Réseaux de neurones

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

Correction des TDs

3MIC semestre 7

Dr. Frédéric de Gournay

Correction des TDs

1 Calculer des adjoints

(15 mins) Exercice 1.1

L'objectif est de retrouver la formule de l'adjoint dans le cas où le produit scalaire n'est pas Euclidien, ou dans le cas où le base n'est pas orthonormée. On se donne E et F deux espaces vectoriels réels de dimension respectives n et m finies. On se donne $(e_i)_{i=1,\dots n}$ et $(f_j)_{j=1,\dots,m}$ deux bases de E et F. On se donne deux produits scalaires sur E et F notés $\langle \bullet, \bullet \rangle_E$ et $\langle \bullet, \bullet \rangle_F$. Pour tout vecteur $u \in E$ (resp. $v \in F$), on note $\tilde{u} \in \mathbb{R}^n$ (resp. $\tilde{v} \in \mathbb{R}^n$) le vecteur des coordonnées de u (resp. de v), i.e. le vecteur tel que

$$u = \sum_{i=1}^{n} \tilde{u}[i] e_i \quad (resp.\ v = \sum_{j=1}^{m} \tilde{v}[j] f_j).$$

Finalement, on note $(\bullet, \bullet)_n$ le produit scalaire usuel de $\mathbb{R}^n$.

- Soit S_E la matrice de taille $n \times n$ donnée par $S_E[i,j] = \langle e_i, e_j \rangle_E$ (on définira de manière similaire S_F plus tard).
  1. Montrer rapidement que la base (e_i) est orthonormale si et seulement si S_E vaut l'identité.
  2. Montrer que pour tout $a, b \in E$, on a $\langle a, b \rangle_E = \langle \tilde{a}, S_E \tilde{b} \rangle_n$
  3. Montrer rapidement que S_E est une matrice symétrique définie positive.
- Pour tout opérateur $\mathcal{A}$ de E dans F, on note $\tilde{\mathcal{A}}$ la matrice $\mathcal{M}_{mn}(\mathbb{R})$ tel que pour tout u de E :

  $$\widetilde{(\mathcal{A}u)} = \tilde{\mathcal{A}} \tilde{u},$$

  ou encore, pour tout i, $\mathcal{A} e_i = \sum_{j=1}^{m} \tilde{\mathcal{A}}[j,i] f_j$. Montrer que $\widetilde{(\mathcal{A}^\star)} = S_E^{-1} \tilde{\mathcal{A}}^T S_F$

Solution de l'Exercice 1.1

- Ici on a majoritairement des rappels et on va passer assez vite sur ces questions
  1. La base (e_i) est orthonormale ssi $\begin{cases} \langle e_i, e_j \rangle_E = 0 & \text{pour tout} i \neq j \\ \langle e_i, e_i \rangle_E = 1 & \text{pour tout} i \end{cases}$. Ce qui est équivalent à $\begin{cases} S_E[i,j] = 0 & \text{pour tout} i \neq j \\ S_E[i,i] = 1 & \text{pour tout} i \end{cases}$. Ce qui est équivalent à dire que S_E vaut l'identité.
  2. Soit $a, b \in E$, on a

  $$\begin{aligned} \langle a, b \rangle_E &= \langle \sum_i \tilde{a}[i] e_i, \sum_k \tilde{b}[k] e_k \rangle_E = \sum_{i,k} \tilde{a}[i] \tilde{b}[k] \langle e_i, e_k \rangle_E \\ &= \sum_{i,k} S_E[i,k] \tilde{a}[i] \tilde{b}[k] = \sum_i \tilde{a}[i] (S_E \tilde{b})[i] = \langle \tilde{a}, S_E \tilde{b} \rangle_n \end{aligned}$$

  3. S_E est une matrice symétrique car $S_E[i,j] = \langle e_i, e_j \rangle_E = \langle e_j, e_i \rangle_E = S_E[j,i]$. Elle est définie positive car pour tout $\tilde{u} \in \mathbb{R}^n$ tel que $\tilde{u} \neq 0$, si on dénote $u = \sum_{i=1}^{n} \tilde{u}[i] e_i$, alors $u \neq 0$ et

  $$\langle S_E \tilde{u}, \tilde{u} \rangle_n = \langle u, u \rangle_E > 0.$$

  Ainsi S_E est bien définie positive.

