Projet Deep Learning

Arnaud FEYEL, Malek BOUZIDI et Noor SEMAAN

March 24, 2024

Contents

1 Résoudre une EDO neuronale

1.1 Rappels de Calculs Différentiels

Supposons que l'on veuille simuler une population d'individus. En d'autres termes, si $N(t)$ est la population t, alors N vérifie

$$\frac{\partial}{\partial t}N(t) = KN(t), \text{ avec } K \in \mathcal{R}$$
$$\implies \quad N(t) = Ce^{Kt}, \text{avec } C \in \mathcal{R}$$

Il est fréquemment plus simple de caractériser une fonction par sa dynamique plutôt que par sa formule explicite. Cela souligne l'importance des équations différentielles.

Néanmoins, la résolution de ces équations peut s'avérer complexe et de multiples défis peuvent apparaître. Notamment, l'existence et l'unicité de la solution ne sont pas toujours garanties, tout comme sa stabilité globale. De plus, si une solution existe, elle n'est pas nécessairement calculable de manière explicite. Prenons par exemple l'équation où N satisfait

$$\left(\frac{\partial}{\partial t}N(t)\right)^3 + y^2 = N(t)y.$$

Une approche couramment employée pour résoudre cette équation est d'utiliser la méthode d'Euler. Cette méthode consiste à discrétiser le problème comme suit :

$$N(t + \Delta t) = N(t) + \sqrt[3]{N(t)y - y^2}\Delta t.$$

En pratique, on utilise plutôt les méthodes de Runge-Kutta.

1.2 Motivations

Parmi tous les réseaux de neurones existants, ce sont les réseaux récurrents de neurones (RNN) sont les plus adaptés pour traiter des données présentant une dépendance temporelle, également appelées séries temporelles.

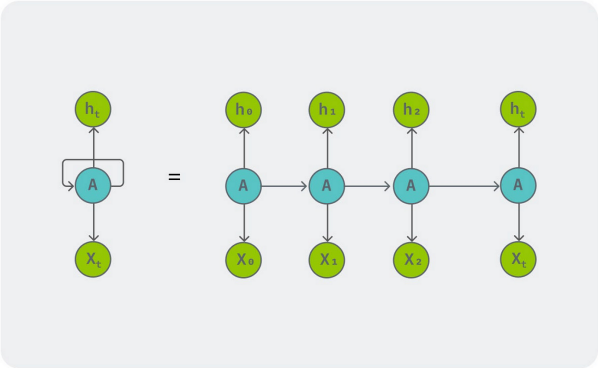

Figure 1: Architecture d'un RNN

Une approche naturelle pour réaliser un apprentissage profond (deep learning) de ces données est d'ajouter un grand nombre de couches au réseau. Pour pallier les problèmes de Gradient Vanishing/Exploding, que nous n'aborderons pas ici, il est courant d'utiliser des réseaux LSTM ou GRU.

Les architectures telles que les RNN construisent des transformations complexes en appliquant séquentiellement des transformations à un état caché dit hidden state.

$$h_{t+1} = h_t + f(h_t, \theta_t)$$

Ici, t varie de 0 à T, et $h_t \in \mathbb{R}^D$. Ces mises à jour étape par étape peuvent être vues comme une discrétisation d'Euler d'une transformation continue. Alors, il est naturel de se demander comment se comporte le réseau avec un grand nombre de couches et des pas plus petits.

Dans notre contexte, nous cherchons à résoudre l'équation suivante :

$$\frac{d}{dt}h(t) = f(h(t), t, \theta) = f_\theta(h(t), t). \tag{1}$$

Nous supposerons que, f est un RNN continu et qu'il satisfait les hypothèses du théorème de Picard. Ainsi, sous ces conditions, résoudre l'équation (2) présente de nombreux avantages. Notamment, cela garantit l'existence et l'unicité de la solution, ainsi que sa stabilité numérique.

1.3 Cadre d'une NEDO

Dans le contexte d'un réseau de neurones, l'objectif est d'évaluer la perte entre la prédiction estimée et le véritable résultat à l'aide d'une fonction de coût. Ensuite, une backpropagation est réalisée afin d'ajuster plus précisément les poids et les biais de chaque couche. La méthode la plus couramment utilisée consiste à effectuer une descente de gradient, définie par l'équation suivante :

$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta L \tag{2}$$

Avec θ les paramètres du réseau et γ le pas d'apprentissage. Pour simplifier, nous supposerons que L est une fonction convexe, ce qui est généralement le cas pour des fonctions de perte telles que MSE, RMSE, etc. Ainsi, le défi principal réside dans le calcul de $\nabla_\theta L$.

Une première approche pourrait consister à utiliser des algorithmes de résolution d'équations différentielles. Plutot que d'utiliser ceux-ci, nous utiliserons la méthode de l'adjoint est souvent préférée.

2 Méthode de l'Adjoint pour Optimiser une NODE

2.1 Approche intuitive

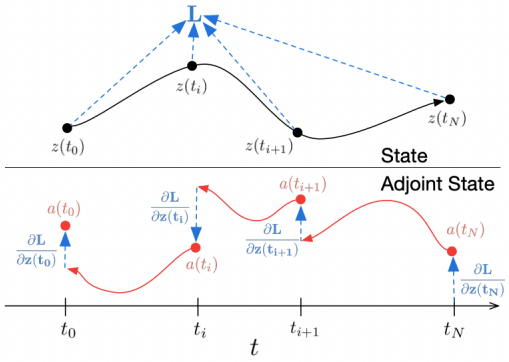

Figure 2: Méthode de l'Adjoint

La méthode de l'adjoint aborde une équation différentielle ordinaire (EDO) augmentée en remontant dans le temps. Ce système augmenté englobe à la fois l'état original et la sensibilité de la perte par rapport à l'état. Lorsque la fonction de perte dépend directement de l'état à divers moments d'observation, l'état adjoint doit être ajusté en fonction de la dérivée partielle de la perte par rapport à chaque observation. Pour l'optimisation de L, les gradients par rapport à θ sont essentiels.

La première étape consiste à établir la relation entre le gradient de la perte et l'état caché $z(t)$ à chaque instant, désigné par l'adjoint $a(t) = \frac{\partial L}{\partial z(t)}$. Ses dynamiques sont définies par une autre EDO, qui peut être interprétée comme une forme continue de la règle de la Chaine Rule :

$$\frac{da(t)}{dt} = -a(t)^T \frac{\partial f(z(t), t, \theta)}{\partial z}$$

Nous pouvons déterminer $\frac{\partial L}{\partial z(t_0)}$ par une invocation supplémentaire d'un solveur d'EDO. Ce solveur doit fonctionner en arrière, en partant de la valeur initiale de $\frac{\partial L}{\partial z(t_N)}$. Une complication survient car la résolution de cette EDO nécessite la connaissance de $z(t)$ sur toute sa trajectoire. Néanmoins, nous pouvons calculer itérativement $z(t)$ et l'adjoint en remontant, en commençant par sa valeur terminale $z(t_N)$. Enfin, le calcul des gradients par rapport aux paramètres θ implique l'évaluation d'une intégrale finale, dépendante à la fois de $z(t)$ et $a(t)$.

