

Neural ODEs, Normalizing Flows et Latent ODEs

Arnaud FEYEL, Malek BOUZIDI, Noor SEMAAN

March 15, 2024

Introduction

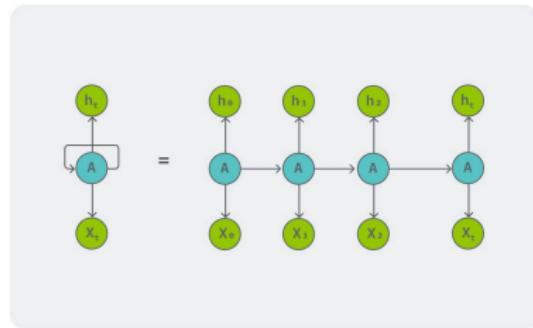


Figure: Architecture d'un RNN

Introduction

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

Idée : ajouter le plus de couches possibles !

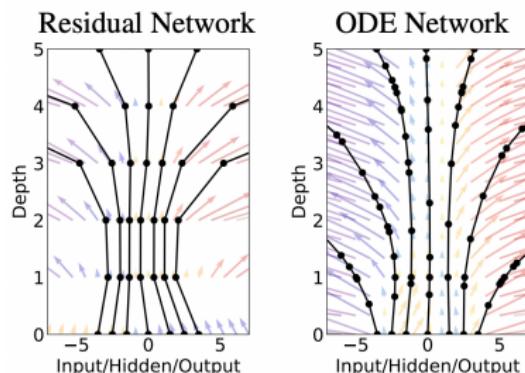


Figure: Réseau Discret vs Continu

Introduction

$h_{t+1} = h_t + f(h_t, \theta_t) \longrightarrow$ Discrétisation par Euler !

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

1. Unicité et Existence de la solution.
2. Facile à résoudre analytiquement et/ou numériquement.
3. Stabilité numérique.
4. "Infinité" de couches.

I - Un peu de théorie

Soient f_θ un réseau de neurone de paramètres θ , L une fonction de coût et z fonction de l'état au temps t .

$$\frac{d}{dt}z(t) = f_\theta(z(t), t)$$

$$\implies z(T) = z(0) + \int_0^T f_\theta(z(t), t) dt.$$

$z(0) = x = z_{input}$, et $z(T) = \hat{y} = z_{output}$ notre prédition.

I - Un peu de théorie

$$L(z(T)) = L \left(z(0) + \int_0^T f(z(t), t, \theta) dt \right)$$

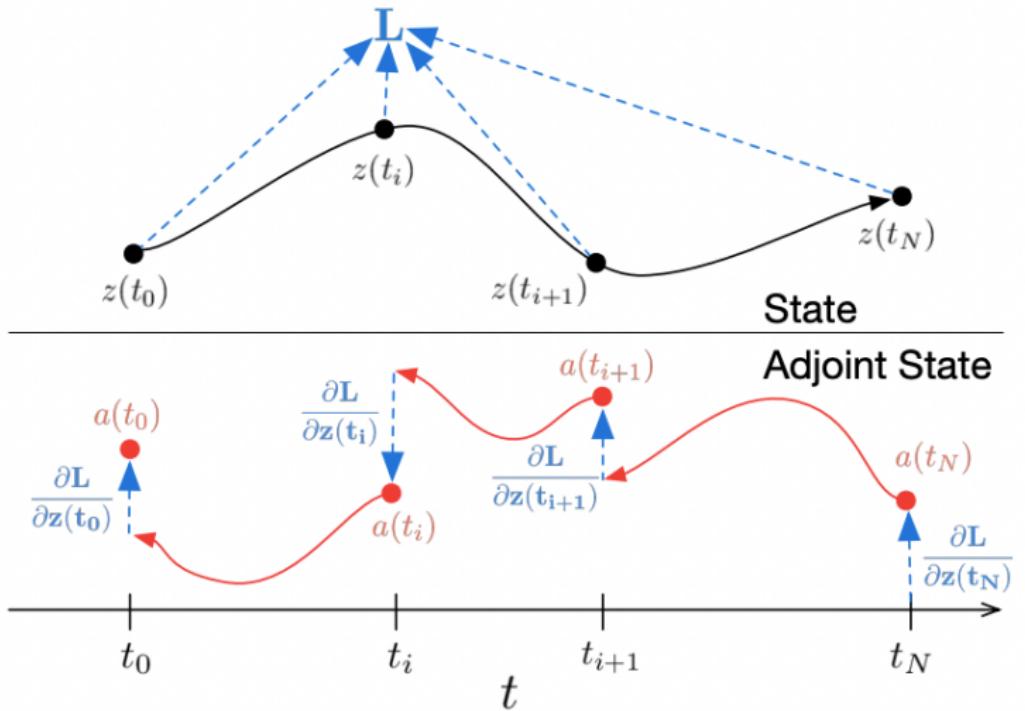
$\theta_{t+1} = \theta_t - \gamma \nabla L_\theta$, γ le pas d'apprentissage.

