pas en calculant le gradient non pas sur l'ensemble des données de l'échantillon, mais en effectuant la moyenne sur un batch.
Au bout d'une epoch, l'ensemble des données a été parcouru, et une nouvelle itération démarre à l'epoch suivante. L'algorithme s'arrête après avoir effectué le nombre d'epoch fixé en paramètre d'entrée.

Validation et phénomène de sur-ajustement (over-fitting)

Une fois entraîné, le réseau doit pouvoir être testé sur des données qu'il n'a jamais rencontrées au cours de la phase d'apprentissage.
Une partie de l'échantillon d'origine ne sera pas utilisé pour l'entraînement et sera réservé pour la validation.

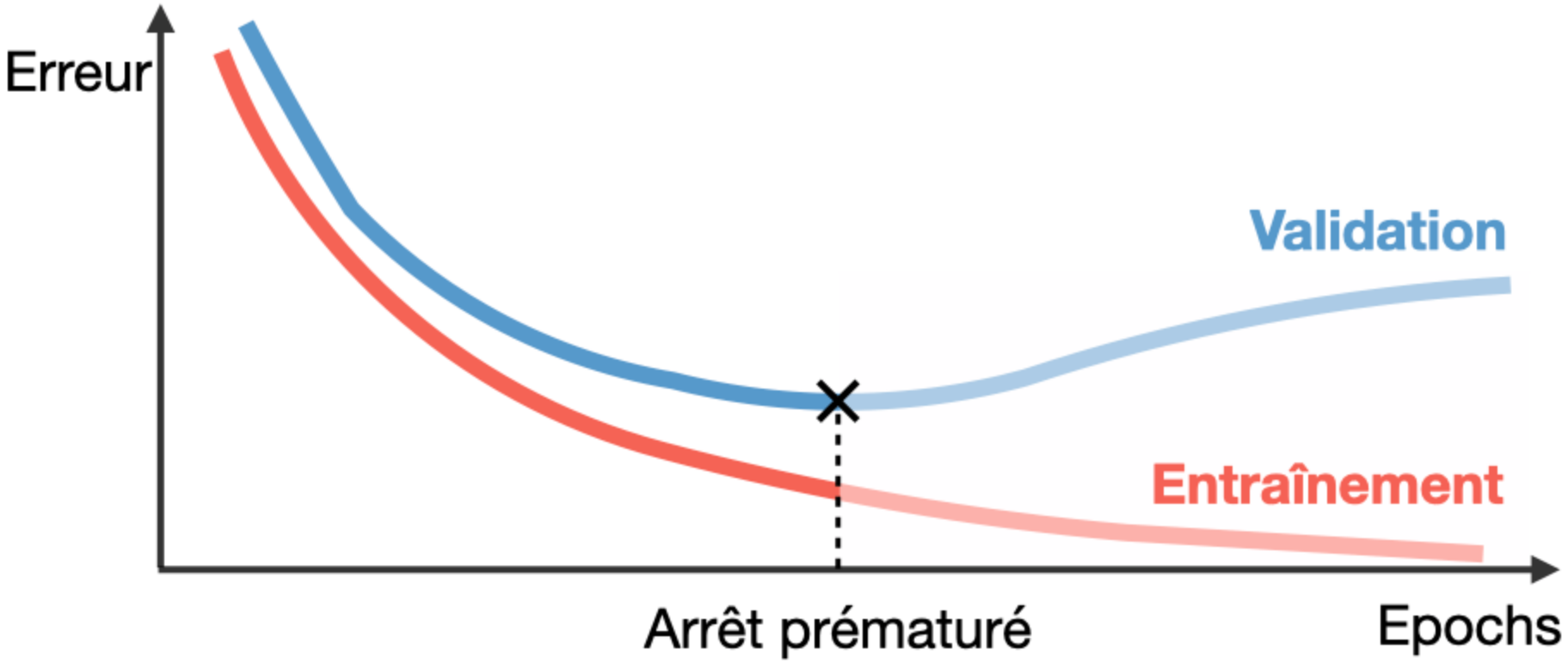
L'évaluation du modèle sur les données de validation va même pouvoir être effectué au cours de la phase d'entraînement.

Ce test consiste à l'issue de chaque epoch à mesurer l'exactitude de prédiction du modèle et sa fonction de coût ou perte, non seulement sur l'ensemble des données d'entraînement, mais aussi sur les données de validation.

