Cours de Data Science : Réseaux de neurones

Marc Tommasi

6 décembre 2021

Outline

Outline

Cybernétique : années 50

- Bio-inspiration
- McCullloch-Pitts 43 : modèle du neurone
- Rosenblatt 58/62 : apprendre les poids
- Widrow-Hoff 60 : Adaline comme une instance de la descente de gradient stochastique
- Minsky-Papert 69 : limitation par l'exemple du XOR

Connexionisme : années 80

- une tâche complexe est le résultat de la composition de tâches simples.
- retropropagation du gradient : Rumelhart 86 / Lecun 87
- utilisation de données non étiquetées pour faciliter l'apprentissage de réseaux profonds
- plus trop de connexion avec les neurosciences (on sait peu de choses sur le cerveau, pour l'instant, s'en rapprocher n'a rien donné).

Tirée de Goodfellow

The choice of the functions used to compute these representations is also loosely guided by neuroscientific observations about the functions that biological neurons compute. Modern neural network research, however, is guided by many mathematical and engineering disciplines, and the goal of neural networks is not to perfectly model the brain. It is best to think of feedforward networks as function approximation machines that are designed to achieve statistical generalization, occasionally drawing some insights from what we know about the brain, rather than as models of brain function

Deep learning : depuis milieu des années 2000

- Beaucoup de données (étiquetées)
- Des machines puissantes (GPU, . . .)
- Explosion de la taille des réseaux (ver de terre il y a 15 ans, aujourd'hui comparable à la grenouille)
- Résolument inspiré par les statistiques, les probabilités, l'optimisation, le calcul numérique, . . .

Évolutions I

Évolution des type de modèles

- Modèle linéaire, un seul neurone (perceptron)
- Modèles à plusieurs couches,
- Réseaux de convolution,
- Modèles non linéaires,
- Modèles récurrents.

Évolution des jeux de données

- de quelques centaines ou milliers d'exemples
- à quelques millions et milliards aujourd'hui
- 10 millions d'exemples étiquetés pour pour avoir des performances proches de l'homme dans bien des cas. . .

Évolutions II

Évolution des performances

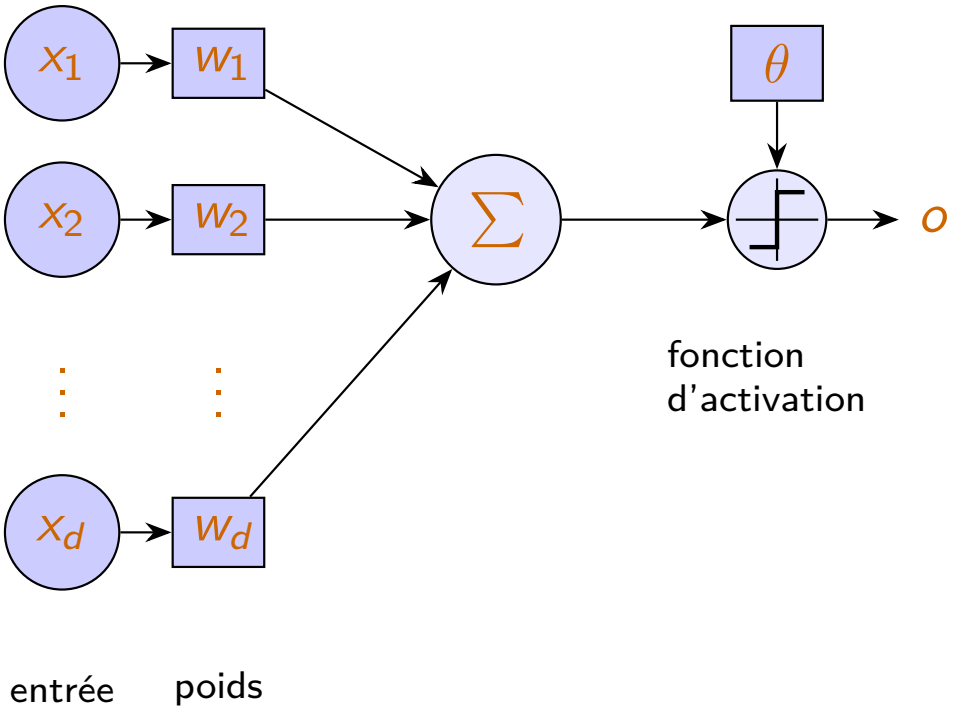
- Des réseaux aujourd'hui avec des millions de neurones
- Impact impressionnant
  - en vision
  - en reconnaissance de la parole