Problème 1: comment déterminer la façon dont ∇L dépend de $z(t)$ pour tout $t \in [0, T[$?

Problème 2: comment calculer $\nabla_\theta L$?

I - Un peu de théorie

Problème 1 \iff calculer l'adjoint $a(t)$ de L !



I - Un peu de théorie

Problème 1 \iff calculer l'adjoint $\mathbf{a(t)}$ de L !

$$\mathbf{a(t)} = \frac{\partial}{\partial z(t)} L$$

Par analogue de la *Chain Rule*

$$\frac{dL}{dh_t} = \frac{dL}{dh_{t+1}} \frac{dh_{t+1}}{dh_t} \sim \frac{dL}{dz(t)} = \frac{dL}{dz(t+\epsilon)} \frac{dz(t+\epsilon)}{dz(t)}$$

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

I - Un peu de théorie

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

$$\implies \frac{d}{dt} a(t) = -a(t) \frac{\partial}{\partial z(t)} f_\theta(z(t), t)$$

Problème 1 : Ok !

(Définition de dérivée + Formule de Taylor)

I - Un peu de théorie

→ Rappel : $f(z(t), t, \theta)$

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

→ On définit le *augmented state*

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

I - Un peu de théorie

→ On dérive

$$\frac{d}{dt} \boldsymbol{a}_{aug}(t) = - \begin{bmatrix} \mathbf{a} \frac{\partial}{\partial z} f & \mathbf{a} \frac{\partial}{\partial \theta} f & \mathbf{a} \frac{\partial}{\partial t} f \end{bmatrix} (t)$$

(Jacobienne + Produit Matrice Vecteur)

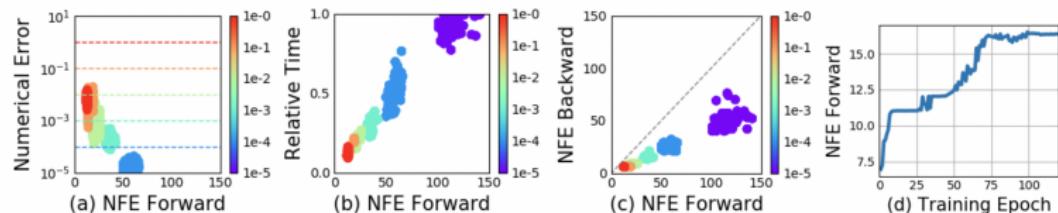
I - Un peu de théorie

- **a(t)** Finalement comment avoir $\frac{\partial}{\partial \theta} L$?
- Rappel : $\mathbf{a(t)} = \frac{\partial}{\partial z(t)} L$

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

Problème 2 : Ok !