2.2 Approche formelle

Soit la fonction z qui décrit la transformation continue au-cours du temps t d'une donnée en input, en fonction d' un réseau de neurone de paramètres θ pour $t \in [0, T]$. On chercher à étudier le comportement de z en fonction du temps, t. En d'autres termes, comment résoudre

$$\frac{\partial}{\partial t} z(t) = f_\theta(z(t), t) \tag{3}$$

Alors, par le théorème fondamental de l'analyse, on a

$$z(T) = z(0) + \int_0^T f_\theta(z(t), t) dt \text{ où } z(0) = x,\ z(T) = \hat{y} \text{ notre prédiction.} \tag{4}$$

Soit L une fonction de coût. Pour la backpropagation à travers le réseau, et sous hypothèse de convexité de L, on utilise la méthode de descente de gradient pour optimiser les paramètres du réseau. En particulier on chercher à calculer

$$\frac{\partial}{\partial \theta} L(z(T), y, \theta).$$

Pour cela, on utilisera la méthode de l'adjoint. Dans un premier temps, montrons que

$$a(t) = \frac{d}{dz(t)}L \implies \frac{d}{dt}a(t) = -a(t)\frac{\partial}{\partial z(t)}f_\theta(z(t), t). \tag{5}$$

Par la Chain Rule "discrète",

$$\frac{dL}{dh_t} = \frac{dL}{dh_{t+1}}\frac{dh_{t+1}}{dh_t}$$

.

Or ici le hidden state $z(t)$ est solution d'une EDO donc nécessairement continue.

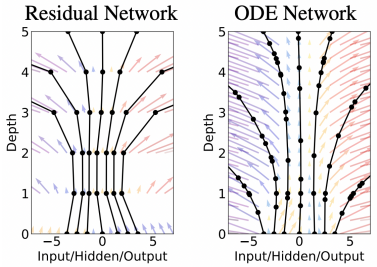

Figure 3: RNN Discret contre NODE Continus

Alors par analogue de la Chain Rule continue,

$$\frac{dL}{dz(t)} = \frac{dL}{dz(t+\epsilon)}\frac{dz(t+\epsilon)}{dz(t)} = \frac{dL}{dT_\epsilon(z(t),t)}\frac{dT_\epsilon(z(t),t)}{dz(t)}. \tag{6}$$

Avec $z(t+\epsilon) = z(t) + \int_t^{t+\epsilon} f_\theta(z(t), t)dt = T_\epsilon(z(t), t)$.
On obtient alors

$$a(t) = a(t+\epsilon)\frac{\partial}{\partial z(t)}T_\epsilon(z(t), t). \tag{7}$$

Par définition de la dérivée :

$$\begin{aligned}
\frac{d}{dt}a(t) &= \lim_{\epsilon\to 0^+} \frac{a(t+\epsilon) - a(t)}{\epsilon} \\
&= \lim_{\epsilon\to 0^+} \frac{a(t+\epsilon) - a(t+\epsilon)\frac{\partial}{\partial z(t)}T_\epsilon(z(t), t)}{\epsilon} \\
&= \lim_{\epsilon\to 0^+} \frac{a(t+\epsilon) - a(t+\epsilon)\frac{\partial}{\partial z(t)}\left[z(t) + \epsilon f_\theta(z(t), t) + \tau(\epsilon^2)\right]}{\epsilon} \\
&= \lim_{\epsilon\to 0^+} \frac{a(t+\epsilon) - a(t+\epsilon)\left[Id + \epsilon\frac{\partial}{\partial z(t)}f_\theta(z(t), t) + \tau(\epsilon^2)\right]}{\epsilon} \\
&= \lim_{\epsilon\to 0^+} \frac{-\epsilon a(t+\epsilon)\frac{\partial}{\partial z(t)}f_\theta(z(t), t) + \tau(\epsilon^2)}{\epsilon} \\
&= -a(t)\frac{\partial}{\partial z(t)}f_\theta(z(t), t) \quad (A).
\end{aligned}$$

Une idée naturelle est de généraliser (A) pour obtenir et gradient de θ et L, de t_0 à t_N.

On pose θ et t comme des états du réseau tels que

$$\frac{\partial}{\partial t}\theta(t) = 0 \text{ et } \frac{d}{dt}t(t) = 1.$$

On combine les 3 paramètres d'un coup pour former un augmented state.

$$\frac{d}{dt}\begin{bmatrix} z \\ \theta \\ t \end{bmatrix}(t) := f_{aug}([z,\theta,t]) = \begin{bmatrix} f([z,\theta,t]) \\ 0 \\ 1 \end{bmatrix}$$

Alors on définit

$$a_{aug} := \begin{bmatrix} a \\ a_\theta \\ a_t \end{bmatrix} \text{ avec } a_\theta = \frac{d}{d\theta(t)}L,\ a_t = \frac{d}{dt(t)}L. \tag{8}$$

$$J(f_{aug}) = \frac{\partial f_{\text{aug}}}{\partial[z,\theta,t]} = \begin{bmatrix} \frac{\partial f}{\partial z} & \frac{\partial f}{\partial \theta} & \frac{\partial f}{\partial t} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}(t) \tag{9}$$

On réutilise (A) appliqué à (6),

$$\frac{d}{dt}a_{aug}(t) = -\begin{bmatrix} a(t) & a_\theta(t) & a_t(t) \end{bmatrix} J(f_{aug}) = -\begin{bmatrix} a\frac{\partial}{\partial z}f & a\frac{\partial}{\partial \theta}f & a\frac{\partial}{\partial t}f \end{bmatrix}(t)$$

On retrouve alors (A) dans la première coordonnée de $\frac{d}{dt}a_{aug}(t)$ et on cherche le gradient de la deuxiement coordonnée.

$$\frac{d}{d\theta}L = a_\theta(t_0) = -\int_{t_N}^{t_0} \frac{\partial}{\partial \theta}f(z(t),t,\theta)dt. \tag{10}$$

Il ne reste alors plus qu'à optimiser le réseau.

2.3 Discussions et limites du modèle

Lors de la modélisation de dynamiques continues en utilisant des réseaux de neurones, la question de l'unicité de la solution se pose. Le théorème d'existence de Picard établit que la solution à un problème de valeur initiale existe et est unique si l'équation différentielle est uniformément Lipschitz continue par rapport à z et continue par rapport à t. Cette condition est satisfaite pour notre modèle si le réseau de neurones a des poids finis et utilise des fonctions d'activations non-linéaires et Lipschitz, telles que tanh ou relu. Alors, il est naturel de se questionner sur le comportement du réseau avec des fonctions d'activations non Lipschitz voir non continues. En pratique, la plupart des fonctions d'activations sont Lipschitz et non linéaires. Alors, on peut aussi se demander si le choix de telles fonctions impacte la convergence du réseaux vers la solution.

