M2 MMS : Réseaux de neurones pour la modélisation

August 11, 2025

Contents

1 Machines et réseaux de neurones **2**

2 Pourquoi construire des machines ? **4**
2.1 Différentes classes de problèmes 4

3 Fonction de perte **6**
3.1 Classification binaire 7
3.2 Classification 7
3.3 Moindre carré 7
3.4 Remarques 7

4 Apprentissage **8**
4.1 Méthode de gradient 8
4.2 Différentation automatique 8
4.3 Apprentissage par étape 8

5 EDO et méthode de collocation **8**
5.1 Méthode de collocation et méthode de Galerkin 9
5.1.1 Méthode de collocation 9
5.1.2 Méthode de Galerkin 9
5.2 Fonction loss 9

6 Réseaux pour les EDO **10**

7 EDO avec paramètres **10**

Introduction

Objectifs du cours

- Connaître les bases des machines et réseaux de neurones

- Savoir les utiliser dans le contexte de modèles basées sur les équations différentielles ordinaires (EDO)
- Savoir utiliser des librairies de l'IA en python.

Chaque chapitre correspond à 1-3 cours/TP.

Librairies

- `tensorflow`
- `pytorch`
- `jax` (avec `Flax` et `Optax`)

Prérequis

- Les base de `python`
- `list`, `dict`, `class`
- `numpy`

1 Machines et réseaux de neurones

Définition 1.1. *Soit $m, n, n_W \in \mathbb{N}$. Une **machine** est une application*

$$\Phi : \mathbb{R}^n \times \mathbb{R}^{n_W} \to \mathbb{R}^m, \qquad (x, \tilde{w}) \to y = \Phi(x, \tilde{w}).$$

***L'apprentissage** consiste à fixer $w = w_*$ de sorte que l'application $x \mapsto \Phi(x, w_*)$ permet de représenter des données ou un modèle physique.*

*On appelle Φ une **machine vectoriel**, si*

$$\Phi(x, \tilde{w}) = \sum_{i=1}^{N} c_i \phi_i(x, w).$$

Dans ce cas nous avons $\tilde{w} = (c, w)$ et $X_\Phi := \operatorname{Im} \Phi = \operatorname{vect} \{\phi_i(x, w) \mid 1 \leq i \leq N\}$. Les fonction $\phi_i^w : x \mapsto \phi_i(x, w)$ peuvent former une base ou non. On les appelle souvent 'features' en anglais, ou un "dictionnaire de fonctions de base". Nous avons alors une application ('feature map')

$$\mathbb{R}^{n_W} \to \{\mathbb{R}^n \to \mathbb{R}^m\}^N \quad w \mapsto (\phi_i^w)_{1 \leq i \leq N} \tag{1}$$

Exemple 1.2. *Voici quelques exemples de machines vectoriels.*

- *L'interpolation de Lagrange d'une fonction univariée continue* $f : [0; 1] \to \mathbb{R}$ *avec les points d'interpolation* $0 = t_0 < \cdots < t_{i-1} < t_i < t_N = 1$.

$$I_n f(t) = \sum_{i=0}^{N} c_i \phi_i(t), \quad \phi_i(t) = \prod_{\substack{j=1 \\ j \neq i}}^{N} \frac{t - t_j}{t_i - t_j}.$$

  Ici, nous avons "$w = \emptyset$*".*

- *Même chose, mais avec les points d'interpolation variables, donc "*$w = (t_i)$*".*

- *Espace* $P^1([0; 1])$*. Similaire au deux précédents.*

- *Espace des séries de Fourier tronquées. Similaire aux précédents.*

Définition 1.3. *On appelle* ***fonction d'activation*** *une fonction* $\sigma : \mathbb{R} \to \mathbb{R}$.

1. *Sigmoide* $\sigma(x) = \frac{1}{1+e^x}, \quad \sigma(\mathbb{R}) \subset]0; 1[$.
2. *Tanh* $\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \sigma(\mathbb{R}) \subset]-1; 1[$.
3. *ReLU* $\mathrm{relu}(x) = x^+ = \max\{0, x\}, \quad \sigma(\mathbb{R}) \subset]-1; 1[$.
4. *Swish* $\mathrm{swish}(x) = x\sigma(\beta x), \quad \sigma(\mathbb{R}) \subset]-1; 1[$.
5. *Id* $\sigma(x) = x, \quad \sigma(\mathbb{R}) = \mathbb{R}$.

