Apprentissage d'ED0 et OdeNet

Emmanuel Franck*,

19 septembre, 2024
Master CMSI, M2, Strasbourg

*MACARON project-team, Université de Strasbourg, CNRS, Inria, IRMA, France

Outline

Apprentissage d’EDO: approche directe

Apprentissage d’EDO: Sindy

Apprentissage d’EDO: ODEnet

Apprentissage d'EDO: approche directe

Apprentissage d'EDO

- On suppose que l'on a des données temporelles. La *i*eme séquence est donnée par:

$$\mathcal{X} = (\mathbf{x}_1^1, ...\mathbf{x}_t^1, ..., \mathbf{x}_T^1), ..., \left(\mathbf{x}_1^i, ...\mathbf{x}_t^i, ..., \mathbf{x}_T^i\right), ..., (\mathbf{x}_1^M, ...\mathbf{x}_t^M ...., \mathbf{x}_T^M)$$

- L'approche statistique/probabiliste utilisée par les RNN ne suppose pas une certaine continuité temporelle.
- En physique, en biologie ou autre il est raisonnable de supposer cette continuité et donc d'apprendre la dynamique sous la forme d'EDO:

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}_\theta(\mathbf{x}(t))$$

ou F_θ est un modèle paramétrique à construire. On peut aussi construire une EDS:

$$\dot{\mathbf{x}}(t) = \mathbf{f}_\theta(\mathbf{x}(t))dt + \sigma dB_t$$

- On peut commencer par construire le jeux de données suivant

$$d\mathcal{X} = \left[\frac{d\mathbf{x}_1^1}{dt}, .., \frac{d\mathbf{x}_1^T}{dt}, ..., \frac{d\mathbf{x}_M^1}{dt}, .., \frac{d\mathbf{x}_M^T}{dt}\right] \in \mathbb{R}^d, \quad \text{avec } \frac{d\mathbf{x}_i^j}{dt} = \frac{\mathbf{x}_i^{j+1} - \hat{\mathbf{x}}_i^{j-1}}{t_{j+1} - t_{j-1}}.$$

Apprentissage d'EDO et stabilité I

Apprentissage d'EDO

Pour apprendre une EDO il suffit de minimiser:

$$\min_\theta \sum_{i=1}^{M} \sum_{j=1}^{T-1} \| \frac{d\mathbf{x}_i^j}{dt} - F_\theta(\mathbf{x}_i^j) \|_2^2$$

sur l'ensemble des trajectoires. Il s'agit d'apprentissage supervisé.

- Si le modèle paramétrique est C^1 on a un modèle localement Lipschitzien. Existence globale ? stabilité en temps long ?

A priori et stabilité

L'idée va être de se restreindre à certaines formes de fonction F_θ qui permettent d'obtenir de la stabilité. On fait un a priori sur la forme de modèle a déterminer.

Apprentissage d'EDO et stabilité II

- La théorie de Lyapunov est un outils important. En effet l'existence d'une fonction de Lyapunov assure un certains nombre propriétés de stabilité et l'existence globale de solutions.

Fonction de Lyapunov

On considère le système associé a un flux f. On appelle une fonction de Lyapunov une fonction V qui satisfait

$$lim_{\|\mathbf{x}\|\to\infty} V(\mathbf{x}) \to \infty, \quad \text{et} \quad \frac{dV(\mathbf{x}(t))}{dt} \leqslant 0, \quad \forall \mathbf{x}(t)$$

avec $\mathbf{x}(t)$ une trajectoire de l'EDO.

Théorème de limitation de Lyapunov

Si on a une fonction de Lyapunov pour un système d'EDO alors toutes les les trajectoires de ce système sont bornées pour $\forall t \geqslant 0$. On a donc existence globale des solutions.

Apprentissage d'EDO et stabilité III

Théorème de stabilité exponentielle de Lyapunov

Si on a une fonction définie positive ($V(\mathbf{x}) \geqslant 0$ et $V(0) = 0$) satisfaisant $\frac{dV(\mathbf{x}(t))}{dt} \leqslant -\alpha V(\mathbf{x}(t))$, il s'agit d'une fonction de Lyapunov qui assure que

$$\| \mathbf{x}(t) \| \leqslant Me^{-\frac{1}{2}\alpha t} \| \mathbf{x}(0) \|$$

- **Idée**: apprendre un flux qui satisfait la stabilité exponentielle de Lyapunov.
- Pour la stabilité exponentielle il nous faut $(\nabla_x V)^t f(\mathbf{x}(t)) \leqslant -\alpha V(\mathbf{x}(t))$.
- Si on donne une fonction Vet une fonction $\hat{f}$ qui ne satisfait pas cette condition on peut toujours la projeter sur l'espace $\{f, \quad (\nabla_x V)^t f(\mathbf{x}(t)) \leqslant -\alpha V(\mathbf{x}(t))\}$.
- Le projecteur est donné par:

$$f(\mathbf{x}) = P(\hat{f}) = \hat{f}(\mathbf{x}) - (\nabla_x V) \frac{Relu((\nabla_x V)^t \hat{f}(\mathbf{x}(t)) + \alpha V(\mathbf{x}(t)))}{\| (\nabla_x V) \|^2}$$

- **Apprentissage**: on projette à chaque étape de gradient le flux appris $\hat{f}_\theta$ sur f_θ.

Apprentissage d'EDO et stabilité IV

- Maintenant reste à savoir comment paramétrer la fonction de Lyapunov.
- **Difficulté**: V doit être positive, coercive et zéro est l'unique optimum.
- Optimum local loin de l'origine ==> la dynamique peut rester bloquée à cet endroit et ne pas décroitre vers zéro
- On choisit donc la paramétrisation suivante:

$$V_\theta(\mathbf{x}) = \sigma(g_\theta(\mathbf{x}) - g_\theta(\mathbf{0})) + \epsilon \parallel \mathbf{x} \parallel^2$$

avec σ_p une fonction d'activation positive, convexe, croissante et g_θ un réseau de neurones convexe par rapport à $\mathbf{x}$.