Cette propriété d'unicité est essentielle pour garantir que notre modèle de réseau de neurones offre des solutions cohérentes et fiables aux problèmes de dynamique continue. Elle renforce la robustesse et la stabilité de notre approche hybride, combinant la rétropropagation à travers les équations différentielles et la méthode adjointe pour l'entraînement des réseaux de neurones.

En outre, il est important de noter que l'approche de l'adjointe offre une alternative efficace à la rétropropagation traditionnelle à travers les solveurs d'ODE, évitant ainsi les erreurs potentielles et les coûts de calcul élevés. Cette méthode permet d'obtenir des gradients précis et fiables, même dans des situations où la fonction de coût dépend de manière complexe des états intermédiaires.

En conclusion, l'utilisation de l'approche hybride combinant la rétropropagation à travers les ODE et la méthode adjointe représente une avancée significative dans le domaine de l'entraînement des réseaux de neurones pour les problèmes de dynamique continue. Elle garantit non seulement des solutions uniques et stables, mais ouvre également la voie à des méthodes d'optimisation plus efficaces et précises, offrant ainsi de nouvelles perspectives pour des applications où la précision et la fiabilité de l'entraînement sont cruciales.

3 Applications

3.1 Contexte

Dans cette partie, nous étudions l'article Neural-ODE for pharmacokinetics modeling de James Lu, Kaiwen Deng, Xinyuan Zhang, Gengbo Liu, et Yuanfang Guan.
Dans cet article, on utilise les méthodes de modélisation par Neural-EDO et on compare ces résultats avec les méthodes usuelles. Cette article est la premiere application des méthodes de NEDO.

3.2 Contexte et Enjeux

En recherche pharmaceutique, l'étude de la pharmacocinétique (PK) de chaque patient est fondamentale. On définit cela comme l'étude de la réaction du corps après l'administration d'une substance, pour toute la durée de l'exposition.

Naturellement, ce type de simulations, que l'on désigne par "PK", constitue un axe crucial dans l'industrie pharmaceutique. L'un des problèmes clés est que les modèles PK, construits selon les méthodes classiques des réseaux neuronaux récurrents (RNN), fonctionnent bien pour un dosage constant. Cependant, leur généralisation pour des dosages différents est très mauvaise. C'est pourquoi cet article utilise des Neural EDO qui se comportent bien mieux lors de la généralisation des modèles de prédictions.

Cette étude est la première à utiliser les Neural EDO pour simuler des PK.

3.3 Dataset et Méthodes

Déterminer le bon dosage et le moment approprié de l'administration pour chaque patient est un enjeu fondamental en recherche pharmaceutique, et cela a permis un grand nombre d'applications dans le développement des médicaments. De manière classique, on utilise les régressions linéaires (LR), la régularisation Lasso, et les forêts aléatoires (RF) pour générer ces modèles. Par exemple, on a réussi à déduire le bon dosage de remifentanil (un anesthésique) dans une population saine.

Dans cette étude, nous utilisons un ensemble de données de pharmacocinétique (PK) du trastuzumab emtansine (T-DM1) : un médicament contre le cancer du sein. Le PK de ce traitement est bien caractérisé et décrit par un modèle linéaire de PK. En pratique, le T-DM1 possède une fenêtre thérapeutique étroite et a été déterminée après des essais à petites doses sur les patients afin de déduire les meilleurs dosages. Deux modèles ont été établis : le premier avec un seul dosage toutes les 3 semaines (Q3W), et le deuxième avec un dosage hebdomadaire (Q1W).

En raison de la nature du traitement, l'ensemble de données de Q1W est beaucoup plus important que celui de Q3W. Ces différences nous permettront de déterminer dans quelle mesure l'extrapolation du réseau est efficace.

Les modèles suivants ont été utilisés.

1. Neural-ODE
2. LSTM Neural Network (outil classique)
3. LightGBM (version plus récente du XGBoost).

Finalement, la performance de chaque modèle est évaluée en testant sa capacité à prédire les concentrations PK chez les patients traités avec le régime opposé.

3.4 Résultats

Les réseaux neuronaux à équations différentielles ordinaires (Neural-ODE), LightGBM et LSTM présentent des performances similaires lorsque les données d'entraînement et de test proviennent des mêmes régimes de traitement. Cependant, les réseaux neuronaux à équations différentielles ordinaires surpassent les autres algorithmes lorsqu'il s'agit de généraliser à de nouveaux régimes.

L'avantage du modèle de réseau neuronal à équations différentielles ordinaires (Neural-ODE) peut résulter de sa capacité à traiter des points de données échantillonnés de manière inégale. Les modèles d'apprentissage profond alternatifs tels que les LSTM ont tendance à supposer un échantillonnage régulier. Neural-ODE

intègre directement les données de dosage et de timing dans le modèle au stade du décodeur, ce qui contribue probablement également à la performance stable lors de l'adoption d'un modèle entraîné sur un régime de traitement différent à un autre régime de traitement.

4 Normalizing Flows et Continuous Normalizing Flows

4.1 Normalizing Flows

L'équation discrétisée $h_{t+1} = h_t + f(h_t, \theta_t)$ apparaît également dans le concept de Normalizing Flows.

On appelle Normalizing Flows la méthode permettant de construire des distributions complexes en transformant une densité de probabilité à travers une série de transformations inversibles.

Cette méthode utilise le théorème de changement de variable pour calculer les changements exacts de probabilité si les échantillons sont transformés à travers une fonction bijective f.

$$z_1 = f(z_0) \Rightarrow \log p(z_1) = \log p(z_0) - \log \left| \det \frac{\partial f}{\partial z_0} \right|$$

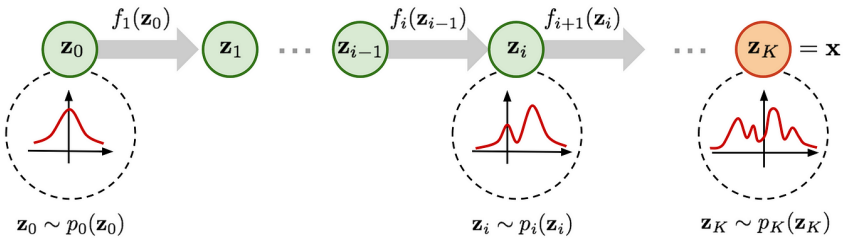

Figure 4: Normalizing Flows

Exemple: Planar Normalizing Flows

$$z(t+1) = z(t) + uh(w^T z(t) + b),$$

$$\log p(z(t+1)) = \log p(z(t)) - \log \left| 1 + u^T \frac{\partial h}{\partial z} \right|$$

En général, le principal obstacle à l'utilisation de la formule de changement de variables est le calcul du déterminant du Jacobien

$$\frac{\partial f}{\partial z}$$

qui a un coût cubique soit en fonction de la dimension de z, soit en fonction du nombre d'unités cachées.

Passer d'un ensemble discret de couches à une transformation continue simplifie le calcul du changement de constante de normalisation.

4.2 Introduction aux Continuous Normalizing Flows

Objectif : Supposons que nous observons une distribution $\mathbb{P}$ de densité π définie sur $\mathbb{R}^{d_1 \times \cdots \times d_k}$. Le but est de trouver une approximation de la distribution $\mathbb{P}$.