- Soit $\mathcal{A}$ un opérateur de E dans F. On note $\mathcal{B}$ l'opérateur tel que $\tilde{\mathcal{B}} = S_E^{-1}\tilde{\mathcal{A}}^T S_F$, on a pour tout u dans E et v dans F

$$\begin{aligned}\langle \mathcal{A}u, v\rangle_F &= \langle S_F\widetilde{\mathcal{A}u}, \tilde{v}\rangle_m = \langle S_F\tilde{\mathcal{A}}\tilde{u}, \tilde{v}\rangle_m = \langle \tilde{u}, \tilde{\mathcal{A}}^T S_F\tilde{v}\rangle_n = \langle \tilde{u}, S_E S_E^{-1}\tilde{\mathcal{A}}^T S_F\tilde{v}\rangle_n \\ &= \langle S_E\tilde{u}, \tilde{\mathcal{B}}\tilde{v}\rangle_n = \langle S_E\tilde{u}, \widetilde{\mathcal{B}v}\rangle_n = \langle u, \mathcal{B}v\rangle_E\end{aligned}$$

(15 mins) Exercice 1.2

Soit $m, n, p \in \mathbb{N}$. Pour tout $x \in \mathcal{M}_{np}(\mathbb{R})$ et $A \in \mathcal{M}_{mn}(\mathbb{R})$, on nomme $A \cdot x \in \mathcal{M}_{mp}(\mathbb{R})$ la matrice définie par, pour tout $1 \le i \le m$ et $1 \le j \le p$.

$$(A \cdot x)_{i,j} = \sum_{k=1}^{n} A_{i,k} x_{k,j}.$$

Donner les adjoints des opérateurs $A \mapsto A \cdot x$ et $x \mapsto A \cdot x$, si on considère les produits scalaires usuels sur les matrices. On concluera que leurs adjoints sont, respectivement, $B \mapsto B \cdot x^T$ et $B \mapsto A^T \cdot B$.

Solution de l'Exercice 1.2

On note $\mathcal{A} : A \mapsto A \cdot x$ et $\mathcal{X} : x \mapsto A \cdot x$, on remarque que $\mathcal{A} \in \mathcal{L}(\mathcal{M}_{mn}(\mathbb{R}) \to \mathcal{M}_{mp}(\mathbb{R}))$ et $\mathcal{X} \in \mathcal{L}(\mathcal{M}_{np}(\mathbb{R}) \to \mathcal{M}_{mp}(\mathbb{R}))$. Ainsi

$$\mathcal{A}^\star \in \mathcal{L}(\mathcal{M}_{mp}(\mathbb{R}) \to \mathcal{M}_{mn}(\mathbb{R})) \text{ et } \mathcal{X}^\star \in \mathcal{L}(\mathcal{M}_{mp}(\mathbb{R}) \to \mathcal{M}_{np}(\mathbb{R})).$$

Pour tout A dans $\mathcal{M}_{mn}(\mathbb{R})$, x dans $\mathcal{M}_{np}(\mathbb{R})$ et B dans $\mathcal{M}_{mp}(\mathbb{R})$, on a

- On commence par $\mathcal{A}$

$$\begin{aligned}\langle \mathcal{A}A, B\rangle &= \langle A \cdot x, B\rangle = \sum_{ij}(A \cdot x)_{ij} B_{ij} = \sum_{ijk} A_{ik} x_{kj} B_{ij} = \sum_{ik} A_{ik} (\sum_j B_{ij} x_{kj}) \\ &= \sum_{ik} A_{ik} (B \cdot x^T)_{ik} = \langle A, B \cdot x^T\rangle\end{aligned}$$

  Ainsi $\mathcal{A}^\star : B \mapsto B \cdot x^T$.

- Pour $\mathcal{X}$, on procède de manière similaire

$$\begin{aligned}\langle \mathcal{X}x, B\rangle &= \langle A \cdot x, B\rangle = \sum_{ij}(A \cdot x)_{ij} B_{ij} = \sum_{ijk} A_{ik} x_{kj} B_{ij} = \sum_{kj} x_{kj} (\sum_i B_{ij} A_{ik}) \\ &= \sum_{kj} x_{kj} (A^T \cdot B) = \langle x, A^T \cdot B\rangle\end{aligned}$$

  Ainsi $\mathcal{X}^\star : B \mapsto A^T \cdot B$.

(30 mins) Exercice 1.3

L'objectif est de construire des espaces de discrétisation de l'ensemble des fonctions.

- On se donne $n+2$ points $a = x_0 < x_1 < \cdots < x_n < x_{n+1} = b$. Pour tout i entre 0 et $n+1$, on note $x_{i+\frac{1}{2}}$ le milieu du segment $[x_i, x_{i+1}]$, on a donc

$$x_{i+1/2} = \frac{x_i + x_{i+1}}{2},$$

- Soit $C_c^\infty([a,b])$ l'ensemble des fonctions C^∞ sur $[a,b]$ qui s'annulent sur un voisinage de a et de b. On note E l'espace vectoriel des $u \in \mathbb{R}^{n+2}$ tels que $u_0 = u_{n+1} = 0$ et F l'espace vectoriels $\mathbb{R}^{n+1}$. pour les diférentier les vecteurs $u \in E$ seront notés $(u_i)_{0\leq i\leq n+1}$ et les vecteurs $v \in F$ seront notés $(v_{i+\frac{1}{2}})_{0\leq i\leq n}$ On construit les applications de discrétisation π_E et π_F qui vont de $C_c^\infty([a,b])$ vers respectivement E et F et qui sont définies par

$$\pi_E(\phi)_i = \phi(x_i) \quad \text{ et } \pi_F(\phi)_{i+\frac{1}{2}} = \phi(x_{i+\frac{1}{2}}).$$

