INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Réseaux de neurones

Transposer comme un dieu, différentier comme une déesse: l'intelligence non artificielle des maths.

Poly de cours

3MIC semestre 7

Dr. Frédéric de Gournay

Publié par INSA de Toulouse

frederic@degournay.fr

Première édition, Janvier 2024.

Table des matières

1 Calculer des Adjoints 5
1.1 Définition, et quelques calculs simples 5
1.2 Exemples sous python 8

2 La programmation orientée objet 11
2.1 Les notions de base sur un programme 11

3 Approximation de fonctions 13
3.1 Position du problème 13
3.2 Premières approximations et discussion 16
3.3 Approximation par réseau de neurone 19

4 Optimisation d'un réseau de neurone 23
4.1 Définitions 23
4.2 Calcul théorique de la rétropropagation du gradient 24
4.3 Structure des fonctions 25
4.4 Calcul effectif de la rétropopagation du gradient 26

CHAPITRE 1

Calculer des Adjoints

1.1 Définition, et quelques calculs simples

L'objectif de ce chapitre est de remettre doucement au clair certaines propriétées de la transposée et clarifier les notions d'adjoint, de transposée et de trans-conjuguée. On se place dans le cadre des espaces vectoriels réels ou complexes, possiblement de dimension infinie.

Définition 1.1.1 Soit E et F deux espaces vectoriels munis chacuns d'un produit scalaire, soit $\mathcal{A}$ une application linéaire de E dans F, alors l'adjoint de $\mathcal{A}$, noté $\mathcal{A}^\star$ est l'unique application linéaire de F dans E telle que

$$\langle \mathcal{A}u, v\rangle_F = \langle u, \mathcal{A}^\star v\rangle_E \quad \forall u \in E, v \in F.$$

- Si on ajoute comme hypothèse que les espaces sont Hilbertiens (complets pour la norme du produit scalaire) et que l'opérateur $\mathcal{A}$ est continu alors l'adjoint existe toujours. Ainsi en dimension finie, les adjoints existent toujours.
- Sous réserve d'existence, on a $\mathcal{A}^{\star\star} = \mathcal{A}$.

Exercice 1.1

1. Soit $E = \mathbb{R}^n$, $F = \mathbb{R}^m$ avec les produits scalaires canoniques et $A \in \mathcal{M}_{mn}(\mathbb{R})$. Si on définit $\mathcal{A} : u \mapsto Au$, montrer que $\mathcal{A}^\star : v \mapsto A^T v$.
2. Soit $E = \mathbb{C}^n$, $F = \mathbb{C}^m$ avec les produits scalaires canoniques et $A \in \mathcal{M}_{mn}(\mathbb{C})$. Si on définit $\mathcal{A} : u \mapsto Au$ alors montrer que $\mathcal{A}^\star : v \mapsto \bar{A}^T v$.

Solution de l'Exercice 1.1

1. Soient $u \in E$ et $v \in F$.

$$\begin{aligned}\langle Au, v\rangle_F &= \sum_{j=1}^{m}(Au)_j v_j = \sum_{j=1}^{m}(\sum_{i=1}^{n} A_{ji}u_i)v_j = \sum_{i=1}^{n} u_i(\sum_{j=1}^{m} A_{ji}v_j) \\ &= \sum_{i=1}^{n} u_i(\sum_{j=1}^{m} A^T_{ij}v_j) = \sum_{i=1}^{n} u_i(A^T v)_i = \langle u, A^T v\rangle_E\end{aligned}$$

2. Il faut juste se souvenir que le produit scalaire canonique dans des espaces complexes est défini comme

$$\langle a, b\rangle_E = \sum_{i=1}^{n} \bar{a}_i b_i.$$

Cela assure que $\langle a, a\rangle = \sum_{i=1}^{n} |a_i|^2 > 0$ pour tout $a \neq 0$. En reprenant le même calcul que précédemment :

$$\begin{aligned}\langle Au, v\rangle_F &= \sum_{j=1}^{m}\overline{(Au)_j} v_j = \sum_{j=1}^{m}(\sum_{i=1}^{n} \bar{A}_{ji}\bar{u}_i)v_j = \sum_{i=1}^{n} \bar{u}_i(\sum_{j=1}^{m} \bar{A}_{ji}v_j) \\ &= \sum_{i=1}^{n} \bar{u}_i(\sum_{j=1}^{m} \bar{A}^T_{ij}v_j) = \sum_{i=1}^{n} \bar{u}_i(\bar{A}^T v)_i = \langle u, \bar{A}^T v\rangle_E\end{aligned}$$

L'exemple mentionné ci-dessus montre que le calcul de l'adjoint est essentiellement une opération de transposition (dans le cas des espaces vectoriels réels de dimension finie) et de transconjugaison (la conjugaison de la transposition) dans le cas des espaces vectoriels complexes. Si une matrice A a des coefficients réels, alors la transconjuguée de A est égale à sa transposée. Ainsi, la définition correcte de l'adjoint est celle de la transposition conjuguée, qui se réduit à la banale transposition dans le cas des espaces réels.

Cependant, les espaces vectoriels complexes ne sont pas rencontrés couramment, sauf dans le contexte de la transformation de Fourier. Par conséquent, il est ainsi usuel de connaître les théorèmes dans le cadre des espaces vectoriels réels et de savoir comment les étendre aux espaces complexes, plutôt que de connaître les théorèmes généraux dans le cadre complexe et de savoir les appliquer aux cadre réel.

Exercice 1.2

Soient a, b deux réels avec $a < b$ et E l'espace vectoriel des fonctions $C^\infty([a, b])$ qui s'annulent sur un voisinage de a et sur un voisinage de b.

1. Justifier rapidement que l'application $\langle \phi, \psi\rangle = \int_a^b \phi(x)\psi(x)dx$ est un produit scalaire sur E.
2. Montrer rapidement que $\mathcal{A} : \phi \mapsto \phi'$ est un endomorphisme de E.
3. Calculer $\mathcal{A}^\star$, montrer $\mathcal{A}^\star = -\mathcal{A}$.
4. On se donne $\mathcal{B} : \phi \mapsto -\phi''$, justifier que $\mathcal{B} = \mathcal{A}^\star\mathcal{A}$. Montrer que $\mathcal{B} = \mathcal{B}^\star$ (l'endomorphisme est dit auto-adjoint). Montrer que pour tout ϕ, $\langle \mathcal{B}\phi, \phi\rangle \geq 0$.

Solution de l'Exercice 1.2

1. On montre que l'application est bien caculable pour tout ϕ et ψ dans E, qu'elle est à valeurs réelles, qu'elle est symétrique, qu'elle est linéaire par rapport à une de ses variables. Ensuite on montre que $\langle \phi, \phi \rangle \geq 0$ pour tout $\phi \in E$. Finalement on montre que si $\langle \phi, \phi \rangle = 0$ alors nécessairement $\phi = 0$.
2. Comme ϕ est dans E, alors ϕ' est bien défini et appartient à E. De plus $\mathcal{A}$ est linéaire.
3. Pour tout ϕ et ψ dans E, on a

$$\begin{aligned} \langle \mathcal{A}\phi, \psi \rangle &= \int_a^b \phi'(x)\psi(x)dx = \underbrace{[\phi(x)\psi(x)]_{x=a}^b}_{=0} - \int_a^b \phi(x)\psi'(x)dx \\ &= -\langle \phi, \mathcal{A}\psi \rangle \end{aligned}$$

   Ainsi $\mathcal{A} = -\mathcal{A}^\star$.
4. on a $\mathcal{B}\phi = -(\phi')' = -(\mathcal{A}\phi)' = \mathcal{A}^\star\mathcal{A}\phi$. Pour tout ϕ, ψ, on a

$$\langle \mathcal{B}\phi, \psi \rangle = \langle \mathcal{A}^\star\mathcal{A}\phi, \psi \rangle = \langle \mathcal{A}\phi, \mathcal{A}\psi \rangle = \langle \phi, \mathcal{A}^\star\mathcal{A}\psi \rangle$$

   Ainsi $\mathcal{B}^\star = \mathcal{B}$. Finalement, pour tout ϕ, on a $\langle \mathcal{B}\phi, \phi \rangle = \langle \mathcal{A}\phi, \mathcal{A}\phi \rangle \geq 0$.

En fait l'application $\langle \mathcal{B}\bullet, \bullet \rangle$ est un produit scalaire sur E.

Proposition 1.1.1 Si E et F sont deux espaces de Hilbert, on note $\mathcal{L}_c(E \to F)$ l'ensemble des fonctions linéaires continues de E dans F.