Réseau convexe

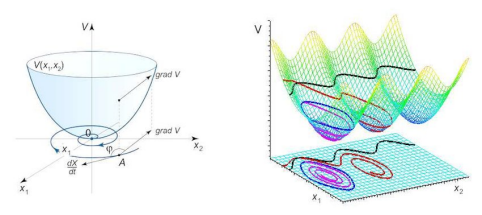
un réseau de première couche: $\mathbf{z}_1 = \sigma_0(W_0\mathbf{x} + b_0)$ et ou les couches suivantes sont définies par:

$$z_{k+1} = \sigma_k(U_k\mathbf{z}_k + W_k\mathbf{x} + \mathbf{b}_k)$$

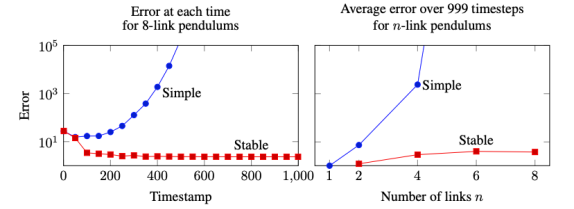
avec U_k, W_k des matrices de poids et $\mathbf{b}_k$ des vecteurs de biais. En prenant les poids U_k sont positifs et les fonctions d'activations σ_k convexe, monotone, croissante est une transformation convexe.

Apprentissage d'EDO et stabilité V

- Fonction de Lyapunov avec un/plusieurs points d'attractions.



- Exemple d'apprentissage de pendule dissipative avec/sans la méthode stable ("Learning Stable Deep Dynamics Model").



Apprentissage d'EDO Hamiltonienne

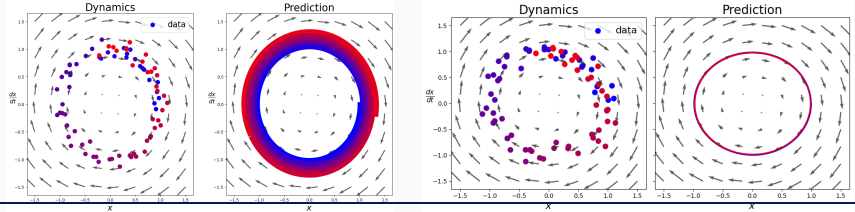
- Autre possibilité pour assurer la stabilité: imposer une structure conservative au modèle recherché. En général cela permet d'assurer des trajectoires bornées.
- On peut voir ca comme un a priori physique sur les modèles recherchés.

HNN

Soit un réseau $H_\theta(x)$ de $\mathbb{R}^{2n}$ dans $\mathbb{R}$ Pour obtenir une ODE Hamiltonienne on minimise:

$$\min_\theta \sum_{i=1}^{M} \sum_{j=1}^{T-1} \| \frac{d\mathbf{x}_i^j}{dt} - \mathcal{J}^{-1} \nabla_x H_\theta(\mathbf{x}_i^j) \|_2^2$$

On apprend un oscillateur sans imposer/en imposant la structure Hamiltonienne



Apprentissage d'EDO Lagrangienne

- Pour imposer une structure conservative on peut imposer une structure Lagrangienne.

Fonction de cout pour l'apprentissage d'une EDO dérivant d'un Lagrangien

Soit un réseau $\mathcal{L}_\theta(x)$ de $\mathbb{R}^{2n}$ dans $\mathbb{R}$. Soit $\left(\mathbf{q}_i^j, ...\right)$ des données de différente trajectoires. Soit $\frac{d\hat{\mathbf{q}}_i^j}{dt}$ et $\frac{d^2\hat{\mathbf{q}}_i^j}{dt^2}$ les reconstruction des dérivées première et seconde. Pour apprendre une ODE dérivant d'un Lagrangien on minimise:

$$\min_\theta \sum_{i=1}^{n}\sum_{j=1}^{T-1} \| \frac{d^2\hat{\mathbf{q}}_i^j}{dt^2} - \left(\frac{\partial \mathcal{L}(\mathbf{q},\dot{\mathbf{q}},t)}{\partial^2 \dot{\mathbf{q}}}\right)^{-1}\left(-\left(\frac{\partial \mathcal{L}(\mathbf{q},\dot{\mathbf{q}},t)}{\partial \mathbf{q}\partial \dot{\mathbf{q}}}\right)\frac{d\hat{\mathbf{q}}_i^j}{dt} + \frac{\partial \mathcal{L}(\mathbf{q},\dot{\mathbf{q}},t)}{\partial \mathbf{q}}\right) \|_2^2$$

- Cette approche admet quelques difficultés. Il faut reconstruire la dérivée seconde des données et donc avoir des données assez propres. Cette approche nécessite de calculer la dérivée seconde du réseau et $\nabla_\theta\left(\left(\frac{\partial \mathcal{L}(\mathbf{q},\dot{\mathbf{q}},t)}{\partial^2 \dot{\mathbf{q}}}\right)^{-1}\right)$.

Apprentissage d'EDO Hamiltonienne non canonique

- Maintenant on s'intéresse au cas non-canonique:

$$\dot{\mathbf{x}} = \mathcal{K}(\mathbf{x})^{-1}\nabla_{\mathbf{x}}\mathcal{H}(\mathbf{x}), \quad + \text{identité de Jacobi.}$$

- Comme précédemment on souhaite apprendre ce type de système à partir de données.
- Pour imposer cette structure on peut apprendre en minimisant:

$$\min_\theta \sum_{i=1}^{M}\sum_{j=1}^{T-1} \| \frac{d\mathbf{x}_i^j}{dt} - \mathcal{K}_\theta(\mathbf{x})^{-1}\nabla_x\mathcal{H}_\theta(\mathbf{x}_i^j) \|_2^2$$

- Le modèle appris est conservatif.
- Cependant pour obtenir un modèle non-canonique il faut aussi satisfaire l'identité de Jacobi.
- Pour obtenir cela on peut imposer:

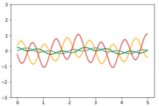
$$\mathcal{K}_{i,j,\theta}(\mathbf{x}) = \frac{\partial V_{i,\theta}(\mathbf{x})}{\partial x_j} - \frac{\partial V_{j,\theta}(\mathbf{x})}{\partial x_i}$$

