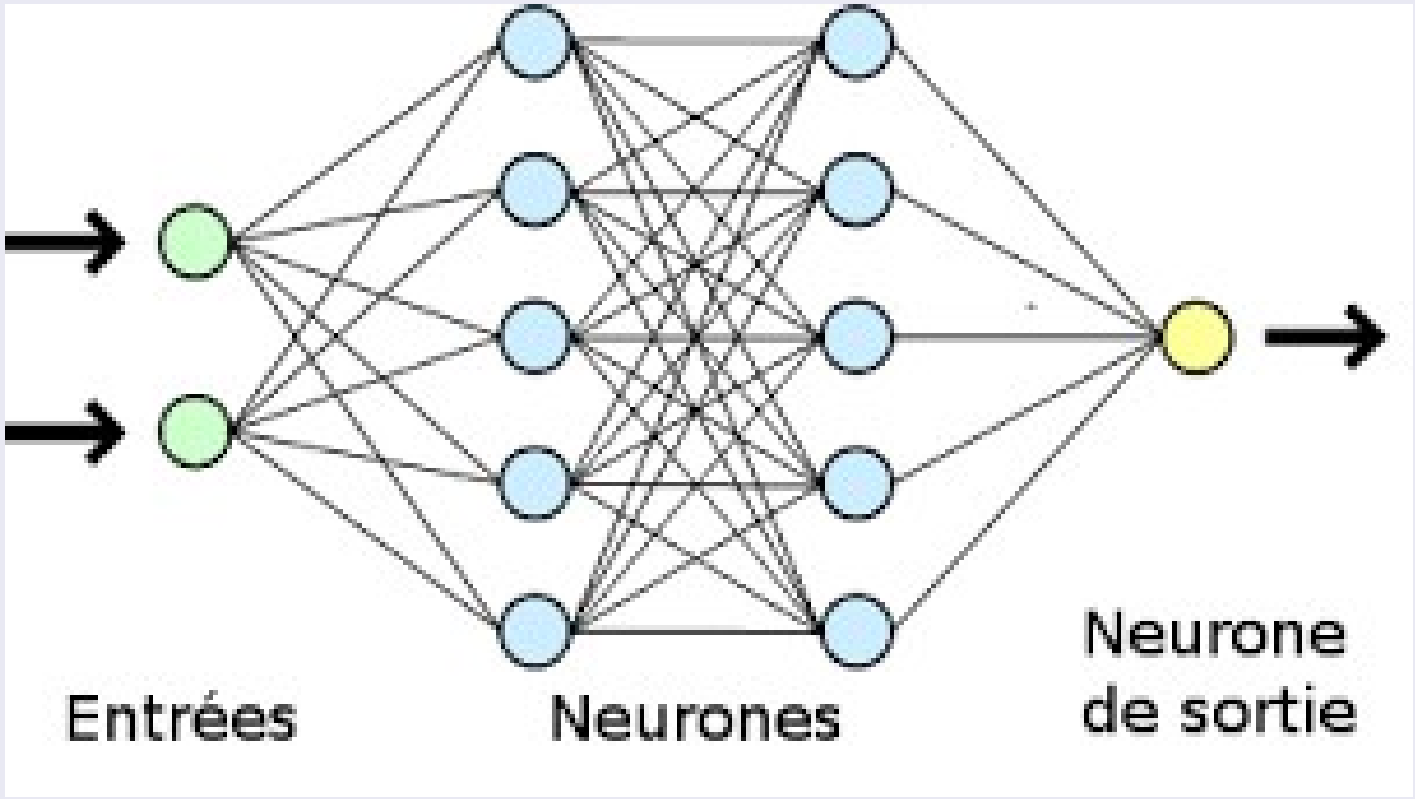
Cours de reconnaissance des formes

cours 3 : réseau de neurones

Adrien CHAN-HON-TONG
ONERA/DTIS département traitement de l'information et système

Aujourd'hui : comprendre ce qu'est un réseau de neurones



La prochaine fois : comprendre comment deep mind peut faire ça

Plan

- Classification supervisé
- Réseau de neurones
- Descente de gradient stochastique