I - Un peu de théorie



- Beaucoup moins de Backward en fonction du Forward, d'où un apprentissage plus rapide !
- Plus on veut de précision, plus on a besoin de samples

I - Applications

Neural-ODE for pharmacokinetics modeling

James Lu, Kaiwen Deng, Xinyuan Zhang, Gengbo Liu, et Yuanfang
Guan

I - Applications - Enjeux

- prédire le PK (pharmacokinetics)
- évolution au cours du temps

Problème : Ok pour 1 seul dosage mais mauvaise généralisation quand le dosage change !

NB: première étude qui utilise les NEDOs

I - Applications - Travaux précédents

- Déduire le dosage de Remifentanil (un anesthésique) dans une population saine
- Multiple Learning Regression, Regression Tree, Multivariate Adaptive Regression Splines, Boosted Regression Trees, RF , Lasso, LSTM

Problème : Données trop irrégulières pour LSTM !

NB: première étude qui utilise les NEDOs

I - Applications - Stratégie

Objectif : prédire le PK associé au T-DM1 (traitment pour le cancer du sein)

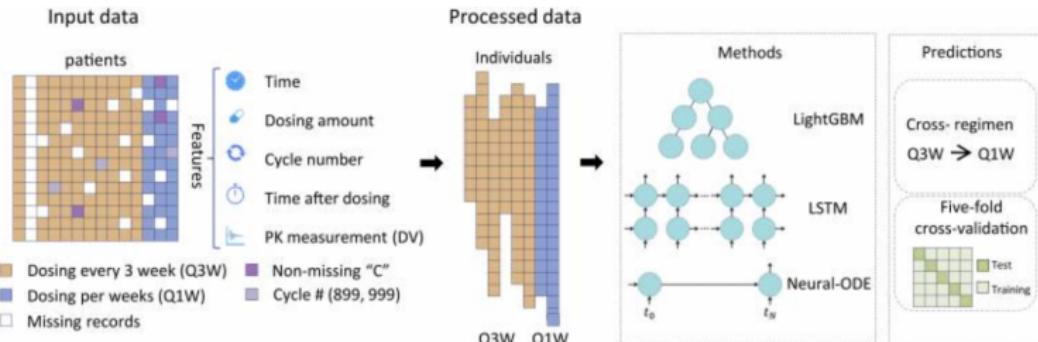
- T-DM1 a une fenêtre thérapeutique étroite
- Tests à petites doses sur les patients

- Traitement Q3W : 1 dose (variable dont la majorité à 3.6 mg/kg) toutes les 3 semaines
- Traitement Q1W : 1 dose (variable) toutes les semaines

I - Applications - Stratégie - Modèles

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

I - Applications - Stratégie



→ Performance de chaque modèle évaluée en testant sa capacité à prédire les concentrations PK chez les patients traités avec le régime opposé.

I - Applications - Résultat

- Comportement similaire sur les mêmes schémas
- NEDO beaucoup plus efficace pour généraliser
- Meilleure performance grâce au temps continu.