1. L'application

$$\begin{aligned} \mathcal{L}_c(E \to F) &\to \mathcal{L}_c(F \to E) \\ \mathcal{A} &\mapsto \mathcal{A}^\star \end{aligned}$$

   est une application linéaire (c'est même une involution).
2. $(\mathcal{A}\mathcal{B})^\star = \mathcal{B}^\star\mathcal{A}^\star$
3. Si $J \subset [\![1, m]\!]$ et $I \subset [\![1, n]\!]$ sont des ensembles d'indices sans répétition de même cardinal p. Si $\mathcal{A} \in \mathcal{L}_c(\mathbb{R}^n \to \mathbb{R}^m)$ est donné par $(\mathcal{A}u)[J] = u[I]$ et $(\mathcal{A}u)[J^c] = 0$ alors $\mathcal{A}^\star$ est donné par $(\mathcal{A}^\star v)[I] = v[J]$ et $(\mathcal{A}^\star v)[I^c] = 0$.

Démonstration

Seul le troisième point mérite une démonstration, on se donne $u \in \mathbb{R}^n$ et $v \in \mathbb{R}^m$. On note B l'opérateur $(\mathcal{B}v)[I] = v[J]$ et $(\mathcal{B}v)[I^c] = 0$.

$$\begin{aligned} \langle u, v \rangle &= \sum_{j=1}^m (\mathcal{A}u)[j]v[j] = \sum_{s=1}^p (\mathcal{A}u)[J[s]]v[J[s]] = \sum_{s=1}^p u[I[s]]v[J[s]] \\ &= \sum_{s=1}^p u[I[s]](\mathcal{B}v)[I[s]] = \sum_{i=1}^n u[i](\mathcal{B}v)[i] = \langle u, \mathcal{B}v \rangle \end{aligned}$$

Ainsi $\mathcal{B} = \mathcal{A}^\star$.

1.2 Exemples sous python

On va mettre en lumière les calculs d'adjoint en utilisant python. On implémentera le produit scalaire par :

```
def scal(a,b) :
  assert a.shape==b.shape
  return np.sum(a*b)
```

1.2.1 Applications de la Proposition 1.1.1

On se donne maintenant un code python qui met en lumière le 3ème point de la Proposition 1.1.1.

```
n,m=6,12
I,J=[4,2,3],[11,0,5]
def A(u) :
        v=np.zeros(m)
        v[J]=u[I]
        return v
def AT(v) :
        u=np.zeros(n)
        u[I]=v[J]
        return u
np.random.seed(42)
u=np.random.randn(n)
v=np.random.randn(m)
print(scal(A(u),v),scal(u,AT(v))) # 0.23993564669581763 0.23993564669581763
```

Il faut aussi comprendre pourquoi le code suivant donne le résultat

```
I,J=[4,2,3],[11,5,5]
print(scal(A(u),v),scal(u,AT(v))) # -0.7829023931300434 -1.0845502163699587
I,J=[4,2,4],[11,0,5]
print(scal(A(u),v),scal(u,AT(v))) # 1.0583081599084143 1.1318902333334042
```

Pour mieux comprendre les problèmes d'affectation, on peut définir l'opérateur B et son adjoint comme étant

```
def B(u) :
    v=np.zeros(m)
    for (i,j) in zip(I,J) :
        v[j]+=u[i]
    return v
def BT(v) :
    u=np.zeros(n)
    for (i,j) in zip(I,J) :
        u[i]+=v[j]
    return u
I,J=[4,2,3],[11,0,5]
print(scal(B(u),v),scal(u,BT(v))) # 0.23993564669581763 0.23993564669581763
I,J=[4,2,3],[11,5,5]
```

```
print(scal(B(u),v),scal(u,BT(v))) # -1.0845502163699587 -1.0845502163699587
I,J=[4,2,4],[11,0,5]
print(scal(B(u),v),scal(u,BT(v))) # 1.0583081599084143 1.0583081599084143
```

Il se trouve que l'opérateur `BT` ici défini est toujours l'adjoint de `B`. Les opérateurs `A` et `B` sont les mêmes si et seulement si J ne contient pas de répétition. De même les opérateurs `AT` et les opérateurs `BT` sont les même si et seulement si I ne contient pas de répétitions.

1.2.2 La dérivation discrète

Dans l'exemple ci-dessous, on utilise aussi la linéarité de l'adjonction pour calculer l'adjoint de l'opérateur de dérivation discrète.

```
def dx(im) :
  n,m=im.shape
  d=np.zeros((n-1,m))
  d+=im[1:,:]
  d+=-im[:-1,:]
  return d
def dxT(d) :
  n,m=d.shape
  im=np.zeros((n+1,m))
  im[:-1,:]+=-d
  im[1:,:]+=d
  return im
np.random.seed(42)
im=np.random.randn(15,23)
d=np.random.randn(14,23)
print(scal(dx(im),d),scal(im,dxT(d))) # -26.30768779905194 -26.30768779905194
```

CHAPITRE 2

La programmation orientée objet

L'objectif ici n'est clairement pas de faire un cours de programmation orienté objet sous Python, il faut cependant connaître les différentes notions d'une classe, on s'intéresse ici aux notions

1. Notion de classe et d'instance de classe.
2. Le constructeur de la classe
3. Les attributs et les méthodes de classe.
4. La syntaxe des classes sur python, l'usage du mot `self` sur python. L'accès aux attributs et méthodes de classes par l'usage du .

On ne s'intéressera pas aux notions suivantes qui sont pourtant nécessaires si vous voulez prétendre savoir faire de la POO.

1. L'héritage de classe, le polymorphisme, le mot clef `super` .
2. Le caractère public/protégé/privé des attributs et l'encapsulation des méthodes.
3. Les accesseurs (get) et mutateurs (set).

2.1 Les notions de base sur un programme

Dans ce chapitre, nous discuterons oralement ensemble d'un petit programme qui illustre les notions nécessaires.

```
1  class Voiture():
2      def __init__(self, Puissance, Vitesse_max ):
3          self.p = Puissance ## Puissance d'accélération de la voiture
4          self.vm = Vitesse_max ## Vitesse max de la voiture avant qu'elle n'explose
5          self.v = 0. ## Vitesse initiale
6      def accelere(self):
7          self.v = self.v+self.p
8          if self.v > self.vm :
9              print("Tu es allé trop vite :",self.v)
10             self.v = 0
11 Ferrari=Voiture(100,300)
12 Megane=Voiture(10,180)
13 print(Ferrari.v,Megane.v)# 0.0,0.0
```

```
14 Ferrari.accelere()
15 print(Ferrari.v,Megane.v)# 100.0,0.0
16 Megane.accelere()
17 print(Ferrari.v,Megane.v)# 100.0,10.0
18 Ferrari.accelere()
19 print(Ferrari.v,Megane.v)# 200.0 10.0
20 Ferrari.p=400
21 Ferrari.accelere() # Tu es allé trop vite : 600.0
22 print(Ferrari.v,Megane.v) #0 10.0
```

CHAPITRE 3

Approximation de fonctions

L'objectif de ce chapitre est de donner un cadre global dans lequel les réseaux de neurones, qui sont une des solutions de choix de l'intelligence artificielle apparaissent relativement naturellement.

3.1 Position du problème

Nous nous intéresserons à un problème où nous avons des variables d'entrées I (en anglais **input data**) et des variables de sorties O (en anglais **output data**). Nous supposons qu'il existe une fonction qui associe les données d'entrées aux données de sortie, l'objectif est de retrouver cette fonction f.

Définition 3.1.1 — Formulation mathématique.

1. On se donne n couples de variables $(I[\ell], O[\ell])_{\ell=1\dots n}$ avec pour tout ℓ, $I[\ell] \in \mathbb{R}^p$ et $O[\ell] \in \mathbb{R}^q$. L'objectif est de trouver une fonction $f : \mathbb{R}^p \mapsto \mathbb{R}^q$ tel que $O \simeq f(I)$ c'est-à-dire que pour tout ℓ, on a :
$$O[\ell] \simeq f(I[\ell]) \quad \forall \ell \in \{1, \dots, n\}$$
2. On suppose que l'ensemble des fonctions admissibles est un ensemble de fonctions paramétrées, c'est à dire qu'il existe $\mathcal{X}$ un sous ensemble d'un espace vectoriel tel que pour tout $\Theta \in \mathcal{X}$ dans cet espace vectoriel on a à disposition f_Θ, une fonction de $\mathbb{R}^p \mapsto \mathbb{R}^q$. L'objectif devient de trouver $\Theta \in \mathcal{X}$ tel que $O \simeq f_\Theta(I)$.
3. Pour donner un sens au symbole $\simeq$, on se donne une fonction **perte** $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$, c'est une fonction qui vérifie
$$\begin{cases} d(O_1, O_2) \geq 0 \\ d(O_1, O_2) = 0 \iff O_1 = O_2 \end{cases}$$
4. Le problème devient
$$\min_{\Theta \in \mathcal{X}} \frac{1}{n} \sum_{\ell=1}^{n} d(f_\Theta(I[\ell]), O[\ell])$$

Les couples (I, O) s'appellent les **données d'entrée/sortie**.

