

Neural Ordinary Differential Equations

Science des données

Alphonse Paix Djamila Azzouz

Master Probabilités et statistiques des nouvelles données

Année 2022–2023



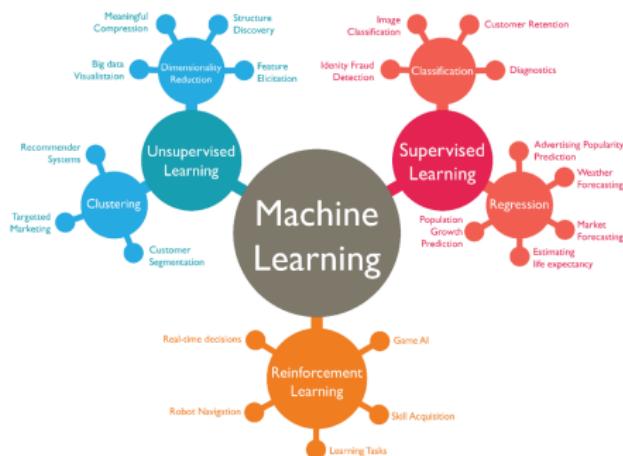
Plan

- Introduction générale
- Méthode de l'état adjoint avec les ODE et comparaison avec les ResNets
- Continuous Normalizing Flow
- Timeseries
- Conclusion

Introduction générale

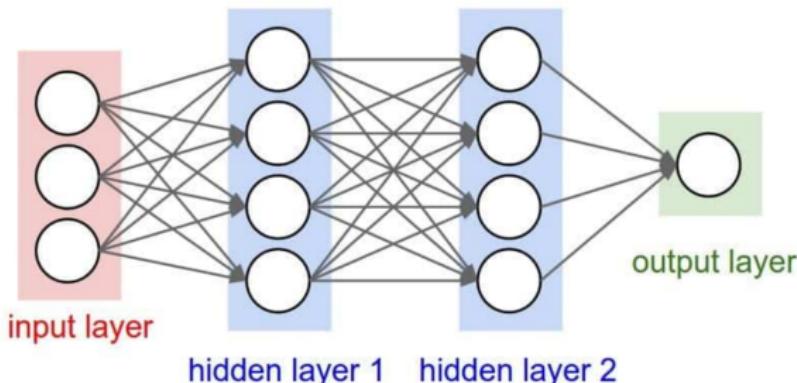
Principe du Machine learning

- $X \in \mathbb{R}^d$: variables explicatives ;
- $Y \in \{0, 1\}$: variable à prédire ;
- $\{(X_1, Y_1), \dots (X_n, Y_n)\}$: observations iid ;
- **Objectif** : prédire Y pour de nouvelles observations de X.



Introduction générale

- Réseaux de neurones :



- Présentation d'une EDO

Une EDO est une fonction mathématique qui décrit la relation entre une fonction inconnue et ses dérivées, elle prend la forme suivante :

$$f(x, y, y', y'', \dots, y^{(n)}) = 0$$

où : y : la fonction inconnue , x : la variable indépendante

EDO neuronale

Formule générale d'une EDO neuronale

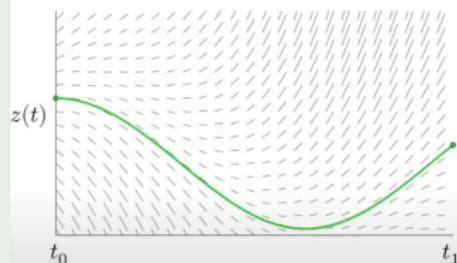
$$\frac{dz}{dt} = f(z(t), t, \theta)$$

Où :

- z : le vecteur qui représente les différents états des données à travers le réseau ;
- f : fonction qui décrit comment les éléments du vecteur d'état évoluent dans le temps.

Solution de l'EDO neuronale

$$z(t_1) = z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta) dt$$



Avantages des ODE-Net

- **Mémoire** : utilisation de la méthode de l'état adjoint à la place d'une rétropagation classique, on ne stocke pas les différents états ;
- **calcul adaptatif** : choix de la tolérance, les modèles adaptent les opérations pour respecter la tolérance, on peut échanger de la précision pour la vitesse en production ;
- **moins de paramètres** avec ODE-Nets en comparaison avec les réseaux classiques ;
- **transformations continues** induisent les modèles de CNF et de time-series.