Exemple d'application : Utiliser un modèle génératif qui approximite π pour produire des images synthétiques des chats. Dans cet exemple, $\mathbb{R}^{d_1 \times \cdots \times d_k} = \mathbb{R}^{3 \times 32 \times 32}$ et $\mathbb{P}$ peut désigner une distribution de probabilité sur les 'images de chats', à partir de laquelle nous disposons d'échantillons empiriques.

Considérons l'EDO neuronale aléatoire suivante :

$$y(0) \sim \mathcal{N}(0, I_{d \times d}), \quad \frac{dy(t)}{dt} = f_\theta(t, y(t)) \quad \text{pour} \quad t \in [0, T] \quad (1)$$

Où $d = \prod_{k=1}^{m} d_k$ (pour simplifier les notations).

Le but est d'entrainer le modèle tel que la distribution de $y(1)$ est approximativement $\mathbb{P}$. C'est ce qu'on appelle **Continuous normalizing flows(CNF)**.

4.2.1 Changement de variables instantané

On utilise le maximum de vraisemblance pour entrainer le modèle, i.e on a besoin d'une expression traitable pour la densité de la distribution de y(1):

Théorème: Changement de variables instantané
Supposons que $f_\theta = (f_{\theta,1}, \ldots, f_{\theta,d})$ est Lipschitz-continue. Soit $p_\theta : [0,T] \times \mathbb{R}^d \to \mathbb{R}$, où $p_\theta(t,\cdot)$ est la densité de $y(t)$ pour chaque instant $t \in [0,T]$. L'indice θ dans p_θ indique la dépendance de f_θ. Alors p_θ évolue selon l'équation différentielle :

$$\frac{d}{dt}\log p_\theta(t, y(t)) = -\sum_{k=1}^{d} \frac{\partial f_{\theta,k}(t, y(t))}{\partial y_k} = -Tr(J(f)) \quad (2)$$

où $y = (y_1, \ldots, y_d) \in \mathbb{R}^d$

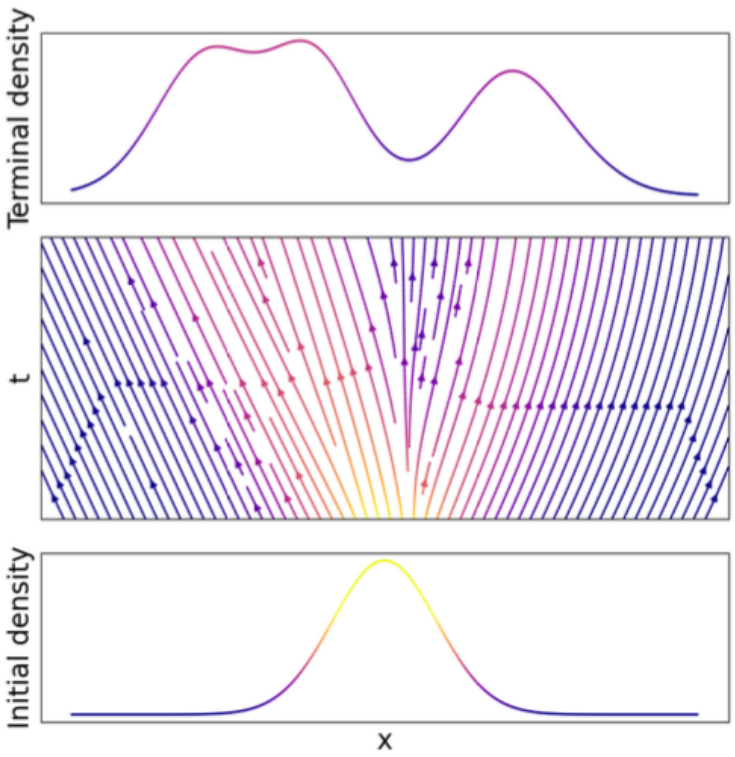

Figure 5: Continuous Normalizing Flows déforme continuellement une distribution en une autre distribution. Les lignes de flux montrent comment les particules de la distribution de base sont perturbées jusqu'à ce qu'elles approximent la distribution cible.

Remarque: Au lieu de log determinant, nous avons maintenant seulent besoin d'une opération Trace. De plus, l'équation différentielle f n'a pas besoin d'être bijective (contrairement au NF), car si l'unicité est satisfaite, alors toute la transformation est automatiquement bijective.

Entraînement: En résolvant l'équation ci-dessus, on peut entraîner un CNF via la méthode du maximum de vraisemblance.

Étant donné une condition terminale quelconque $x \in \mathbb{R}^d$, soit $y(t, x)$ la solution de l'EDO

$$y(T, x) = x, \quad \frac{dy(t, x)}{dt} = f_\theta(t, y(t, x)) \quad \text{pour} \quad t \in [0, T] \quad (3)$$

qui sera résolue rétroactivement(backwards-in-time) depuis $t = T$ jusqu'à $t = 0$.

Étant donné un lot d'échantillons empiriques $y_1, \ldots, y_N \in \mathbb{R}^d$, le maximum de vraisemblance stipule qu'en ce qui concerne θ, nous devrions minimiser $-\frac{1}{N}\sum_{i=1}^{N} - \log p_\theta(T, y_i)$.
En intégrant entre 0 et T et en substituant dans (2), nous obtenons

$$-\frac{1}{N}\sum_{i=1}^{N} \log p_\theta(T, y_i) = -\frac{1}{N}\sum_{i=1}^{N}[\log p_\theta(0, y(0, y_i)) - \int_0^T \sum_{k=1}^{d} \frac{\partial f_{\theta,k}}{\partial y_k}(t, y(t, y_i))] \quad (4)$$

À partir d'un échatillon $y_i \in \mathbb{R}^d$, on peut résoudre l'équation (3) rétroactivement.

On obtient progressivement $y(t, x)$ pour $t = T$ à $t = 0$. Ceci est un "input" pour la partie droite de l'équation (4). L'intégrale peut être résolue comme une partie de la résolution rétrograde dans le temps (backwards-in-time solve). On a juste besoin de la concatener avec l'équation (3) pour former un système d'équations différentielles.

Enfin, on évalue $\log p_\theta(0, y(0, y_N))$ - Rappel : $p_\theta(0, .)$ suit une distribution normale [équation (1)] - et on l'ajoute à la valeur de l'intégrale afin d'obtenir le résultat de (4).

Ainsi, l'équation (4) est rétroactivée et le paramétre θ est mis à jour grace à la méthode de la descente de gradient.