Outline

Un problème de classification supervisée binaire

- Les données sont décrites par d valeurs réelles : $\boldsymbol{x} \in \mathbb{R}^d$
- Les résultats sont binaires : on doit associer une valeur binaire y à chaque donnée (y vaut -1 ou 1)
- On dispose de données étiquetées : $\{(\boldsymbol{x}_1, y_1) \ldots, (\boldsymbol{x}_m, y_m)\}$
- On cherche la meilleure fonction qui permet d'obtenir les résultats (étiquettes y) en fonction des entrées (données $\boldsymbol{x}$).

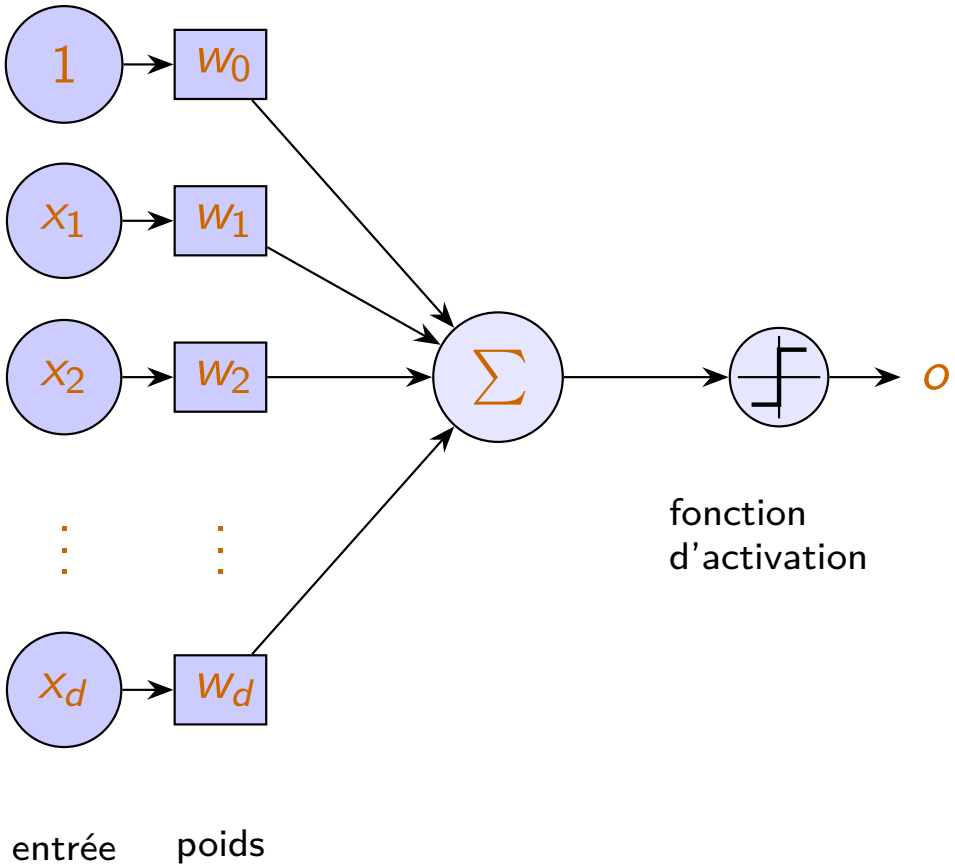
Perceptron linéaire à seuil



- Entrées : d valeurs $x_1, \ldots x_d$;
  Sortie : o une valeur binaire
- Fonctionnement : d coefficients (synaptiques) $w_1, \ldots, w_d$, le biais θ ,

$$o = \begin{cases} 1 & \text{si } \boldsymbol{w}^\top \boldsymbol{x} > \theta \\ -1 & \text{sinon} \end{cases}$$