Descente de gradient avec les EDO

Problème

Pour des réseaux à temps continu, la rétropropagation et la descente de gradient ont un cout mémoire élevé et font grimper l'erreur

Solution

Méthode de l'état *adjoint*, consiste à résoudre une seconde équation différentielle, obtenue à partir de la première, en temps inverse

Adjoint sensitivity method

On note L l'erreur. On pose $a(t) = \frac{\partial L}{\partial z(t)}$ l'adjoint.

Adjoint state

La sensibilité a vérifie l'équation différentielle

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

Cette équation doit être résolue à partir de la valeur initiale de $\frac{\partial f}{\partial z(t_1)}$, en remontant le temps. Les $z(t)$ successifs doivent être recalculés en même temps que les sensibilités.

Pour pouvoir optimiser le modèle, il faut calculer le gradient de la perte par rapport aux paramètres (*dynamics*). On cherche donc à évaluer $\frac{\partial L}{\partial \theta}$ qui dépend à la fois de $z(t)$ et de $a(t)$:

Gradient de la perte par rapport à θ

$$\frac{\partial L}{\partial \theta} = \int_{t_1}^{t_0} a(t)^T \frac{\partial f(z(t), t, \theta)}{\partial \theta} dt$$

Les produits $a(t)^T \frac{\partial f}{\partial z}$ et $a(t)^T \frac{\partial f}{\partial \theta}$ sont évalués numériquement par différentiation automatique.

Schématisation

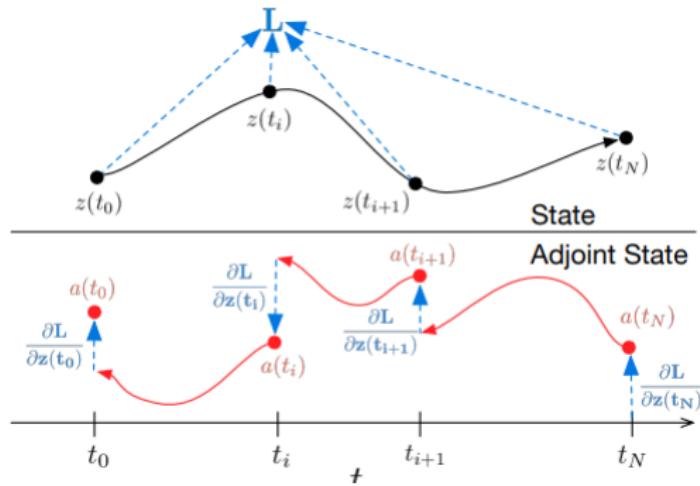


Figure – Méthode de l'état adjoint

Algorithme pour l'adjoint sensitivity method

Algorithm 1 Reverse-mode derivative of an ODE initial value problem

```
Input: dynamics parameters  $\theta$ , start time  $t_0$ , stop time  $t_1$ , final state  $\mathbf{z}(t_1)$ , loss gradient  $\frac{\partial L}{\partial \mathbf{z}(t_1)}$ 
 $s_0 = [\mathbf{z}(t_1), \frac{\partial L}{\partial \mathbf{z}(t_1)}, \mathbf{0}_{|\theta|}]$                                 ▷ Define initial augmented state
def aug_dynamics([ $\mathbf{z}(t)$ ,  $\mathbf{a}(t)$ ,  $\cdot$ ],  $t, \theta$ ):          ▷ Define dynamics on augmented state
    return [ $f(\mathbf{z}(t), t, \theta)$ ,  $-\mathbf{a}(t)^\top \frac{\partial f}{\partial \mathbf{z}}$ ,  $-\mathbf{a}(t)^\top \frac{\partial f}{\partial \theta}$ ]      ▷ Compute vector-Jacobian products
    [ $\mathbf{z}(t_0)$ ,  $\frac{\partial L}{\partial \mathbf{z}(t_0)}$ ,  $\frac{\partial L}{\partial \theta}$ ] = ODESolve( $s_0$ , aug_dynamics,  $t_1, t_0, \theta$ )      ▷ Solve reverse-time ODE
return  $\frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}$                                               ▷ Return gradients
```