Plein d'autres son possible. Attention : l'image de σ *est importante.*

Définition 1.4. *Pour* $f : \mathbb{R} \to \mathbb{R}$ *on définit* $f : \mathbb{R}^n \to \mathbb{R}^n$ *(sans distinction de notation !) composante par composante,* $(f(x))_i = f(x_i), 1 \leq i \leq n$.

Un ***réseau à une couche ('perceptron')*** $(W, b, \sigma), W \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$ *est l'application*

$$\Phi(x, W, b, \sigma) : \mathbb{R}^n \to \mathbb{R}^m, \qquad x \mapsto \sigma(Wx + b). \tag{2}$$

Un ***réseau multi-couches (MLP, 'multi-layer-perceptron')*** *est la composition de réseau à une couche* $\Phi_i = \Phi_i(W_i, b_i, \sigma_i)$, $1 \leq i \leq L$ *avec*

$$\Phi(x, \tilde{w}) : \mathbb{R}^n \to \mathbb{R}^m \qquad \Phi(x, \tilde{w}) = \Phi_L \circ \cdots \circ \Phi_1. \tag{3}$$

Évidemment nous avons $\tilde{w} = (W_i, b_i)_{1 \leq i \leq L}$ *et*

$$W_i \in \mathbb{R}^{n_i \times n_{i-1}}, \quad b_i \in \mathbb{R}^{n_i}, \quad n_0 = n, \quad n_L = m.$$

Nous appelons plus généralement ***machine***

Remarque 1.5. *Un MLP est une machine vectoriel, si* $\sigma_L = \mathrm{id}$. *Nous avons* $N = n_{L-1}$, *si* $b_L = 0$ *et* $N = n_L$ *sinon. Le fait d'avoir un biais* $b_L \neq 0$ *rajoute la fonction de base* $\phi_0 = 1$, *donc, dans ce cas* b_L *joue aussi le rôle d'un coefficient.*

Voici quelques types de couches.

- 'Linear layer avec poids ('weight') W et biais ('bias') b'. $x \mapsto Wx + b$.
- 'Activation layer'. $x \mapsto \sigma(x)$.
- 'Convolutional layer'. Une convolution s'écrit comme une multiplication matrice-vecteur :

  (1D) $y_i = \sum_{k=-m}^{m} w_k x_{i+k}$, **(2D)** $y(i,j) = \sum_{k=-m}^{m} \sum_{l=-m}^{m} w(k,l)x(i+k, j+l)$.

- 'Residual layer' $\Phi : \mathbb{R}^n \to \mathbb{R}^n$, $\Phi(x) = x + \sigma(Wx + b)$
- 'Attention layer' : $\Phi(x) = \mathrm{softmax}\left(n^{-1/2} K Q^{\mathsf{T}}\right) V$ $(K, Q, V \in \mathbb{R}^{n\times n})$

  $$\mathrm{softmax}_\beta : \mathbb{R}^n \to]0;1[^n \quad x \mapsto \left(\frac{e^{\beta x_i}}{\sum_{j=1}^n e^{\beta x_j}}\right)_{1\leq i\leq n}.$$

  C'est le gradient de $\mathrm{logsumexp}_\beta : \mathbb{R}^n \to \mathbb{R}, \quad x \mapsto \beta \log\left(\sum_{j=1}^n e^{\beta x_j}\right)$
- 'Dropout layer' Il s'agit de réduire le nombre de paramètres de la matrice W.
- 'Implicit layer' Ici Φ correspond à la solution d'une équation, par exemple une équation différentielle.
- 'SVM' exemple d'un problème d'optimisation, aussi 'implicit'.

2 Pourquoi construire des machines ?

- Expliquer les relatons entre x et y : dans le cas unidimensionnel, on interprète de façon géométrique et algébrique $y = ax$ ou $y = bx^2$.
- Prédire le résultat y associé à un x.

2.1 Différentes classes de problèmes

- **Classification (classement)**
  - binaire : il y seulement deux caractéristiques (que l'on peut appeler 0 et 1, $y \in \{0, 1\}$ ou -1 et +1) : chat ou chien, appareil fonctionne ou pas, passager survit à la catastrophe du *Titanic*,...

– classes multiples ('pattern recognition'): reconnaître un chiffre donné $y \in \{0, 1, \cdots, 9\}$, différents types de vélos,...

- **Régression** : $y \in \mathbb{R}$ dans le cas scalaire. Étant donné $(x_i, y_i)_{1 \leq i \leq n_D}$ on cherche f tel que

$$l(f) := \frac{1}{2n_D} \sum_{i=1}^{n_D} (y_i - f(x_i))^2$$

soit le plus petit possible. Ce problème dépend de la classe de fonction $f \in \mathcal{F}$. Si $\mathcal{F}$ est assez grand nous nous rappelons que $l(f_{\text{lag}}) = 0$ pour le polynôme d'interpolation de Lagrange f_{lag}, car interpolation signifie $y_i = f(x_i)$. On se rappelle également que ce processus est instable pour $n_D \to +\infty$ (et même $n_D \approx 10$, phénomène de Runge). On s'intéresse plutôt à une classe $\mathcal{F}$ avec peu de paramètre. Par exemple Gauß avait pris $\mathcal{F} = \{x \mapsto a + bx \mid a, b \in \mathbb{R}\}$, ce qui permet une interprétation des paramètres.

- **Simplification de modèle** : étant donné un modèle compliqué Φ^* que l'on ne peut évaluer, on cherche une approximation Φ. Pour l'exemple d'une EDO nonlinéaire (10)

$$\Phi^* : U \subset \mathbb{R}^n \to C^1(I, \mathbb{R}^n), \quad \Phi^*(u_0) = \text{solution de (10) sur l'intervalle } I$$

Premièrement, Φ^* ne prend pas des valeurs en un espace de dimension finie. Deuxièmement, on ne connaît pas $\Phi^*(u_0)(t)$ pour un temps $t \in I$. On est obligé de calculer une approximation par une méthode numérique. Et on peut utiliser un réseaux de neuronnes, qui devient alors une méthode numérique. Il y a différentes technique qui rentre dans cette catégorie

  – **Méthodes de Galerkine** (Séries de Fourier, FEM, ondelettes, méthodes spectrales) où l'on construit des fonctions de base appropriée pour le modèle en question. Ces méthodes dépendent de paramètres. Dans le cas des séries de Fourier il s'agit des indices que l'on utilise et dans le cas de la FEM, il s'agit par exemple du maillage. Depuis longtemps il existe des algorithmes qui cherche à optimiser ces paramètres (méthodes adaptatives) en fonction de la 'solution' $\Phi^*(u_0)$. Il s'agit de l'approximation non-linéaire. Une caractéristique des méthode de Galerkine est que les coefficients sont obtenus par la résolution d'un système d'équations nonlinéaires. Dans les réseaux de neuronnes, même si la machine est vectorielle, on minimise une fonction perte.
  – **Méthode de réduction de modèle**, POD où on essaie d'extraire l'information essentielle pour construire une base adaptée. Ces méthodes sont souvent combinées avec une méthode de Galerkine.

- **Mélange de régression et simplification**. Si nous avons un modèle Φ^* et des observations (données), on peut facilement combiner les deux si le problème d'apprentissage est une minimisation : il suffit de rajouter une mesure des écarts entre modèle et observations à la fonction à minimiser (la perte). Cela est souvent utilisé comme argument en faveur des réseaux de neuronnes. Toutefois, minimiser la somme de deux fonctions f et g dépend d'une pondération que l'on choisit a priori et le résultat peut être très différent des résultats des autres problèmes qui se posent naturellement

$$\begin{cases} \min\{f(x) + \alpha g(x) \mid x \in \mathbb{R}^n\}, \\ \min\{f(x) \mid x \in \mathbb{R}^n \cap \{\|g(x)\| \leq \varepsilon\}\}, \\ \min\{g(x) \mid x \in \mathbb{R}^n \cap \{\|f(x)\| \leq \delta\}\}. \end{cases}$$

3 Fonction de perte

Définition 3.1. *On appelle* ***données*** *(data) un ensemble* $\mathcal{D} = (x_i, y_i)_{1 \leq i \leq n_D}$, $x_i \in \mathbb{R}^n$, $y \in \mathbb{R}^m$. *Alors* $\mathcal{D} \subset \mathbb{R}^{n_D^*} \approx (\mathbb{R}^n \times \mathbb{R}^m)^{n_D}$. *Souvent, on partage les données en données d'apprentissage et données de test (et éventuellement données de validation).*

Définition 3.2. *On appelle* ***fonction de perte*** *(loss) une fonction* $\widetilde{L} : \mathbb{R}^{n_W} \times \mathbb{R}^{n_D^*} \to \mathbb{R}$ *dont la minimisation est l'****apprentissage*** *(learning). On suppose avoir* $L : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ *telle que*

$$\widetilde{L}(w, \mathcal{D}) = \frac{1}{n_D} \sum_{i=1}^{n_D} L(\Phi(x_i, w), y_i). \tag{4}$$

Le problème d'apprentissage est alors la minimisation de $\widehat{L} : \mathbb{R}^{n_W} \to \mathbb{R}$

$$\min\left\{\widehat{L}(w) \;\middle|\; w \in \mathbb{R}^p\right\}, \quad \widehat{L}(w) = \widetilde{L}(w, \mathcal{D}). \tag{5}$$

Remarque 3.3. *On peut avoir une régularisation* $\widehat{L}(w) = \widetilde{L}(w, \mathcal{D}) + \frac{\alpha}{2}\|w - w_0\|^2$ *avec* w_0 *des valeurs données des paramètres du modèle, ou, encore plus complexe, des contraintes sur les paramètres du modèle.*

Exemple 3.4.
- *(moindre carrés,* l^2*)* $L(z, y) = \frac{1}{2}\|y - z\|^2$.
- *(*l^1*)* $L(z, y) = \|y - z\|_{l^1(\mathbb{R}^m)}$.

On rappelle les notations du calcul différentiel. $F : \mathbb{R}^n \to \mathbb{R}^m$ est différentiable en x s'il existe $A \in \mathbb{R}^{m \times n}$ tel que

$$\lim_{\substack{h \to 0 \\ h \neq 0}} \frac{\|R(x,h)\|}{\|h\|} = 0 \quad (R(x,h) := F(x+h) - F(x) - Ah), \quad F'(x) := A.$$

($F'(x) = Df(x)$ est plus simple à écrire). Dans le cas $m = 1$ on écrit $\nabla f(x) := f'(x)^\mathsf{T}$. Pour des fonctions de plusieurs variables, par exemple $F(x,z)$ on écrit $F'_x(x,z)$ et $F'_z(x,z)$ pour les dérivées "partielles".

Par les théorèmes du calcul différentiel nous avons.

Théorème 3.5. *Si L et Φ sont continûment différentiables, alors $\widetilde{L} \in C^1(\mathbb{R}^{nW})$, le gradient de $\widetilde{L}$ s'écrit*

$$\nabla\widetilde{L}(w) = \frac{1}{n_D}\sum_{i=1}^{n_D} \Phi'_w(x_i,w)^\mathsf{T}\nabla_z L(\Phi(x_i,w),y_i) \tag{6}$$

3.1 Classification binaire

On utilise souvent la 'binary cross entropy' (si la dernière couche finit avec une sigmoïde)

$$y_i \in \{-1,+1\}, \quad l(x) := -\frac{1}{n_D}\sum_{i=1}^{n_D} y_i\log(z_i)+(1-y_i)\log(1-z_i); \quad z_i = \Phi(x_i,w) \tag{7}$$

3.2 Classification

Avec n_C classes, $y_{ij} \geq 0$, $\sum_{j=1}^{n_C} y_{ij} = 1$ pour tout i.

$$y_i \in [\![1,\cdots,n_c]\!] \quad l(x) := -\frac{1}{n_D}\sum_{i=1}^{n_D}\sum_{j=1}^{n_C} y_{ij}\log(z_{ij}); \quad z_{ij} = \Phi(x_i,w). \tag{8}$$

On interprète les z_{ij} comme la probabilité que

3.3 Moindre carré

$$l(x) = \frac{1}{2}\sum_{i=1}^{m} |y_i - \Phi(x_i,w)|^2. \tag{9}$$

3.4 Remarques

Il y a une certaine ambiguïté dans nos définition de couches et de fonction de perte : on peut toujours intégrer les dernières couches dans la fonction de parte.

Théorème 3.6. *Si la machine contient une couche de la forme $x \to \sigma(Wx+b)$, on peut modifier la fonction de perte telle que la machine est vectorielle.*

Proof. On écrite $x \mapsto \Phi(x,W,b) = \sigma(Wx+b)$ comme $\Phi_2 \circ \Phi_1$ avec $x \mapsto \Phi_1(x,W) = Wx$ et $y \mapsto \Phi_2(y,b) = \sigma(y+b)$. On a alors

$$L(\Phi(x,W,b),y) = L_2(\Phi_1(x,W),y), \quad L_2(x,y) := L(\Phi_2(x),y).$$

□

Exemple 3.7. *Pour les problème de classification on utilise généralement la fonction logistique.*

4 Apprentissage

Définition 4.1. *Un problème d'apprentissage de machine 'machine learning' consiste en trois ingrédients*

- *Une machine* $\Phi(x, w)$
- *Une fonction de perte (ce qui implique généralement les données et pratiquement leur traitement)*
- *Un algorithme d'optimisation (y compris critère d'arrêt).*

4.1 Méthode de gradient

4.2 Différentation automatique

4.3 Apprentissage par étape

5 EDO et méthode de collocation

On considère $U \subset \mathbb{R}^n$ ouvert, $u_0 \in U$ et une fonction $f \in C^1(U, \mathbb{R}^n)$. L'équation différentielle ordinaire (EDO) d'ordre un autonome

$$\begin{cases} \dfrac{du}{dt} = f(u) \\ u(0) = u_0. \end{cases} \tag{10}$$

On rappelle le théorème de Cauchy-Lipschitz. S'il existe $r > 0$ et $L > 0$ tel que

$$\|f(u) - f(v)\| \leq L\|u - v\| \quad \forall u, v \in U \cap B_r(u_0), \tag{11}$$

alors il existe $T > 0$ et une solution unique de (10) sur $[0, T]$. $f \in C^1(U, \mathbb{R}^n)$ est suffisant pour (11).

Remarque 5.1.

1. *Il existe un temps maximal* $T \in \mathbb{R} \cup \{+\infty\}$ *d'existence de solution. Si* $T < +\infty$, *nous avons* $\lim\limits_{t \nearrow T} |u(t)| = +\infty$.
2. *L'équation non-autonome*

$$\frac{du}{dt} = f(t, u)$$

s'écrite comme (10) *avec* $\widetilde{u} = (u, v)$, $\widetilde{f}(\widetilde{u}) = (f(v, u), 1)$ *et* $\widetilde{u_0} = (u_0, 0)$, *la variable* v *est le temps.*

5.1 Méthode de collocation et méthode de Galerkin

5.1.1 Méthode de collocation

Une méthode de collocation sur un intervalle $I = [0; T]$ consiste à choisir un espace $X_N \subset C^1(I, \mathbb{R}^n)$ de dimension $N + 1$ et des points (maillage)

$$\tau = (0 = t_0 < t_1 < \cdots < t_N = T)\,. \tag{12}$$

On impose les équations à une fonction $u_N \in X_N$

$$u(0) = u_0, \quad \frac{du}{dt}(t_i) = f(u(t_i)) \quad 1 \leq i \leq N. \tag{13}$$

Soit $(\phi_j)_{0\leq j\leq N}$ une base de X_N. (13) est alors converti en un système algébrique pour les coefficient $c \in \mathbb{R}^{N+1}$

$$u(t) = \sum_{j=0}^{N} c_j\phi_j(t), \quad \sum_{j=0}^{N} c_j\phi_j(0) = u_0, \quad \sum_{j=0}^{N} c_j\frac{d\phi_j}{dt}(t_i) = f(\sum_{j=0}^{N} c_j\phi_j(t)).$$

Exemple 5.2. *Pour $X_N = B^1(I)$, l'espace des spline quadratique de classe C^1 sur le **même** maillage $(t_i)_{0\leq i\leq N}$ on obtient un schéma de type Crank-Nicoloson. Plus généralement, les méthodes de collocation font partie des méthode de Runge-Kutta implicites.*

5.1.2 Méthode de Galerkin

Une méthode de Galerkin consiste à choisir deux espaces X_N et Y_N associé à τ et l'approximation est définie par

$$u \in u_0 + X_N : \int_0^T \frac{du}{dt}v = \int_0^T f(u)v \quad \forall v \in Y_n. \tag{14}$$

5.2 Fonction loss

Une première idée est d'utiliser un espace $\tilde{X} = X_\Phi$ généré par une machine vectorielles Φ est d'utiliser la fonction perte

$$\frac{1}{2}\|\tilde{u}(0) - u_0\|^2 + \sum_{i=0}^{N}\frac{1}{2}\|\frac{d\tilde{u}}{dt}(t_i) - f(\tilde{u}(t_i)\|^2$$

Il est alors facile de rajouter un term avec des données (PINN) $\mathcal{D} = (\tilde{t}_k, \tilde{w}_k)$ (mesures expermientales)

$$l_{\text{PINN}}(\tilde{u}) = \frac{1}{2}\|\tilde{u}(0) - u_0\|^2 + \sum_{i=0}^{N}\frac{1}{2}\|\frac{d\tilde{u}}{dt}(t_i) - f(\tilde{u}(t_i)\|^2 + \sum_{k=1}^{d}\frac{1}{2}\|\tilde{u}(\tilde{t}_k) - \tilde{w}_k\|^2$$

Remarque 5.3. *Dans le cas sans données ($d = 0$), si le MLP produit $B^1(I)$, la machine produit la solution de Crank-Nicolson. Une difficulté est le conditionnement du problème.*

6 Réseaux pour les EDO

1. Pour l'exemple de l'EDO d'ordre deux scalaire
   (a) Que se passe-t-il si l'on a plus de fonctions de base que de points de collocations ?
   (b) Étudier la convergence et l'erreur en fonction du nombre de point de collocation (et du nombre de fonctions de base).
   (c) Changer les conditions de limites : on remplace $u(T) = 0$ par $u'(0) = \alpha$.
   (d) Que se passe-t-il si l'on varie le nombre de couches ?
   (e) Que se passe-t-il si l'on change la fonction d'activation ?
   (f) Que se passe-t-il si l'on change la perte : par exemple pondération différente, l_1 ?
   (g) Essayer d'implémenter les conditions de bord comme contraintes.
   (h) Plotter les fonctions de base

7 EDO avec paramètres

Dans la pratique, les modèles mathématiques contiennent des paramètres (physique ou non). Soit $n_p \in \mathbb{N}$ et $p = (p_0, p_1) \in \mathbb{R}^{n_p}$ et

$$\begin{cases} \dfrac{du}{dt} = f(u, p_1) \\ u(0) = u_0(p_0). \end{cases} \tag{15}$$

Nous avons alors une application

$$S : \mathbb{R}^{n_p} \to C^1(I, \mathbb{R}^n) \qquad p \to u(p). \tag{16}$$

Des questions typiques sont

- Déterminer des paramètres à partir de mesures.
- Plus modestement : déterminer la sensibilité des solution par rapport aux paramètre.
- Plus ambitieux : déterminer les mesures les plus importantes.
- Dans un autre registre : trouver des valeurs critiques des paramètre. Les valeurs critiques sont celles quand la solution $p \to u(p)$ change de comportement, par exemple des points de bifurcation.

La clé à toutes ces questions est l'étude de l'application S définie en (16). Si elle est différentiable nous avons

$$\left\{\begin{array}{l} \qquad u := S(p), \quad \delta u := S'(p)(\delta p) \\ \left\{\begin{array}{r} \dfrac{d\delta u}{dt} = f'_u(u, p_1)\delta u + f'_p(u, p_1)\delta p_1 \\ \delta u(0) = u'_0(p_0)(\delta p_0). \end{array}\right. \end{array}\right. \tag{17}$$