Perceptron linéaire à seuil



- Entrées : d valeurs $x_1, \ldots x_d$;
  Sortie : o une valeur binaire
- Fonctionnement : $d+1$ coefficients (synaptiques) $w_0, \ldots, w_d$, entrée $x_0 = 1$,

$$o = \begin{cases} 1 & \text{si } \boldsymbol{w}^\top \boldsymbol{x} > 0 \\ -1 & \text{sinon} \end{cases}$$

Interprétation

Example (Exercices)

- Quel est le perceptron linéaire à seuil qui calcule un OU logique entre n variables binaires (prenant les valeurs 0 ou 1)
- Donnez une représentation graphique de ce que vous avez proposé dans le cas de deux variables : représentez par des points de deux couleur les valeurs possibles et la frontière de décision pour la sortie (considérant cette fois que les valeurs des x_i peuvent être réelles).
- Représentez maintenant les points correspondants à une fonction XOR.
- Démontrez qu'un perceptron linéaire à seuil ne peut séparer ces points correspondant au XOR.

Algorithme par correction d'erreur I

Intuition

- considérons des valeurs pour $\boldsymbol{w}$, comment les adapter en fonction d'un exemple $(\boldsymbol{x}, y)$?
- rappelons que les poids w_i associés aux entrées x_i qui sont à 0 ne participent pas à la décision.
- Supposons $y = 1$.
  - Si $\boldsymbol{w}^\top \boldsymbol{x} > 0$ alors on n'a rien a faire
  - Si $\boldsymbol{w}^\top \boldsymbol{x} <= 0$ alors il faut augmenter les poids associés aux entrées x_i qui sont à 1.
- Supposons $y = -1$.
  - Si $\boldsymbol{w}^\top \boldsymbol{x} <= 0$ alors on n'a rien a faire
  - Si $\boldsymbol{w}^\top \boldsymbol{x} > 0$ alors il faut baisser les poids associés aux entrées x_i qui sont à 1.

Algorithme par correction d'erreur II

L'algorithme d'apprentissage

- Entrée : Un échantillon $\{(\boldsymbol{x}_1, y_1) \ldots, (\boldsymbol{x}_m, y_m)\}$
- $t \leftarrow 0$
- Initialiser les poids $\boldsymbol{w}_0$ à des valeurs aléatoires
- Répéter
  - Choisir/recevoir un exemple k, $(\boldsymbol{x}_k, y_k)$ dans S.
  - Calculer la sortie $\hat{y} = \text{sign}(\boldsymbol{w}_t^\top \boldsymbol{x}_k)$
  - $t \leftarrow t + 1$
  - Si $(\hat{y} \neq y_k)$ alors $\boldsymbol{w}_t \leftarrow \boldsymbol{w}_{t-1} + y_k \boldsymbol{x}_k$
- Jusqu'à ...
- Sortie : $\boldsymbol{w}_t$

Algorithme par correction d'erreur III

Exemple avec le OU

- On commence avec $w_0 = 0$; $w_1 = 1$ et $w_2 = -1$.
- On passe tous les exemples de l'échantillon :
  $S = \{((1,0,0),0),((1,0,1),1),((1,1,0),1),((1,1,1),1)\}$.
- Calculer les sorties successives et les mises à jour

Propriétés de cet algorithme I

Si l'échantillon est linéairement séparable, si les exemples sont présentés équitablement, l'algorithme s'arrête et calcule un séparateur linéaire pour S.

Theorem (Terminaison)

Soit $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m \in \mathbb{R}^d$ tels que il existe $r > 0$ avec $\|\boldsymbol{x}_k\| \leq r$ pour tout $k \in [1, m]$. Si il existe $\rho > 0$ et $\boldsymbol{v} \in \mathbb{R}^d$, $\|\boldsymbol{v}\| = 1$, tels que pour tout $k \in [1, m]$, $\rho \leq y_k(\boldsymbol{v}^\top \boldsymbol{x}_k)$, alors le nombre d'itérations de l'algorithme du perceptron avec $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ en entrée est borné par r^2/ρ^2.