4.2.2 Exemple

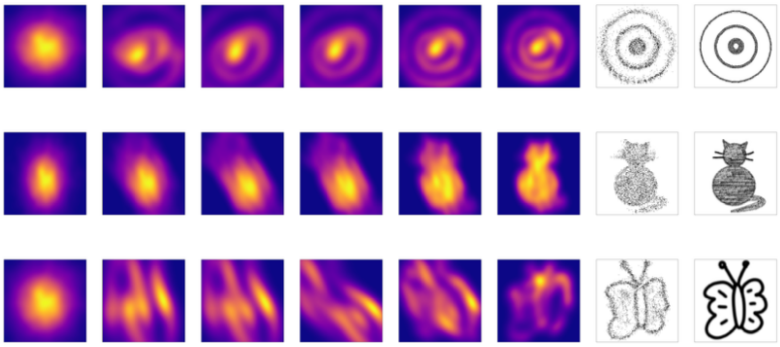

Figure 6: Exemple de CNF(cible, chat et papillon)

Les six premières images montrent l'évolution de la distribution du CNF de t = 0 à t = T, transformant une distribution normale en la distribution souhaitée.
L'avant-dernière image montre des échantillons provenant du CNF appris.
La dernière image montre l'image utilisée pour spécifier la distribution souhaitée.

Grâce à l'exemple, on peut voir comment CNF est capable de lire des images 2D complexes (cible, chat et papillon). Il s'agit de 3 images de styles différents. Ceci est une preuve de la flexibilité de CNF.

4.2.3 Estimation efficace de la trace du Jacobien

Les équations (2) et (4) sont basées sur l'expression $\sum_{k=1}^{d} \frac{\partial f_{\theta,k}(t,y)}{\partial y_k}$ qui est évalué via la distribution d'autodifférentiation. Cependant, l'autodifferentiation calcule le produit des Jacobiens ce qui n'est pas le cas ici. Cela nécessiterait des opérations répétées coûteuses pour chaque composante. Donc, on peut utiliser d'autres méthodes :

Estimateur de trace de Hutchinson

Soit $A \in \mathbb{R}^{d\times d}$ une matrice quelconque. Soit ε une variable aléatoire sur $\mathbb{R}^d$ telle que $\mathbb{E}[\varepsilon] = 0 \in \mathbb{R}^d$ et $\mathrm{Cov}[\varepsilon] = I_{d\times d}$. Alors,

$$\mathrm{tr}(A) = \mathbb{E}_\varepsilon[\varepsilon^\top A\varepsilon].$$

L'approximation de Monte-Carlo dérivée de cette équation est connue sous le nom d'estimateur de trace de Hutchinson.

Le Trace-Jacobian

Le fait que le membre de droite de l'équation (2) soit un Trace-Jacobian se révèle maintenant utile. Nous avons que

$$\sum_{k=1}^{d} \frac{\partial f_{\theta,k}(t,y)}{\partial y_k} = \mathrm{tr}\left(\frac{\partial f_\theta(t,y)}{\partial y}\right) = E_\varepsilon\left[\varepsilon^\top \frac{\partial f_\theta(t,y)}{\partial y}\varepsilon\right].$$

En substituant dans l'équation (4), nous obtenons

$$\frac{1}{N}\sum_{i=1}^{N} \log p_\theta(T, y_i) = -\frac{1}{N}\sum_{i=1}^{N}\left[\log p_\theta(0, y(0, y_i)) - E_\varepsilon\left[\int_0^T \varepsilon^\top \frac{\partial f_\theta}{\partial y}(t, y(t, y_i))\varepsilon\, dt\right]\right].$$

4.3 Comparaison entre CNF et NF

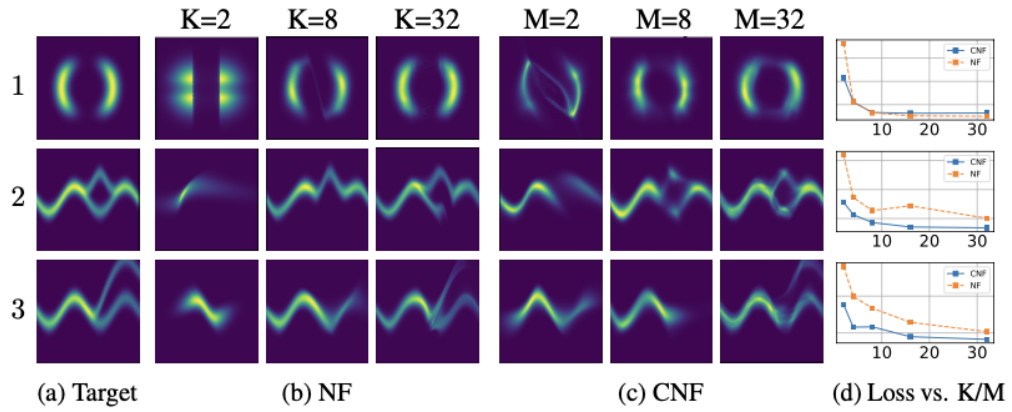

Figure 7: Comparaison entre NF et CNF. La capacité du modèle des NF est déterminée par leur profondeur (K), tandis que les CNF peuvent également augmenter leur capacité en augmentant leur largeur (M), ce qui les rend plus faciles à entraîner

NF	CNF
Worst-Case Cost : $O(D^3)$	Worst-Case Cost : $O(D^2)$
Nécessite f inversible	Ne nécessite pas f inversible

Table 1: Tableau de comparaison

5 EDOs Latentes

5.1 Introduction

Les EDO latentes (équations différentielles ordinaires) combinent des réseaux de neurones avec des équations différentielles pour modéliser des processus dynamiques. Elles introduisent des variables latentes, représentant des états cachés, dans les formulations traditionnelles d'ODE. Ces modèles largement basés sur des Autoencodeurs Variationnels (VAE) captent des dépendances temporelles complexes et permettent d'apprendre des systèmes dynamiques à partir de données, offrant des représentations flexibles des phénomènes en évolution dans le temps. Les ODE latentes trouvent des applications dans divers domaines tels que la physique, la biologie et l'apprentissage automatique, où elles excellent dans la modélisation et la prévision de processus dynamiques, tels que le suivi de trajectoires, l'analyse de données chronologiques et la compréhension de motifs temporels.

5.2 Méthodologie

5.2.1 Cadre

On considère l'espace de séries temporelles d-dimensionnelles irrégulièrement échantillonné

$$TS(\mathbb{R}^2) = \{((t_0, x_0), ..., (t_n, x_n)) \mid n \in \mathbb{N}, t_j \in \mathbb{R}, x_j \in \mathbb{R}^d, t_0 = 0, t_j < t_{j+1}\}$$

Pour simplifier, on considère qu'il n'y a pas de données manquantes - donc cas entièrement observé. Cependant, les constructions ci-dessous s'étendent également au cas partiellement observé.

Pour un certain $x \in \mathbb{R}^{d_m}$, soit $y_0 = g_\theta(z) \in \mathbb{R}^{d_t}$ et soit $y(t, y_0)$ la solution de l'EDO neuronale:

$$y(0, y_0) = y_0 \qquad \frac{dy}{dt}(t, y_0) = f_\theta(y, (t, y_0)) \tag{11}$$

On considère la distribution $p_{\theta, y(t, y_0)}$ pour chaque temps t.