  En d'autres termes, l'espace E est la discrétisation des fonctions $C_c^\infty([a,b])$ prises aux points x_i et l'espace F est la discrétisation des fonctions $C_c^\infty([a,b])$ prises aux points $x_{i+\frac{1}{2}}$.
- On se donne sur E et F des produits scalaires qui sont des versions discrétisées de $\langle \phi, \psi\rangle = \int \phi\psi$.

$$\langle u, v\rangle_E = \sum_{i=0}^{n}(x_{i+1}-x_i)\frac{u_{i+1}v_{i+1}+u_iv_i}{2} \text{ et } \langle u, v\rangle_F = \sum_{i=0}^{n}(x_{i+1}-x_i)u_{i+\frac{1}{2}}v_{i+\frac{1}{2}}.$$

  Autrement dit, pour E on regarde une approximation par la méthode des trapèzes et pour F par le point du milieu.
- Si $\mathcal{D}$ est l'endomorphisme de $C_c^\infty([a,b])$ défini par $\mathcal{D} : \phi \mapsto \phi'$, une discrétisation logique de $\mathcal{D}$ est l'opérateur $D : E \to F$ défini par

$$\forall u \in E \quad (Du)_{i+1/2} = \frac{u_{i+1}-u_i}{x_{i+1}-x_i}$$

1. Vérifier que

$$\langle u, v\rangle_E = \sum_{i=1}^{n}(x_{i+1/2}-x_{i-1/2})u_iv_i$$

2. Vérifier que

$$(D^\star v)_i = -\frac{v_{i+1/2}-v_{i-1/2}}{x_{i+1/2}-x_{i-1/2}} \qquad \text{pour tout } 1 \leq i \leq n.$$

3. Vérifier que $\mathcal{D}^\star = -\mathcal{D}$

Solution de l'Exercice 1.3

1. On a

$$\begin{aligned}\langle u, v\rangle_E &= \sum_{i=0}^{n}(x_{i+1}-x_i)\frac{u_{i+1}v_{i+1}+u_iv_i}{2}\\ &= \sum_{i=1}^{n+1}(x_i-x_{i-1})\frac{u_iv_i}{2} + \sum_{i=0}^{n}(x_{i+1}-x_i)\frac{u_iv_i}{2}\end{aligned}$$

   Comme u_i et v_i s'annulent en $i=0$ et $i=n+1$, ces deux sommes sont en fait des sommes de i allant de 1 à n et on obtient

$$\langle u, v\rangle_E = \sum_{i=1}^{n}(x_{i+1}-x_{i-1})\frac{u_iv_i}{2}$$

Correction des TDs

Ensuite, il suffit de remarque que $x_{i+1/2} - x_{i-1/2} = \frac{(x_{i+1}-x_{i-1})}{2}$

2. On note B l'opérateur de F dans E défini par

$$(Bv)_i = -\frac{v_{i+1/2} - v_{i-1/2}}{x_{i+1/2} - x_{i-1/2}} \quad \text{pour tout } 1 \leq i \leq n.$$

On utilise la formule du produit scalaire :

$$\begin{aligned} \langle Du, v\rangle_F &= \sum_{i=0}^{n} (x_{i+1} - x_i)(Du)_{i+1/2} v_{i+1/2} = \sum_{i=0}^{n} (u_{i+1} - u_i) v_{i+1/2} \\ &= \sum_{i=1}^{n+1} u_i v_{i-1/2} - \sum_{i=0}^{n} u_i v_{i+1/2} \end{aligned}$$

On utilise encore le fait que $u_0 = u_{n+1} = 0$ pour montrer que les deux sommes du dessus sont en fait des sommes de i allant de 1 à n. On obtient donc

$$\langle Du, v\rangle_F = \sum_{i=1}^{n} u_i (v_{i-1/2} - v_{i+1/2}) = \sum_{i=1}^{n} (x_{i+1/2} - x_{i-1/2}) u_i (Bv)_i.$$

Ainsi on a bien $B = D^\star$.

3. on a bien $\int_a^b \phi' \psi = -\int_a^b \phi \psi'$ pour tout $(\phi, \psi) \in C_c^\infty([a, b])$ par intégration par partie. Cela a été fait en cours.