Apprentissage d'EDO Hamiltonienne non canonique II

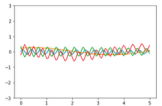
- On va minimiser

$$\min_\theta \sum_{i=1}^{M} \sum_{j=1}^{T-1} \| \frac{d\mathbf{x}_i^j}{dt} - \mathcal{K}(\mathbf{x})^{-1} \nabla_x \mathcal{H}_\theta(\mathbf{x}_i^j) \|_2^2 \quad \text{avec } \mathcal{K}_{i,j}(\mathbf{x}) = \frac{\partial V_{i,\theta}(\mathbf{x})}{\partial x_j} - \frac{\partial V_{j,\theta}(\mathbf{x})}{\partial x_i}$$

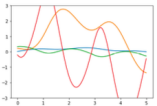
- Comparaison en temps court de plusieurs approches sur le double pendule



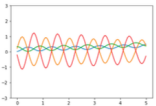
(a) Ground truth



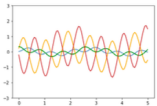
(c) HNN



(d) LNN

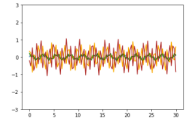


(e) Skew Matrix Learning

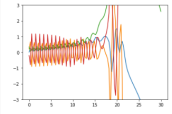


(f) Neural Symplectic Form

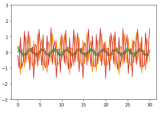
- En temps long



(a) Ground truth



(d) Skew Matrix Learning



(e) Neural Symplectic Form

Apprentissage d’EDO: approche directe

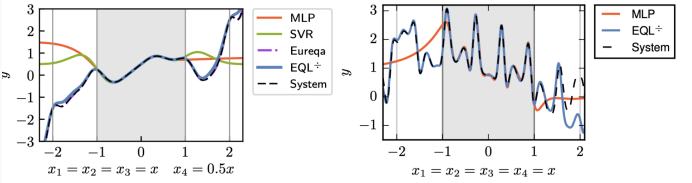
Apprentissage d’EDO: Sindy

Apprentissage d’EDO: ODEnet

Apprentissage d'EDO: Sindy

Apprentissage d'EDO et méthode Sindy

- L'apprentissage d'EDO à partir de ANN est très général mais ne donne pas des équations explicables. De plus l'extrapolation en temps est souvent mauvaise.
- **Méthode symbolique**: on cherche une expression analytique.
- Symbolique vs nn:



- Exemple de Modèle:

$$\begin{cases} \frac{dS(t)}{dt} = -\frac{\beta S(t) I(t)}{N} \\ \frac{dI(t)}{dt} = \frac{\beta S(t) I(t)}{N} - \gamma I(t) \\ \frac{dR(t)}{dt} = \gamma I(t) \end{cases}, \quad \begin{cases} \frac{dx(t)}{dt} = \sigma(y(t) - x(t)) \\ \frac{dy(t)}{dt} = x(t)(\rho - z(t)) - y(t) \\ \frac{dz(t)}{dt} = x(t)y(t) - \beta z(t) \end{cases}$$

- Les flux physique **f** recherchés sont souvent des fonctions simples.

Apprentissage d'EDO et méthode Sindy II

- Sindy est l'approche la plus simple de modèle symbolique.

Modèle Sindy

On définit le modèle dans le cas scalaire $x(t) \in \mathbb{R}$. On se donne un certains nombre de fonctions $(f_1(x), ..., f_m(x))$. Le modèle SINDy est donné par

$$f_\xi(x) = \sum_{i=1}^{m} \xi_i f_i(x)$$

avec comme paramètres $\xi = (\xi_1, ..., \xi_m)$.

- **Dilemme**: on veut utiliser peu de fonctions f_i pour coller à la physique mais beaucoup de fonctions pour avoir un modèle assez riche physiquement.
- **Idée**: prendre un grand dictionnaire de fonctions avec une contrainte de parcimonie.

Apprentissage d'EDO et méthode Sindy III

- On commence par introduire la matrice des données:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^T(t_1) \\ \mathbf{x}^T(t_2) \\ \vdots \\ \mathbf{x}^T(t_m) \end{bmatrix} = \begin{bmatrix} x_1(t_1) & x_2(t_1) & \cdots & x_d(t_1) \\ x_1(t_2) & x_2(t_2) & \cdots & x_d(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(t_m) & x_2(t_m) & \cdots & x_d(t_m) \end{bmatrix} \in \mathcal{M}_{n,d}(\mathbb{R}), \quad \dot{\mathbf{X}} = \begin{bmatrix} \dot{\mathbf{x}}^T(t_1) \\ \dot{\mathbf{x}}^T(t_2) \\ \vdots \\ \dot{\mathbf{x}}^T(t_m) \end{bmatrix}$$

- Ensuite on va construire la matrice des nonlinéarité:

$$\Theta(\mathbf{X}) = \begin{bmatrix} 1 & \mathbf{X} & \mathbf{f}_1(\mathbf{X}) & \mathbf{f}_2(\mathbf{X}) & \cdots & \mathbf{f}_m(\mathbf{X}) \end{bmatrix} \in \mathcal{M}_{n,m}(\mathbb{R})$$

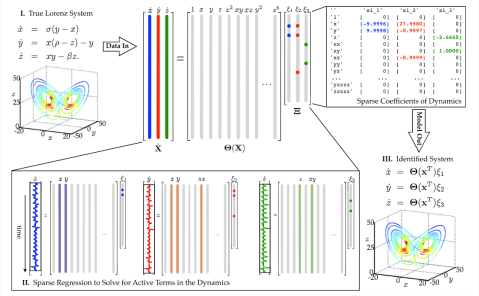
  Lorsqu'on note $\mathbf{f}_1(\mathbf{X})$ il s'agit du vecteur contenant la nonlinéarité $(t_1, ..., t_n)$.
- Les nonlinéarités seront du type x_1^2, $x_1x_2...x_d$, $sin(x_4)$ etc.

Sindy

$$\xi^* = \underset{\xi}{\operatorname{argmin}} \frac{1}{2} \left\| \dot{\mathbf{X}} - \Theta\left(\mathbf{X}^T\right) \xi \right\|_2^2 + \lambda \|\xi\|_1,$$

avec $\xi \in \mathcal{M}_{m,d}(\mathbb{R})$.