5.2.2 Autoencodeur Variationnel

Les VAE sont une architecture de réseaux de neurones entraînés de manière non-supervisée. L'encodeur du VAE va générer $\mathbb{E}(z)$ et $\mathbb{V}(z)$, tel que $z \sim \mathcal{N}(\mathbb{E}[z], \mathbb{V}(z))$. Par la suite, un échantillonnage de $\mathbb{E}(z)$ et $\mathbb{V}(z)$ va permettre au VAE de générer le vecteur latent z. Ce dernier est ensuite donné en entrée (*input*) au décodeur afin de générer des données x' correspondant aux données séries temporelles d'entrée x.

Le but est donc de reconstruire les données d'entrées avec un maximum de précision. Par conséquent, on cherche souvent à minimiser une fonction de perte d'où un critère d'optimisation *end-to-end*.

La Figure 1 nous permet de visualiser la structure du VAE qui sera détaillée par la suite.

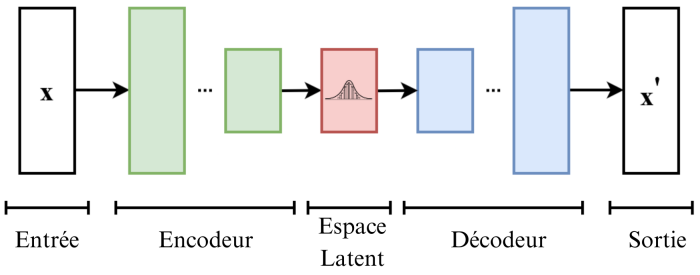

Figure 8: Schéma de la structure d'une Auto-Encodeur Variationnel

Afin d'illustrer cette architecture, on considère un exemple où l'échantillon est de taille 250 et l'espace latent est de taille 25.

5.2.3 Encodeur

La Figure 2 schématise l'architecture de l'encodeur.

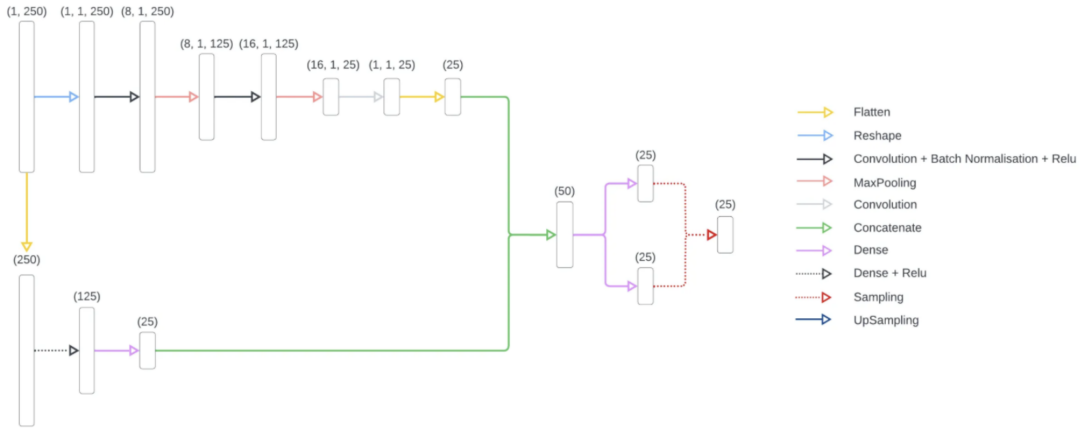

Figure 9: Architecture de l'Encodeur sur un échantillon de taille 250

L'encodeur du VAE est une certaine fonction $\nu_\theta : TS(\mathbb{R}^d) \to \mathbb{R}^{d_m} \times (0, \infty)^{d_m}$; souvent choisi comme un RNN.

Il est formé d'une partie convolutionnelle et d'une partie dense.

La partie convolutionnelle est composée d'une succession de convolutions chacune accompagnée d'une *batch normalization* suivie par une fonction d'activation ReLU et finalement par un MaxPooling. Cette boucle est accompagnée d'une convolution simple pour enfin finir avec un Flatten. Cette partie résulte en un vecteur de taille 25.

```
# Convolution
    x_conv = x.unsqueeze(0).unsqueeze(0)
    x_conv = F.relu(self.convbatchNorm1(self.conv1(x_conv)))
    x_conv = self.maxPool1(x_conv)
    x_conv = F.relu(self.convbatchNorm2(self.conv2(x_conv)))
    x_conv = self.maxPool2(x_conv)
    x_conv = self.conv3(x_conv)
    x_conv = x_conv.view(25)
```

De plus, la partie dense consiste en une couche dense avec fonction d'activation ReLU. Ensuite, une couche simple est appliquée et on obtient un vecteur, également de taille 25.

```
# Dense
    x = x.flatten()  # pour avoir shape (250)
    x_dense = F.relu(self.dense1(x))
    x_dense = self.dense2(x_dense)
```

Ces deux vecteurs sont ensuite concaténés pour en former un nouveau de taille 50. On lui applique deux couches denses simples ce qui résulte à nouveau en deux vecteurs de taille 25.

```
# Concatenation
    x_concat = torch.cat((x_conv, x_dense), 0)
    mu = self.concatDense1(x_concat)  # moyenne
    var = torch.exp(self.concatDense2(x_concat))  # variance
```

Ces vecteurs équivalent la moyenne μ et écart-type σ. Ces derniers permettent l'échantillonnage du vecteur latent.

```
# Vecteur latent
    z = mu + var * self.N.sample(mu.shape)
    self.kl = 0.5 * torch.sum(var - torch.log(var) - 1 + mu ** 2)  # divergence KL
```

Output: statistique d'une distribution normale multivariée avec une covariance diagonale qu'on note $q_{\theta,x} = N(\mu_{\theta,x}, diag(\sigma^2_{\theta,x}))$ with $(\mu_{\theta,x}, \sigma_{\theta,x}) = \nu_\theta(x)$.

Si l'encodeur est un RNN, on peut l'exécuter en arrière (*backwards*) dans le temps sur TS. Dans ce cas, le décodeur commence donc là où se termine l'encodeur.

5.2.4 Décodeur

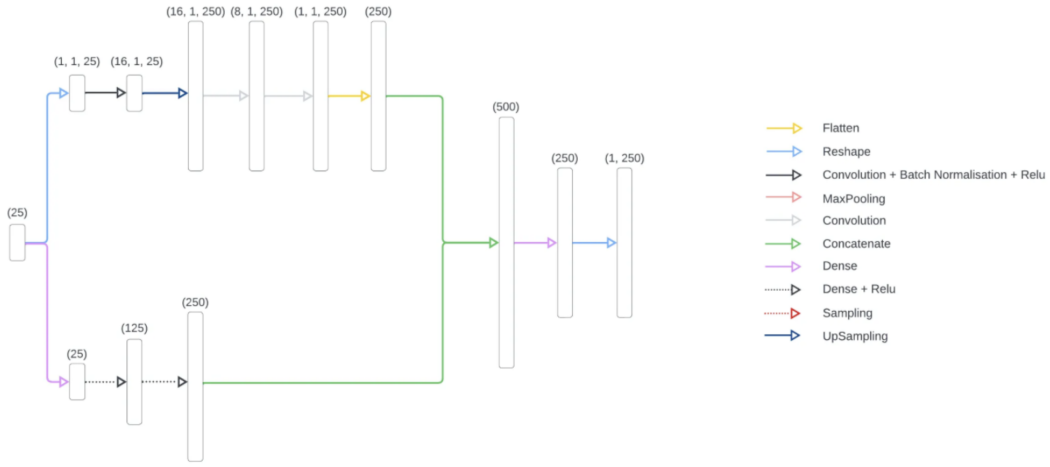

Figure 10: Architecture du Décodeur sur un échantillon de taille 250

Le décodeur a une structure similaire à celle de l'encodeur; il est également formé d'une partie convolutionnelle et d'une partie dense. Il prend en entrée le vecteur latent résultant de l'encodeur.

