Projet ALD - Neural ODEs

Mehdi LAKBAR - Léo PORTE

Janvier 2024

1 Introduction

La Neural ODE (Neural Ordinary Differential Equations, Chen et al. 2018) est une approche où l'on utilise des équations différentielles ordinaires pour modéliser la transformation des données dans un réseau de neurones. Contrairement aux réseaux de neurones traditionnels où les transformations sont discrètes (c'est-à-dire, une séquence fixe de couches), une Neural ODE représente cette transformation comme une fonction continue. En pratique, cela signifie que vous avez un système dynamique gouverné par une ODE où l'état du système représente les données à un certain niveau de transformation, et la dérivée de l'état (par rapport à la "profondeur" du réseau ou à un "temps" fictif) est une fonction paramétrée par les poids du réseau.

Un pendule amorti est un système dynamique classique caractérisé par des équations différentielles. Ce problème est donc parfaitement adapté à l'archictecture que nous allons explorer dans ce compte-rendu.

2 Formalisation du problème

Considérons un pendule amorti, avec l'équation différentielle qui décrit sa dynamique donnée par :

$$\ddot{\theta}_t + \omega_0^2 \sin(\theta_t) + \alpha \dot{\theta}_t = 0 \tag{1}$$

où :

- θ_t est l'angle du pendule par rapport à la direction verticale à l'instant t,
- $\dot{\theta}_t$ est la vitesse angulaire du pendule à l'instant t,
- $\ddot{\theta}_t$ est l'accélération angulaire du pendule à l'instant t,
- $\omega_0 = \sqrt{\frac{g}{l}}$ est une constante qui dépend de la gravité g et de la longueur l du pendule,
- $\alpha = \frac{k}{ml^2}$ est le coefficient de friction.

Le vecteur d'état du pendule à l'instant t est :

$$\mathbf{Y}_t = (\theta_t, \dot{\theta}_t) \tag{2}$$

La dynamique du pendule est alors décrite par :

$$\dot{\mathbf{Y}}_t = f(\mathbf{Y}_t) = (\dot{\theta}_t, -\omega_0^2 \sin(\theta_t) - \alpha\dot{\theta}_t) \tag{3}$$

où $f(\mathbf{Y}_t)$ dépend uniquement de l'état actuel $\mathbf{Y}_t$. La preuve de cette équation se trouve dans le notebook associé à ce compte-rendu, sous la question 1.1.

3 Prédiction de l'État d'un Pendule Amorti avec Neural ODE

L'objectif est de prédire l'état $\mathbf{Y}_t$ du pendule amorti à un instant t, en partant d'un état initial $\mathbf{Y}_0$, en utilisant l'approche Neural ODE. Le réseau de neurones va apprendre la dynamique sous-jacente du système représentée par la fonction $f(\mathbf{Y}_t; \theta)$.

3.1 Architecture du Réseau de Neurones

Le réseau de neurones prend l'état actuel $\mathbf{Y}_t$ en entrée et produit la dérivée de l'état, $\dot{\mathbf{Y}}_t = f(\mathbf{Y}_t; \theta)$, où θ sont les paramètres du réseau.

3.2 Processus de Prédiction

Pour prédire l'état à un instant t, on suit le processus suivant :

1. Initialiser le système avec l'état initial : $\mathbf{Y}_0$.
2. À chaque instant t, utiliser le réseau de neurones pour estimer la dérivée de l'état : $\dot{\mathbf{Y}}_t = f(\mathbf{Y}_t; \theta)$.
3. Utiliser un solveur ODE numérique pour intégrer la dérivée et mettre à jour l'état du système :

$$\mathbf{Y}_{t+1} = \mathbf{Y}_t + \int_t^{t+\Delta t} f(\mathbf{Y}_s; \theta) ds \tag{4}$$

4. Répéter les étapes 2 et 3 jusqu'à atteindre l'instant souhaité.

Le réseau de neurones et le solveur ODE travaillent conjointement pour prédire l'évolution du système dans le temps, en partant de l'état initial.

3.3 Entraînement du Modèle

Le réseau est entraîné en ajustant les paramètres θ pour que la trajectoire prédite par le modèle corresponde aux données observées. On utilise des données de trajectoires réelles du pendule pour effectuer cet apprentissage.

3.4 Utilisation du Modèle pour la Prédiction

Une fois le modèle entraîné, on peut l'utiliser pour prédire l'état du pendule à n'importe quel instant t, en partant de n'importe quel état initial $\mathbf{Y}_0$. Cela permet de simuler la dynamique du pendule sous différentes conditions initiales et paramètres.

4 Résultats

Voici nos résultats de prédiction de trajectoire :

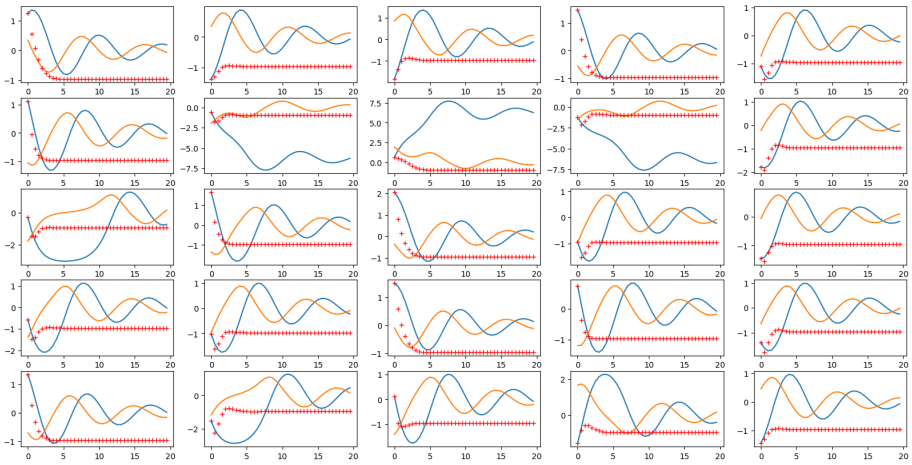
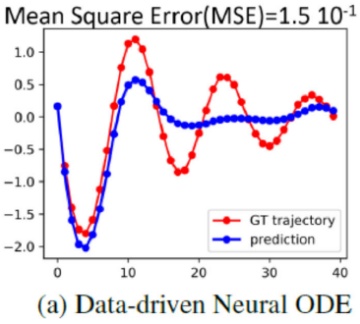


Figure 1: Résultats sur le batch de test de notre modèle (bleu = θ, orange = $\dot{\theta}$, rouge = prédiction de θ)

En comparant nos résultats avec ceux du cours, à savoir :



(a) Data-driven Neural ODE

Figure 2: Résultats cours Data-driven model

Et en sachant que notre MSE varie entre 3 et 10, nous pouvons en conclure

que bien que nos prédictions ont un certain sens si l'on s'en tient aux résultats du cours (des variations sur la courbe amoindries avec le temps), force est de constater que nos résultats sont en deçà de ce qui pourrait être attendu. Notre MSE fait un bond énorme par rapport à celle du cours, puisque nos trajectoires prédites sont trop "lisses".

Les raisons de cet écart pourraient être attribuées au modèle utilisé pour le DerivativeEstimator, qui serait trop simple (MLP de 2 layers avec une activation ReLU). Une autre hypothèse serait que notre training loop ne fasse pas assez d'epochs, mais nous choisissons de maintenir ces résultats puisque les hyper-paramètres utilisés étaient prédéfinis. Dans tous les cas, un problème de manque de complexité du modèle peut aisément être pointé par rapport à d'autres méthodes que nous verrons plus tard dans ce compte-rendu.

5 Remplacement des residual networks par des ODEs pour de l'apprentissage supervisé

Dans la section intitulée "Replacing residual networks with ODEs for supervised learning" du papier de **Chen et al. 2018**, les auteurs explorent l'utilisation de Neural ODEs pour l'apprentissage supervisé. Ils expérimentent avec des architectures de modèle où des réseaux résiduels classiques sont remplacés par des solveurs d'ODE, introduisant ainsi des dynamiques continues dans les modèles de réseaux de neurones.

Les Neural ODEs sont une généralisation des réseaux résiduels où la transformation de la couche est dictée par une ODE. Au lieu de couche par couche, un Neural ODE utilise un solveur d'ODE pour déterminer la transformation continue des caractéristiques d'entrée en sortie. Cela permet d'adapter dynamiquement la complexité du modèle en fonction des données, une caractéristique qui n'est pas possible avec les architectures fixes traditionnelles.

Pour résoudre les problèmes de valeur initiale ODE numériquement, les auteurs utilisent la méthode Adams implicite. Cette méthode est préférée pour sa robustesse par rapport aux méthodes explicites comme Runge-Kutta, bien qu'elle nécessite de résoudre un problème d'optimisation non linéaire à chaque étape, rendant la rétropropagation directe à travers l'intégrateur difficile. Pour surmonter cela, ils implémentent la méthode de sensibilité adjointe dans le cadre autograd de Python. En termes d'architecture de modèle, ils expérimentent avec un réseau résiduel qui effectue deux sous-échantillonnages et applique six blocs résiduels standards, remplacés par un module ODESolve dans la variante ODE-Net. Ils comparent également avec un réseau ayant la même architecture mais avec des gradients rétropropagés directement à travers un intégrateur de Runge-Kutta, appelé RK-Net.

La performance des ODE-Nets et des RK-Nets est trouvée être similaire à celle des ResNets traditionnels tout en utilisant moins de paramètres. Les ODE-Nets permettent également un contrôle de l'erreur, car les solveurs d'ODE peuvent approximativement assurer que la sortie est dans une tolérance donnée

de la solution vraie, permettant ainsi un équilibre entre précision et coût computationnel.

La profondeur du réseau dans le contexte des Neural ODEs n'est pas clairement définie comme dans les réseaux traditionnels. Au lieu de cela, elle est liée au nombre d'évaluations des dynamiques d'état caché nécessaires, qui est délégué au solveur d'ODE et dépend de l'état initial ou de l'entrée. Cette quantité peut être interprétée comme un nombre implicite de couches, qui s'adapte et augmente au cours de la formation, probablement en réponse à une complexité croissante du modèle.

5.1 Implémentation

Nous avons repris l'implémentation de l'auteur pour nos expérimentations (`https://github.com/rtqichen/torchdiffeq/`), en l'adaptant au contexte Google Colab.

5.2 Caractéristiques et résultats

Le nombre de paramètres de notre réseau (208266) est légèrement inférieur à celui du papier (220k). Cela peut être attribué à notre manière de définir le réseau qui est légèrement différente de celle du papier (utilisation de nn.Sequential au lieu des listes).

L'impact sur les résultats de cette différence semble minime, car au bout de 20 epochs nous avons déjà une test error de 0.55%, contre 0.42% dans le papier pour 160 epochs. Étant donné que nos résultats sont du même ordre que ceux du papier et que l'accuracy augmente proportionnellement au nombre d'epochs, nous pouvons dire que nos résultats sont cohérents et pourraient même concurrencer ceux du papier, puisque notre modèle permet une meilleure généralisation.

Enfin, nous obtenons les graphiques suivants :

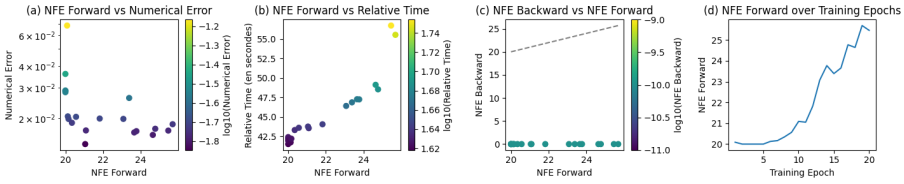


Figure 3: Nos résultats

Et ceux du papier sont les suivants :

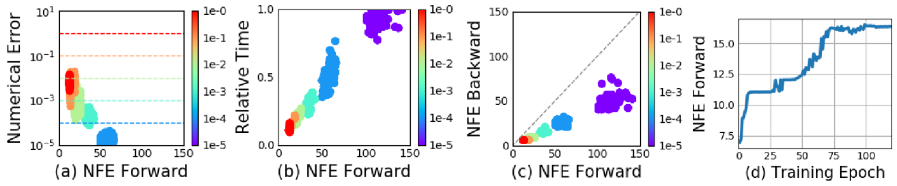


Figure 4: Résultats du papier

A nombre d'epochs égal, nos résultats sont cohérents avec ceux du papier.

6 4D-Var

Le 4D-Var (**Learning 4DVAR inversion directly from observations.** Arthur Filoche, Julien Brajard, Anastase Charantonis, Dominique Béréziat. ICASSP 2023), ou assimilation de données variationnelle en quatre dimensions, est une méthode d'optimisation qui intègre des observations au sein d'un modèle dynamique pour améliorer les prévisions des systèmes physiques, comme le pendule amorti. Cette technique s'avère cruciale pour préciser les états cachés et les paramètres non observables du système, tels que le coefficient de frottement dans le cas d'un pendule. Par l'ajustement des états et paramètres pour correspondre aux observations dans une fenêtre temporelle, le 4D-Var réduit les incertitudes et affine les conditions initiales, ce qui est particulièrement bénéfique pour les prédictions à court terme des systèmes non linéaires.

L'efficacité de cette méthode repose sur la capacité à modéliser les non-linéarités et à ajuster la trajectoire du système pour qu'elle soit la plus probable au regard des observations disponibles. L'implémentation du 4D-Var pour le pendule amorti requiert un modèle numérique robuste, une série temporelle d'observations précises, et un algorithme d'optimisation performant, capable de minimiser les écarts entre les prédictions du modèle et les observations tout en gérant les non-linéarités inhérentes au système.

Concrètement, le 4D-Var va permettre de prédire le y0 à partir des observations faites en aval, de sorte à réaliser un meilleur forecast dans le cas de nouvelles observations faites à partir de nouveaux y0.

6.1 Implémentation

Pour implémenter cette technique, nous avons dû redéfinir le modèle physique du pendule, et la loss utilisée sera une MSE sur les angles du pendule uniquement sur certains pas de temps (T_5, T_10, T_15, T_20 dans notre cas).

Pour l'optimiseur, nous avons choisi l'optimiseur LBFGS. L'algorithme L-BFGS est préféré dans le contexte de l'assimilation de données 4D-Var pour des

systèmes comme le pendule amorti en raison de sa convergence rapide sans le besoin de calculer explicitement la Hessienne. Cet optimiseur est adapté aux fonctions non linéaires complexes et offre une gestion efficace de la mémoire, une caractéristique importante lors de la manipulation de modèles dynamiques. L-BFGS ajuste automatiquement la taille du pas, éliminant la nécessité de définir un taux d'apprentissage spécifique, et est robuste face aux fonctions objectifs bruitées ou mal conditionnées, ce qui est essentiel pour intégrer des observations dans les modèles pour affiner les prédictions du système.

6.2 Résultats

Voici un échantillon de résultat :