Algorithme de résolution



- Algorithm de résolution: STLSQ
  - On résout une succession de problème Ridge $\xi^* = \underset{\xi}{\operatorname{argmin}} \frac{1}{2} \left\| \dot{\mathbf{X}} - \mathbf{\Theta}\left(\mathbf{X}^T\right) \xi \right\|_2^2 + \lambda \|\xi\|_2^2$
  - On enlève les poids de ξ qui ont une norme inférieur à ϵ.
- Algorithm de résolution : SR3
  - On résout $\underset{\xi_{k+1}}{\operatorname{argmin}} \frac{1}{2} \left\| \dot{\mathbf{x}} - \mathbf{\Theta}\left(\mathbf{x}^T\right) \xi_k \right\|_2^2 + \frac{1}{2\nu} \parallel C\xi_k - \mathbf{w}_k \parallel_2^2$
  - On applique la parcimonie au poids $\mathbf{w}$: $\mathbf{w}_{k+1} = P_{\lambda,\mu}\left(\xi_{k+1}\right)$
- Sindy pour les EDO Hamiltoniennes et Lagrangiennes: on paramètrise $\mathcal{H}$ ou $\mathcal{L}$.
- Librairie python: `https://pypi.org/project/pysindy/`.

Weak Sindy

- L'approche Sindy classique utilise:

$$\dot{\mathbf{x}} - \Theta\left(\mathbf{x}^T\right)\xi = 0$$

- La formulation faible consiste donc à multiplier par une fonction test en temps à support compact sur $[0, T]$ nommée $\phi(t)$ et intégrer on obtient un résidu:

$$\mathcal{R}(\mathbf{x}) = \int_0^T \left(\dot{\mathbf{x}} - \Theta\left(\mathbf{x}^T\right)\xi\right)\phi(t)dt$$

- En integrant par partie on retrouve

$$\mathcal{R}(\mathbf{x}) = \int_0^T \mathbf{x}(t)\dot{\phi}(t)dt + \int_0^T \left(\Theta\left(\mathbf{x}^T\right)\xi\right)\phi(t)dt$$

- On discrétise l'intégrale pour obtenir le résidu qu'on minimisera:

$$\mathcal{R}_n(\mathbf{x}) = \sum_{j=1}^{n}\left(\mathbf{x}(t_j)\dot{\phi}(t_j) + \sum_{i=1}^{M}\xi_i f_i(\mathbf{x}(t_j))\phi(t_j)\right)$$

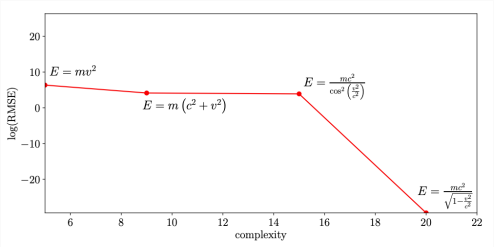
- On écrit sous la forme d' un problème au moindre carré et on résout cela avec STLSQ.

Au dela de Sindy: la régression symbolique

- Sindy est un modèle permettant d'obtenir des EDO analytique.
- Elle nécessite de correctement choisir ses fonctions de bases.
- Pour aller au delà: Régression Symbolique

Régression Symbolique

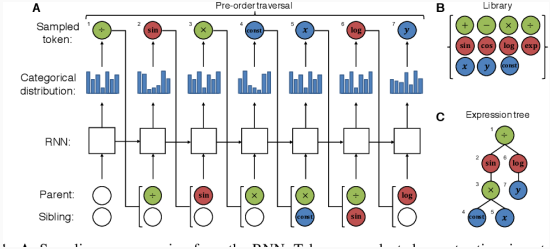
Soir des données $(\mathbf{x}_1, ..., \mathbf{x}_n)$ et $(\mathbf{y}_1, ..., \mathbf{y}_n)$ tel que $\mathbf{y} = f(\mathbf{x}) + \epsilon$. Un algorithme de régression symbolique est un processus qui permet d'obtenir une formule analytique à partir des données.

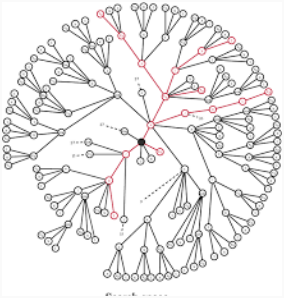


- Logiciel Phi-SO. W. Tenachi. Strasbourg

Régression symbolique I

- Approche Renforcement/algorithme génétique
- Il s'agit de méthodes d'optimisation sans gradient qui vont explorer l'espace des séquences.

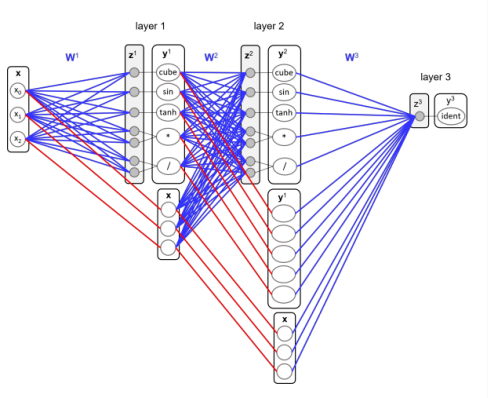




- Transformer/LLM qui relie les données et les séquences.
- Réseaux de neurones et contrainte de parcimonie.

Régression symbolique II

- On met comme fonctions d'activation: les fonctions classiques.
- Cela génère une formule. Pour qu'elle soit compacte on impose la parcimonie.



- Beaucoup de variantes dans le modèle, l'apprentissage, etc
- La contrainte de parcimonie est faible au début puis augmente. Pruning etc.

Apprentissage d’EDO: approche directe

Apprentissage d’EDO: Sindy

Apprentissage d’EDO: ODEnet

Apprentissage d'ODE: ODEnet

Resnet et lien avec les EDO

- Ici on va d'abord proposer une architecture générale de réseau avant d'appliquer ses idées aux EDO.
- On se souvient de l'approche "ResNet" qui consiste à écrit une couche comme:

$$z^{l+1} = z^l + block_\theta(z^l)$$

- L'ajout de l'identité permettait notamment d'éviter des explosions/disparitions de gradient.
- Forme est très proche de la discrétisation d'une EDO ou le réseau correspondrait au flux.
- Avec cette analogie, on voit que mettre plus et plus de couches reviendraient à discrétiser plus finement l'EDO.