II - Normalizing Flows

II - Normalizing Flows

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

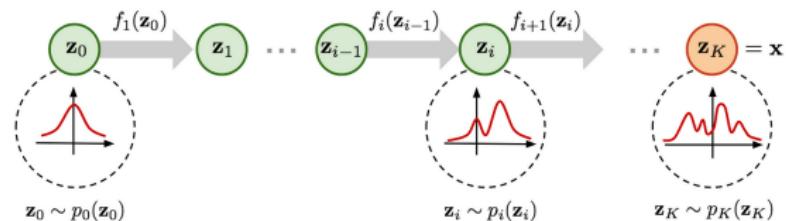


Figure: Normalizing Flows

II - Planar Normalizing Flows

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

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

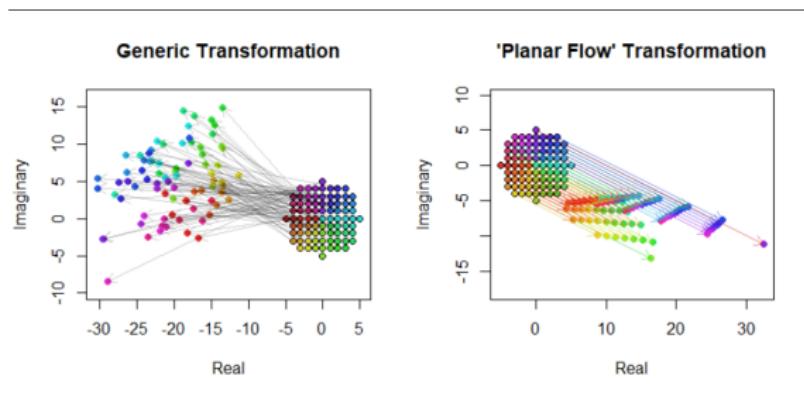


Figure: À droite: "planar flow". Les flèches sont colorées en fonction de la valeur de h .

II - Introduction aux Continuous Normalizing Flows

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

II - Introduction aux Continuous Normalizing Flows - Exemple



Figure: Images synthétiques des chats

II - Continuous Normalizing Flows (CNF)

Considérons l'EDO neuronale aléatoire suivante :

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

où $d = \prod_{k=1}^m d_k$

II - Changement de variable instantané

Supposons que $f_\theta = (f_{\theta,1}, \dots, f_{\theta,d})$ est Lipschitz-continue. Soit $p_\theta : [0, T] \times \mathbb{R}^d \rightarrow \mathbb{R}$, 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 :

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

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

II - Théorème de changement de variable instantané

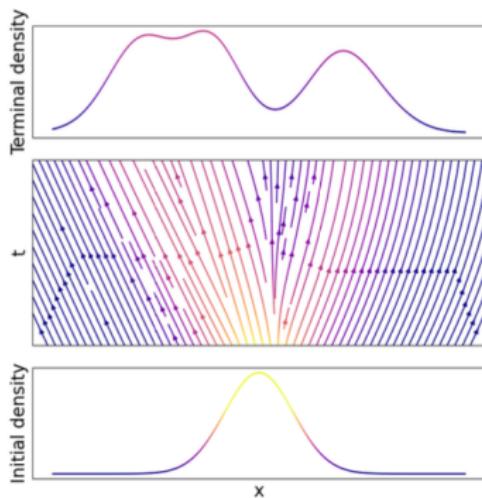


Figure: Continuous Normalizing Flows déforme continuellement une distribution en une autre distribution. Les lignes de flux montrent comment les particules de la distribution de base sont perturbées jusqu'à ce qu'elles approximent la distribution cible.

II - Continuous Normalizing Flows (CNF) - Entrainement

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

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

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

II - Continuous Normalizing Flows (CNF) - Entrainement

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

II - Continuous Normalizing Flows (CNF) - Entrainement

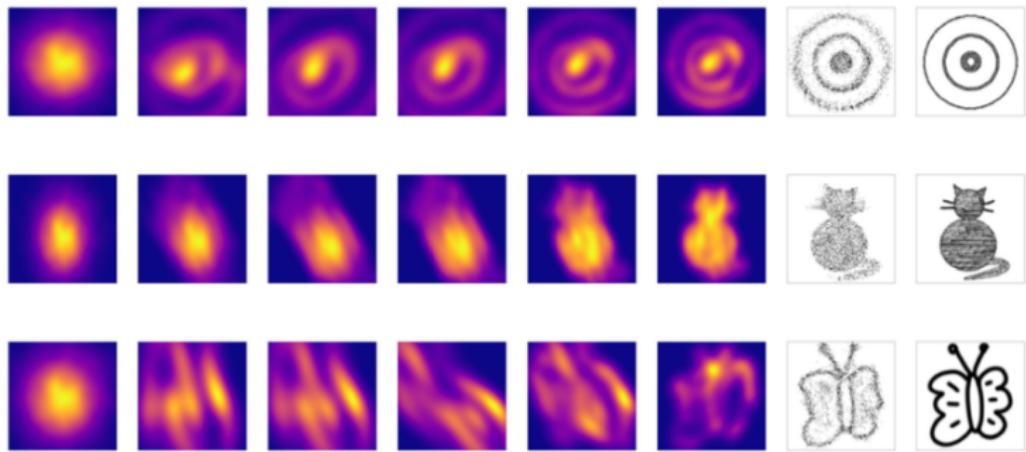
$$-\frac{1}{N} \sum_{i=1}^N \log p_\theta(T, y_i) = -\frac{1}{N} \sum_{i=1}^N [\log p_\theta(0, y(0, y_i)) - \int_0^T \sum_{k=1}^d \frac{\partial f_{\theta,k}}{\partial y_k}(t, y(t, y_i))] \,$$