D'une manière générale, on constate alors que la perte d'entrainement diminue à chaque époque, et que l'exactitude d'entrainement augmente.

En revanche, cette tendance s'inverse sur les données de validation : alors que la perte de validation commence par diminuer à chaque époch, et l'exactitude de validation à augmenter, les performances du modèle passent par un point culminant et se dégradent ensuite sur les données de validation au bout de seulement quelques époques. Ce phénomène est représentatif d'un sur-ajustement (overfitting) du modèle aux données d'entraînement : le modèle est en sur apprentissage sur les données d'entrainement si bien qu'il n'est plus capable de généraliser de manière pertinente sur des données en dehors de cet ensemble d'apprentissage.

Le phénomène de sur-ajustement (overfitting)



Représentation et traitement des données textuelles

- Les données textuelles sont une des formes de données séquentielles les plus répandues.

- Un réseau de neurones n'acceptant en entrée que des données numériques, il est nécessaire de procéder à la **vectorisation** du texte.

- Il existe principalement deux façons de procéder à cette vectorisation :

    - Segmenter le texte en mots, et transformer chaque mot en vecteurs,
    - Segmenter le texte en caractères, et transformer chaque caractère en vecteur

- Les unités de décomposition d'un texte (mots, ou caractères) sont appelées des **jetons (tokens)** et la décomposition d'un texte en jetons s'appelle l'analyse lexicale **(tokenization)**
- Notons que chatGPT ne traite ni avec des mots ou des caractères, mais avec des jetons dont la longueur varie entre l'affixe (préfixe, suffixe, infixe) et le mot complet.

Représentation des mots et caractères avec l'encodage one-hot

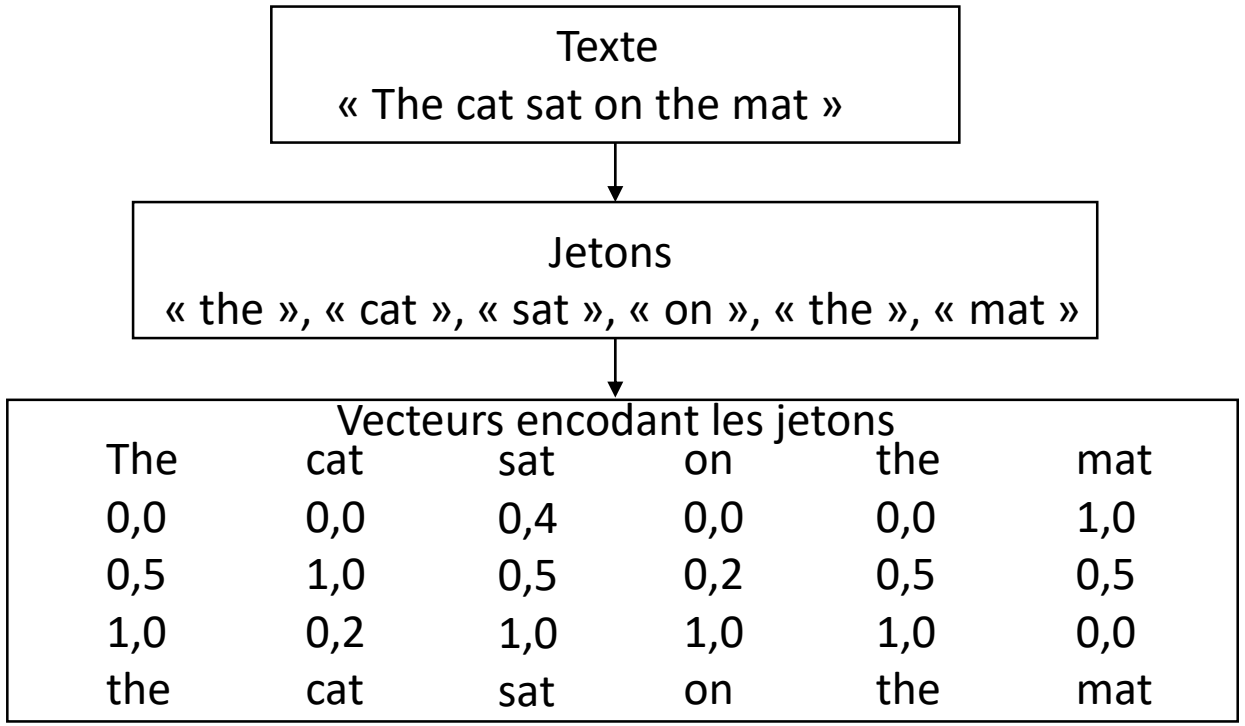
- On commence par associer un index unique à chaque mot

- N étant la taille du vocabulaire, l'indice i de chaque mot est transformé en un vecteur de dimension N, dont les composantes sont toutes nulles, sauf pour la composante d'indice i qui est égale à 1
- L'encodage one-hot peut également s'appliquer au niveau des caractères.