Idée

Proposer une architecture avec un nombre arbitraire de couches à l'aide d'EDO.

OdeNet

On se donne une donnée d'entrée $\mathbf{x} \in \mathbb{R}^d$ et une donnée de sortie $\mathbf{y} \in \mathbb{R}^d$. On appelle un Neural ODE ou OdeNet une processus de transformation de la forme:

$$\mathbf{z}(0) = h_{\theta_i}(\mathbf{x}), \quad \mathbf{y} = h_{\theta_e}(\mathbf{z}(T))$$

avec $h_{\theta_i} : \mathbb{R}^d \to \mathbb{R}^m$, $h_{\theta_e} : \mathbb{R}^m \to \mathbb{R}^{d_{out}}$, et $\mathbf{z}(t)$ qui satisfait

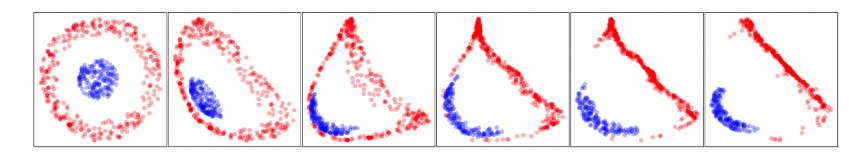
$$\frac{d\mathbf{z}(t)}{dt} = \mathbf{f}_\theta(t, \mathbf{z}(t))$$

une équation différentielle dont le flux $\mathbf{f}_\theta : \mathbb{R}^m \to \mathbb{R}^m$ dépend de paramètres θ.

- f_θ un bloc de réseaux de neurones (MLP ou CNN).
- On parle de **Neural ODE** si $m = d$ et h_{θ_i}. Pas d'augmentation de dimension.
- On parle de **Neural ODE augmentée** si $m > d + d_{out}$, h_{θ_i} et h_{θ_e} des réseaux de neurones. Le second peut contenir le classifieur final.

ODEnet II

- Pour la classification ce type de réseaux revient à apprendre une transformation géométrique régulière qui sera appliquée par la discrétisation de l'EDO et il suffit d'ajouter à la fin un classifieur de type MPC (h_{θ_e}).
- Exemple avec un OdeNet classique;



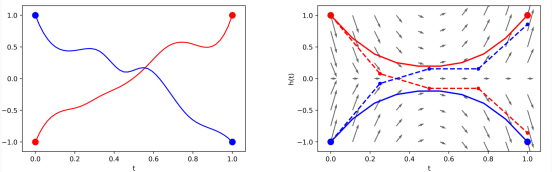
- **Question**: Quel est l'intérêt des EDO augmentée ?
  - On va montrer que les OdeNet sont moins expressive que les OdeNet augmentée
  - En discrétisant on évite ce problème mais cela se traduit par d'importante difficultés numériques.

ODEnet III

- On va se placer dans le cadre $1D$. On chercher a approcher la fonction $g(x) = -x$.
- Elle transforme $x = -1$ en $> y = 1$ et vice-versa.
- Si on decide d'approcher cette transformation par une EDO on doit donc construire une EDO tel qu'il existe deux trajectoires $z_1(t)$ et $z_2(t)$ tel que

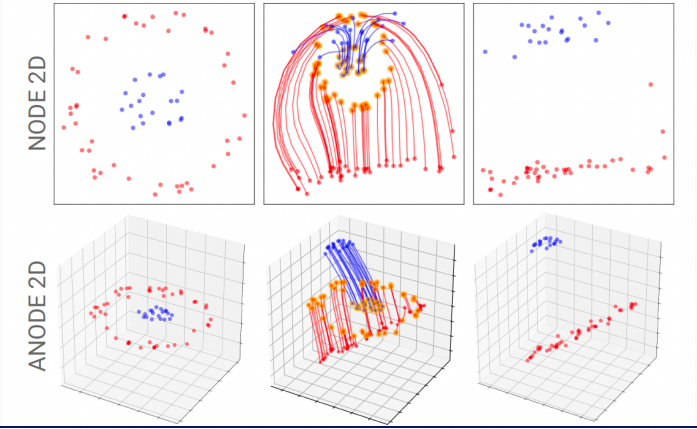
$$z_1(t=0) = -1, \quad z_1(t=1) = 1, \quad \text{et} \quad z_2(t=0) = 1, \quad z_2(t=1) = -1$$

- Construire une EDO générant ce type de trajectoire est impossible. En effet l'unicité des solutions de Cauchy-Lipschitz nous dit que les trajectoires d'une EDO ne peuvent pas ses croiser.
- Pas vrai au niveau discret mais cela se paye par des difficultés d'apprentissage et des couts importants.



Remarque

En augmentant la dimension on peut se ramener un problème ou il y a aura pas besoin de changement topologique dans la transformation.



OdeNet calcul du gradient I

- La fonction de coût qu'on cherche à minimiser est donnée par

$$L(\theta) = \sum_{i=1}^{N_{data}} \| h_{\theta_e}((\mathbf{z}(T))_i) - \mathbf{y}_i \|_2^2$$

  avec $\theta = (\theta_1, ..., \theta_n)$ et $(\mathbf{z}(0))_i = h_{\theta_i}(\mathbf{x}_i)$.
- En pratique il faut maintenant calculer le gradient de notre fonction de coût par rapport à nos entrées $(\mathbf{z}(t), \theta, t)$.