Figure – Algorithme pour calculer dans le backward pass les $\mathbf{z}(t)$, les sensibilités ainsi que la dérivée de L par rapport à θ en un seul appel à un solveur

Remplacement des ResNets par des EDOs

Adams est utilisé comme solveur d'EDO avec une implémentation python de la méthode de l'état adjoint pour entraîner un ODENet dans le cas supervisé. Ce modèle est comparé avec presque le même mais où la méthode de l'état adjoint est remplacée par une rétropropagation directe avec Runge-Kutta, et avec un ResNet constitué de 6 couches. Les résultats sont résumés dans le tableau suivant.

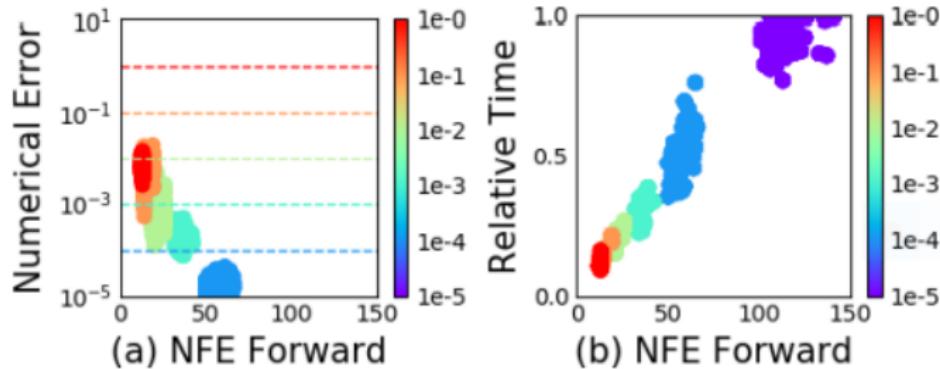
Comparaison

	Erreur test	# params	Mémoire	Temps
1-Layer MLP	1.60%	0.24M	-	-
ResNet	0.41%	0.60M	$\mathcal{O}(L)$	$\mathcal{O}(L)$
RK-Net	0.47%	0.22M	$\mathcal{O}(\hat{L})$	$\mathcal{O}(\hat{L})$
ODE-Net	0.42%	0.22M	$\mathcal{O}(1)$	$\mathcal{O}(\hat{L})$

L est le nombre de couches dans le réseau résiduel et \hat{L} est le nombre d'évaluations réalisées par le solveur. On observe un coût mémoire, en temps et un nombre de paramètres bien moins important pour l'ODENet en comparaison du ResNet.

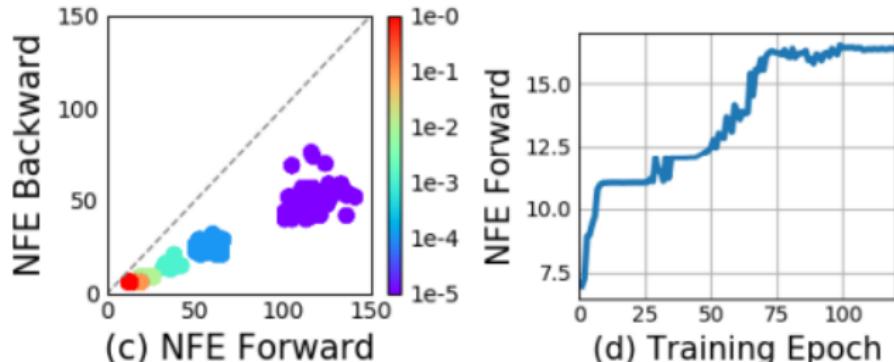
Erreurs et temps de calcul

NFE = Nombre d'évaluations réalisées par le solveur



L'erreur et le temps de calcul sont des fonctions linéaires du NFE
(on peut diminuer la tolérance mais avec un coût en temps plus important et vice-versa)

Nombre d'évaluations et complexité



Le nombre d'évaluations pour la méthode de l'état adjoint est à peu près la moitié de celui nécessaire lors du premier passage : coût en calcul et en mémoire plus avantageux face à une rétropropagation classique

Profondeur d'un ODE-Net

Dans le cas d'un ODE-Net la notion de profondeur n'est pas tout à fait claire.