Définition 3.1.2 — Version stochastique. Le problème au dessus peut être reformulé dans le cadre légèrement plus général de la théorie des probabilités. Pour cela il faut supposer que l'on se donne deux variables aléatoires I et O à valeurs respectivement dans $\mathbb{R}^p$ et $\mathbb{R}^q$. L'objectif est de minimiser

$$\min_{\Theta\in\mathcal{X}} \mathbb{E}\left(d(f_\Theta(I), O)\right).$$

Définition 3.1.3 — Exemple : l'espérance conditionnelle. On se place dans le cadre stochastique, dans le cas où I et O sont de carré intégrable, $f_\Theta = \Theta$ et $\mathcal{X}$ est l'ensemble des fonctions mesurables et d est la norme L^2 c'est-à-dire $d(O_1, O_2) = \frac{1}{2}\|O_1 - O_2\|_2^2$, alors le problème devient

$$\inf_{\Theta} \mathbb{E}\left(\|\Theta(I) - O\|^2\right).$$

La solution de ce problème est donnée par $\Theta(I) = \mathbb{E}(O|I)$.

Définition 3.1.4 — Exemple : les moindres carrés linéaire. On suppose que les fonctions f_Θ sont affines, c'est à dire que $X = \mathcal{M}_{qp}(\mathbb{R}) \times \mathbb{R}^q$ et que pour chaque $\Theta = (A, b)$ on a $f_\Theta(x) = Ax + b$. De plus on prend comme fonction perte la norme L^2. Dans ce cas le problème devient

$$\min_{(A,b)\in X} \frac{1}{n}\sum_{\ell=1}^{n}\sum_{j=1}^{q}\left(\sum_{i=1}^{p} A_{ji}I_i[\ell] + b_j - O_j[\ell]\right)^2.$$

Il est facile de s'apercevoir que le problème de dessus est quadratique par rapport aux variables A et b, ainsi ce sont des problèmes de moindre carrés linéaires. Vous avez déjà vu des problèmes de moindre carrés linéaires, ils ont une nette tendance à admettre une unique solution qui est aisément calculable. On va redonner le résultat dans le théorème suivant

Théorème 3.1.1 Soit $X = \mathcal{M}_{qp}(\mathbb{R}) \times \mathbb{R}^q$ et soit le problème

$$\min_{(A,b)\in X} \frac{1}{n}\sum_{\ell=1}^{n}\sum_{j=1}^{q}\left(\sum_{i=1}^{p} A_{ji}I_i[\ell] + b_j - O_j[\ell]\right)^2. \tag{3.1}$$

Définissons $\mathcal{I} \in \mathcal{M}_{n,p+1}(\mathbb{R})$ par

$$\mathcal{I} = \begin{pmatrix} 1 & I_1[1] & \dots & I_i[1] & \dots & I_p[1] \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots \\ 1 & I_1[\ell] & \dots & I_i[\ell] & \dots & I_p[\ell] \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots \\ 1 & I_1[n] & \dots & I_i[n] & \dots & I_p[n] \end{pmatrix}$$

On suppose que $\mathcal{I}$ est injective (ce qui est très probable si n est grand). Dans ce cas il existe un seul (A, b) qui minimise (3.1). Pour calculer ce couple (A, b) il faut, pour

tout $j = 1, \dots, q$ définir $\theta_j \in \mathbb{R}^{p+1}$ comme

$$\theta_j = \begin{pmatrix} b_j \\ A_{j1} \\ \vdots \\ A_{jp} \end{pmatrix},$$

Alors le couple optimal (A, b) vérifie les **équations normales**:

$$\mathcal{I}^T \mathcal{I} \theta_j = \mathcal{I}^T Y_j, \quad \forall j = 1, \dots, q$$

Démonstration

On rappelle le problème d'optimisation

$$\min_{A \in \mathcal{M}_{qp}(\mathbb{R}), b \in \mathbb{R}^q} \sum_{\ell=1}^{n} \sum_{j=1}^{q} (\sum_{i=1}^{p} A_{ji} I_i[\ell] + b_j - O_j[\ell])^2.$$

Tout d'abord notons $(A_j)_{1 \le j \le q}$ les vecteurs colonnes de A, le problème original (3.1) se sépare en q problèmes indépendants :

$$\min_{A_j \in \mathbb{R}^p, b_j \in \mathbb{R}} \sum_{\ell=1}^{n} (\sum_{i=1}^{p} A_{ji} X_i[\ell] + b_j - Y_j[\ell])^2.$$

On remarque que $(\mathcal{I}\theta_j)[\ell] = \sum_{i=1}^{p} A_{ji} I_i[\ell] + b_j$, ainsi le j^{eme} problème est encore un problème classique d'optimisation quadratique

$$\min_{\theta_j \in \mathbb{R}^{p+1}} \frac{1}{2} \|\mathcal{I}\theta_j - O_j\|^2 = \min_{\theta_j \in \mathbb{R}^{p+1}} \frac{1}{2} (\mathcal{I}\theta_j, \mathcal{I}\theta_j) - (\mathcal{I}\theta_j, O_j) + \frac{1}{2} (O_j, O_j)$$

On peut le mettre sous la forme :

$$\min_{\theta_j \in \mathbb{R}^{p+1}} \frac{1}{2} (\mathcal{I}^\star \mathcal{I} \theta_j, \theta_j) - (\theta_j, \mathcal{I}^\star O_j).$$

On rappelle que ce problème admet une unique solution donnée par $\mathcal{I}^T \mathcal{I} \theta_j = \mathcal{I}^\star O_j$, si la matrice $C = \mathcal{I}^\star \mathcal{I}$ est symétrique définie positive. La matrice C est symétrique , elle est toujours positive car pour tout θ, on a

$$(C\theta, \theta) = (\mathcal{I}^\star \mathcal{I} \theta, \theta) = \|\mathcal{I}\theta\|^2 \ge 0.$$

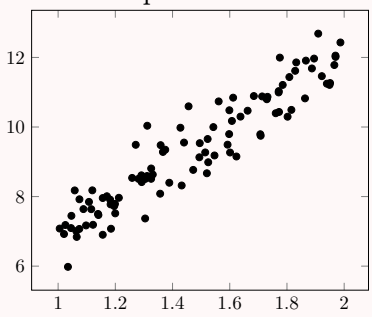
Pour finir, la matrice C est définie car si $(C\theta, \theta) = 0$ alors $\mathcal{I}\theta = 0$ and comme $\mathcal{I}$ est injective alors $\theta = 0$.

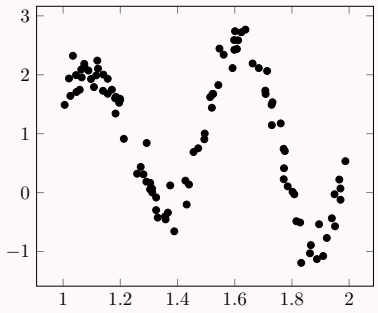
3.1.1 Exemples de jeux de données

Afin d'introduire les réseaux de neurones, nous discuterons de jeux de données différents

Exercice 3.1: Exemple unidimensionnel

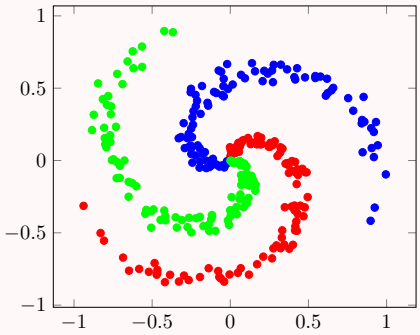
Pour les deux premiers jeux de données, nous aurons $n = 100$ données et $p = q = 1$, comme les entrées et les sorties sont des réels, on peut représenter les entrées en abscisse et les sorties en ordonnées. Il suffit ensuite de trouver une fonction qui approxime au mieux le nuage de points $(I[\ell], O[\ell])_\ell$. Les deux premiers jeux de données sont représentés ci-dessous





Exercice 3.2: Classification supervisée

Ici on suppose que l'on a $n = 100$ données et $p = 2$. Chaque donnée d'entrée $I[\ell] \in \mathbb{R}^2$ appartient à une certaine classe (ici "rouge", "vert" ou "bleu"). On représente les différentes classes en colorant les données d'entrées par une couleur différente. Afin d'encoder les couleurs dans un espace vectoriel, les données de sorties $O[\ell]$ seront un vecteur de $\mathbb{R}^3$. Par convention $O[\ell] \in \left\{ \begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix} \right\}$ ne peut valoir qu'un des trois vecteurs de base de $\mathbb{R}^3$. Chacun de ces vecteurs est associé à une classe (ici rouge, vert, bleu).