Classification

On voudrait un système qui prend une observation $x \in \mathbb{R}^D$
En déduit la classe de cette observation $y \in \{-1, 1\}$

Exemple : à partir d'une image $x \in \mathbb{R}^D$, je voudrais savoir si c'est une image de chat $y \in \{-1, 1\}$.

chat

non chat

non chat

Classification

On voudrait un système qui prend une observation $x \in \mathbb{R}^D$
En déduit la classe de cette observation $y \in \{-1, 1\}$

Exemple : à partir d'une image $x \in \mathbb{R}^D$, je voudrais savoir si c'est une image de chat $y \in \{-1, 1\}$.

chat

non chat

non chat

Mais, on ne sait pas définir le lien entre x et y.

Classification supervisée

base d'apprentissage $(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$

on cherche $\Phi \in \mathbb{R}^{\mathbb{R}^D}$ tel que $\forall i \in \{1, ..., n\}, sign(\Phi(x_i)) \approx y_i$

Si $\Phi(x_i) \approx y_i$, est ce qu'on a trouvé le système qu'on voulait ?

Classification supervisée

base d'apprentissage $(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$

on cherche $\Phi \in \mathbb{R}^{\mathbb{R}^D}$ tel que $\forall i \in \{1, ..., n\}, sign(\Phi(x_i)) \approx y_i$

Si $\Phi(x_i) \approx y_i$, est ce qu'on a trouvé le système qu'on voulait ?
$\rightarrow$ NON

$$\Phi(x) = \begin{array}{ll} \text{si } \exists i / x = x_i & \text{retourner } y_i \\ \text{sinon} & \text{retourner -1} \end{array}$$

Classification supervisée

base d'apprentissage $(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$

base de test $(x'_1, y'_1), ..., (x'_n, y'_n) \in \mathbb{R}^D \times \{-1, 1\}$

on cherche $\Phi \in \mathbb{R}^{\mathbb{R}^D}$ tel que $\forall i \in \{1, ..., n\}, sign(\Phi(x_i)) \approx y_i$

dans l'espoir que $\forall i \in \{1, ..., n\}, sign(\Phi(x'_i)) \approx y'_i$

problème mathématiquement mal posé : à une même base d'apprentissage peuvent correspondre des bases de tests différentes $\rightarrow$ aucune garantie possible

Classification supervisée

base d'apprentissage $(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$
base de test $(x'_1, y'_1), ..., (x'_n, y'_n) \in \mathbb{R}^D \times \{-1, 1\}$

on se donne une famille de fonction $x \to \Phi(x, \theta)$, θ est un ensemble de poids, paramètres, coefficients

on optimise θ avec l'objectif que $\forall i \in \{1, ..., n\}, sign(\Phi(x_i, \theta)) \approx y_i$

dans l'espoir que $\forall i \in \{1, ..., n\}, sign(\Phi(x'_i, \theta)) \approx y'_i$

Classification supervisée

base d'apprentissage $(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$
base de test $(x'_1, y'_1), ..., (x'_n, y'_n) \in \mathbb{R}^D \times \{-1, 1\}$

sélection : on se donne une famille de fonction $x \to \Phi(x, \theta)$

optimisation : on optimise θ avec l'objectif que $sign(\Phi(x_i, \theta)) \approx y_i$

évaluation : dans l'espoir que $\forall i \in \{1, ..., n\}, sign(\Phi(x'_i, \theta)) \approx y'_i$

classification supervisée : étant donnée un problème réel, on cherche une famille Φ tel qu'après optimisation, on ait une bonne évaluation

Classification linéaire supervisée

base d'apprentissage $(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$
base de test $(x'_1, y'_1), ..., (x'_n, y'_n) \in \mathbb{R}^D \times \{-1, 1\}$