- r : est la norme maximale des données
- $\boldsymbol{v}$: est l'hyperplan séparateur des données (unitaire)
- ρ : est la *marge*

Propriétés de cet algorithme II

Sketch of proof, borne inférieure

Supposons qu'à l'étape n, l'algorithme a déjà effectué M mises à jour aux temps t_1, t_2, ..., t_M.

$$\begin{aligned}\|\boldsymbol{w}_n\|^2 &= \|\boldsymbol{w}_{t_M} + y_{t_M}\boldsymbol{x}_{t_M}\|^2 \\ &= \|\boldsymbol{w}_{t_M}\|^2 + \|\boldsymbol{x}_{t_M}\|^2 + 2y_{t_M}\boldsymbol{w}_{t_M}^\top \boldsymbol{x}_{t_M}\end{aligned}$$

Comme par l'algorithme, $y_{t_M}\boldsymbol{w}_{t_M}^\top \boldsymbol{x}_{t_M} < 0$ on a

$$\|\boldsymbol{w}_n\|^2 \leq \|\boldsymbol{w}_{t_M}\|^2 + \|\boldsymbol{x}_{t_M}\|^2$$

.

Par induction

$$\|\boldsymbol{w}_n\|^2 \leq \|\boldsymbol{w}_0\|^2 + \sum_{i=1}^{M}\|\boldsymbol{x}_{t_i}\|^2 \leq r^2 M$$

Propriétés de cet algorithme III

Sketch of proof, borne supérieure

On a $\boldsymbol{v}$ qui est unitaire (de norme 1) donc, pour tout vecteur $\boldsymbol{w}$, la norme de $\boldsymbol{w}$ est plus grande que le produit scalaire $\boldsymbol{v}^\top \boldsymbol{w}$. Ici,

$$\begin{aligned}\|\boldsymbol{w}_n\| &\geq \boldsymbol{v}^\top \boldsymbol{w}_n \\ &\geq \boldsymbol{v}^\top (\boldsymbol{w}_{t_M} + y_{t_M} \boldsymbol{x}_{t_M}) \\ &\geq \boldsymbol{v}^\top \boldsymbol{w}_{t_M} + y_{t_M} \boldsymbol{v}^\top \boldsymbol{x}_{t_M} \\ &\geq \boldsymbol{v}^\top \boldsymbol{w}_{t_M} + y_{t_M} \rho\end{aligned}$$

Par induction (on suppose qu'on initialise $\boldsymbol{w}_0$ à 0)

$$\|\boldsymbol{w}_n\| \geq M\rho$$

En conclusion on a $\|\boldsymbol{w}_n\|^2 \geq M^2\rho^2$ et $\|\boldsymbol{w}_n\|^2 \leq r^2 M$. Donc

$$M \leq r^2/\rho^2$$

Propriétés et limitations

- Plus la marge est petite, plus l'algorithme peut prendre du temps à converger.
- On remarque que le vecteur $\boldsymbol{w}$ est calculé avec des combinaisons des exemples $\boldsymbol{x}$.
- Les poids sont entiers
- Pas de garantie de convergence dans le cas non séparable
- De nombreux efforts pour réaliser des approximations et tolérer des erreurs
- On peut voir l'algorithme comme une descente de gradient stochastique...

Rappel descente de gradient

Algo générique

- Entrée : Un échantillon $S = \{(\boldsymbol{x}_1, y_1) \ldots, (\boldsymbol{x}_m, y_m)\}$, une fonction de perte f
- Initialiser les poids $\boldsymbol{w}_0$ à des valeurs aléatoires
- Pour t de 1 à T
  - Prendre $\boldsymbol{x}, y$ dans S
  - $\boldsymbol{w}_t \leftarrow \boldsymbol{w}_{t-1} - \eta \nabla_{\boldsymbol{w}} f(\boldsymbol{w}, \boldsymbol{x}, y)$
- Sortie : $\boldsymbol{w}_T$

- De nombreuses fonctions de perte peuvent être utilisées selon les cas (MSE (régression linéaire etc. . .), logistique (régression logistique), exponentielle (dans Adaboost), Hinge loss
- Hinge Loss : $\max(0, 1 - y_k \boldsymbol{w}^\top x_k)$
  - Dérivée : $-y_k x_k$ si les signes de y_k et $\boldsymbol{w}^\top x_k$ diffèrent
  - On retrouve l'algorithme par correction d'erreur avec $\eta = 1$