- Les représentations de mots obtenues à partir d'un encodage one-hot sont creuses, et de haute dimension (la taille du vocabulaire)

Utilisation des embeddings de mots

- Cette méthode consiste à utiliser des vecteurs à virgule flottante de faible dimension pour représenter des mots.
- les embeddings de mots ne sont pas choisis au hasard et font l'objet d'un apprentissage, en même temps que la tâche principale à résoudre, ou sont hérités d'un apprentissage sur une autre tâche (dans ce dernier cas, on parle **d'embedding de mots pré-entraînés**)
- L'espace des embeddings de mots possède une structure telle que les relations géométriques entre les vecteurs de mots reflètent alors les relations sémantiques entre les mots.

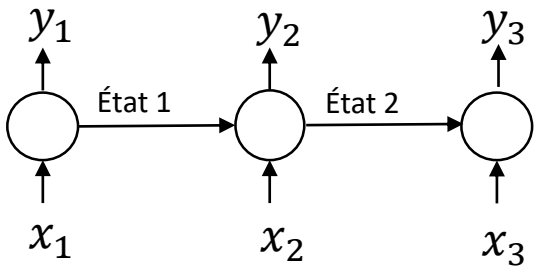


Modèles de réseaux pour le traitement des données textuelles

Les exemples de réseaux vus jusqu'à présent, qu'ils soient entièrement connectés, ou convolutifs, traitent la séquence complète en une seule fois. Ils ne permettent pas de traiter chaque entrée de manière séquentielle en conservant un état des données traitées précédemment. De tels réseaux sont dits réseaux de neurones profonds à propagation avant (feedforward networks)

Les réseaux récurrents s'inspirent du mode de lecture des humains qui décode la phrase mot par mot en gardant un souvenir de ce qui a été déjà lu.

Un réseau récurrent parcourt dans l'ordre les différents éléments de la séquence, et calcule une sortie pour chaque élément de la séquence parcourue. A chaque étape, la sortie est calculée en utilisant un état qui est calculé et mémorisé lors de l'étape précédente. Pour le premier élément de la séquence, l'état n'est pas défini et est initialisé avec un vecteur dont toutes les valeurs sont nulles. Ainsi, si la séquence d'entrée de jetons est (x_1, x_2, x_3,…) le réseau produit séquentiellement les sorties (y_1, y_2, y_3,…)



Applications des réseaux de neurones récurrents au traitement de données textuelles

Évaluation de critiques de films :

Le réseau comprend une première couche d'embedding qui prend en entrée une séquence d'entiers dont les valeurs sont les indices des mots ou jetons dans le dictionnaire, et produit en sortie la séquence correspondante vectorisée, avec des vecteurs d'embedding de dimension 32.

Cette couche est suivie par une couche récurrente SimpleRNN.
La dernière couche est une couche dense avec une sortie unique représentant la probabilité que la critique soit positive.

```
model = Sequential()
model.add(Embedding(max_feature, 32))
model.add(SimpleRNN(32))
Model.add(dense(1, activation = 'sigmoid'))
```

Modèles de langage (ex ChatGPT)

La méthode universelle pour générer des données séquentielles en apprentissage profond consiste à entraîner un réseau à prédire le prochain jeton.

Par exemple, étant donnée l'entrée «the cat is on the ma», le réseau est entraîné à produire la cible t, qui est le caractère ou le jeton suivant.

Pendant l'entraînement, la sortie d'un tel modèle sera une couche dense dont la dimension de la sortie sera égale à la taille du dictionnaire des jetons sur laquelle sera appliquée une fonction softmax représentant la distribution de probabilités pour le prochain jeton.

Le principe de génération d'un texte est de partir d'un texte d'amorce pour générer le mot ou le jeton suivant, puis de répéter ce processus en ajoutant la sortie obtenue à l'entrée du réseau.