3.2 Premières approximations et discussion

3.2.1 Les moindres carrés linéaires

Nous commençons par appliquer les moindres carrés linéaires aux problèmes unidimensionnels de l'exercice 3.1, les résultats sont montrés dans la Figure 3.1. Dans cette figure, on trace par une droite rouge la fonction $i \mapsto f_\Theta(i)$ qui est la meilleure approximation affine du jeu de données (I, O).

Le problème majeur est que la relation entre I et O n'est pas forcément affine, autant fonctionne bien dans le premier cas (à gauche dans la Figure 3.1), autant dans le deuxième cas (à droite dans la Figure 3.1) on pense bien qu'on ne peut pas approximer les couples

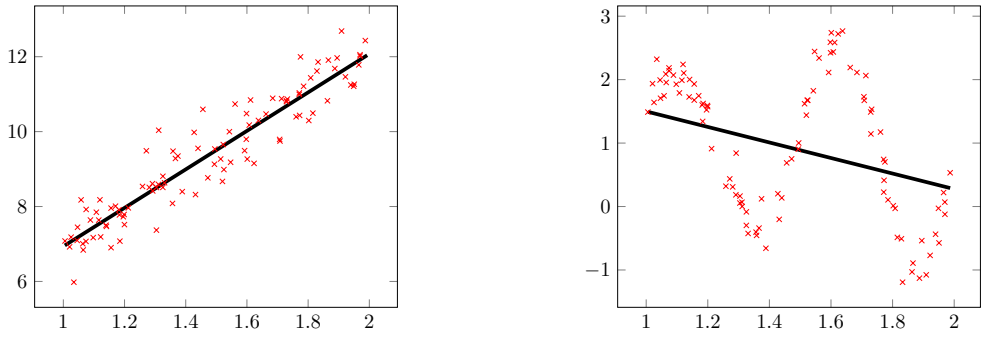


Figure 3.1: Moindres carrés linéaires, on essaie d'approximer un nuage de points par une droite.

entrée-sortie par des droites..

3.2.2 Assimilation de données

Une des solutions pour le deuxième jeu de donnée est de demander d'où il vient, de quelle expérience physique, etc... On supposera ici qu'il y a un modèle sous-jacent et que les ingénieurs pensent que l'on devrait avoir

$$y = f_\Theta(x) \text{ avec } f_\Theta : x \mapsto \Theta_1 x \cos(\Theta_2 + 12x) + \Theta_3.$$

Ici le vecteur $\Theta = (\Theta_1, \Theta_2, \Theta_3)$ représente des coefficients physiques qui sont inconnus par les ingénieurs qui nous ont donné ce problème. Leur objectif est donc de retrouver ces coefficients depuis le jeu de données. Ce processus s'appelle **l'assimilation de données**. On ne change pas la fonction coût et on résout le problème

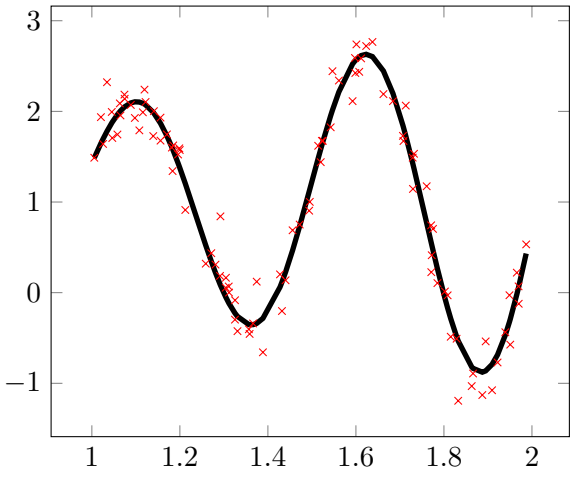


Figure 3.2: Apprentissage par assimilation de données. On retrouve des paramètres d'un modèle physique pré-déterminé.

$$\min_\Theta d(f_\Theta(X), Y) = \min_\Theta \sum_\ell \frac{1}{2}(f_\Theta(I[\ell]) - O[\ell])^2 = \min_\Theta \frac{1}{2}\|F(\Theta)\|^2.$$

Pour faire cela, on applique une méthode de Gauss-Newton avec

$$F(\Theta) = (f_\Theta(I[\ell]) - O[\ell])_{1 \leq \ell \leq n}$$

On rappelle que la méthode de Gauss-Newton consiste à calculer les Θ^k suivants :

$$\Theta^{k+1} = \Theta^k - d \text{ avec } J^\star J d = J^\star F(\Theta),$$

où J est la matrice $n \times 3$ qui est la matrice Jacobienne de F au point Θ et dont la ligne numéro ℓ est donnée par

$$J_{\ell,\bullet} = (I[\ell]\cos(\Theta_2 + 12I[\ell]) + \Theta_3, -\Theta_1 I[\ell]\sin(\Theta_2 + 12I[\ell]) + \Theta_3, 1)$$

Le résultat de l'algorithme de Gauss-Newton est donné dans la Figure 3.2. Ici on obtient de bon résultats car on est parti d'une bonne modélisation physique du problème considéré.

3.2.3 Formatage des données

On s'intéresse maintenant à l'approximation des moindres carrés aux données de l'exercice 3.2 (la spirale). On est bien embêté car on ne sait pas trop quoi faire d'un vecteur de $\mathbb{R}^3$. Effectivement on a une interprétation claire pour les données si elles sont un des vecteurs de la base canonique mais la méthode des moindres carrés nous rend $AI + b$ qui est n'importe quel vecteur de $\mathbb{R}^3$.

Pour résoudre ce problème, on pose toujours $\Theta = (A, b)$ et au lieu de poser $f_\Theta(I) = AI+b$ on va effectuer une transformation supplémentaire $g : \mathbb{R}^3 \mapsto \mathbb{R}^3$ sur $AI + b$ et on posera $f_\Theta(I) = g(AI + b)$. Cette transformation g est donnée par :

$$g(z) = \frac{1}{e^{z_1} + e^{z_2} + e^{z_3}} \begin{pmatrix} e^{z_1} \\ e^{z_2} \\ e^{z_3} \end{pmatrix} = \frac{e^z}{(e^z, \mathbf{1})}$$

Ainsi si z est un vecteur de $\mathbb{R}^3$, $g(z)$ est un vecteur de $\mathbb{R}^3$ dont toutes les coordonnées sont positives et dont la somme des coordonnées vaut 1. Ainsi si $g(z) = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$, on dira que le vecteur a la probabilité a de valoir $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, la probabilité b de valoir $\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ et la probabilité c de valoir $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$. De plus on utilise une fonction perte spéciale dans ce cas qui est :

$$d(g(z), Y) = -(Y, \log(g(z))) = \log(\sum_{i=1}^{3} e^{z_i}) - (z, Y) = \log((e^z, \mathbf{1})) - (z, Y)$$

Ce qui nous permet d'obtenir l'expression

$$\nabla_z d(g(z), Y) = g(z) - Y$$

Ainsi on résout le problème suivant

$$\min_{A,b} d(g(AI + b)), Y)$$

avec une méthode de gradient sur A et b. On obtient le résultat en Figure 3.3 (à gauche). On représente non seulement les données mais aussi la valeur de $g(AX + b)$ pour différents points de l'espace. On voit une séparation de l'espace en trois parties. Cette séparation se fait avec des droites. Sur la Figure 3.3 à droite, on représente aussi le résultat avec un réseau de neurones profond.

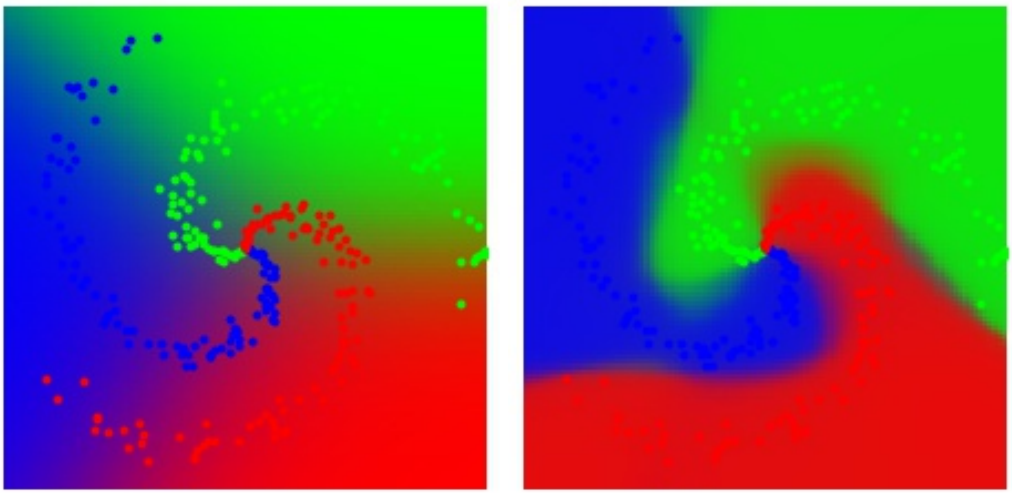


Figure 3.3: Modèle linéaire avec formatage des données (à gauche) vs réseau de neurones profond (à droite)

3.3 Approximation par réseau de neurone

Afin de comprendre un peu mieux ce que l'on a appris de la section précédente on va faire un schéma explicatif des trois modèles que l'on a vu précédemment. Cette représentation est faite dans la Figure 3.4.