Gradient

Soit $\mathbf{z}(t)$ solution. On se donne le gradient $\frac{\partial L}{\partial \mathbf{z}(T)}$ alors les gradients sont donnés par:

$$\frac{dL}{d\theta} = -\int_T^0 \mathbf{a}(t) \frac{\partial \mathbf{f}(\mathbf{z}(t), t, \theta)}{\partial \theta} dt, \qquad \frac{dL}{d\mathbf{z}(t)} = -\mathbf{a}(t) \frac{\partial \mathbf{f}(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}} dt$$

avec $\mathbf{a}(t)$ solution de:

$$\begin{cases} \dfrac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t) \dfrac{\partial \mathbf{f}(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}} \\ \mathbf{a}(t = 0) = \frac{\partial L}{\partial \mathbf{z}(T)} \end{cases}$$

OdeNet calcul du gradient II

- Preuve:
  - On commence par construire le gradient par rapport à l'état.
  - On définit
$$\mathbf{a}(t) = \frac{dL}{d\mathbf{z}(t)}$$
  - On va montrer formellement que ce nouvel état (appelé état adjoint) satisifait
$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^\top \frac{\partial \mathbf{f}(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}$$
  - Pour ca, on l'équivalent continu de la formule de récurrence qu'on a avec les réseaux a nombre fini de couches.
  - On commence par remarquer que:
$$\mathbf{a}(t) = \frac{dL}{\partial \mathbf{z}(t)} = \frac{dL}{d\mathbf{z}(t+\varepsilon)} \frac{d\mathbf{z}(t+\varepsilon)}{d\mathbf{z}(t)}$$
    et par définition de $\mathbf{a}(t)$
$$\mathbf{a}(t) = \mathbf{a}(t+\epsilon) \frac{d\mathbf{z}(t+\varepsilon)}{d\mathbf{z}(t)}$$
  - A partir de la on peut attaquer le calcul de la dérivée de $\mathbf{a}(t)$

OdeNet calcul du gradient III

- Preuve:

$$
\begin{aligned}
\frac{d\mathbf{a}(t)}{dt} &= \lim_{\varepsilon \to 0^+} \frac{\mathbf{a}(t+\varepsilon) - \mathbf{a}(t)}{\varepsilon} \\
&= \lim_{\varepsilon \to 0^+} \frac{\mathbf{a}(t+\varepsilon) - \mathbf{a}(t+\varepsilon)\frac{\partial}{\partial \mathbf{z}(t)}\mathbf{z}(t+\varepsilon)}{\varepsilon} \\
&= \lim_{\varepsilon \to 0^+} \frac{\mathbf{a}(t+\varepsilon) - \mathbf{a}(t+\varepsilon)\frac{\partial}{\partial \mathbf{z}(t)}\left(\mathbf{z}(t) + \varepsilon \mathbf{f}(\mathbf{z}(t), t, \theta) + \mathcal{O}(\varepsilon^2)\right)}{\varepsilon} \\
&= \lim_{\varepsilon \to 0^+} \frac{\mathbf{a}(t+\varepsilon) - \mathbf{a}(t+\varepsilon)\left(I + \varepsilon \frac{\partial \mathbf{f}(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}(t)} + \mathcal{O}(\varepsilon^2)\right)}{\varepsilon} \\
&= \lim_{\varepsilon \to 0^+} \frac{-\varepsilon \mathbf{a}(t+\varepsilon)\frac{\partial \mathbf{f}(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}(t)} + \mathcal{O}(\varepsilon^2)}{\varepsilon} \\
&= \lim_{\varepsilon \to 0^+} -\mathbf{a}(t+\varepsilon)\frac{\partial \mathbf{f}(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}(t)} + \mathcal{O}(\varepsilon) \\
&= -\mathbf{a}(t)\frac{\partial \mathbf{f}(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}(t)}
\end{aligned}
$$

OdeNet calcul du gradient IV

- Preuve:
  - On va maintenant en déduire le gradient complet. Pour cela on va réécrire une ODEnet de la façon suivante:

$$\frac{d}{dt}\begin{pmatrix} \mathbf{z}(t) \\ \theta(t) \end{pmatrix} = \begin{pmatrix} \mathbf{f}(\mathbf{z}(t), \theta) \\ 0 \end{pmatrix}$$

  - De la même façon on peut introduire

$$\mathbf{a}_{total}(t) = \begin{pmatrix} \mathbf{a}(t) \\ \mathbf{a}_\theta(t) \end{pmatrix}$$

    avec $\mathbf{a}_\theta = \frac{dL}{d\theta(t)}$ Lorsqu'on applique la formule du gradient a l'équation agrandie on obtient:

$$\frac{d\mathbf{a}_{total}(t)}{dt} = -\begin{pmatrix} \mathbf{a}(t)\frac{\partial \mathbf{f}(\mathbf{z}(t),\theta,t)}{\partial \mathbf{z}(t)} \\ \mathbf{a}(t)\frac{\partial \mathbf{f}(\mathbf{z}(t),\theta,t)}{\partial \theta(t)} \end{pmatrix}$$

    On intègre en temps pour obtenir le gradient par rapport aux paramètres θ. Pour obtenir le résultat en partant du principe que $\theta(T) = 0$ (cela revient a dire qu'initialement les poids sont nuls).

OdeNet Algorithme

Dual agrandi

On définit le dual agrandi d'un ODEnet:

$$\frac{d}{dt}\begin{pmatrix}\mathbf{z}(t)\\ \mathbf{a}(t)\\ \theta(t)\end{pmatrix} = F_a\begin{pmatrix}\mathbf{z}(t)\\ \mathbf{a}(t)\\ \theta(t)\end{pmatrix} = \begin{pmatrix}\mathbf{f}(\mathbf{z}(t),\theta,t)\\ -\mathbf{a}(t)\frac{\partial \mathbf{f}(\mathbf{z}(t),\theta,t)}{\partial \mathbf{z}(t)}\\ -\mathbf{a}(t)\frac{\partial \mathbf{f}(\mathbf{z}(t),\theta,t)}{\partial \theta}\end{pmatrix}$$

Algorithme de calcul de gradient

- Soit θ les paramètre courant du réseau.
- On définit l'état initial $s_0 = \left(\mathbf{z}(t_1), \frac{\partial L}{\mathbf{z}(t_1)}, 0_{|\theta|}\right)$.
- On résout l'EDO duale afin de calculer le gradient

$$\left(\mathbf{z}(t_0), \frac{\partial L}{\mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}\right) = \text{OdeSolve}(s_0, F_a, t_0, t_1, \theta)$$

- On renvoit les gradients $\frac{\partial L}{\mathbf{z}(t_0)}$ et $\frac{\partial L}{\partial \theta}$.