II - Continuous Normalizing Flows (CNF) - Entrainement

$$-\frac{1}{N} \sum_{i=1}^N \log p_\theta(T, y_i) = -\frac{1}{N} \sum_{i=1}^N [\log p_\theta(0, y(0, y_i)) - \int_0^T \sum_{k=1}^d \frac{\partial f_{\theta,k}}{\partial y_k}(t, y(t, y_i))] \quad (2)$$

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

II - Continuous Normalizing Flows (CNF) - Exemple



Exemple d'application de CNF: cible, chat et papillon

II - Continuous Normalizing Flows (CNF) - Distribution d'autodifferentiation

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.

II - Continuous Normalizing Flows (CNF) - 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 $\text{Cov}[\varepsilon] = I_{d \times d}$. Alors,

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

II - Continuous Normalizing Flows (CNF) - Trace-Jacobian

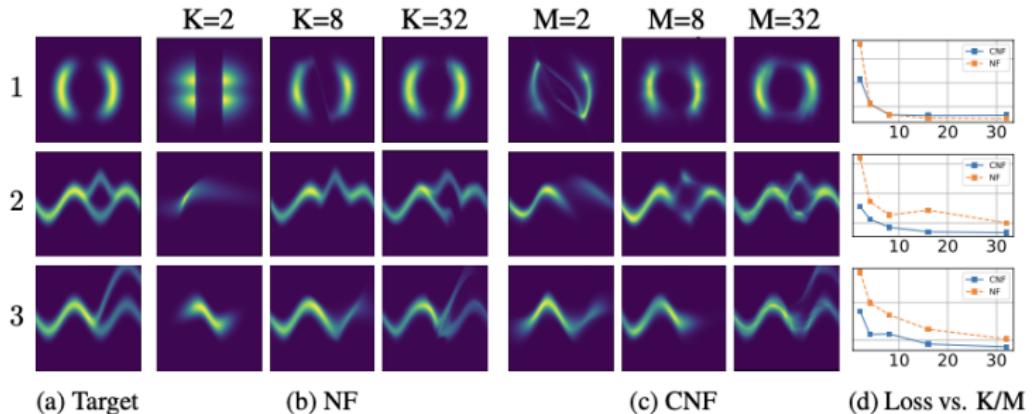
Nous avons que

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

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

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

II - Comparaison entre CNF et NF



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.

II - Comparaison entre CNF et NF

article multirow

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

Table: Tableau de comparaison

III - EDO Latentes

III - Objectif

Modéliser les distributions ayant une composante variable dans le temps.