on optimise w, b avec l'objectif que $sign(w^T x_i + b) \approx y_i$

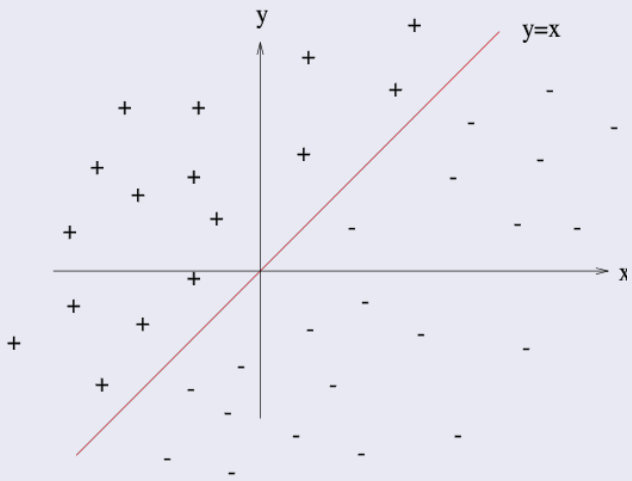
dans l'espoir que $\forall i \in \{1, ..., n\}, sign(w^T x'_i + b) \approx y'_i$

Classification linéaire supervisée

base d'apprentissage $(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$
base de test $(x'_1, y'_1), ..., (x'_n, y'_n) \in \mathbb{R}^D \times \{-1, 1\}$

on optimise w, b avec l'objectif que $sign(w^T x_i + b) \approx y_i$

dans l'espoir que $\forall i \in \{1, ..., n\}$, $sign(w^T x'_i + b) \approx y'_i$



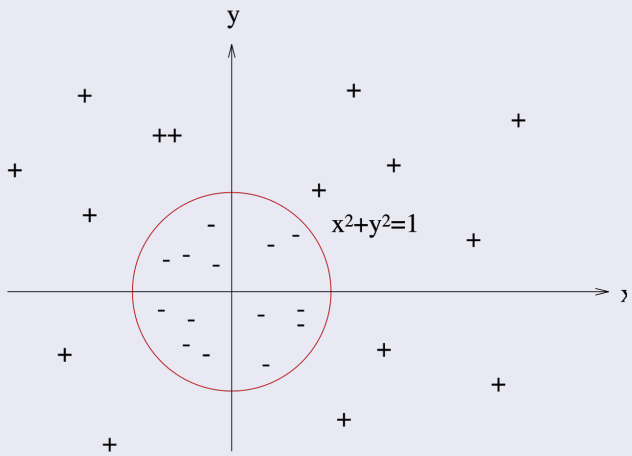
Avantage : optimisation bien étudiée, simplicité (théorie de la généralisation)
Inconvénient : modèle trop simpliste pour capturer des distributions complexes

Classification linéaire supervisée

base d'apprentissage $(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$
base de test $(x'_1, y'_1), ..., (x'_n, y'_n) \in \mathbb{R}^D \times \{-1, 1\}$

on optimise w, b avec l'objectif que $sign(w^T x_i + b) \approx y_i$

dans l'espoir que $\forall i \in \{1, ..., n\}$, $sign(w^T x'_i + b) \approx y'_i$



Avantage : optimisation bien étudiée, simplicité (théorie de la généralisation)
Inconvénient : modèle trop simpliste pour capturer des distributions complexes

classification supervisée par réseau de neurones

base d'apprentissage $(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$
base de test $(x'_1, y'_1), ..., (x'_n, y'_n) \in \mathbb{R}^D \times \{-1, 1\}$

on optimise θ avec l'objectif que $sign(DL(x_i, \theta)) \approx y_i$

dans l'espoir que $\forall i \in \{1, ..., n\}, sign(DL(x'_i, \theta)) \approx y'_i$

Plan

- Classification supervisé
- Réseau de neurones
- Descente de gradient stochastique

Réseau de neurones

1 filtre linéaire

$x \in \mathbb{R}^D \rightarrow x|w \in \mathbb{R}$

Réseau de neurones

1 filtre linéaire

$x \in \mathbb{R}^D \rightarrow x|w \in \mathbb{R}$

K filtres linéaires ?

$x \in \mathbb{R}^D \rightarrow (x|w_k)_{k \in \{1,\ldots,K\}} \in \mathbb{R}^K$

comment les recombiner pour produire une décision ? ?

Réseau de neurones

1 filtre linéaire

$x \in \mathbb{R}^D \rightarrow x|w \in \mathbb{R}$

K filtres linéaires + 1 filtre linéaire !