2 La programmation orientée objet

(15 mins) Exercice 2.1

J'ai effacé une partie d'un programme que vous devez re-écrire, vous devez cependant comprendre tout le programme pour arriver à rétablir la partie manquante à la place du commentaire `##TODO` dans la fonction `adjoint()`.

```python
class my_strange_linear_op() :
    def __init__(self,input_len) :
        assert isinstance(input_len,int)
        assert input_len > 2
        self.input_len=input_len
        self.output_len=2*input_len
        np.random.seed(42)
        self.A=np.random.randn(self.output_len,self.input_len)
    def direct(self,x) :
        assert x.shape==(self.input_len,)
        u=self.A@x
        u[0]=4*u[0]
        u[1]+=u[0]
        return u
    def adjoint(self,u) :
        assert u.shape==(self.output_len,)
```

```
        ##TODO
    def test(self) :
        np.random.seed(42)
        x=np.random.randn(self.input_len)
        u=np.random.randn(self.output_len)
        print("0 ?=? ",np.sum(self.direct(x)*u)-np.sum(x*self.adjoint(u)))
import numpy as np
e=my_strange_linear_op( 4 )
e.test() # 0 ?=?  -1.7763568394002505e-15
```

Solution de l'Exercice 2.1

```
def adjoint(self,u) :
    assert u.shape==(self.output_len,)
    v=np.copy(u)
    v[0]+=v[1]
    v[0]=4*v[0]
    x=self.A.T@v
    return x
```

3 Approximation de fonctions

(10 mins) Exercice 3.1

Soit une suite $(I_\ell, O_\ell)_{\ell=1,\dots,L}$ un jeu de donnée avec I_ℓ et O_ℓ dans $\mathbb{R}$ pour tout ℓ. On suppose que tous les I_ℓ sont deux à deux différents. On cherche une fonction $f_\Theta : \mathbb{R} \to \mathbb{R}$ avec $f(I) \simeq O$ tel que $f_\Theta(x) = \Theta_2 x^2 + \Theta_1 x + \Theta_0$. Ici notre ensemble de fonction est paramétré par $\Theta \in \mathbb{R}^3$. On utilisera la fonction perte $d(\hat{o}, o) = \frac{1}{2}|o - \hat{o}|^2$.

1. Montrez que le problème d'approximation est un problème de moindre carré linéaire. Ecrire les équations normales.
2. Dans le cas $L = 3$, comment s'appelle le problème ? dans le cas $L < 3$ y-a-t'il existence et unicité des solutions ?

Solution de l'Exercice 3.1

1. On rappelle que le problème s'écrit

$$\min_{\Theta \in \mathbb{R}^3} \sum_{\ell=1}^{L} d(O_\ell, f_\Theta(I_\ell))^2.$$

En remplaçant, cela donne

$$\min_{\Theta \in \mathbb{R}^3} \sum_{\ell=1}^{L} \frac{1}{2} \left(O_\ell - \Theta_2 I_\ell^2 - \Theta_1 I_\ell - \Theta_0\right)^2.$$

Pour tout $\Theta \in \mathbb{R}^3$, on défini $F(\Theta) \in \mathbb{R}^L$ par

$$F(\Theta)_\ell = -O_\ell + \Theta_2 I_\ell^2 + \Theta_1 I_\ell + \Theta_0 \text{ pour tout } 1 \leq \ell \leq L.$$

Dans ce cas, en notant $\| \bullet \|$ la norme Euclidienne usuelle, le problème est

$$\min_{\Theta \in \mathbb{R}^3} \sum_{\ell=1}^{L} \frac{1}{2} (F(\Theta)_\ell)^2 \text{ ou encore } \min_{\Theta \in \mathbb{R}^3} \frac{1}{2} \|F(\Theta)\|^2.$$

On s'aperçoit ensuite que $F(\Theta) = A\Theta - b$ si on pose

$$A = \begin{pmatrix} 1 & I_1 & I_1^2 \\ \vdots & \vdots & \vdots \\ 1 & I_L & I_L^2 \end{pmatrix} \text{ et } b = \begin{pmatrix} O_1 \\ \vdots \\ O_L \end{pmatrix}.$$

On minimise $f : \Theta \mapsto \frac{1}{2}\|A\Theta - b\|^2$. On connaît par cœur son gradient (moi en tout cas) et sa Hessienne qui sont données par :

$$\nabla f(\Theta) = A^T(A\Theta - b) \text{ et } H[f](\Theta) = A^T A.$$

Il est facile (essayez !!) de montrer que $H[f](\Theta) \succeq 0$ et donc que f est convexe sur $\mathbb{R}^3$. Ainsi toute solution de $\nabla f(\Theta) = 0$ est solution du problème et on cherche Θ solution de

$$A^T A\Theta = A^T b$$

2. Dans le cas $L = 3$, il faut trouver un polynôme de degré 3 qui passe par trois points (diffférents) donnés, il y a une et une seule solution. De plus la matrice A est carrée et inversible, elle vaut

$$A = \begin{pmatrix} 1 & I_1 & I_1^2 \\ 1 & I_2 & I_2^2 \\ 1 & I_3 & I_3^2 \end{pmatrix}.$$

C'est une matrice dite de **Vandermonde**, son déterminant est non nul et on a donc $\Theta^\star = -A^{-1}b$ et $F(\Theta^\star) = 0$. Dans le cas $L < 3$, il n'y a plus unicité des solutions mais il y a existence d'une infinité de Θ tel que $F(\Theta) = 0$. Effectivement il suffit d'ajouter des données $(I_{L+1}, O_{L+1}), \ldots, (I_3, O_3)$ pour que l'on ait exactement 3 jeux de données toutes différentes et se ramener au cas $L = 3$.

4 Optimisation d'un réseau de neurones

(20 mins) Exercice 4.1

On s'intéresse à la fonction suivante de $\theta = (\theta_0, \theta_1, \theta_2) \in \mathbb{R}^3$

$$x_4(\theta) = \sqrt{\theta_2^2 + (\theta_1 \cos(\theta_0 + 2))^2}$$

- Calculez directement le gradient de $\theta \mapsto x_3(\theta)$.
- On définit $\mathcal{F}_0(x_0, \theta_0) = \cos(\theta_0 + x_0)$, $\mathcal{F}_2(x_2, \theta_2) = \theta_2^2 + x_2^2$. Définir $\mathcal{F}_1$ et $\mathcal{F}_3$ pour que x_4 soit défini par la récurence suivante

$$x_{s+1} = \mathcal{F}(x_s, \theta_s) \quad x_0 = 2$$

Vous donnerez les formules de $X = (x_i)_{i=0,;4}$

- Utiliser la formule de rétropropagation de gradient pour trouver le gradient de x_4 par rapport à θ (on exprimera tout en fonction de $X = (x_i)_{i=0,\dots,4}$). Comparez avec la méthode de la première question.

Solution de l'Exercice 4.1

- On trouve

$$\nabla_\theta x_4 = \frac{1}{2\sqrt{\theta_2^2 + (\theta_1 \cos(\theta_0+2))^2}} \begin{pmatrix} 2(\theta_1 \cos(\theta_0+2)\theta_1(-\sin(\theta_0+2)) \\ 2\theta_1 \cos^2(\theta_0+2) \\ 2\theta_2 \end{pmatrix}$$

- Il faut prendre $\mathcal{F}_1(x_1,\theta_1) = \theta_1 x_1$ et $\mathcal{F}_3(x_3) = \sqrt{x_3}$. Ainsi on a

$$\begin{cases} x_1 & = \cos(\theta_0 + x_0) \\ x_2 & = \theta_1 x_1 \\ x_3 & = \theta_2^2 + x_2^2 \\ x_4 & = \sqrt{x_3} \end{cases}$$

- On a $\hat{x}_4 = 1$
  1. Avec $\dot{x}_4 = \frac{1}{2\sqrt{x_3}}\dot{x}_3$, on conclut que $\hat{x}_3 = \frac{\hat{x}_4}{2\sqrt{x_3}} = \frac{1}{2\sqrt{x_3}}$
  2. Avec $\dot{x}_3 = 2\dot{\theta}_2\theta_2 + 2\dot{x}_2 x_2$ on obient

$$\begin{cases} \hat{x}_2 & = 2x_2\hat{x}_3 = \frac{2x_2}{2\sqrt{x_3}} \\ \hat{\theta}_2 & = 2\theta_2\hat{x}_3 = \frac{2\theta_2}{2\sqrt{x_3}} \end{cases}$$

  3. Avec $\dot{x}_2 = \dot{\theta}_1 x_1 + \theta_1 \dot{x}_1$ on obient

$$\begin{cases} \hat{x}_1 & = \theta_1\hat{x}_2 = \frac{2x_2\theta_1}{2\sqrt{x_3}} \\ \hat{\theta}_1 & = x_1\hat{x}_2 = \frac{2x_2x_1}{2\sqrt{x_3}} \end{cases}$$

  4. Avec $\dot{x}_1 = \dot{\theta}_0(-\sin(\theta_0+x_0)) + \dot{x}_0(-\sin(\theta_0+x_0))$ on obient

$$\begin{cases} \hat{x}_0 & = -\sin(\theta_0+x_0)\hat{x}_1 = \frac{2x_2\theta_1}{2\sqrt{x_3}}(-\sin(\theta_0+x_0)) \\ \hat{\theta}_0 & = -\sin(\theta_0+x_0)\hat{x}_1 = \frac{2x_2\theta_1}{2\sqrt{x_3}}(-\sin(\theta_0+x_0)) \end{cases}$$

  On retrouve bien $\nabla_\theta x_4 = \begin{pmatrix} \hat{\theta}_0 \\ \hat{\theta}_1 \\ \hat{\theta}_2 \end{pmatrix}$

(15 mins) Exercice 4.2: Gradient implicite

On s'intéresse à la fonction $y = \mathcal{F}(x,\theta)$ où $x \in \mathbb{R}^n$, $\theta \in \mathcal{M}_{nn}(\mathbb{R})$ tel que θ^{-1} existe et y est défini de manière implicite par

$$\theta y = x.$$

1. Pour tout $\dot{\theta}$ et $\dot{x}$, donnez la formule de $\dot{y} = (\partial_x\mathcal{F})\dot{x} + (\partial_\theta\mathcal{F})\dot{\theta}$
2. Pour tout $\hat{y}$, donnez la formule de $\hat{\theta} = (\partial_\theta\mathcal{F})^\star\hat{y}$ et $\hat{x} = (\partial_x\mathcal{F})^\star\hat{y}$

Solution de l'Exercice 4.2

1. On trouve $\dot\theta y + \theta \dot y = \dot x$ ou encore
$$\dot y = \theta^{-1}(\dot x - \dot\theta y)$$
2. On se donne $\hat y$ quelconque
   - On suppose $\dot\theta = 0$, on cherche $\hat x$ tel que
   $$\langle \dot y, \hat y\rangle = \langle \theta^{-1}\dot x, \hat y\rangle = \langle \dot x, \theta^{-\star}\hat y\rangle.$$
   Ainsi $\hat x = \theta^{-\star}\hat y$.
   - On suppose $\dot x = 0$, on cherche $\hat\theta$
   $$\begin{aligned}\langle \dot y, \hat y\rangle &= -\langle \theta^{-1}\dot\theta y, \hat y\rangle = -\langle \dot\theta y, \theta^{-\star}\hat y\rangle = -\sum_i (\dot\theta y)_i(\theta^{-\star}\hat y)_i \\ &= -\sum_{ikl} \dot\theta_{ik} y_k (\theta^{-1})_{li}\hat y_l.\end{aligned}$$
   Ainsi $\hat\theta_{ik} = -\sum_l y_k(\theta^{-1})_{li}\hat y_l$.

5 Exemple d'annale

(40 mins) Exercice 5.1

On se donne la couche $y = \mathcal{F}(x,\theta)$ définie pour tout $x \in \mathcal{M}_{NL}(\mathbb{R})$, $\theta = (A,B)$ avec $A \in \mathcal{M}_{MN}(\mathbb{R})$ et $B \in \mathbb{R}^M$ par
$$y_{ml} = \sum_{n=1}^N A_{mn}^2 \cos(x_{n\ell}^2) + B_m \text{ pour tout } 1 \le \ell \le L \text{ et } 1 \le m \le M.$$
Ainsi y est une matrice de $\mathcal{M}_{ML}(\mathbb{R})$.

1. Si $\dot x$ et $\dot\theta = (\dot A, \dot B)$ sont donnés, donnez la formule de $\dot y = (\partial_x\mathcal{F})\dot x + (\partial_\theta\mathcal{F})\dot\theta$.
2. Pour tout $\hat y$, donnez la formule de $\hat\theta = (\partial_\theta\mathcal{F})^\star\hat y$ et $\hat x = (\partial_x\mathcal{F})^\star\hat y$. On notera $\hat\theta = (\hat A, \hat B)$.
3. On note ci-dessous une classe `interro` qui implémente cette couche. Donnez les lignes de code qu'il faut mettre à la place des balises `#TODO1` à `#TODO6` pour implémenter correctement cette couche. On rappelle la structure de la classe `Parameter` et de la classe `Dense` en préambule.

```
import numpy as np

class Parameter() :
    def __init__(self,shape) :
        self.shape=shape
        np.random.seed(42)
        self.grad=np.zeros(self.shape)
        self.size=self.grad.size
        self.data=np.random.randn(self.size).reshape(self.shape)
```

```
class Dense() :
    def __init__(self,nb_entree,nb_sortie) :
        A=Parameter((nb_sortie,nb_entree))
        b=Parameter((nb_sortie,1))
        self.list_params=[A,b] # Liste des paramètres
        self.save=None # Objet pour sauver des infos dans le forward
    def forward(self,x) :
        self.save=np.copy(x)
        (A,b)=[p.data for p in self.list_params]
        y=A@x+b
        return y
    def backward(self,hat_y) :
        x=self.save
        (A,b)=[p.data for p in self.list_params]
        hat_A=hat_y@x.T
        hat_b=np.sum(hat_y,axis=1)
        hat_x=A.T@hat_y
        self.list_params[0].grad=hat_A
        self.list_params[1].grad=hat_b
        for p in self.list_params :
            p.grad=p.grad.reshape(p.shape)
        return hat_x

class Interro()
  def __init__(self,nb_entree,nb_sortie) :
        self.list_params=[Parameter((nb_sortie,nb_entree)),Parameter((nb_sortie,1))]
        self.save=None
    def forward(self,x) :
        self.save=np.copy(x)
        (A,b)=[p.data for p in self.list_params]
        y=     None #TODO1 : change this
        return y
    def backward(self,hat_y) :
        x=     None #TODO2 : change this
        (A,b)= None #TODO3 : change this
        hat_A= None #TODO4 : change this
        hat_b= None #TODO5 : change this
        hat_x= None #TODO6 : change this
        self.list_params[0].grad=hat_A
        self.list_params[1].grad=hat_b
        for p in self.list_params :
            p.grad=p.grad.reshape(p.shape)
        return hat_x
```

Solution de l'Exercice 5.1

1. On différentie et on trouve

$$\dot{y}_{ml} = \sum_{n=1}^{N} 2\dot{A}_{mn}A_{mn}\cos(x_{n\ell}^2) - 2A_{mn}^2\dot{x}_{n\ell}x_{n\ell}\sin(x_{n\ell}^2) + \dot{B}_m$$

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

$$\begin{aligned} \langle \dot{y}, \hat{y} \rangle &= \sum_{ml}(\sum_{n} -2A_{mn}^2\dot{x}_{n\ell}x_{n\ell}\sin(x_{n\ell}^2))\hat{y}_{ml} \\ &= \sum_{n\ell}\dot{x}_{n\ell}\left(\sum_{m} -2A_{mn}^2 x_{n\ell}\sin(x_{n\ell}^2)\hat{y}_{ml}\right) = \langle \hat{x}, \dot{x} \rangle \end{aligned}$$

   avec $\hat{x}_{nl} = -2x_{n\ell}\sin(x_{n\ell}^2)\left(\sum_m A_{mn}^2\hat{y}_{ml}\right)$.
   - On suppose $\dot{x} = 0$ et $\dot{B} = 0$, on cherche $\hat{A}$ tel que

$$\begin{aligned} \langle \dot{y}, \hat{y} \rangle &= \sum_{ml}(\sum_{n} 2\dot{A}_{mn}A_{mn}\cos(x_{n\ell}^2))\hat{y}_{ml} \\ &= \sum_{mn}\dot{A}_{mn}\left(\sum_{l} 2A_{mn}\cos(x_{n\ell}^2)\hat{y}_{ml}\right) = \langle \hat{A}, \dot{A} \rangle \end{aligned}$$

   avec $\hat{A}_{mn} = 2A_{mn}\left(\sum_l \cos(x_{n\ell}^2)\hat{y}_{ml}\right)$.
   - On suppose $\dot{x} = 0$ et $\dot{A} = 0$, on cherche $\hat{B}$ tel que

$$\langle \dot{y}, \hat{y} \rangle = \sum_{ml}(\dot{B}_m)\hat{y}_{ml} = \sum_{m}\dot{B}_m(\sum_{l}\hat{y}_{ml}) = \langle \hat{B}, \dot{B} \rangle$$

   avec $\hat{B}_m = \sum_l \hat{y}_{ml}$.

3. On a :

```
        y=(A*A)@np.cos(x*x) + b                          #TODO1
        x=self.save                                      #TODO2
        (A,b)=[p.data for p in self.list_params]         #TODO3
        hat_A=2*A*(hat_y@np.cos(x**2).T)                 #TODO4
        hat_b=np.sum(hat_y,axis=1)                       #TODO5
        hat_x=-2*x*np.sin(x**2)*((A*A).T@hat_y)          #TODO6
```

6 correction TP1

```
import numpy as np
class Parameter() :
    def __init__(self,shape) :
        self.shape=shape
        np.random.seed(42)
        self.grad=np.zeros(self.shape)
        self.size=self.grad.size

```

```
class Arctan() :
    def __init__(self) :
        self.list_params=[] # Liste des paramètres
        self.save=None # Objet pour sauver des infos dans le forward
    def forward(self,x) :
        self.save=np.copy(x)
        return np.arctan(x)
    def backward(self,hat_y) :
        hat_x=hat_y/(1+self.save**2)
        return hat_x

class Dense() :
    def __init__(self,nb_entree,nb_sortie) :
        A=Parameter((nb_sortie,nb_entree))
        b=Parameter((nb_sortie,1))
        self.list_params=[A,b] # Liste des paramètres
        self.save=None # Objet pour sauver des infos dans le forward
    def forward(self,x) :
        self.save=np.copy(x)
        (A,b)=[p.data for p in self.list_params]
        return A@x + b
    def backward(self,hat_y) :
        x=self.save
        (A,b)=[p.data for p in self.list_params]
        hat_A=hat_y@x.T
        hat_b=np.sum(hat_y,axis=1)
        hat_x=A.T@hat_y
        self.list_params[0].grad=hat_A
        self.list_params[1].grad=hat_b
        for p in self.list_params :
            p.grad=p.grad.reshape(p.shape)
        return hat_x

class Loss_L2() :
    def __init__(self,D) :
        self.D=np.copy(D)
        self.list_params=[]
        self.save=None
    def forward(self,x) :
        self.save=np.copy(x)
        return 0.5*np.linalg.norm(x-self.D)**2
    def backward(self,hat_y) :
        x=self.save
        return (x-self.D)*hat_y

class Sequential() :
    def __init__(self,list_layers) :
        self.list_params=[] # Liste des paramètres
        self.list_layers=list_layers
```

```
        for l in self.list_layers :
            self.list_params+=l.list_params
        self.save=None # Objet pour sauver des infos dans le forward
    def forward(self,x) :
        z=np.copy(x)
        for l in self.list_layers :
            z=l.forward(z)
        return  z
    def backward(self,hat_y) :
        z=np.copy(hat_y)
        for l in reversed(self.list_layers) :
            z=l.backward(z)
        return z
```

7 correction TP2

7.1 Algorithme dysfonctionnel

```
np.random.seed(13)
N=Neur.Sequential([Neur.Dense(1,12),Neur.Sigmoid(),Neur.Dense(12,1)])
N_a=Neur.Sequential([N,Neur.Loss_L2(y)])

nbiter=2000
step=3.e-6
cost=np.zeros((nbiter+1))
for k in range(nbiter) :
    cost[k]=N_a.forward(x)
    N_a.backward(1.)
    for p in N.list_params :
        p.data=p.data-step*p.grad
cost[nbiter]=N_a.forward(x)
y2=N.forward(x)
plt.plot(x[0,:],y[0,:])
plt.plot(x[0,:],y2[0,:])
plt.show()
plt.plot(cost)
```

7.2 Interfacage

```
def Size(N) :
    n=0
    for p in N_a.list_params :
        n+=p.data.size
    return n

def set_data(N_a,u) :
    # Fonction qui remplace les paramètres de N_a par le grand vecteur de u
    start=0
    for p in N_a.list_params :
        size=p.data.size
```

```
            p.data=u[start:start+size].reshape(p.shape)
            start=start+size

def get_data(N_a) :
    n=0
    for p in N_a.list_params :
        n+=p.data.size
    grad=np.zeros(n)
    # Fonction qui remplace les paramètres de N_a par le grand vecteur de u
    start=0
    for p in N_a.list_params :
        size=p.data.size
        grad[start:start+size]=p.data.ravel()
        start=start+size
    return grad

def get_grad(N_a) :
    n=0
    for p in N_a.list_params :
        n+=p.data.size
    grad=np.zeros(n)
    # Fonction qui remplace les paramètres de N_a par le grand vecteur de u
    start=0
    for p in N_a.list_params :
        size=p.data.size
        grad[start:start+size]=p.grad.ravel()
        start=start+size
    return grad
```

7.3 Deuxièmes algorithmes d'interfaçage

```
np.random.seed(13)
N=Neur.Sequential([Neur.Dense(1,12),Neur.Sigmoid(),Neur.Dense(12,1)])
N_a=Neur.Sequential([N,Neur.Loss_L2(y)])

def func(u):
    set_data(N_a,u)
    return N_a.forward(x)

def nablafunc(u):
    set_data(N_a,u)
    c=N_a.forward(x)
    N_a.backward(1.)
    return get_grad(N_a)
```

8 correction TP3

Correction des TDs

8.1 Classe Sigmoïde

```
class Sigmoid() :
    def __init__(self) :
        self.list_params=[] # Liste des paramètres
        self.save=None # Objet pour sauver des infos dans le forward
    def forward(self,x) :
        self.save=np.copy(x)
        return  1./(1+np.exp(-x))
    def backward(self,hat_y) :
        x=self.save
        hat_x=np.exp(-x)/(1+np.exp(-x))**2*hat_y
        return hat_x
```

8.2 Classe SoftMax

```
class Softmax() :
    def __init__(self) :
        self.list_params=[]
        self.save=None
    def forward(self,x) :
        y=np.exp(x)
        y=y/y.sum(axis = 0)
        self.save=np.copy(y)
        return y
    def backward(self,hat_y) :
        y=self.save
        c=np.sum(hat_y*y,axis=0)
        return y*(hat_y-c)
```

8.3 Classe KL

```
class KL() :
    def __init__(self,O) :
        self.O=O
        self.list_params=[]
        self.save=None
    def forward(self,x) :
        self.save=np.copy(x)
        return -np.sum(self.O*np.log(x))
    def backward(self,hat_y) :
        x=self.save
        return -self.O/x*hat_y
```

8.4 Algorithme d'optimisation

```
tau = .01/15
niter = 6000
np.random.seed(42)
import Neural as Neur
N=Neur.Sequential([Neur.Dense(2,15),Neur.Arctan(),Neur.Dense(15,3),Neur.Arctan()
```

```
                  ,Neur.Softmax()])
N_a=Neur.Sequential([N,Neur.KL(O)])
cost=np.zeros(niter)
for it in np.arange(0,niter):
    cost[it] = N_a.forward(I)
    N_a.backward(1.)
    for p in N_a.list_params :
        p.data=p.data-tau*p.grad
    if (it%400 == 0) : # ici toutes les 400 itérations, on affiche le réseau de neurone
        print(it,cost[it])
        plot_N(N)
        plt.show()
plt.plot(cost)
```

INSA TOULOUSE
135 avenue de Rangueil
31400 Toulouse

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

INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
TOULOUSE