On y représente les trois modèles de la Section 3.2, par convention x_s représente les données au fil du temps, on commence avec $x_0 = I$ et au bout d'un certain temps, il existe un S ($S = 1$ dans les deux premiers cas et $S = 2$ dans le deuxième) tel que x_S vaut l'approximation de O par le programme. Les données x_S passent par la couche de perte, elles sont comparées avec O et on calcule x_{S+1} qui est un réel qui est la qualité de l'approximation de O par x_S. L'objectif est de minimiser x_{S+1} par rapport à Θ, les paramètres qui interviennent dans les couches intermédiaires (les boîtes grises). On retient de cette représentation les différentes conventions

- On note $(x_s)_{s=0,..,S+1}$ les données au cours du temps. Par convention, $x_0 = I$, $x_S \simeq O$ et $x_{S+1} \in \mathbb{R}$ représente à quel point x_S est proche de O. La variable x_{S+1} est notée dans un cercle rouge.
- On note par des boîtes grises des transformations élémentaires qui modifient les données.
- On note par une boîte bleue la fonction perte. La boîte bleue transforme toujours x_S en x_{S+1}.

En se basant sur cette convention il est facile de se dire que l'on pourrait étendre le idées de la section précédente en considérant n'importe quel programme qui ait la forme générale de la Figure 3.5 (gauche), on appelle de tels programmes des réseaux de neurones. Dans cette figure on représente des programmes, où les boîtes grises seront remplacées par des fonctions du genre $y = \mathcal{F}(x, \theta)$, où θ sont des paramètres à ajuster. On appelle ces boîtes grises des `modules` et ils sont censés être choisis par le constructeur du réseau de neurone pour avoir une certaine fonctionnalité (module linéaire, convolutionnel, d'activation, d'attention, transformeur, etc...). La fonction perte est un module mais est dessinnée avec une boîte bleue. Dans la figure de gauche, le programme dessine un graphe entre les modules, ce graphe est appelé **graphe computationnel**. Dans la figure de droite, le

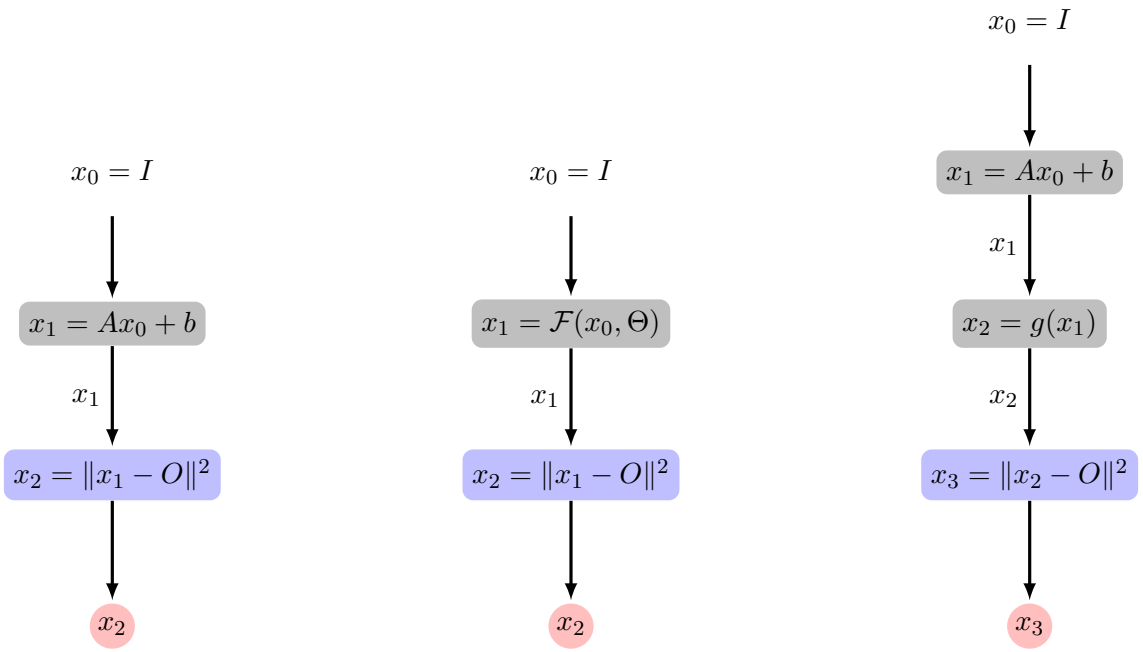


Figure 3.4: Représentation schématique des programmes de moindre carrés linéaire (gauche), d'assimilation de données (milieu) et de formatage des données (droite). Les schémas sont de profondeur S différente ($S = 1$ dans les deux premiers cas et $S = 2$ dans le dernier). La variable x_S est l'approximation de O par l'algorithme et la variable x_{S+1} est un réel qui est la qualité de l'approximation de O par x_S. Si dans les deux premiers cas, on pose $\Theta = (A, b)$, l'objectif est de minimiser x_{S+1} par rapport à Θ.

graphe computationnel est beaucoup plus simple, on dit qu'il est **séquentiel**. Un théorème de la théorie des graphes nous dit que tout graphe acyclique orienté peut toujours se mettre sous la forme d'un graphe séquentiel, ainsi on ne considérera pour la théorie que des réseaux de neurones de la forme de celui de droite. Pour comprendre comment on passe d'un graphe computationnel quelconque à un graphe séquentiel, il suffit de penser que x_s représente la copie de la mémoire de l'ordinateur au cours du temps, elle évolue de manière séquentielle.

3.3.1 Type de transformations élémentaires

Une transformation élémentaire est une fonction $y = \mathcal{F}(x, \theta)$ qui transforme les données x en des données y et qui dépendent de paramètres θ qui seront appris par le réseau.

- Si l'application $x \mapsto \mathcal{F}(x, \theta)$ est affine, alors la couche est dite **linéaire** (c'est étrange mais c'est comme cela). Quitte à écrire les vecteurs x et y dans une base, il existe donc une matrice A et un vecteur b tels que $\mathcal{F}(x, \theta) = Ax + b$. Le vecteur b est appelé le **biais**. L'immense majorité des couches à paramètre sont des couches linéaires.
- On a déjà vu la couche **Softmax** donnée par $y_i = \dfrac{e^{x_i}}{\sum_i e^{x_i}}$. Cette couche sert à transformer un vecteur quelconque en vecteur dont les coordonnées sont positives et qui somment à 1. La couche **Softmax** est une version régulière de la fonction argmax d'un vecteur. L'interprétation de la sortie d'une couche Softmax est que y représente un vecteur de probabilités.
- Il est assez inutile de composer les couches linéaires, entre elles car la composition

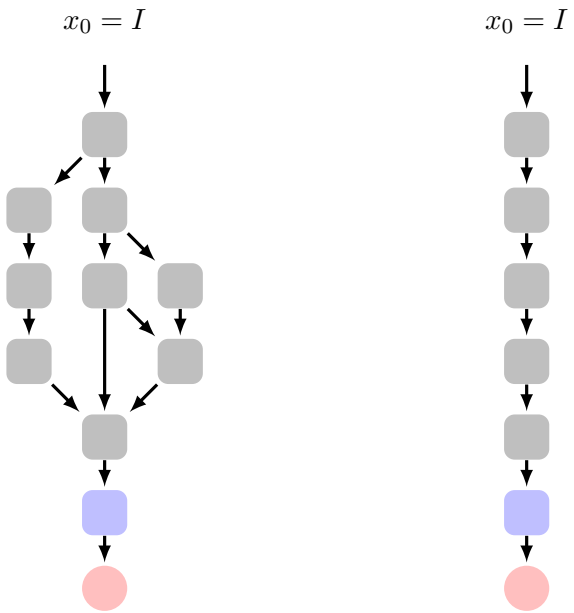


Figure 3.5: Représentation schématique de réseaux de neurones, à gauche un réseau de neurone avec son graphe computationnel. Chaque boîte grise est une couche, la boîte bleue est la fonction perte et on optimise la sortie en rouge par rapport aux paramètres du réseau. A droite, un réseau séquentiel.

de deux fonctions linéaires est une fonction linéaire. Ainsi il est d'usage d'interposer entre deux couches linéaires une couche dite d'**activation**. Pour définir une couche d'activation, il suffit de se donner une fonction ϕ de $\mathbb{R}$ dans $\mathbb{R}$ et de l'appliquer coordonnées par coordonnées. Ainsi $y_i = \phi(x_i)$ pour tout i

On récapitule ces définitions dans les tableaux ci-dessous.