Outline

Description

- La sortie des neurones d'une couche sont l'entrée des neurones de la couche suivante
- On ne crée pas de cycle
- Le réseau réalise des compositions de plusieurs fonctions
- Ces réseaux sont appelés feed-forward

Autres fonctions d'activation

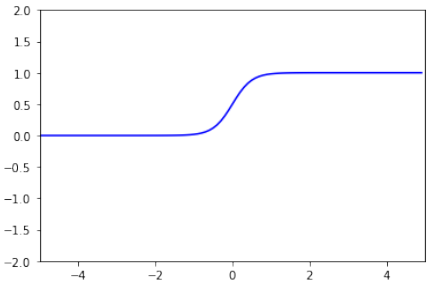
- La fonction d'activation à seuil fait un changement très brutal dans un réseau
- Ce changement rend l'apprentissage très difficile : de petits changements ont de gros effets !
- Passage à des fonctions plus « lisses »
  - la sigmoïde par exemple.

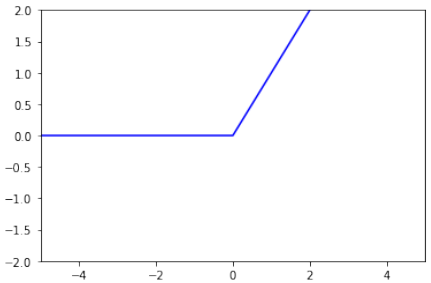
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

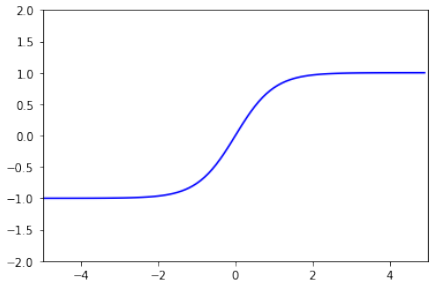
la dérivée

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

  - La tangente hyperbolique (tanh) ou la hinge/ReLU.







Compositions de fonctions

Chain rule

- $(f \circ g)' = (f' \circ g) \cdot g'$ ou encore
- Si $F(x) = f(g(x))$, alors $F'(x) = f'(g(x))g'(x)$, ou encore
- Si $z = f(y)$ et $y = g(x)$, alors $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$

Dans le cas d'un réseau de neurones

- On réalise la composition d'une fonction d'activation (comme par exemple un ReLU, tanh, ...) avec un produit scalaire sur une couche,
- qu'on compose avec d'autres couches...

Expressivité

La non linéarité est importante

- Si on ne calcule que des produits scalaires, cela ne sert à rien !
- Si on a une activation RELu ou tanh, ou sigmoïde au augmente la capacité de la classe de fonctions représentées
- Mais le problème d'optimisation des poids devient non convexe

Constats

- Les réseaux de neurones sont très expressifs (toutes les fonctions calculables).
- D'un point de vue ERM, la classe des fonctions est bien trop large et il faut des heuristiques pour limiter l'expressivité.
- Les problèmes d'optimisation sont importants : non convexité, taille des problèmes très importants (les paramètres sont les poids et il peut y en a énormément)
- Demande expertise et ingénierie pour sélectionner des architectures, résoudre les problèmes d'optimisation, appliquer les heuristiques,...
- La fonction calculée par le réseau est difficile à interpréter : quelle importance de tel paramètre, pourquoi tel résultat,...

Outline

Principe

- Rappel : on considère des architectures feed-forward.
- On voit le réseau comme un graphe acyclique de composition de fonctions : fonctions associées à chaque neurone et fonction d'activation
- Forward : avec un exemple en entrée on compose les fonctions suivant le graphe pour calculer la sortie
- On calcule l'erreur à la sortie du réseau (par exemple par $\frac{1}{2}(\hat{y} - y)^2$).
- Backward : on calcule la modification à apporter à l'ensemble des poids sur chaque couche.
  - C'est un calcul de gradient
  - appliqué de façon récursive sur l'ensemble des couches
  - résolu par la chain rule,
  - i.e. : backpropagation ou rétropropagation du gradient.