$x \in \mathbb{R}^D \rightarrow (x|w_k)_{k\in\{1,\dots,K\}} \in \mathbb{R}^K \rightarrow (x|w_k)\,|w' \in \mathbb{R}$

Réseau de neurones

SVM = 1 filtre linéaire

$$x \in \mathbb{R}^D \rightarrow x|w \in \mathbb{R}$$

K filtres linéaires + 1 filtre linéaire = 1 filtre linéaire :-(

$$x \in \mathbb{R}^D \rightarrow (x|w_k)_{k\in\{1,\dots,K\}} \in \mathbb{R}^K \rightarrow (x|w_k)\,|w' = x|\sum_k w'_k w_k \in \mathbb{R}$$

Réseau de neurones

avec un petit peu de non linéarité ?

$relu\,(x) = \max(x, 0)$

sigmoide $sigm\,(x) = \frac{1}{1+\exp(-x)}$

tangente hyperbolique $tanh\,(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}$

Réseau de neurones

SVM = 1 filtre

$$x \in \mathbb{R}^D \rightarrow x|w \in \mathbb{R}$$

K filtres + relu + 1 filtre = réseau de neurones 1950

$$x \in \mathbb{R}^D \rightarrow (relu\,(x|w_k))_{k\in\{1,\dots,K\}} \in \mathbb{R}^K \rightarrow (relu\,(x|w_k))\,|w' \in \mathbb{R}$$

Réseau de neurones

SVM = 1 filtre linéaire

$x \in \mathbb{R}^D \rightarrow x|w \in \mathbb{R}$

K filtres + relu + 1 filtre = réseau de neurones 1950

$x \in \mathbb{R}^D \rightarrow (relu\,(x|w_k))_{k\in\{1,\dots,K\}} \in \mathbb{R}^K \rightarrow (relu\,(x|w_k))\,|w' \in \mathbb{R}$

avec K grand on peut approximer n'importe quelle fonction
Mais ce n'est pas le but : ce qui compte c'est la performance en test

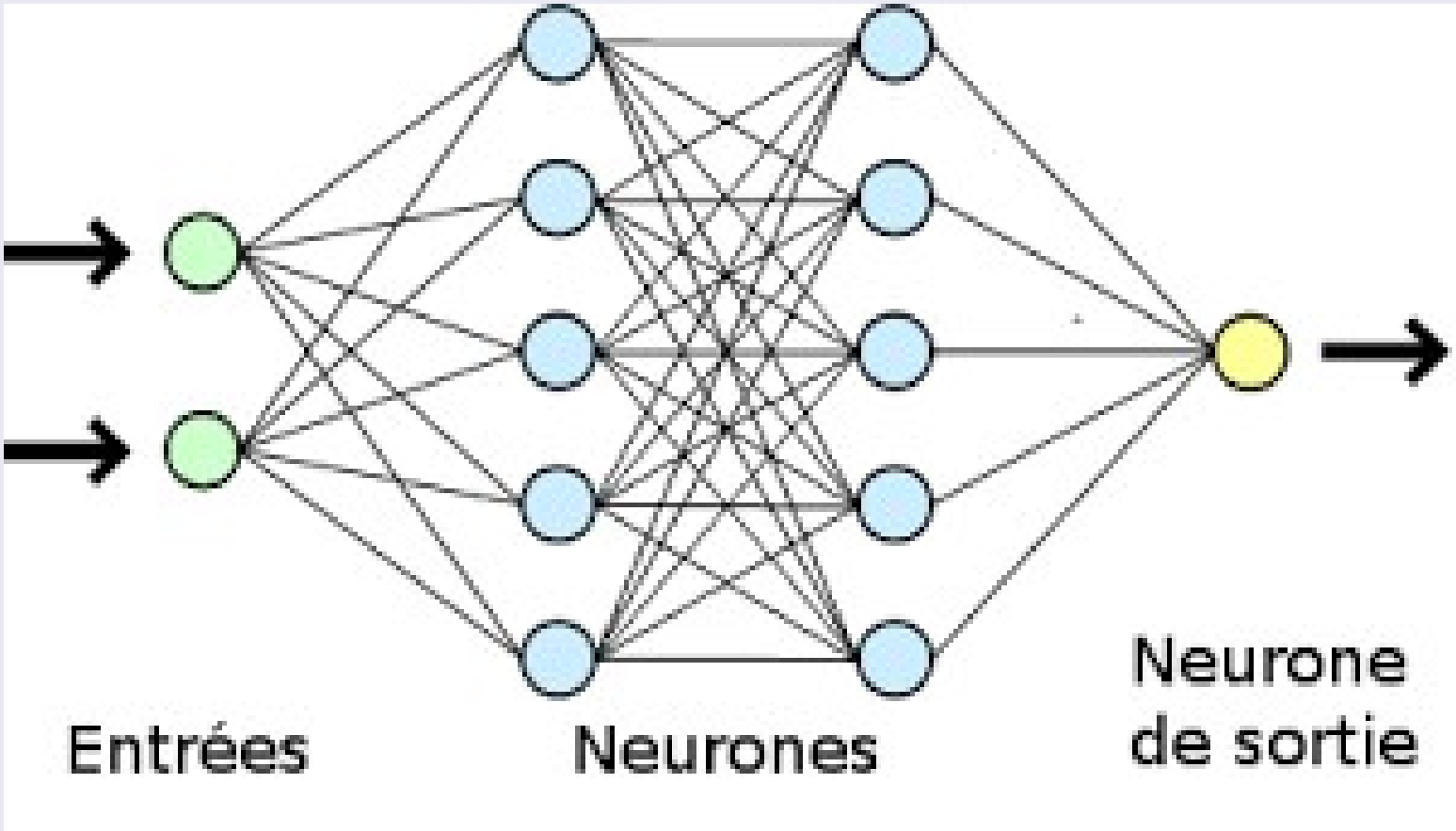
Or expérimentalement, les réseaux sont meilleurs pour l'apprentissage quand ils sont profonds et pas épais