Définition 3.3.1 — Quelques transformations élémentaires.

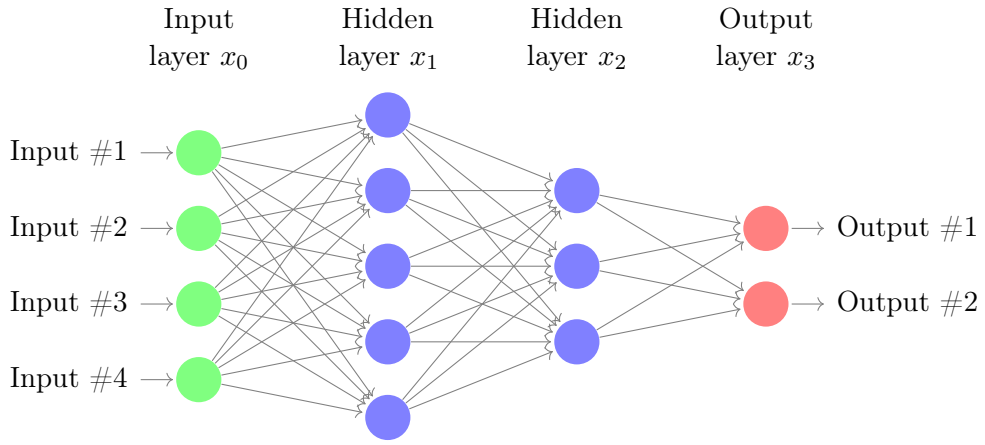
Nom	Formule	Paramètres θ
Dense (Linéaire)	$y = Ax + b$	A,b
Softmax	$y_i = \frac{e^{x_i}}{\sum_i e^{x_i}}$	Aucun
Activation	$y_i = \phi(x_i)$	Aucun

Définition 3.3.2 — Différentes fonctions d'activation.

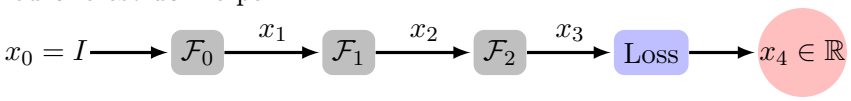
Nom	Formule	avantagee
RELU	$\phi(x) = \max(x, 0)$	facile à calculer ainsi que sa dérivée
Sigmoid	$\phi(x) = \arctan(x)$	différentiable et à valeurs bornées
Heaviside	$\phi(x) = 1_{x \geq 0}$	facile à calculer et bornée (mais dérivée nulle).

3.3.2 Représentation usuelle

Vous avez déjà vu des représentations de réseaux de neurones, elle ressemblent à ce genre de choses :



Cela représente un réseau de neurone où pour $s = 0, 1, 2$, les couches $x_{s+1} = \mathcal{F}(x_s, \theta) = \phi(Ax_s + b)$ sont des couches linéaires suivies d'une fonction d'activation ϕ. Les flèches qui lient x_s à x_{s+1} repésentent les coefficients de la matrice A. Le biais b et la fonction d'activation et la fonction perter ne sont pas montrés. Dans notre représentation, ce réseau de neurone est donné par :

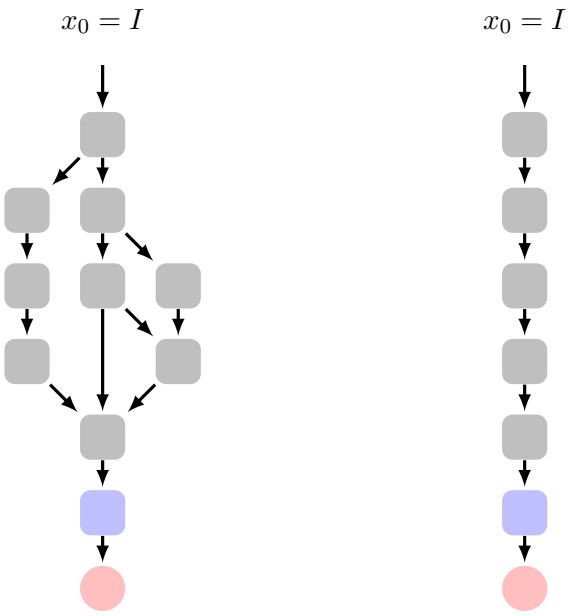


CHAPITRE 4

Optimisation d'un réseau de neurone

4.1 Définitions

On rappelle le schéma récapitulatif de réseaux de neurones



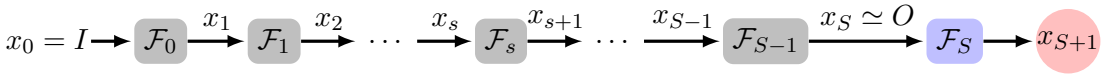
Sachant que chaque couche a potentiellement des paramètres, l'objectif est de dériver la sortie du réseau par rapport aux paramètres qui interviennent dans le réseau. Le graphe de calcul peut être atrocement compliqué et on a l'impression de devoir différentier un monstre. Heureusement, il y a une méthode pour le faire de manière automatique, cela s'appelle la **rétropropagation du gradient**. Nous allons dans ce chapitre démontrer la formule de rétropropagation et montrer comment elle s'applique. Il se trouve qu'un réseau de neurone quelconque (de la forme de gauche) se met sous la forme séquentielle (à droite). Donc nous allons nous restreindre à l'étude des réseaux qui ressemblent à ceux de droite.

Définition 4.1.1 — Mode forward. Un réseau de neurone est la donnée de $S+1$ fonctions (couches) $\mathcal{F}_s : \mathcal{H}_s \times \mathcal{G}_s \to \mathcal{H}_{s+1}$ où les espaces $\mathcal{H}_s$ et $\mathcal{G}_s$ sont des espaces de Hilbert. On supposera que l'on a toujours $\mathcal{H}_{S+1} = \mathbb{R}$. Etant donné $\Theta = (\theta_s)_{s=0,\dots,S}$ où $\theta_s \in \mathcal{G}_s$ et étant donné $x_0 \in \mathcal{H}_0$, le calcul direct (forward) du réseau de neurone est le calcul de $X(\Theta) = (x_s(\Theta))_{s=0,\dots S+1}$ de la récurrence :

$$x_{s+1}(\Theta) = \mathcal{F}_s(x_s(\Theta), \theta_s) \quad \text{pour tout } s = 0, \dots, S$$

L'ensemble des variables Θ sont les paramètres du réseau de neurone, $x_0 = I$ sont les données d'entrées, x_S est l'approximation des données de sortie et $x_{S+1} = d(x_S, O) \in \mathbb{R}$ est la qualité de l'approximation. Ainsi la dernière couche $\mathcal{F}_S$ est la couche de perte.

On donne ci-dessous un petit schéma explicatif du réseau de neurone et de son mode forward :



On n'écrira pas la dépendance de x_s en fonction de Θ et on écrira donc souvent x_s en lieu de $x_s(\Theta)$. L'objectif est de calculer le gradient de $\Theta \mapsto x_{S+1}(\Theta)$. Avant de calculer le gradient, nous montrons qu'il existe une dérivée directionnelle.

4.2 Calcul théorique de la rétropropagation du gradient

4.2.1 Existence de la dérivée directionnelle

Avant de calculer le gradient, nous montrons qu'il existe une dérivée directionnelle.