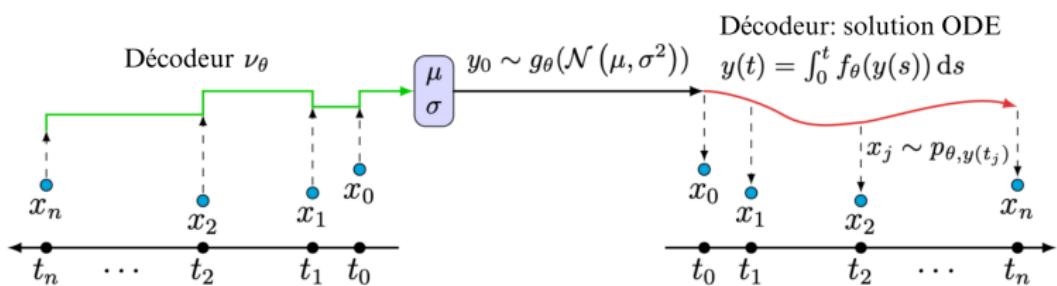


Figure: Aperçu général du modèle EDO latentes

III - Cadre - Séries Temporelles

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

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

Pour simplifier, on admet qu'il n'y a pas de données manquantes.

III - Cadre - Solution de l'EDO neuronale

Pour $x \in \mathbb{R}^{d_m}$, soit $y_0 = f_\theta(z) \in \mathbb{R}^{d_t}$

Soit $y(t, y_0)$ la solution de l'EDO neuronale:

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

Existence et unicité: théorème de Picard

III - Auto-Encodeur Variationnel (VAE)

Architecture: Réseaux de neurones entraînés de manière non-supervisée.

But: reconstruire les données d'entrées avec un maximum de précision

Exemple: échantillon de taille 250, espace latent de dimension 25

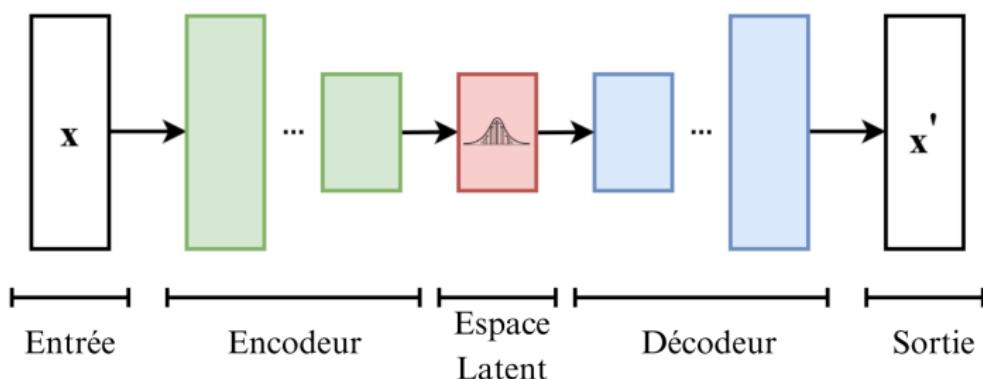


Figure: Schéma de la structure d'une Auto-Encodeur Variationnel

III - Encodeur - Aperçu

Définition: $\nu_\theta : TS(\mathbb{R}^d) \rightarrow \mathbb{R}^{d_m} \times (0, \infty)^{d_m}$