La partie convolutionnelle est formée d'une convolution suivie d'une *batch normalization* et l'application d'une ReLU. Par la suite, on insert des échantillons à valeur nulle entre les échantillons originaux afin d'augmenter le taux d'échantillonnage. On appelle cette étape échantillonnage suraugmenté (*upsampling*). Dans le cadre de l'exemple choisi, elle permet de revenir à une taille de 250. On applique ensuite deux convolutions et un Flatten pour obtenir un vecteur de taille 250.

```
# Couches convolutives
    z_conv = z.reshape((1, 25)).unsqueeze(0).unsqueeze(0)
    z_conv = F.relu(self.convbatchNorm1(self.conv1(z_conv)))
    z_conv = self.upsampling1(z_conv)
    z_conv = self.ConvUp1(z_conv)
    z_conv = self.ConvUp2(z_conv)
    z_conv = z_conv.flatten()
```

La partie dense commense par l'application d'une couche dense. Par la suite, deux couches denses sont appliquées suivie chacune d'une ReLU. Un Flatten est finalement appliqué résultant en un vecteur de taille 250.

```
# Couches denses
    z_dense = self.dense1(z)
    z_dense = F.relu(self.dense2(z_dense))
    z_dense = F.relu(self.dense3(z_dense))
    z_dense = z_dense.flatten()
```

Ensuite une concaténation des deux vecteurs résulte en un vecteur de taille 500.

```
# Concaténation
    z_dense = torch.cat((z_conv, z_dense), 0)
```

La seconde partie dense est composée de deux couches denses avec ReLU.

```
# Secondes couches denses
    z_final = self.concatDense1(z_dense)
    z_final = z_final.reshape(1, 250)
```

Afin de formaliser ce concept, soient $d_l, d_m > 0$ les dimensions de deux espaces latents.

Soit θ le vecteur des paramètres appris. Soient:

$$f_\theta : \mathbb{R}^{d_l} \to \mathbb{R}^{d_l},$$

$$g_\theta : \mathbb{R}^{d_m} \to \mathbb{R}^{d_l}$$

deux réseaux de neurones paramétrisés par θ.

Soit $p_{\theta,y} : \mathbb{R}^{d_l} \to [0, \infty)$ une densité de probabilité paramétrisée par $y \in \mathbb{R}^{d_l}$ et θ.

Pour un certain $z \in \mathbb{R}^{d_m}$, soit $y_0 = g_\theta(z) \in \mathbb{R}^{d_l}$ et soit $y(t, y_0)$ la solution de la *neural ODE* (1). En remplaçant y_0 par $g_\theta(z)$, on obtient une expression dont les composantes sont paramétrisées par θ

$$y(0, g_\theta(z)) = g_\theta(z) \qquad \frac{dy}{dt}(t, g_\theta(z)) = f_\theta(y(t, g_\theta(z))) \tag{12}$$

On considère $p_{\theta,y(t,y_0)}$ pour chaque temps t.

Output: l'ensemble des $t \to p_{\theta,y(t,y_0)}$.

C'est-à-dire que, pour un certain $z \in \mathbb{R}^{d_m}$, le décodeur est appliqué à l'espace latent $\mathbb{R}^{d_l}$ à partir duquel l'EDO évolue. A chaque temps t, la valeur latente paramétrise une distribution de probabilité.

5.2.5 Entraînement

Comme chaque modèle de réseaux de neurones, on doit entraîner le VAE de manière optimale. En effet, pour un certain batch ou dataset d'une série temporelle $x_1, ..., x_N \in TS(\mathbb{R}^d)$, le critère d'optimisation *end-to-end* est de trouver θ minimisant la fonction de perte

$$\frac{1}{N} \sum_{i=1}^{N} [\mathbb{E}_{y_0 \sim q_{\theta,x_i}}[-log p_{\theta,y_0}(x_i)] + KL(q_{\theta,x_i} || N(0, I_{d_l \times d_l}))]$$

où KL représente la divergence Kullback-Leider - qui sert à minimiser la distance entre q et p - dont l'expression est

$$KL(q||p) = \int_x q(x) log \frac{q(x)}{p(x)} dx$$

C'est le critère d'optimisation standard de la VAE. Pour $p_{\theta,y}$ raisonnable (par exemple Gaussienne), cette expression peut être évaluée par backpropagation.

Le premier terme du critère d'optimisation garantit que le décodeur apprenne à répliquer les échantillons donnés en input. Le deuxième terme garantit que la distribution initiale dans l'espace latent correspond à une distribution connue.

5.2.6 Echantillonnage

L'échantillonnage à partir du modèle est assez simple. On échantillonne un certain $z \sim \mathcal{N}(0, I_{d_m \times d_m})$ pour ensuite évaluer $y_0 = g_\theta(z)$. Par la suite, on évalue l'équation (1) *forward* dans le temps.

Output: $p_{\theta,y(t,y_0)}$; si une statistique ponctuelle est requise alors la moyenne de $p_{\theta,y(t,y_0)}$ peut être renvoyée.

Les inputs ne sont pas régulièrement espacées et différents éléments du lot ne sont pas échantillonnés aux mêmes moments. On parle donc d'échantillonnage irrégulier.

Cependant, l'ouptput s'étend sur l'intervalle de temps continu [0,12] de telle sorte que nous obtenons des échantillons à tout moment (différent du RNN analogue, qui se limiterait à produire des sorties uniquement à des timestamps discrets prédéfinis).

De plus, le choix de la densité de probabilité $p_{\theta,y(t,y_0)}$ est dépendant du comportement cherché pendant l'échantillonnage durant le temps d'inférence.

Remarque: en apprentissage automatique, le temps d'inférence fait référence à la période pendant laquelle un modèle entraîné effectue des prédictions ou génère des résultats en fonction de nouvelles données d'entrée non vues. C'est la phase au cours de laquelle le modèle applique ses connaissances acquises pour prendre des décisions ou fournir des résultats.

Si seul une statistique ponctuelle est requise, alors le choix de $p_{\theta,y(t,y_0)}$ est un choix de fonction de coût (Gaussienne, Laplace...).

Si l'output complet du modèle $p_{\theta,y(t,y_0)}$ est demandé, alors des choix plus expressifs de la $p_{\theta,y}$ sont intéressants (par exemple *normalising flow*).

5.2.7 Extrapolation

L'EDO latente reproduit bien les données et montre de bonne qualités d'extrapolation sur un intervalle quatre fois plus longs que l'intervalle de son entraînement.