Théorème 4.2.1 — Mode tangent. Soit $\Theta = (\theta_s)_s$ un jeu de paramètres et $X(\Theta) = (x_s)_s$ la solution du forward associé. On suppose que chaque $\mathcal{F}_s$ est une fonction C^1. On note $\partial_\theta \mathcal{F}_s$ la matrice Jacobienne de $\mathcal{F}_s(x_s, \bullet)$ au point θ_s et $\partial_x \mathcal{F}_s$ la matrice Jacobienne de $\mathcal{F}_s(\bullet, \theta_s)$ au point x_s. Pour toute direction $\dot\Theta = (\dot\theta_s)_s$ On note $\dot X = (\dot X_p)_{0 \le p \le s+1}$ le vecteur défini par

$$\begin{cases} \dot x_0 & = 0 \\ \dot x_{s+1} & = (\partial_\theta \mathcal{F}_s)\, \dot\theta_s + (\partial_x \mathcal{F}_s)\, \dot x_s \quad \forall 0 \le s \le S \end{cases}$$

Alors

$$X(\Theta + \dot\Theta) = X(\Theta) + \dot X + o(\|\dot\Theta\|)$$

Démonstration

La démonstration du Théorème 4.2.1 se fait par récurence sur s, on suppose que

$$x_s(\Theta + \dot\Theta) = x_s(\Theta) + \dot x_s + o(\|\dot\Theta\|),$$

puis on prend la définition de x_{s+1} et on fait un DL de Taylor

$$\begin{aligned} x_{s+1}(\Theta + \dot\Theta) &= \mathcal{F}_s(x_s(\Theta + \dot\Theta), \theta_s + \dot\theta_s) \\ &= \mathcal{F}_s(x_s(\Theta) + \dot x_s + \mathcal{O}\|\dot\Theta\|, \theta_s + \dot\theta_s) \\ &= \mathcal{F}_s(x_s(\Theta), \theta_s) + (\partial_\theta \mathcal{F}_s)\, \dot\theta_s + (\partial_x \mathcal{F}_s)\, \dot x_s + \mathcal{O}\|\dot\Theta\| \\ &= x_{s+1}(\Theta) + \dot x_{s+1} + \mathcal{O}\|\dot\Theta\| \end{aligned}$$

Nous sommes maintenant capable de montrer la formule de rétropropagation

Théorème 4.2.2 — Rétropopagation. Avec les notations du Théorème 4.2.1, définissons $\hat{\theta} = (\hat{\theta}_s)_{0\leq s\leq S}$ et $\hat{X} = (\hat{x}_s)_{0\leq s\leq S+1}$ définis par les récurrences rétrogrades suivantes

$$\begin{cases} \hat{x}_{s+1} & = 1 \\ \hat{x}_s & = (\partial_x \mathcal{F}_s)^\star \, \hat{x}_{s+1} \quad \forall 0 \leq s \leq S \\ \hat{\theta}_s & = (\partial_\theta \mathcal{F}_s)^\star \, \hat{x}_{s+1} \quad \forall 0 \leq s \leq S \end{cases}$$

Alors le gradient de $\Theta \mapsto x_{S+1}(\Theta)$ est donné par le vecteur $\hat{\Theta} = (\hat{\theta}_s)_{0\leq s\leq S}$.

Démonstration

Tout d'abord, on remarque que pour tout p, on a

$$\begin{aligned} \langle \dot{x}_{s+1}, \hat{x}_{s+1} \rangle &= \langle (\partial_\theta \mathcal{F}_s)\, \dot{\theta}_s + (\partial_x \mathcal{F}_s)\, \dot{x}_s, \hat{x}_{s+1} \rangle \\ &= \langle \dot{\theta}_s, (\partial_\theta \mathcal{F}_s)^\star \, \hat{x}_{s+1} \rangle + \langle \dot{x}_s, (\partial_x \mathcal{F}_s)^\star \, \hat{x}_{s+1} \rangle \\ &= \langle \dot{\theta}_s, \hat{\theta}_s \rangle + \langle \dot{x}_s, \hat{x}_s \rangle \end{aligned}$$

Ainsi en sommant toutes ces inégalités, on trouve

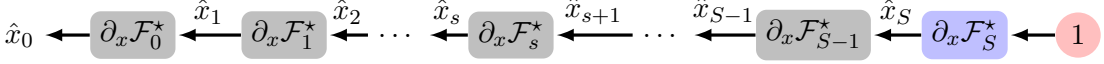
$$\langle \dot{x}_{S+1}, \hat{x}_{S+1} \rangle = \langle \dot{x}_0, \hat{x}_0 \rangle + \sum_{s=0}^{S} \langle \dot{\theta}_s, \hat{\theta}_s \rangle$$

Ou encore, en utilisant $\hat{x}_{S+1} = 1$ et $\dot{x}_0 = 0$, on obtient $\dot{x}_{S+1} = \langle \dot{\Theta}, \hat{\Theta} \rangle$. Pour toute direction $\dot{\Theta}$ on a donc

$$x_{S+1}(\Theta + \dot{\Theta}) = x_{S+1}(\Theta) + \langle \dot{\Theta}, \hat{\Theta} \rangle + o(\|\dot{\Theta}\|).$$

Ainsi le gradient de $\Theta \mapsto x_{S+1}(\Theta)$ est donné par le vecteur $\hat{\Theta}$.

On représente un calcul de rétropropagation, ici seul le calcul de $\hat{X}$ est représenté



4.3 Structure des fonctions

Nous détaillons dans cette section la structure globale de notre librairie, nous commençons par la classe `Parameter` qui vise à stocker dans un même endroit θ_s et $\hat{\theta}_s$. La variable θ_s sera stockée dans l'attribut `data` et la variable $\hat{\theta}_s$ sera stockée dans l'attribut `grad`. On initialise la classe `Parameter` avec la taille des paramètres.

```
1 class Parameter() :
2     def __init__(self,shape) :
3         self.shape=shape
4         np.random.seed(42)
5         self.grad=np.zeros(self.shape)
6         self.size=self.grad.size
7         self.data=np.random.randn(self.size).reshape(self.shape)
```

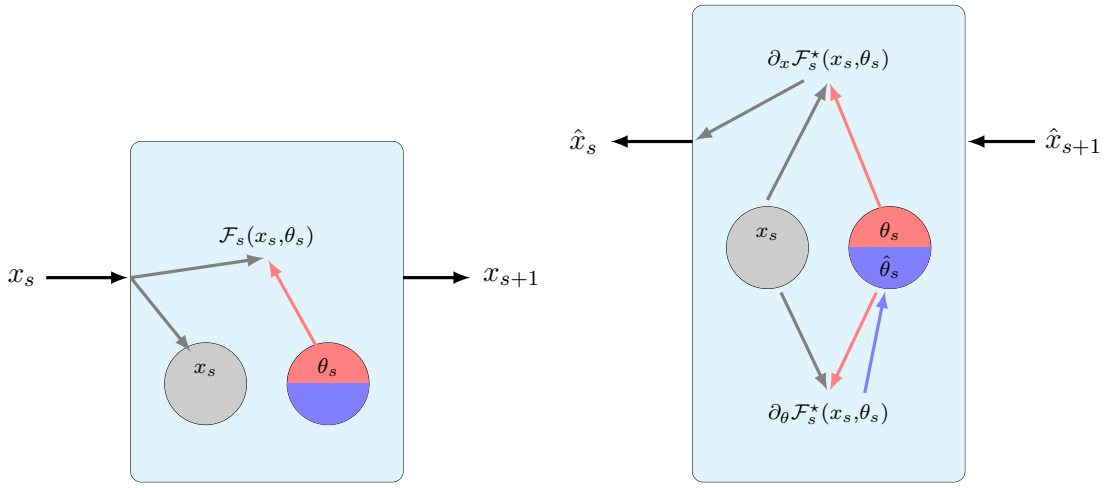


Figure 4.1: Fonctionnement d'une couche en mode forward (à gauche) et en mode backward (à droite). On représente les paramètres avec un cercle rouge et bleu

Ensuite on discute de la classe `Layer` dont les instances seront les couches du réseau de neurone, cette classe doit posséder

- L'attribut `list_params` qui est la liste des paramètres (des instances de `Parameter`) de la couche. Dans cet attribut sont stockés les couples θ_s et $\hat{\theta}_s$.
- L'attribut `save` qui sert à sauver des informations.
- La méthode `y=forward(x)` qui permet de calculer $x_{s+1} = \mathcal{F}_s(x_s, \theta_s)$. Ici
  1. x_s est donné par `x`, en entrée et `y` est la variable x_{s+1}.
  2. θ_s est récupéré depuis les attributs `data` de `list_params`.
  3. La fonction `forward` remplit la variable `save` en fonction des besoins de la fonction `backward`.
- La méthode `x=backward(y)` qui permet de calculer $\hat{x}_s = (\partial_x\mathcal{F}_s)^\star\hat{x}_{s+1}$ et $\hat{\theta}_s = (\partial_\theta\mathcal{F}_s)^\star\hat{x}_{s+1}$. Ici
  1. $\hat{x}_{s+1}$ est donné par `y` en entrée et `x` est la variable $\hat{x}_s$.
  2. θ_s est récupéré depuis les attributs `data` de éléments de `list_params` et si besoin x_s est récupéré depuis l'attribut `save`.
  3. $\hat{\theta}_s$ est sauvé dans l'attributs `grad` des éléments de `list_params`.