Rétropropagation du gradient I

Description et notations

- w_{ij} : poids du neurone i vers le neurone j (donc d'une couche vers une autre couche.
- o_i est la sortie calculée de i, o_s est la sortie finale,
- $i \to j$ signifie que j est successeur direct du neurone i, une des entrées de j est la sortie o_i du neurone i,
- z_j est l'entrée pondérée $z_j = \sum_{i \to j} w_{ij} o_i$,
- l'erreur est calculée par l'erreur quadratique $E = \frac{1}{2}(o_s - y)^2$
- la fonction d'activation de tous les neurones est la sigmoïde

$$o_i = \sigma(z_i) = \frac{1}{1 + e^{-z_i}}$$

- la dérivée $\sigma'(z) = \sigma(z)(1 - \sigma(z))$, donc $\frac{\partial o_i}{\partial z_i} = o_i(1 - o_i)$.

Rétropropagation du gradient II

Ce qu'on cherche

- On veut estimer les poids.
- On cherche un minimum de la fonction d'erreur (la fonction de perte).
- L'erreur dépend de tous les poids w_{ij}.
- On regarde la dérivée de la fonction d'erreur par rapport à tout poids w_{ij} : le gradient est l'ensemble des dérivées partielles $\frac{\partial E}{\partial w_{ij}}$.
- Par la chain rule :

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}} = \frac{\partial E}{\partial z_j} o_i$$

car $\frac{\partial z_j}{\partial w_{ij}} = \frac{\partial \sum_{i \to j} w_{ij} o_i}{\partial w_{ij}} = o_i$

- Il faut ensuite calculer $\frac{\partial E}{\partial z_i}$ selon deux cas : une couche cachée ou la couche de sortie.

Rétropropagation du gradient III

Sur la couche de sortie

Influence de z_s sur le calcul de l'erreur :

$$\frac{\partial E}{\partial z_s} = \frac{\partial E}{\partial o_s}\frac{\partial o_s}{\partial z_s} = \frac{\partial E}{\partial o_s} o_s(1-o_s) = (o_s - y)o_s(1-o_s)$$

car $\frac{\partial E}{\partial o_s} = \frac{\partial [1/2(o_s-y)^2]}{\partial o_s} = o_s - y$ et $\frac{\partial o_s}{\partial z_s}$ est la dérivée de la sigmoïde.

Sur les couches intermédiaires

Influence de z_i sur le calcul de l'erreur par l'intermédiaire des z_j suivants dans le réseau :

$$\frac{\partial E}{\partial z_i} = \sum_{i \to j} \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial z_i} = \sum_{i \to j} \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial o_i}\frac{\partial o_i}{\partial z_i} = \sum_{i \to j} \frac{\partial E}{\partial z_j} w_{ij} o_i(1-o_i)$$

car $\frac{\partial z_j}{\partial o_i} = \frac{\partial \sum_{i \to j} w_{ij} o_i}{\partial o_i} = w_{ij}$

Rétropropagation du gradient IV

Au bilan

- On peut calculer l'influence d'un poids sur l'erreur par

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial z_j} o_i$$

- qui demande de calculer récursivement

$$\frac{\partial E}{\partial z_i} = o_i(1 - o_i) \sum_{i \to j} \frac{\partial E}{\partial z_j} w_{ij}$$

- jusqu'à la couche de sortie :

$$\frac{\partial E}{\partial z_i} = o_i(1 - o_i)(o_i - y)$$

Notes complémentaires

- L'algorithme final est un algorithme de descente de gradient classique (voir l'algo générique plus haut)
- Le calcul du gradient se fait par la rétropropagation.
- Différentes fonctions d'activation s'appliquent et différentes topologies de graphe de fonction sont possibles. Ils ne changent pas l'algorithme fondamentalement.
- L'algorithme de rétropropagation s'écrit avec des opérations matricielles. Tout le calcul peut donc se faire par des produits de matrices.
  - d'où l'importance de disposer de GPUs
  - d'où l'existence de librairies comme TensorFlow

Difficultés liées à l'apprentissage

- Initialisation
- Régularisation
- Vanishing gradient problem
  - Les produits successifs donnés par la formule de récurrence font apparaître pour certaines fonctions d'activation comme tanh ou la sigmoïde des valeurs très faibles
  - Si la profondeur est grande, les gradients disparaissent et l'apprentissage est difficile et ne se fait pas.
- Consommation énergétique
- Attaques adverses
- Vie privée