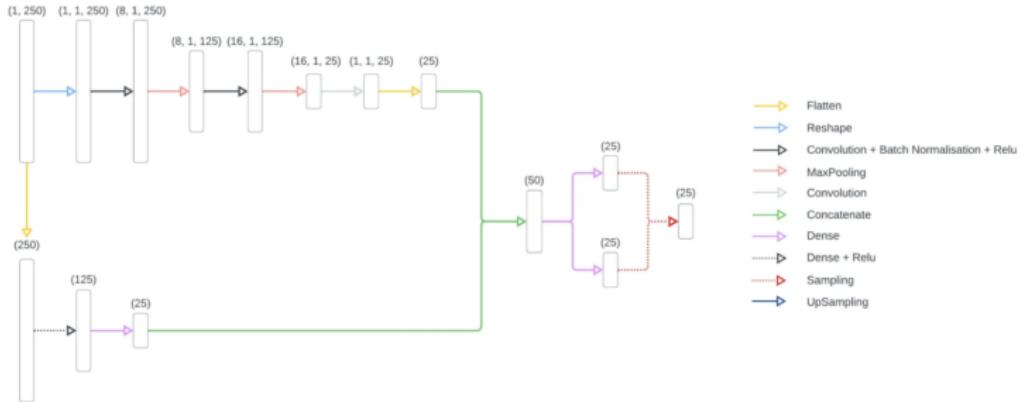


Figure: Architecture de l'Encodeur sur un échantillon de taille 250

III - Encodeur - Structure

Input: série temporelle $x \in \mathbb{R}$

1. Reshape

2. **Partie convolutionnelle:**

- ▶ Convolution
- ▶ Normalisation par lots (*batch normalization*)
- ▶ Fonction d'activation ReLU
- ▶ MaxPooling
- ▶ Répéter
- ▶ Convolution simple

3. Flatten

Output: vecteur de taille 25

III - Encodeur - Structure

4. Partie dense:

- ▶ Couche dense avec ReLU
- ▶ Couches dense simple

Output: second vecteur de taille 25.

5. Concaténation des deux vecteurs

Output: vecteur de taille 50.

6. 2 couches denses simples

Output: deux vecteurs μ et σ^2 de taille 25

III - Encodeur - Output

Output:

$$q_{\theta,x} = \mathcal{N}(\mu_{\theta,x} := \mu, \text{diag}(\sigma_{\theta,x}^2 := \sigma^2)) \text{ ou } (\mu_{\theta,x}, \sigma_{\theta,x}) = \nu_{\theta}(x)$$

Vecteur latent $z \sim \mathcal{N}(\mu_{\theta,x}, \sigma_{\theta,x}^2).$

III - Décodeur - Aperçu

Input: vecteur latent z

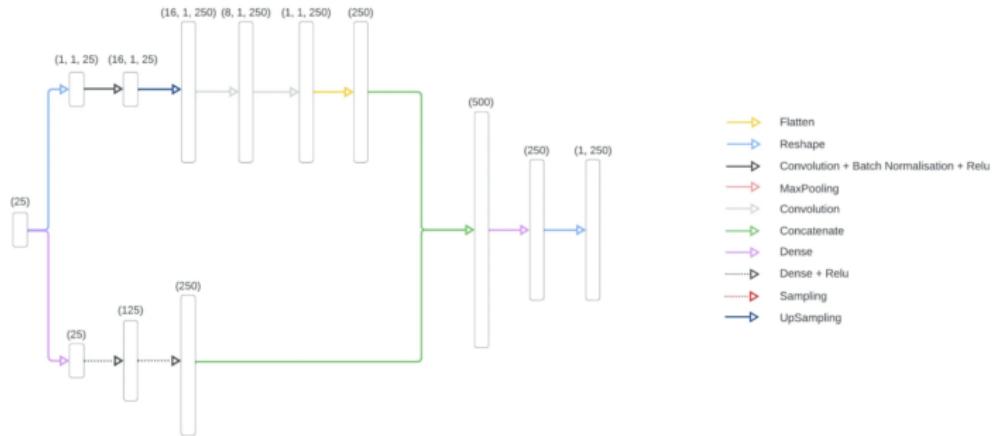


Figure: Architecture du décodeur sur un échantillon de taille 250

III - Décodeur - Cadre

- ▶ Deux espaces latents de dimensions $d_l, d_m > 0$
- ▶ Un vecteur de paramètres appris θ
- ▶ Deux réseaux de neurones paramétrisés par θ :

$$f_\theta : \mathbb{R}^{d_l} \rightarrow \mathbb{R}^{d_l}$$

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

Soit $y_0 = g_\theta(z) \in \mathbb{R}^{d_l}$

EDO neuronale:

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

III - Décodeur - Structure

Input: vecteur latent créé par l'encodeur.