4.4 Calcul effectif de la rétropopagation du gradient

Ici on montre comment on calcule $\hat{x}_s$ et $\hat{\theta}_s$ dans deux exemples de couches. On procéde toujours de la même manière

- On écrit la couche sous la forme $y = \mathcal{F}(x, \theta)$
- On écrit la formule du tangent c'est-à-dire qu'on écrit l'équation $\dot{y} = (\partial_x\mathcal{F})\dot{x} + (\partial_\theta\mathcal{F})\dot{\theta}$
- L'objectif est de calculer $\hat{x}$ et $\hat{\theta}$ si on se donne $\hat{y}$. On rappelle qu'on doit obéir à l'équation

$$\langle\hat{y}, \dot{y}\rangle = \langle\hat{x}, \dot{x}\rangle + \langle\hat{\theta}, \dot{\theta}\rangle.$$

  1. On se donne $\hat{y}$, on suppose que $\dot{\theta} = 0$ et on cherche $\hat{x}$ tel que on ait

$$\langle\hat{y}, \dot{y}\rangle = \langle\hat{x}, \dot{x}\rangle$$

2. On se donne $\hat{y}$, on suppose que $\dot{x} = 0$ et on cherche $\hat{\theta}$ tel que on ait

$$\langle \hat{y}, \dot{y} \rangle = \langle \hat{\theta}, \dot{\theta} \rangle$$

4.4.1 Si $\mathcal{F}$ est affine

Théorème 4.4.1 — Calcul de la rétropopagation si $\mathcal{F}$ est affine. On s'intéresse à une couche du genre $y = \mathcal{F}(x, \theta)$ où $x \in \mathcal{M}_{pn}(\mathbb{R})$, $y \in \mathcal{M}_{qn}(\mathbb{R})$, $\theta = (A, b)$ avec $A \in \mathcal{M}_{qp}(\mathbb{R})$ et $b \in \mathbb{R}^q$ et la couche est définie par

$$y_{j\ell} = \sum_{i=1}^{p} A_{ji} x_{i\ell} + b_j.$$

Etant donné $\hat{y} \in \mathcal{M}_{qn}(\mathbb{R})$, alors $\hat{x}$ et $\hat{\theta} = (\hat{A}, \hat{b})$ sont donnés par

- $\hat{x}_{i\ell} = \sum_j A_{ji} \hat{y}_{j\ell}$.
- $\hat{A}_{ji} = \sum_\ell x_{i\ell} \hat{y}_{j\ell}$.
- $\hat{b}_j = \sum_\ell \hat{y}_{j\ell}$.

Démonstration

1. calcul du tangent : on a

$$\dot{y}_{j\ell} = \sum_{i=1}^{p} A_{ji} \dot{x}_{i\ell} + \dot{A}_{ji} x_{i\ell} + \dot{b}_j$$

2. calcul de $\hat{x}$: on suppose $\dot{A} = \dot{b} = 0$, on a pour tout $\hat{y} \in \mathcal{M}_{qn}(\mathbb{R})$

$$\langle \dot{y}, \hat{y} \rangle = \sum_{j\ell} \dot{y}_{j\ell} \hat{y}_{j\ell} = \sum_{j\ell} \left(\sum_i A_{ji} \dot{x}_{i\ell} \right) \hat{y}_{j\ell} = \sum_{i\ell} \dot{x}_{i\ell} \left(\sum_j A_{ji} \hat{y}_{j\ell} \right) = \langle \dot{x}, \hat{x} \rangle,$$

   où on définit $\hat{x}_{i\ell} = \sum_j A_{ji} \hat{y}_{j\ell}$.

3. calcul de $\hat{A}$: on suppose $\dot{x} = \dot{b} = 0$, on a pour tout $\hat{y} \in \mathcal{M}_{qn}(\mathbb{R})$

$$\langle \dot{y}, \hat{y} \rangle = \sum_{j\ell} \dot{y}_{j\ell} \hat{y}_{j\ell} = \sum_{j\ell} \left(\sum_i \dot{A}_{ji} x_{i\ell} \right) \hat{y}_{j\ell} = \sum_{ji} \dot{A}_{ji} \left(\sum_\ell x_{i\ell} \hat{y}_{j\ell} \right) = \langle \dot{A}, \hat{A} \rangle,$$

   où on définit $\hat{A}_{ji} = \sum_\ell x_{i\ell} \hat{y}_{j\ell}$.

4. calcul de $\hat{b}$: on suppose $\dot{x} = \dot{A} = 0$, on a pour tout $\hat{y} \in \mathcal{M}_{qn}(\mathbb{R})$

$$\langle \dot{y}, \hat{y} \rangle = \sum_{j\ell} \dot{y}_{j\ell} \hat{y}_{j\ell} = \sum_{j\ell} \dot{b}_j \hat{y}_{j\ell} = \sum_j \dot{b}_j \left(\sum_\ell \hat{y}_{j\ell} \right) = \langle \dot{b}, \hat{b} \rangle,$$

   où on définit $\hat{b}_j = \sum_\ell \hat{y}_{j\ell}$.

4.4.2 Si $\mathcal{F}$ est une fonction d'activation

Théorème 4.4.2 Soit ϕ une fonction C^1, si $y = \mathcal{F}(x, \theta)$ où x est un vecteur de $\mathbb{R}^n$ et y un vecteur de $\mathbb{R}^n$ avec

$$\forall i \quad y_i = \phi(x_i).$$

alors le calcul de la rétropropagation du gradient donne

$$\hat{x}_i = \phi'(x_i)\hat{y}_i$$

Démonstration

1. calcul du tangent : on a
$$\dot{y}_i = \phi'(x_i)\dot{x}_i$$
2. calcul de $\hat{x}$. Pour tout $\hat{y} \in \mathbb{R}^n$:
$$\langle \dot{y}, \hat{y} \rangle = \sum_i \left(\phi'(x_i)\dot{x}_i\right) \hat{y}_i = \sum_i \dot{x}_i \left(\phi'(x_i)\hat{y}_i\right) = \langle \dot{x}, \hat{x} \rangle,$$
où on définit $\hat{x}_i = \phi'(x_i)\hat{y}_i$.

4.4.3 Si $\mathcal{F}$ est la fonction SoftMax

Théorème 4.4.3 On se donne la couche sans paramètre $y = \mathcal{F}(x, \theta)$ qui pour tout $x \in \mathcal{M}_{pn}(\mathbb{R})$ calcule $y \in \mathcal{M}_{pn}(\mathbb{R})$ par

$$y_{i\ell} = \frac{e^{x_{i\ell}}}{\sum_{j=1}^{p} e^{x_{j\ell}}}$$

Alors le calcul de rétropropagation du gradient donne

$$\hat{x}_{i\ell} = y_{i\ell}\hat{y}_{i\ell} - \sum_j y_{i\ell}y_{j\ell}\hat{y}_{j\ell}$$

Démonstration

1. calcul du tangent : on a
$$\dot{y}_{i\ell} = \frac{e^{x_{i\ell}}}{\sum_{j=1}^{p} e^{x_{j\ell}}}\dot{x}_{i\ell} - \sum_{j=1}^{p} \frac{e^{x_{i\ell}} e^{x_{j\ell}}}{\left(\sum_{k=1}^{p} e^{x_{k\ell}}\right)^2}\dot{x}_{j\ell} = y_{i\ell}\dot{x}_{i\ell} - \sum_j y_{i\ell}y_{j\ell}\dot{x}_{j\ell}$$
2. calcul de $\hat{x}$. Pour tout $\hat{y} \in \mathcal{M}_{pn}(\mathbb{R})$:
$$\begin{aligned} \langle \dot{y}, \hat{y} \rangle &= \sum_{i\ell} \dot{y}_{i\ell}\hat{y}_{i\ell} = \sum_{i\ell} \left(y_{i\ell}\dot{x}_{i\ell} - \sum_j y_{i\ell}y_{j\ell}\dot{x}_{j\ell} \right) \hat{y}_{i\ell} \\ &= \sum_{i\ell} \dot{x}_{i\ell} \left(y_{i\ell}\hat{y}_{i\ell} - \sum_j y_{i\ell}y_{j\ell}\hat{y}_{j\ell} \right) = \langle \dot{x}, \hat{x} \rangle, \end{aligned}$$
où on définit $\hat{x}_i = y_{i\ell}\hat{y}_{i\ell} - \sum_j y_{i\ell}y_{j\ell}\hat{y}_{j\ell}$.

INSA TOULOUSE
135 avenue de Rangueil
31400 Toulouse

Tél : + 33 (0)5 61 55 95 13
www.insa-toulouse.fr

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE