Notes sur les Flots Normalisants Continus

Hugo Gangloff

1er décembre 2023

Dans la suite, les Flots Normalisants Continus sont abrégés CNFs (*Continuous Normalizing Flows*), les Flots Normalisants, NFs (*Normalizing Flows*), et les Équations aux Différentielles Ordinaires Neuronales, NODEs (*Neural Ordinary Differential Equations*).

Revue de littérature : [PNR+21].

1 Rappels sur les Flots Normalisants

Article de référence : [RM15]

Théorème du changement de variable Soit $f\colon \mathbb{R}^d \to \mathbb{R}^d$ un C^1-difféomorphisme et $z_0 \sim q_0$, alors $z = f(z_0)$ suit une loi q de densité

$$q(z) = q_0(f^{-1}(z))|\det \mathrm{Jac}_{f^{-1}}(z)| = q_0(z_0)|\det \mathrm{Jac}_f(z_0)|^{-1}, \tag{1}$$

où Jac_f représente la matrice jacobienne :

$$\begin{bmatrix} \frac{\partial f_1}{\partial z_1} & \cdots & \frac{\partial f_1}{\partial z_d} \\ \vdots & \cdots & \vdots \\ \frac{\partial f_d}{\partial z_1} & \cdots & \frac{\partial f_d}{\partial z_d} \end{bmatrix}.$$

Flot Normalisant L'idée est de composer des C^1 difféomorphismes, $f_1, \dots, f_K$, tel que $z_K = f_K \circ \cdots \circ f_1(z_0)$, et le théorème du changement de variable donne

$$q_K(z_K) = q_0(z_0) \prod_{k=1}^{K} |\det \mathrm{Jac}_{f_k}(z_{k-1})|^{-1}. \tag{2}$$

Dans la suite, on pose $T_\lambda \triangleq f_{K,\lambda} \circ \cdots \circ f_{1,\lambda}$ et λ sont les paramètres des fonctions f.

Inférence variationnelle On peut utiliser les NFs pour faire de l'inférence variationnelle, *i.e.*, maximiser une borne inférieure de la vraisemblance afin d'approximer une loi *a posteriori*, *i.e.*,

$$\begin{aligned} \max_{q,\theta} \mathcal{L}_{q,\theta} &= \max_{q,\theta} \mathbb{E}_q \left[\log \frac{p_\theta(x,z)}{q(z)} \right], \\ &= \max_{\lambda,\theta} \left\{ \mathbb{E}_{q_0}[\log p_\theta(x, T_\lambda(z_0)) - \mathbb{E}_{q_0}[\log q_{K,\lambda}(T_\lambda(z_0))]] \right\}, \end{aligned} \tag{3}$$

où l'on a utilisé une forme de NF pour q et où $\log q_{K,\lambda}(T_\lambda(z_0)) = \log q_0(z_0) - \sum_{k=1}^{K} \log |\det \mathrm{Jac}_{f_{k,\lambda}}(z_{k-1})|$. q_0 est supposée connue et traditionnellement, $q_0 = \mathcal{N}(0, I_d)$.

Modèle On peut aussi vouloir apprendre un modèle avec les NFs, *i.e.*, estimer une loi $p_\lambda(x)$ liée à un jeu de données. Cela peut notamment permettre d'évaluer la densité de données futures ou alors de tirer des nouvelles données. Supposons que nous disposons d'un jeu de données $\{x_i\}_{i=1}^N$ et que nous posons un modèle de NF défini par $\begin{cases} z_{i,K} \triangleq x_i, \\ z_{i,0} = T_\lambda^{-1}(z_{i,K}), \end{cases}$, alors l'apprentissage se fait classiquement par maximisation de la log-vraisemblance,

$$\begin{aligned} \max_\lambda \log p_\lambda(x_1, \dots, x_N) &= \frac{1}{N} \sum_{i=1}^N \log q_{K,\lambda}(x_i) \\ &= \frac{1}{N} \sum_{i=1}^N \left[\log q_0(T_\lambda^{-1}(z_{i,K})) + \log |\det \mathrm{Jac}_{T_\lambda^{-1}}(z_{i,K})| \right] \\ &= \frac{1}{N} \sum_{i=1}^N \left[\log q_0(T_\lambda^{-1}(z_{i,K})) + \sum_{k=1}^K \log |\det \mathrm{Jac}_{f_{k,\lambda}^{-1}}(z_{i,k})| \right]. \end{aligned} \tag{4}$$

Supposons que l'on puisse expliciter l'inverse de T_λ^{-1}, alors le tirage d'un nouvel échantillon une fois T_λ^{-1} appris se fait par le tirage de $z_{i,0} \sim q_0$ et on calcule $z_{i,K} = T_\lambda(z_{i,0})$.

Planar NF Ce type de flot est défini par la récursion

$$z_{k+1} = f_{k+1,\lambda}(z_k) = z_k + u \underbrace{h(w^T z_k + b)}_{\text{hidden unit}}, \tag{5}$$

avec $\lambda = \{u, w, b\}$ où $u \in \mathbb{R}^d$, $w \in \mathbb{R}^d$, $b \in \mathbb{R}$ les paramètres du modèle et h une non-linéarité. Nous n'avons ici qu'une seule unité cachée (ou neurone). Dans le cas où l'on compose K fois cette transformation :

$$\begin{cases} z_K = f_K \circ \cdots \circ f_1(z_0) \\ \log q_K(z_K) = \log q_0(z_0) - \sum_{k=1}^K \log |1 + u_k^T h'(w_k^T z_{k-1} + b_k) w_k|, \end{cases} \tag{6}$$

la dernière égalité est obtenue grâce au *matrix determinant lemma*.

On note que nous n'avons pas de formule explicite de l'inverse de ce NF, ainsi, nous serions *a priori* coincés si nous voulions effectuer des tirages selon un densité apprise par un *Planar NF*. En effet, l'Éq. 4 montre que le flot appris va de $z_{i,K}$ vers $z_{i,0}$ pour pouvoir évaluer la densité $q_{K,\lambda}(z_{i,K})$; or le tirage de nouveaux échantillons suivant cette loi nécessite l'inversion du T_λ^{-1} appris.

2 Introduction aux Équations aux Différentielles Ordinaires Neuronales

Réseau de neurones résiduel Ce type de réseaux peuvent s'écrire sous la forme

$$z_{k+1} = z_k + f_{\lambda,k}(z_k), k \in \{0, \dots, K\}. \tag{7}$$

Avec les bonnes contraintes sur g_φ, ce type de réseaux peuvent être inversibles.

Neural Ordinary Differential Equation On se demande alors ce qu'il se passe si on ajoute plus de couches et que l'on fait des pas plus petits dans les réseaux de neurones résiduels. On définit alors une NODE en voyant la variable z comme une fonction du temps et en décrivant ses variations temporelles entre t_0 et t_1 via un réseau de neurones $g_\varphi \colon [t_0, t_1] \times \mathbb{R}^d \to \mathbb{R}^d$:

$$\frac{\mathrm{d}z(t)}{\mathrm{d}t} = g_\varphi(t, z(t)), t \in [t_0, t_1]. \tag{8}$$

Problème à Valeur Initiale Abrégé IVP pour *Initial Value Problem*, c'est la donnée d'une (N)ODE et d'une valeur initiale :

$$\begin{cases} \frac{\mathrm{d}z(t)}{\mathrm{d}t} = g_\varphi(t, z(t)), \\ z(t_0) = z_0. \end{cases} \tag{9}$$

Un solveur d'EDO est capable de nous donner la solution à l'IVP, *i.e.*,

$$z(t_1) = z(t_0) + \int_{t=t_0}^{t=t_1} g_\varphi(t, z(t))\mathrm{d}t. \tag{10}$$

Pour la même complexité calculatoire, je peux résoudre l'équation à rebours, *i.e.*,

$$z(t_0) = z(t_1) + \int_{t=t_1}^{t=t_0} g_\varphi(t, z(t))\mathrm{d}t = z(t_1) - \int_{t=t_0}^{t=t_1} g_\varphi(t, z(t))\mathrm{d}t. \tag{11}$$

Discrétisation avant d'Euler (*Forward Euler discretization*) de l'Éq. 8 donne

$$\frac{z(t+h) - z(t)}{h} = g_\varphi(t, z(t)), \tag{12}$$

ce qui nous fait retomber sur l'Éq. 7.

Théorème d'existence de Picard Lindelhöf Il garantit l'existence et l'unicité d'une solution à une NODE (Éq. 8) sous certaines conditions. Soit $g_\varphi \colon [t_0, t_1] \times \mathbb{R}^d \to \mathbb{R}^d$ telle que g_φ est continue en sa première variable t et uniformément Lipshitz par rapport à sa deuxième variable $z(t)$ (*i.e.*, $\exists C > 0, \forall z_1, \forall z_2, \forall t \in [t_0, t_1]$, on a $\|g_\varphi(t, z_1) - g_\varphi(t, z_2)\| \leq C\|z_1 - z_2\|$) alors il existe une unique solution à l'IVP de l'Éq. 9. On note que les réseaux de neurones classiques utilisant des fonctions d'activation tanh or ReLU sont Lipshitz. On remarque ainsi, qu'à l'inverse des NF, nous ne posons aucune restriction sur l'inversibilité ou la tractabilité du Jacobien du réseau de neurones g_φ.

Dérivation automatique à travers un solveur d'EDO La phase d'apprentissage consistera à apprendre les paramètres φ du réseau de neurones. On va donc devoir faire de la dérivation automatique (*automatic differentiation* ou *autodiff*) à travers un solveur d'EDO. En effet, imaginons que nous souhaitons utiliser un modèle de NODE pour de l'apprentissage supervisé. Nous disposons de paires de données $\{(x_i, y_i)\}_{i=1}^N$ et nous pouvons proposer comme architecture :

$$x_{i,t_0} (\triangleq x_i) \to \text{NODE}(g_\varphi) \to x_{i,t_1} \to \text{Couche linéaire} \to \hat{y}_i. \tag{13}$$

Avec une loss de type MSE, $\mathcal{L} = \sum_{i=1}^N (y_i - \hat{y}_i)^2$ où

$$\hat{y}_i = w x_{i,t_1} + b = w \underbrace{\left(x_{i,t_0} + \int_{t=t_0}^{t=t_1} g_\varphi(t, x_{i,t})\mathrm{d}t \right)}_{\text{ODESOLVE}(g_\varphi, x_{i,t_0}, t_0, t_1)} + b.$$

(Note : la couche linéaire est un ajout qui correspondrait à un tâche sous-jacente de classification par exemple.)

On va donc devoir évaluer $\frac{\partial \mathcal{L}}{\partial \varphi}$ et on voit clairement les dérivées qui vont devoir se propager à travers le solveur (ici représenté par un boîte noire nommée ODESOLVE), *i.e.*, on va devoir calculer des $\frac{\partial \text{ODESOLVE}}{\partial \varphi}$. Un tel calcul n'est pas possible *a priori* avec le solveur classique `odeint` de `scipy`. Cependant, l'autodiff à travers un solveur d'EDO est rendue possible grâce aux librairies modernes d'apprentissage profond et certaines proposent déjà des solveurs compatibles avec l'autodiff (`diffrax` en `JAX`, `torchode` en `pytorch`, etc.). Le coût d'une dérivation automatique à travers un solveur d'EDO est coûteux en terme de mémoire car il faut stocker tous les résultats intermédiaires des itérations du solveur dans le cas d'une autodiff de type *backpropagation*. Sur cet aspect, un des apports de [CRBD18] est de proposer une nouvelle approche pour calculer les gradients sans devoir dériver à travers le solveur ; ce dernier est ainsi considéré comme une boîte noire.

Réseaux de neurones informés par la physique Il est important d'avoir en tête la différence entre les NODEs et les réseaux de neurones informés par la physique [KKL+21] (PINNs pour *Physics Informed Neural Networks*). Ces derniers considèrent l'EDO connue et la solution est approchée par un réseau de neurones, à la différence des NODEs où le terme de droite de l'EDO (Éq. 8) est appris, par un réseau de neurones.

3 Les Flots Normalisants Continus

Théorème de changement de variable instantanné L'outil principal derrière les CNFs est le théorème de changement de variable instantanné [CRBD18]. Si $z(t)$ est une variable aléatoire continue, dépendante de t, de densité $p(z(t))$, dont la dynamique est décrite par 8, alors la log-densité de $z(t)$ suit également une NODE donnée par

$$\frac{\partial \log p(z(t))}{\partial t} = -\mathrm{tr}(\mathrm{Jac}_{g_\varphi(t,\cdot)}(z(t))). \tag{14}$$

La preuve est en annexe de [CRBD18]. Pour un IVP avec $\log p(z(t_0))$ comme valeur initiale, un solveur donne donc la solution $\log p(z(t_1)) = \log p(z(t_0)) + \int_{t=t_0}^{t=t_1} -\mathrm{tr}(\mathrm{Jac}_{g_\varphi(t,\cdot)}(z(t)))\mathrm{d}t$.

Estimation de la densité $p_\varphi(x)$ - Apprentissage des paramètres On observe $\{x_i\}_{i=1}^N$ et on pose $z_i(t_1) \triangleq x_i$ et $z_i(t_0) \triangleq z_i$. Comme on l'a vu dans l'Éq. 4, on veut calculer :

$$\begin{aligned}\max_\varphi \log p_\varphi(x_1, \dots, x_N) &= \max_\varphi \sum_{i=1}^N \log p_\varphi(z_i(t_1)), \\ &= \max_\varphi \sum_{i=1}^N \left[\log p_0(z_i(t_0)) + \int_{t=t_0}^{t=t_1} -\mathrm{tr}(\mathrm{Jac}_{g_\varphi(t,\cdot)}(z(t)))\mathrm{d}t\right].\end{aligned} \tag{15}$$

On voit que pour évaluer la dernière ligne, il nous faut calculer le z_i correspondant à x_i et également pouvoir évaluer $z(t), \forall t \in [t_0, t_1]$. Il nous faut donc résoudre en parallèle les deux NODEs : l'équation sur $z(t)$ et celle sur sa log-densité. Ces calculs peuvent s'écrire en combinant les deux équations (Éq. 8 et 14) en une NODE dans un espace d'états augmenté. Nous pouvons en effet écrire :

$$\begin{bmatrix} z_i(t_0) \\ \log p_\varphi(z_i(t_1)) - \log p_0(z_i(t_0)) \end{bmatrix} = \begin{bmatrix} x_i \\ 0 \end{bmatrix} + \int_{t=t_1}^{t=t_0} \begin{bmatrix} g_\varphi(t, z(t)) \\ \mathrm{tr}(\mathrm{Jac}_{g_\varphi(t,\cdot)}(z(t))) \end{bmatrix} \mathrm{d}t, \tag{16}$$

ce qui correspond à résoudre, à rebours entre t_1 et t_0, l'IVP suivant dans un espace d'états augmenté :

$$\begin{cases} \frac{\partial}{\partial t} \begin{bmatrix} z(t) \\ \log p_\varphi(x(t)) - \log p_0(z(t)) \end{bmatrix} = \begin{bmatrix} g_\varphi(t, z(t)) \\ \mathrm{tr}(\mathrm{Jac}_{g_\varphi(t,\cdot)}(z(t))) \end{bmatrix}, \\ \begin{bmatrix} z(t_1) \\ \log p_\varphi(x(t_1)) - \log p_0(z(t_1)) \end{bmatrix} = \begin{bmatrix} x_i \\ 0 \end{bmatrix}. \end{cases} \tag{17}$$

On remarque que le signe $-$ devant la trace disparaît car nous avons inversé les bornes de l'intervalle. Ainsi, avec un seul appel au solveur, nous pouvons obtenir z_i, calculer sa log-densité sous la loi initiale et additioner ce résultat la solution de la deuxième équation pour reformer $\log p_\varphi(x_i)$.

Tirage de nouveaux échantillons L'entraînement d'une NODE sert donc à déterminer g_φ. Pour procéder à un tirage selon $z(t_1) \triangleq x \sim p_\varphi(x)$ définie par le CNF, il suffit de résoudre l'IVP suivant entre t_0 et t_1 :

$$\begin{cases} \frac{\mathrm{d}z(t)}{\mathrm{d}t} = g_\varphi(t, z(t)), \\ z(t_0) \sim p_0(z), \end{cases} \tag{18}$$

où $p_0 = \mathcal{N}(0, I)$.

Évaluation de la densité Nous évaluons la densité $p_\varphi(x)$ de la même manière que celle décrite dans le paragraphe sur l'apprentissage des paramètres.

Considérations pratiques

— **Estimateur de la trace d'Hutchinson** Dans [GCB+19], afin de réduire le coût calculatoire de l'évaluation de la trace, il est proposé d'utiliser l'estimateur stochastique de la trace d'Hutchinson. Ainsi,

$$\mathrm{tr}(\mathrm{Jac}_{g_{\varphi,m}(t,\cdot)}(z(t))) = \mathbb{E}_{p_\epsilon}[\epsilon^T \mathrm{Jac}_{g_{\varphi,m}(t,\cdot)}(z(t))\epsilon], \tag{19}$$

où $p_\epsilon = \mathcal{N}(0, I_d)$. On peut montrer que $\frac{1}{N}\sum_{n=1}^{N} \epsilon_n^T \mathrm{Jac}_{g_{\varphi,m}(t,\cdot)}(z(t))\epsilon_n$ est un estimateur sans biais de la trace de $\mathrm{Jac}_{g_{\varphi,m}(t,\cdot)}(z(t))$.

— **Plusieurs unités cachées** Nous avons vu que les CNFs ont une complexité calculatoire plus faible que celle des NFs. Ce gain peut être utilisé pour utiliser plusieurs unités cachées (*hidden units*) [CRBD18]. Une NODE avec M unités cachées est définie par

$$\frac{\mathrm{d}z(t)}{\mathrm{d}t} = \sum_{m=1}^{M} g_{\varphi,m}(t, z(t)), \tag{20}$$

et la NODE sur la log-densité devient

$$\frac{\partial \log p(z(t))}{\partial t} = -\mathrm{tr}(\mathrm{Jac}_{\sum_{m=1}^{M} g_{\varphi,m}(t,\cdot)}(z(t))) = -\sum_{m=1}^{M} \mathrm{tr}(\mathrm{Jac}_{g_{\varphi,m}(t,\cdot)}(z(t))). \tag{21}$$

La deuxième égalité suit car le jacobien d'une somme de fonctions est égal à la somme des jacobiens de cette fonction et par linéarité de l'opération trace également.

— ***Gating mechanism*** [CRBD18]. L'idée est d'enrichir le modèle en introduisant une dépendance en t supplémentaire, sachant que cette dernière n'est pas nécessairement utilisée dans g_φ (voir le cas des *Planar CNF*). On ajoute un MLP prenant en entrée le temps devant chaque fonction $g_{\varphi,m}$, le rôle étant de donner une importance différente à chaque $g_{\varphi,m}$ en fonction du temps. La NODE sur les états s'écrit alors

$$\frac{\mathrm{d}z(t)}{\mathrm{d}t} = \sum_{m=1}^{M} \sigma_m(t) g_{\varphi,m}(t, z(t)), \tag{22}$$

où $\sigma_m(t) \in [0, 1]$ est un MLP.

Planar CNF Nous prenons exemple sur le *Planar NF* (Éq. 5) et dans l'Éq. 8 nous posons $g_\varphi(t, z(t)) = uh(w^T z(t) + b)$, alors $\mathrm{Jac}_{g_\varphi(t,\cdot)}(z(t)) = u\mathrm{Jac}_h^T(z(t))$ et $\frac{\partial \log p(z(t))}{\partial t} = -\mathrm{tr}(u\mathrm{Jac}_h^T(z(t))) = u^T \mathrm{Jac}_h(z(t))$ (la dernière égalité vient du fait que la trace d'un *outer product* de deux vecteurs est égale à l'*inner product* de ces deux vecteurs. Comme nous l'avons déjà évoqué, les soucis de calcul de l'inverse liés au *Planar NF* disparaissent dans le contexte des flots continus car la transformation dans les deux sens est tout aussi facile.

En pratique, nous allons combiner le *Planar CNF* avec le *gating mechanism* et plusieurs unités cachées.

Références

[CRBD18] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31 : Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6572–6583, 2018.

[GCB+19] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD : free-form continuous dynamics for scalable reversible generative models. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[KKL+21] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6) :422–440, 2021.

[PNR+21] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *The Journal of Machine Learning Research*, 22(1) :2617–2680, 2021.

[RM15] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.