Profondeur d'un ODE-Net

On peut considérer la profondeur comme le nombre d'évaluations effectuées par le solveur, mais ce nombre n'est pas fixé et varie selon les entrées.

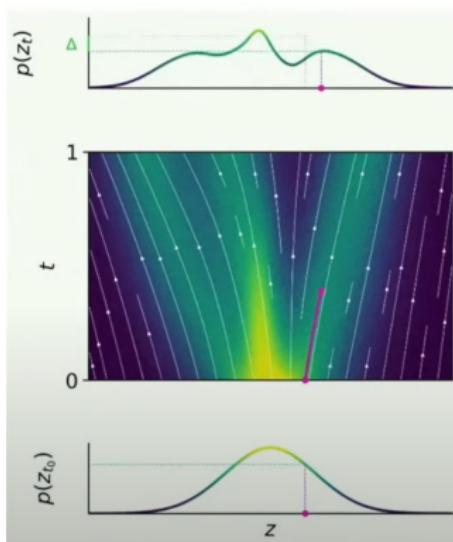
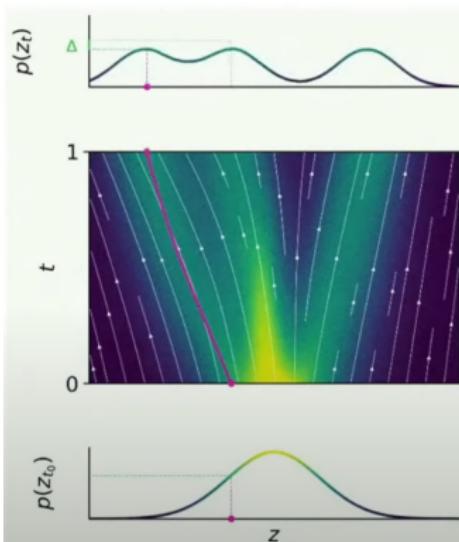
La profondeur est un détail qui dépend de l'entrée et qui est déterminée par le solveur lors de la résolution.

Continuous Normalizing Flow

CNF

Une famille de modèles de densité paramétrique qui utilise une transformation continue :

- Passer d'une densité de base simple comme une gaussienne en une densité paramétrique plus complexe



Formule de changement de variables VS Changement instantané des variables

Théorème du changement de variables

$$z_1 = f(z_0) \implies \log p(z_1) = \log p(z_0) - \log(\det \frac{\partial f}{\partial z_0})$$

- f doit être inversible.

Problème

- Le déterminant à un coût cubique $O(D^3)$.
- Nécessite beaucoup de couches.

Solution

Restreindre la jacobienne de la fonction dynamique de sorte qu'elle ait un jacobien structuré pour pouvoir utiliser les astuces algébriques afin de calculer efficacement le déterminant avec un coût de $O(D^2)$ ou $O(D)$.

Théorème du changement instantané de variables

$$\frac{dz}{dt} = f(z(t), t) \implies \frac{\partial \log(p(z(t)))}{\partial t} = -\text{tr}\left(\frac{\partial f}{\partial z(t)}\right)$$

Avantage

- f n'est pas nécessairement inversible.
- Utiliser n'importe quelle structure de f .
- Coût de la trace est toujours $O(D)$.
- Nécessite moins de couches.

Cas pratique

Exemple 1 d'utilisation du changement instantané de variables :

Nous prenons une densité normalisé qui est simple, nous la laissons suivre une EDO qui est définie par un réseau de neurone

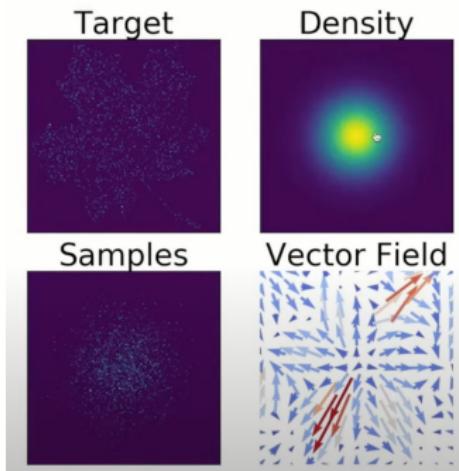


Figure – Avant la transformation

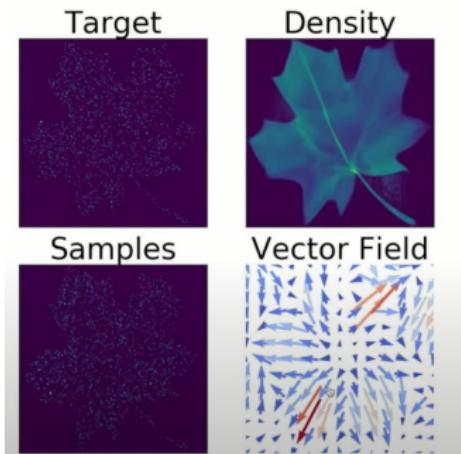
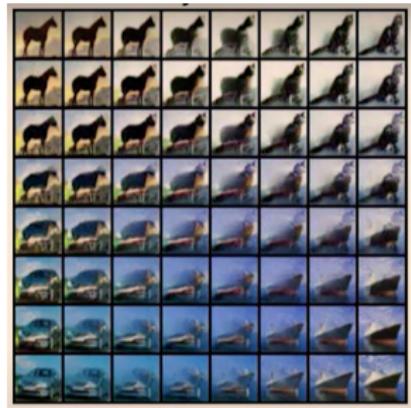
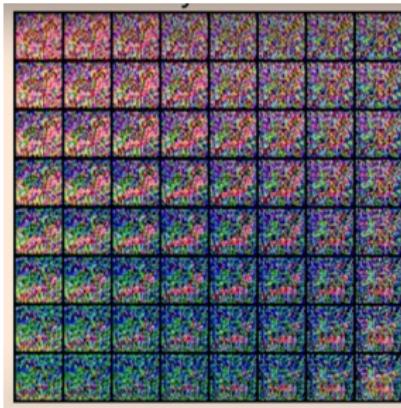
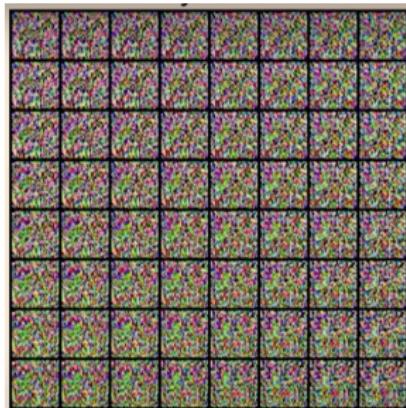


Figure – Après la transformation

Cas pratique

Exemple 2

Transformation du bruit gaussien en image de taille moyenne.



ODE pour les times-series

Problème

Des données qui ne sont pas réparties de manière régulière posent des difficultés aux réseaux de neurones en général. Les données manquantes, mal distribuées peuvent être gérées selon différentes stratégies (imputation, génération...)

Le modèle mis en place ici considère chaque série par une trajectoire, définie par un état initial $z(t_0)$, des paramètres propres à toute la série et des temps d'observations t_0, t_1, \dots, t_N

Latent state

On a alors

$$z_{t_0} \sim p(z_{t_0})$$

et

$$z_{t_1}, z_{t_2}, \dots, z_{t_N} = \text{Solveur}(z_{t_0}, f, \theta_f, t_0, \dots, t_N)$$

où l'observation x_{t_i} est rattaché à l'état latent z_{t_i} par

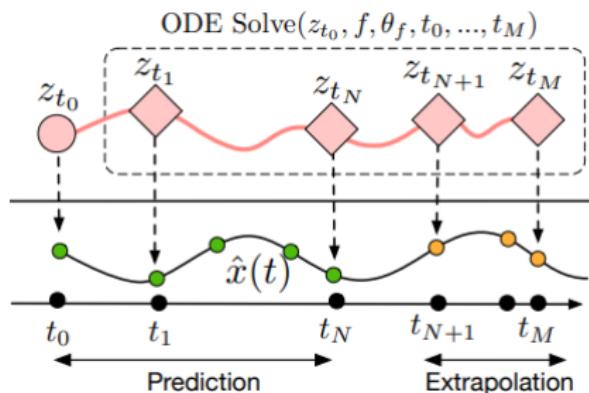
$$x_{t_i} \sim p(x|z_{t_i}, \theta_x)$$

f est une fonction ne dépendant pas du temps définie par :

$$f(z(t), \theta_f) = \frac{\partial z_t}{\partial t}$$

et paramétrisée par un réseau de neurones.

Extrapolation



Comme f ne dépend pas du temps, la trajectoire est unique et on est capable d'extrapoler les éléments de l'espace latent pour pouvoir faire des prédictions (n'importe où dans le temps)

Vraisemblance avec les processus de Poisson

L'observation d'un élément apporte des informations sur l'espace latent sous-jacent. On peut paramétriser l'arrivée des événements comme fonction de l'espace latent :

$$\mathcal{P}(\text{événement à l'instant } t | z(t)) = \lambda(z(t))$$

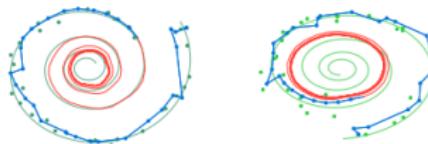
La log-vraisemblance pour des observations iid est donnée par :

$$\sum_i \log \lambda(z(t_i)) - \int_{t_0}^{t_\tau} \lambda(z(t)) dt$$

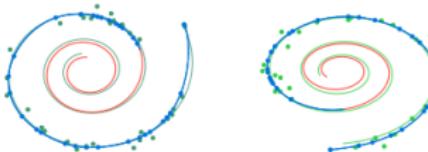
où λ est encore une fois paramétrisée par un réseau. La trajectoire et la vraisemblance peuvent être calculés en un seul appel un à solveur

Comparaison avec les RNN

Le modèle se comporte mieux qu'un RNN sur un time-série à intervalle non régulier (meilleures prédictions, meilleure extrapolation).

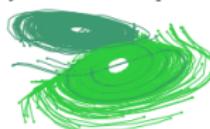


(a) Recurrent Neural Network



(b) Latent Neural Ordinary Differential Equation

- Ground Truth
- Observation
- Prediction
- Extrapolation



(c) Latent Trajectories

RMSE

Un jeu de données est généré et est tiré à chaque fois sans remise un nombre de points (ici 30, 50...). Du bruit est ajouté. Les modèles sont entraînés sur les tirages et le RMSE est calculé sur les 100 points au-delà de ceux utilisés en entraînement : « Predictive RMSE »

Nombre d'observations	30/100	50/100	100/100
RNN	0,3937	0,3202	0,1813
Latent ODE	0,1642	0,1502	0,1346

Inconvénients avec les ODE

- **Minibatching** : moins évident qu'avec des réseaux classiques.
On peut s'en sortir en combinant les équations différencielles, mais induit un coût supplémentaire par exemple si on veut contrôler l'erreur (on est obligé d'évaluer plus de fois que si on avait considéré les équations individuellement) ;
- **unicité** de la solution : condition de Lipschitz par rapport aux deux paramètres. En pratique c'est vérifié si les réseaux utilisent les bonnes fonctions d'activation ;
- on doit choisir une **tolérance** explicitement ;
- la **reconstruction de la trajectoire** lors de la méthode de l'état adjoint : peut induire une erreur numérique, on utilise le *checkpointing* : on stocke quelques valeurs de z (l'argument du coût mémoire est moins fort mais on garde une bonne précision)

Conclusion

Merci pour votre attention

Références

- <https://www.youtube.com/watch?v=MX1RJELWONc>
- <https://www.youtube.com/watch?v=V6nGT0Gakyg>
- Neural Ordinary Differential Equations :
Ricky T. Q. Chen*, Yulia Rubanova*, Jesse Bettencourt*,
David Duvenaud