Réseau de neurones

- 1 filtre = SVM
- K filtres + relu + 1 filtre = réseau de neurones 1950
- K_1 filtres + relu + K_2 filtre + relu + ... + K_r filtre + relu + 1 filtre = deep learning

Maintenant vous savez ce qu'est un réseau de neurones !

Réseau de neurones



Réseau de neurones

classification supervisée de vecteurs

base d'apprentissage $(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$
base de test $(x'_1, y'_1), ..., (x'_n, y'_n) \in \mathbb{R}^D \times \{-1, 1\}$

on choisit une structure de réseau $DL(x, \theta) = K_1$ filtres + relu + K_2 filtre + relu + ... + K_r filtre + relu + 1 filtre appliqué à x

on optimise θ avec l'objectif que $sign(DL(x_i, \theta)) \approx y_i$

dans l'espoir que $\forall i \in \{1, ..., n\}, sign(DL(x'_i, \theta)) \approx y'_i$

Comment est ce qu'on optimise θ ? ?

Plan

- Classification supervisé
- Réseau de neurones
- Descente de gradient stochastique

La descente de gradient

f est une fonction dérivable de $\mathbb{R}^D$ dans $\mathbb{R}$ alors
$\forall u, h \in \mathbb{R}^D,\ f(u+h) = f(u) + \nabla f_u | h + \varepsilon(h)h$ [a]
Donc si $\nabla f_u \neq 0$ alors il existe $\lambda > 0$ tel que $f\left(u - \lambda \nabla f_u\right) < f(u)$

---

a. $\varepsilon(h) \underset{h \to 0}{\to} 0$

La descente de gradient

input : f, u_0

1. $u = u_0$
2. calculer ∇f_u
3. si $\nabla f_u \approx 0$ ou early stopping alors sortir
4. $\lambda = 1$
5. tant que $f\left(u - \lambda \nabla f_u\right) \geq f(u)$ faire $\lambda = 0.5\lambda$
6. $u = u - \lambda \nabla f_u$
7. go to 2

cet algorithme converge vers un point u^* tel que $\nabla f_u = 0$

La descente de gradient

input : f, u_0

1. $u = u_0$
2. calculer ∇f_u
3. si $\nabla f_u \approx 0$ ou early stopping alors sortir
4. $\lambda = 1$
5. tant que $f(u - \lambda \nabla f_u) \geq f(u)$ faire $\lambda = 0.5\lambda$
6. $u = u - \lambda \nabla f_u$
7. go to 2

Attention

u^* peut ne pas être un minimum local (a fortiori global)
La convergence peut être lente

La descente de gradient pour nous

u **correspond à** w **les poids**

f ?

La descente de gradient pour nous

u correspond à w les poids

f est la fonction de perte, elle doit être minimale quand on a atteint le comportement espéré

exemple :
$(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$

$$f(w) = \sum_{i=1}^{n} relu(1 - y_i DL(x_i, w))$$

La descente de gradient pour nous

u correspond à θ les poids

f est la fonction de perte, elle doit être minimale quand on a atteint le comportement espéré

exemple :
$(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$

$$f(\theta) = \sum_{i=1}^{n} relu(1 - y_i DL(x_i, \theta))$$

Est ce que c'est fini ?

Les limites 1/3

$$f(\theta) = \sum_{i=1}^{n} relu(1 - y_i DL(x_i, \theta))$$

si $n = 1000000$ ça veut dire qu'à chaque fois que je veux calculer ∇f_θ je dois faire 1000000 calculs - chacun avec plusieurs étages de plusieurs filtres ! et pour trouver λ n'en parlons pas !

Les limites 2/3

$DL(x_i, \theta)$ n'est pas vraiment dérivable
$\rightarrow$ pseudo dérivé $\rightarrow$ on pert la certitude de converger
est ce que c'est grave sur un problème déjà non convexe ?

Les limites 3/3

NIPS 2017 [a] : on peut mettre un temps exponentiel à sortir d'un point de selle
est ce que c'est grave vu qu'on va faire du early stopping ?

a. Gradient Descent Can Take Exponential Time to Escape Saddle Points

SGD

Les limites 2 et 3 sont marginales.

La SGD est une solution à la limite 1 et dans une moindre mesure à la limite 3 (NIPS 2017 [a] : la descente de gradient perturbée s'échappe des points de selle)

---

a. How to Escape Saddle Points Efficiently

La descente de gradient stochastique

f est une fonction dérivable de $\mathbb{R}^D$ dans $\mathbb{R}$
et que $f(u) = \sum_{i=1}^{n} q_i(u)$
alors, en faisant comme une descente de gradient mais en prenant 1 des q_i tiré aléatoirement à la place de f et avec un pas (λ) fixe sélectionné a priori
on converge en espérance vers le minimum dans le cas convexe

La descente de gradient stochastique pour nous

input : $x_1, y_1, ..., x_n, y_n, \theta_0$

1. $\theta = \theta_0$
2. $iter = 0$
3. tirer i au hasard dans 1,...,n
4. $f = relu(1 - y_i DL(x_i, \theta))$
5. calculer ∇f_θ
6. $\theta = \theta - \lambda_{iter} \nabla f_\theta$
7. $iter = iter + 1$
8. si condition d'arrêt alors sortir
9. go to 3

SGD

Mais, c'est pas vraiment de l'optimisation ! ? !

Mais, c'est pas vraiment de l'optimisation ! ? !

En apprentissage, l'optimisation est un outil pas une finalité !

Si la performance sur la base d'apprentissage est faible, il est probable que optimiser puisse apporter en test. C'est rarement le cas avec des réseaux de neurones !
Or, si la performance sur la base d'apprentissage est déjà forte, mieux optimiser conduit souvent à une moins bonne performance en test.

BILAN

classification supervisée de vecteurs par deep learning

base d'apprentissage $(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$
base de test $(x'_1, y'_1), ..., (x'_n, y'_n) \in \mathbb{R}^D \times \{-1, 1\}$

on choisit une structure de réseau $DL(x, \theta) = K_1$ filtres + relu + K_2 filtre + relu + ... + K_r filtre + relu + 1 filtre appliqué à x

on optimise θ par SGD avec l'objectif que $sign(DL(x_i, \theta)) \approx y_i$

dans l'espoir que $\forall i \in \{1, ..., n\}, sign(DL(x'_i, \theta)) \approx y'_i$

Voilà, vous savez faire du deep learning !

BILAN

Enfin, vous avez l'idée à gros traits...

En pratique,

il faut faire beaucoup pour comprendre un peu