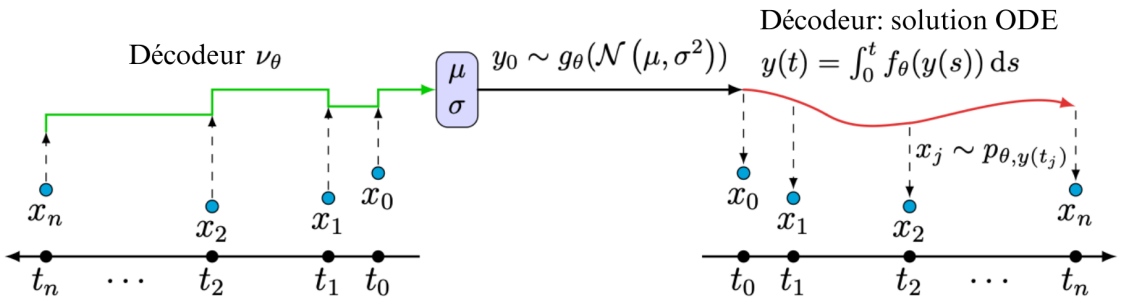

Figure 11: Aperçu général du modèle latent ODE

5.3 Application - Oscillateurs à amplitudes décroissantes

Afin d'illustrer la théorie, prenons un exemple relativement simple: un ensemble de données d'oscillateurs à amplitudes décroissantes. On considère une série temporelle bidimensionnelle composée (d'observations discrètes) de:

$$y(t) = exp(At)y_0$$

avec $y_0, y(t) \in \mathbb{R}^2$, $A \in \mathbb{R}^{2\times 2}$ et telle que les valeurs propres de A soient complexes avec des composantes réelles négatives. Les échantillons ressemblent à des ondes sinusoïdales et cosinusoïdales à amplitudes décroissantes.

On considère également que $y_0 \sim \mathcal{N}(0, I_{2\times 2})$ et on génère des données à partir de l'équation ci-dessus à des horadatages irrégulièrement échantillonnés sur [0,3]. Les horodatages ne sont pas régulièrement espacés et ne sont pas cohérents entre différents élements du lot.

On ajuste une EDO latente à cet ensemble de données. Au moment du test, l'EDO est résolue sur le plus grand intervalle $[0, 12]$. Nous constatons qu'à la fin de l'entraînement, d'excellents échantillons sont produits, même s'ils sont sur un intervalle de temps quatre fois plus grand que celui sur lequel le modèle a été formé.

Cette approche en temps continu traîte sans problème plusieurs types d'échantillonnage irréguliers: les données d'entrée ne sont pas régulièrement espacées et différents éléments du lot ne sont pas échantillonnés en même temps.

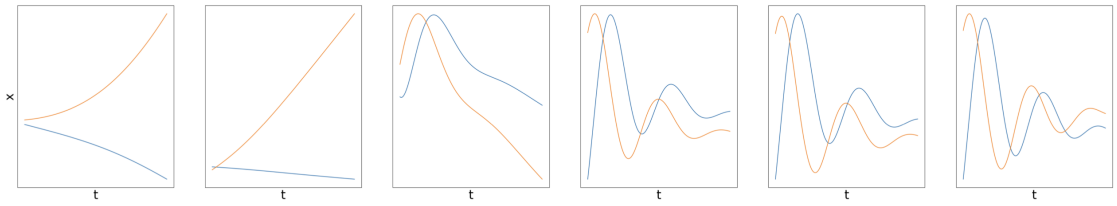

Figure 12: Graphe des échantillons $y : [0, 12] \to \mathbb{R}^2$ tirés du modèle d'EDO latentes. L'image à l'extrême gauche est un échantillon du modèle non entraîné. L'image à l'extrême droite est un échantillon d'un modèle entièrement entraîné. Entre les deux se trouvent des échantillons de modèles partiellement entraînés. La qualité augmente à mesure que la formation progresse.

Cependant, la sortie s'étend sur l'intervalle de temps continu $[0, 12]$, de sorte que nous obtenons des échantillons à tout moment.

5.3.1 Quelques détails computationnels

Soit $y(0)$ indépendamment échantillonné pour chaque chemin échantillonné à partir d'une distribution normale standard bi-dimensionnelle. On résoud l'équation (1) sur l'intervalle de temps $[0, T]$ où $T \sim \mathcal{U}[2, 3]$ de manière indépendante pour chaque chemin. On observe ensuite ces derniers à 20 points différents dans le temps échantillonés à partir de $\mathcal{U}[0, T]$.

L'équation différentielle est résolue sur chaque $[0, T]$ au moment de l'entraînement et sur l'intervalle plus grand $[0, 12]$ lors du test. Le solveur utilisé est Dormand-Prince 5(4) avec un pas de temps fixe de $0, 4$. Chaque opération est effectuée avec une précision en virgule flottante de 32 bits (donc plus ou moins 6 ou 7 chiffres décimaux de précision. On utilise l'optimiseur Adam - avec une taille de lot de 256 et un taux d'apprentissage de 10^{-2} - qu'on entraîne pour 250 pas.

Conclusion

Les modèles basés sur les Équations Différentielles Ordinaires (EDO) latentes, notamment les Neural Ordinary Differential Equations (NODE) et les Latent Ordinary Differential Equations (LODE), ont grandement amélioré la modélisation des processus dynamiques en combinant des réseaux de neurones avec des équations différentielles. Ces modèles, souvent intégrés avec des techniques d'Autoencodeurs Variationnels (VAE) et des Normalizing Flows, sont capables de capturer des dépendances temporelles complexes et d'apprendre des systèmes dynamiques à partir de données. Ils se distinguent par leur capacité à traiter efficacement des échantillonnages irréguliers et à fournir des échantillons sur un intervalle de temps continu, tout en offrant une excellente qualité de reconstruction et d'extrapolation des données. Ces avancées ouvrent de nouvelles perspectives dans des domaines tels que la physique, la biologie et l'apprentissage automatique, où ils sont devenus des outils précieux pour la modélisation et la prévision de processus dynamiques.

Malgré les avancées significatives apportées par les NODE, LODE et Normalizing Flows, plusieurs défis et opportunités subsistent. Il serait essentiel d'explorer davantage la flexibilité et la robustesse de ces modèles sur des données réelles complexes et de comprendre leur adaptabilité à différents types de problèmes dynamiques. L'intégration synergique de ces modèles avec d'autres techniques d'apprentissage profond pourrait conduire à des avancées majeures. L'optimisation des paramètres, la recherche de nouvelles architectures et le développement de méthodes d'interprétabilité pour ces modèles sont également des axes de recherche prometteurs. De plus, l'extension de ces approches pour traiter des données multivariées et spatio-temporelles ouvrirait de nouvelles voies pour leur application dans divers domaines scientifiques et industriels. Enfin, il serait bénéfique de promouvoir des études interdisciplinaires pour maximiser l'impact de ces technologies et faciliter leur adoption dans la pratique clinique et la prise de décision dans des environnements réels.