Quelques caractéristiques de ChatGPT

Chemin neuronal le plus long composé de 400 couches principales

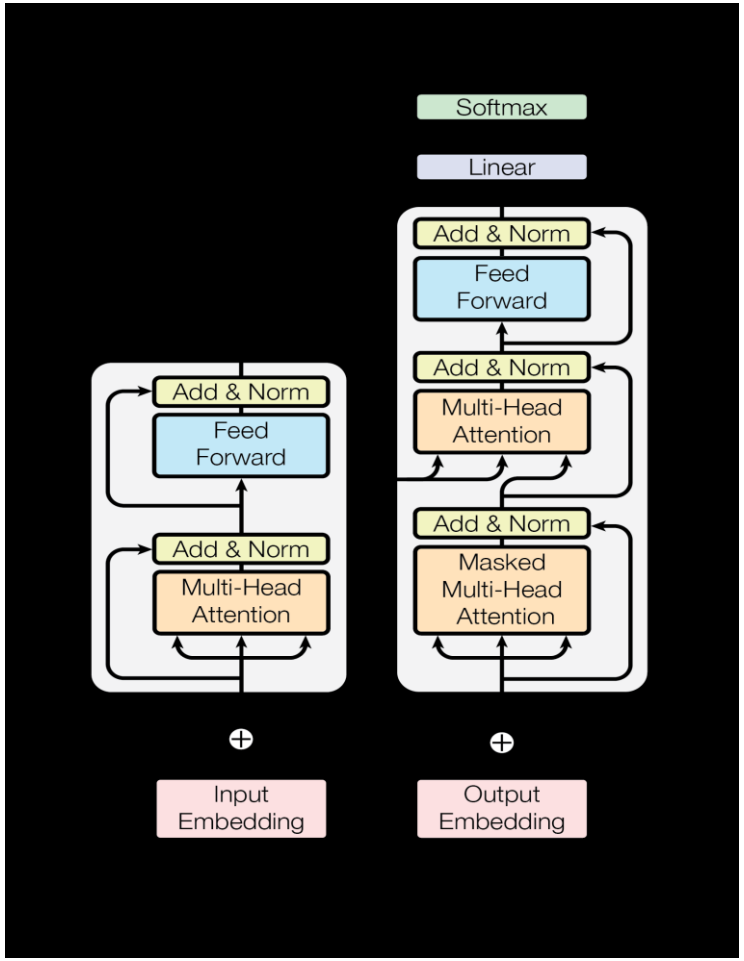
Plusieurs millions de neurones comportant 175 milliards de paramètres

Environ 50 000 jetons possibles

La couche d'embedding prend donc en entrée une séquence de n jetons représentés par des entiers compris entre 1 et 50 000 et les convertit en une séquence de vecteurs d'embedding (de longueur 768 pour GPT2 et 12 288 pour GPT3).

Blocs d'attention : 12 pour GPT2 et 96 pour GPT3.

Architecture du transformer (« attention is all you need »)



Transformer : le mécanisme d'attention

L'entrée d'un transformer est un vecteur X = $(x_1, \ldots, x_n)$ où x_i est un vecteur d'embedding.

Un transformeur est composée d'un assemblage de blocs (transformer blocks) qui sont eux-mêmes des réseaux multicouches comprenant en particulier une couche mettant en œuvre un mécanisme d'attention (auto-attention ou attention croisée).

Chaque bloc transformer prend en entrée un vecteur de type $(x_1, \ldots, x_n)$ et calcule en sortie une séquence de vecteurs de même longueur $(y_1, \ldots, y_n)$

Pour chaque jeton x_i, le réseau calcule un coefficient d'attention de ce jeton avec chacun des jetons x_j qui le précède, y compris lui-même, noté α_{ij}. Ces coefficients sont ensuite normalisés par le biais d'une fonction softmax.

Le module d'attention calcule un vecteur intermédiaire de sortie $(v_1, \ldots, v_n)$ à l'aide d'une couche neuronale simple entièrement connectée (ce qui revient à multiplier le vecteur X par une matrice W^V

La sortie de la couche d'attention est le vecteur Y = $(y_1, \ldots, y_n)$ avec $y_i = \sum_{j<i} \alpha_{ij} . v_j$

Transformer : le mécanisme d'attention

Calcul des coefficients d'attention :

Pour chaque x_i on calcule une « *query* » q_i et une « *key* » k_i qui sont des vecteurs de même dimension en appliquant deux couches linéaires au vecteur X :

$q_i = W^Q x_i$
$k_i = W^K x_i$

Les coeffcients α_{ij} sont les produits scalaires $q_i . k_j$ avant normalisation softmax.