1. Reshape

2. **Partie convolutionnelle:**

- ▶ Convolution
- ▶ Normalisation par lots (*batch normalization*)
- ▶ Fonction d'activation ReLU
- ▶ Upsampling (pour revenir à la taille de 250)
- ▶ Deux convolutions simples
- ▶ Flatten

→ vecteur de taille 250

III - Décodeur - Structure

3. Partie dense:

- ▶ Deux couches denses avec ReLU
 - ▶ Flatten
- second vecteur de taille 250

4. Concaténation des deux vecteurs

→ vecteur de taille 500

5. Couche dense simple

→ vecteur de taille 250

III - Décodeur - Densité de probabilité et Output

On choisit une densité de probabilité dépendant du comportement cherché pendant l'échantillonnage durant le temps d'inférence:

$$p_{\theta,y} : \mathbb{R}^{d_I} \rightarrow [0, \infty)$$

paramétrisée par $y \in \mathbb{R}^{d_I}$.

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

- ▶ Statistique ponctuelle requise: $p_{\theta,y}$ fonction de coût.
- ▶ Output complet du modèle requis: choix plus expressifs (par exemple *normalising flow*).

III - Entraînement

Soit un batch/dataset d'une série temporelle $x_1, \dots, x_n \in TS(\mathbb{R}^d)$.

Critère d'optimisation end-to-end: minimiser θ par rapport à

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

avec KL : divergence Kullback-Leider

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

→ minimiser distance entre p et q

III - Entraînement

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

- ▶ Premier terme: décodeur apprend à répliquer les échantillons donnés en input.
- ▶ Second terme: distribution initiale dans l'espace latent correspond à une distribution connue.

Evaluation par **back propagation**.

III - Application - Cadre

Cadre: oscillateurs à amplitudes décroissantes

Série temporelle: observations discrètes de

$$y(t) = \exp(At)y_0 \quad (1)$$

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.

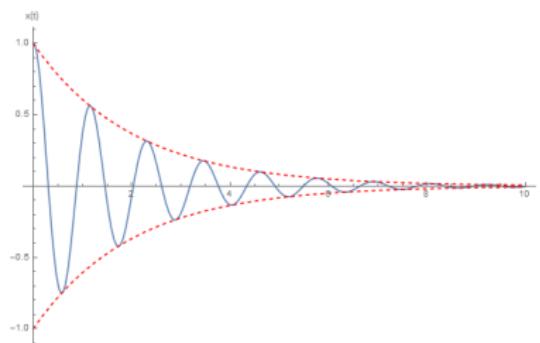


Figure: Forme des échantillons

III - Application - Processus

→ On considère $y_0 \sim \mathcal{N}(0, I_{2 \times 2})$

1. Générer des données à partir de $y(t) = \exp(At)y_0$

tel que t irrégulièrement échantillonnés sur $[0, 3]$

2. Ajuster EDO latente

3. Tester l'EDO sur $[0, 12]$, 4 fois plus grand que $[0, 3]$.

III - Application - Résultats

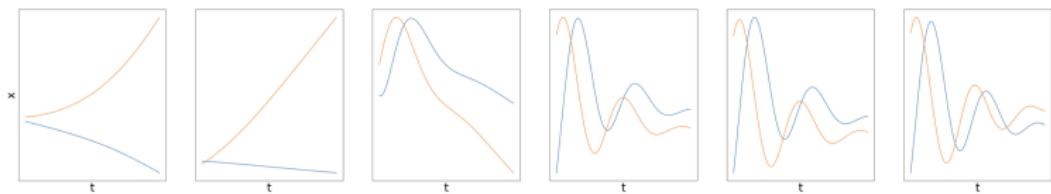


Figure: Graphe des échantillons $y : [0, 12] \rightarrow \mathbb{R}^2$ tirés du modèle d'EDO latentes.

De gauche à droite: différent stades d'entraînement du modèles de non entraîné à entraîné.

- Qualité augmente à mesure que la formation progresse.
- Bonne reproduction des données.

- Bonnes qualités d'extrapolation sur un intervalle **4 fois plus long** que l'intervalle d'entraînement!

Merci!