Cout mémoire de l'OdeNet

- Si on ne résout pas le duale augmenté mais le dual il faut stocker la trajectoire $\mathbf{z}(t)$.
- Cout mémoire en $O(mT)$ avec T le nombre de pas de temps de couche.

Mémoire OdeNet

La méthode OdeNet utilise la réversibilité de l'EDO (issue de Cauchy-Lipschitz) pour recalculer $\mathbf{z}(t)$ rapidement et éviter le stockage. On a donc un cout mémoire en $O(m)$

- La réversibilité n'est pas immédiate. Il faut EDO suffisamment régulière. Il peut avoir des conditions sur l'ensemble des conditions initiales etc.
- Ce point sera re-discuter plus tard.

Apprentissage d'EDO avec des OdeNet

- On va maintenant revenir a notre problème initiale d'apprentissage d'EDO.
- On se donne un ensemble de trajectoires en temps

$$\mathcal{X} = \left[\mathbf{x}_1^1, ..., \mathbf{x}_1^T, ..., \mathbf{x}_M^1, ..., \mathbf{x}_M^T\right] \in \mathbb{R}^d.$$

- Les approches précédentes d'apprentissage on des défauts:
  - Si l'écart en temps entre deux échantillons est trop grand l'estimation de la dérivée nécessaire à l'apprentissage sera mauvaise.
  - on peut apprendre un flux proche du flux recherché mais qui générera des solutions instables en temps long.

Apprentissage d'EDO par contrôle optimal

L'apprentissage d'EDO par contrôle optimal apprend $\mathbf{f}_\theta$ en résolvant

$$\min_\theta \left(\sum_{i=1}^{M} \sum_{j=1}^{T-1} \| \mathbf{x}_{\theta,i}(t_j) - \mathbf{x}_i^j \|_2^2 \right)$$

avec $\mathbf{x}_{\theta,i}(t)$ solution de

$$\begin{cases} \frac{d\mathbf{x}_{i,\theta}(t)}{dt} = \mathbf{f}_\theta(\mathbf{x}_{i,\theta}(t)) \\ \mathbf{x}_{i,\theta}(t_0) = \mathbf{x}_i^0 \end{cases}$$

Apprentissage d'EDO avec des OdeNet II

- On calcule le gradient continue de la même façon que pour un OdeNet classique.
- Ici la fonction a minimiser qui fait apparaitre plusieurs pas de temps.
- Pour obtenir le gradient on remarque que:

$$\frac{\partial L}{\partial \theta} = \sum_{i=1}^{T} \frac{\partial L}{\partial \mathbf{z}(t_i)} \frac{\partial \mathbf{z}(t_i)}{\partial \theta}$$

Ensuite on applique l'algorithme de calcul de gradient sur chaque intervalle $[t_i, t_{i+1}]$.

Algorithme de calcul de gradient

- Soit θ les paramètre courant du réseau.
- $\forall i \in \{T, ..., 1, 0\}$ et pour chaque trajectoire:
  - On définit l'état initial $s_i = \left(\mathbf{z}(t_i), \frac{\partial L}{\partial \mathbf{z}(t_i)}, \theta\right)$.
  - On résout l'EDO duale: $a_i(t) = \text{OdeSolve}(s_i, F_{a_i}, t_i, t_{i-1}, \theta)$ avec F_{a_i} le flux duale.
- On retourne le gradient $\frac{\partial L}{\partial \theta} = -\sum_{i=1}^{T} \int_{t_i}^{t_{i-1}} \mathbf{a}_i(t)^\top \frac{\partial \mathbf{f}(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}} dt$.

ODEnet et structure

- Si on applique les OdeNet au problème d'apprentissage d'EDO c'est principalement la méthode d'entrainement qui varie par rapport aux deux approches précédentes.

Résumé

On va pas calculer $\mathbf{f}_\theta$ de façon supervisé en estimant une valeur de sortie à l'aide des dérivées discrètes. On va résoudre un problème de contrôle optimal en demandant aux trajectoires issues de $\mathbf{f}_\theta$ d'être le plus proche possible des trajectoires de références.

- En pratique on peut incorporer une structure Hamiltonienne canonique ou non voir Lagrangienne sans que le principe change.

Odenet du second ordre

- Au même titre qu'il y a des EDO du 1er et du second ordre, on peut le faire au niveau OdeNet.
- Au lieu de OdeNet $\frac{d\mathbf{z}(t)}{dt} = \mathbf{f}_\theta(t, \mathbf{z}(t))$ on la remplace par

$$\frac{d}{dt}\mathbf{w}(t) = \frac{d}{dt}\begin{pmatrix} \mathbf{z}(t) \\ \mathbf{y}(t) \end{pmatrix} = \begin{pmatrix} \mathbf{y}(t) \\ \mathbf{f}_\theta(t, \mathbf{z}(t), \mathbf{y}(t)) \end{pmatrix}$$

  et

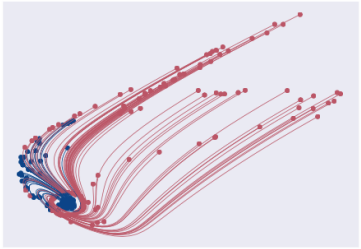
$$\mathbf{w}(t=0) = \begin{pmatrix} \mathbf{z}(t=0) \\ g_{\theta_0}(\mathbf{z}(t=0)) \end{pmatrix}$$

- Il s'agit d'un cas particulier de OdeNet augmentée ou on impose une structure particulière qui rappelle celle des systèmes phyiques.

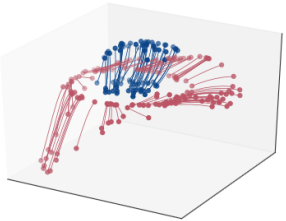
Odenet du second ordre II

- La formulation d'ordre 2 permet d'avoir des croisements de trajectoires.
- les trajectoires sur $\mathbf{w}(t)$ peuvent pas se croiser mais celle sur $\mathbf{z}(t)$

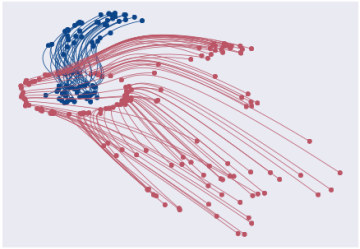
NODE



ANODE(1)



SONODE



Odenet du second ordre III

- On ne détaillera pas les calculs mais l'état adjoint est donné par

$$\ddot{\mathbf{r}} = \mathbf{r}^T \frac{\partial \mathbf{f}}{\partial \mathbf{x}} - \dot{\mathbf{r}}^T \frac{\partial \mathbf{f}}{\partial \dot{\mathbf{x}}} - \mathbf{r}^T \frac{d}{dt}\left(\frac{\partial \mathbf{f}}{\partial \dot{\mathbf{x}}}\right)$$

- Cela nous donne le gradient:

$$\frac{dL}{d\theta} = -\int_{t_n}^{t_0} \mathbf{r}^T \frac{\partial \mathbf{f}}{\partial \theta} dt$$

Conclusion

Les Odenet donne une nouvelle approche intéressante qui sera très importante pour définir des modèles génératifs.

Régression symbolique: exemple

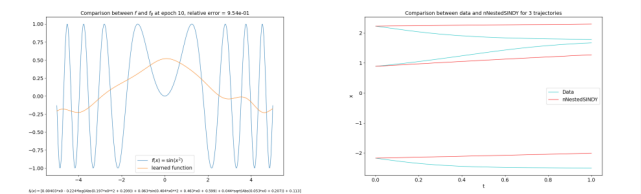
- On propose une exemple simple. Un Sindy a deux couches.
- 1er couche:

$$\phi_1(\mathbf{x};\theta) = \sum_{i=1}^{p} \theta_i P_i(\mathbf{x}), \qquad \text{avec } P_i \text{ des polynômes.}$$

- 2ème couche:

$$\phi_2(\mathbf{x};\theta) = \sum_{i=1}^{p} \theta_i R_i(\mathbf{x}), \qquad \text{avec } R_i \text{ des fonctions nonlinéaires comme: sin, cos, exp.}$$

- On chercher l'EDO $x' = sin(x^2)$. On couple Odenet + Sindy 2 couches



(A) After 10 epochs: $f_\theta(x) = 0.004x - 0.224\log(|0.197x^2 + 0.209|) + 0.063\sin(0.404x^2 + 0.463x + 0.599) + 0.044\sqrt{|0.053x + 0.207|} + 0.113.$

Régression symbolique: exemple

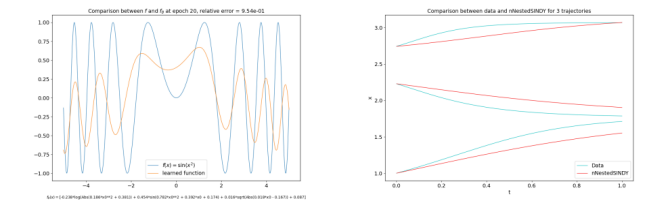
- On propose une exemple simple. Un Sindy a deux couches.
- 1er couche:

$$\phi_1(\mathbf{x};\theta) = \sum_{i=1}^{p} \theta_i P_i(\mathbf{x}), \qquad \text{avec } P_i \text{ des polynômes.}$$

- 2ème couche:

$$\phi_2(\mathbf{x};\theta) = \sum_{i=1}^{p} \theta_i R_i(\mathbf{x}), \qquad \text{avec } R_i \text{ des fonctions nonlinéaires comme: sin, cos, exp.}$$

- On chercher l'EDO $x' = sin(x^2)$. On couple Odenet + Sindy 2 couches



(B) After 20 epochs: $f_\theta(x) = -0.238\log(|0.186x^2+0.381|)+0.454\sin(0.782x^2+0.392x+0.174)+0.016\sqrt{|0.018x-0.167|}+0.087$.

Régression symbolique: exemple

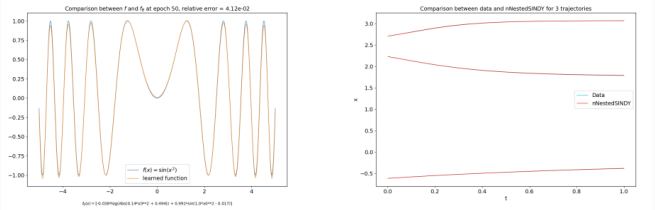
- On propose une exemple simple. Un Sindy a deux couches.
- 1er couche:

$$\phi_1(\mathbf{x};\theta) = \sum_{i=1}^{p} \theta_i P_i(\mathbf{x}), \qquad \text{avec } P_i \text{ des polynômes.}$$

- 2ème couche:

$$\phi_2(\mathbf{x};\theta) = \sum_{i=1}^{p} \theta_i R_i(\mathbf{x}), \qquad \text{avec } R_i \text{ des fonctions nonlinéaires comme: sin, cos, exp.}$$

- On chercher l'EDO $x' = sin(x^2)$. On couple Odenet + Sindy 2 couches



(A) After 50 epochs: $f_\theta(x) = -0.038\log(|0.14x^2 + 0.494|) + 0.991\sin(x^2 - 0.017)$.

Régression symbolique: exemple

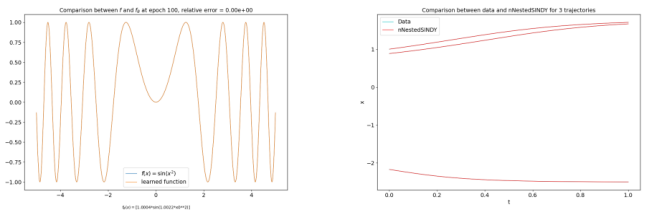
- On propose une exemple simple. Un Sindy a deux couches.
- 1er couche:

$$\phi_1(\mathbf{x};\theta) = \sum_{i=1}^{p} \theta_i P_i(\mathbf{x}), \qquad \text{avec } P_i \text{ des polynômes.}$$

- 2ème couche:

$$\phi_2(\mathbf{x};\theta) = \sum_{i=1}^{p} \theta_i R_i(\mathbf{x}), \qquad \text{avec } R_i \text{ des fonctions nonlinéaires comme: sin, cos, exp.}$$

- On chercher l'EDO $x' = sin(x^2)$. On couple Odenet + Sindy 2 couches



(B) After 100 epochs: $f_\theta(x) = 1.0004\sin(1.0022x^2)$.