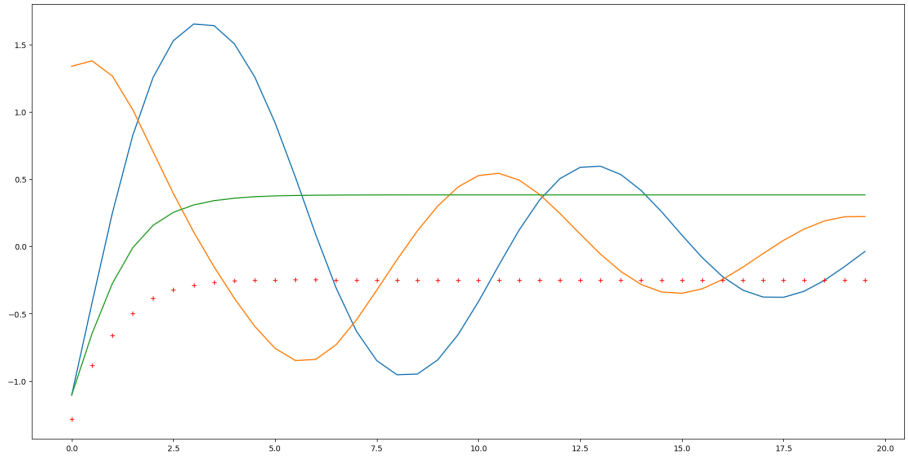


Figure 5: Résultat obtenu avec 4D-Var

Dans ce résultat, la courbe bleue correspond à l'angle du pendule, la courbe orange à la vitesse angulaire, la courbe verte à la prédiction du forecaster et la courbe rouge en pointillés à la prédiction du forecaster en utilisant le y0 obtenu avec le 4D-Var.

Nous pouvons voir que le y0 prédit n'est pas exactement là où est le vrai y0 (il est un peu plus bas). Notre modèle pour le 4D-Var apparaît ici comme étant robuste car il a prédit un résultat très similaire à celui avec le vrai y0, ce qui démontre également que notre forecaster est bien efficient, alors que cela ne sautait pas aux yeux au premier abord.

Notre forecaster manque donc uniquement d'expressivité mais est cohérent, nous avons pu nous en assurer grâce à 4D-Var.

7 APHYNITY

Le cadre APHYNITY (**Augmenting Physical Models with Deep Networks for Complex Dynamics Forecasting.** Yuan Yin, Vincent Le Guen, Jérémie Dona, Emmanuel de Bézenac, Ibrahim Ayed, Nicolas Thome, Patrick Gallinari. ICLR 2021.) représente une avancée significative dans la modélisation et la prévision des systèmes dynamiques tels que le pendule amorti, combinant habilement les modèles physiques basés sur les principes de la dynamique classique avec les capacités de modélisation des données des réseaux de neurones profonds. Cette méthode hybride permet de combler les lacunes des modèles purement physiques ou purement basés sur les données, en tirant parti de la précision des équations différentielles bien établies tout en capturant les dynamiques subtiles ou complexes à travers un apprentissage automatique avancé. APHYNITY sépare intentionnellement les contributions du modèle en deux composantes distinctes : une qui encapsule la physique connue et une autre qui est inférée à partir des données, ce qui permet une analyse plus fine des dynamiques du système.

En se concentrant sur une décomposition en composants physiques et pilotés par les données, APHYNITY est capable de fournir une estimation plus précise des paramètres physiques du système. Cela est réalisé en ajustant les paramètres du modèle pour qu'ils correspondent aux données observées, une étape cruciale pour la prévision précise de la dynamique du pendule. Cette approche basée sur la trajectoire est particulièrement pertinente pour les systèmes comme le pendule amorti, où la temporalité et la séquence des états sont essentielles pour comprendre et prédire le comportement.

En outre, APHYNITY se distingue par sa flexibilité et son adaptabilité. Le cadre peut être appliqué à divers niveaux d'approximation des modèles physiques, ce qui le rend particulièrement utile dans les situations où une connaissance physique complète n'est pas disponible ou serait autrement trop coûteuse à simuler avec précision. En veillant à ce que le composant piloté par les données capture uniquement les aspects de la dynamique non décrits par le modèle physique, APHYNITY améliore non seulement la généralisation du modèle prédictif mais aussi son interprétabilité, facilitant ainsi la compréhension des mécanismes sous-jacents et des comportements imprévus du système étudié.

7.1 Implémentation

Étant donné qu'APHYNITY a été testé sur le pendule amorti avec un modèle physique exactement identique au nôtre, nous allons utiliser l'implémentation officielle (`https://github.com/yuan-yin/APHYNITY`) pour entraîner le modèle pour notre contexte et l'utiliser pour faire les prédictions ensuite.

Nous avons choisi le modèle le plus basique possible, c'est-à-dire celui avec les paramètres par défaut définis dans le git. Plus exactement, nous avons pris

le modèle incomplet (simplification des PDEs), avec augmentation des données.

7.2 Résultats

Voici un échantillon de nos résultats :

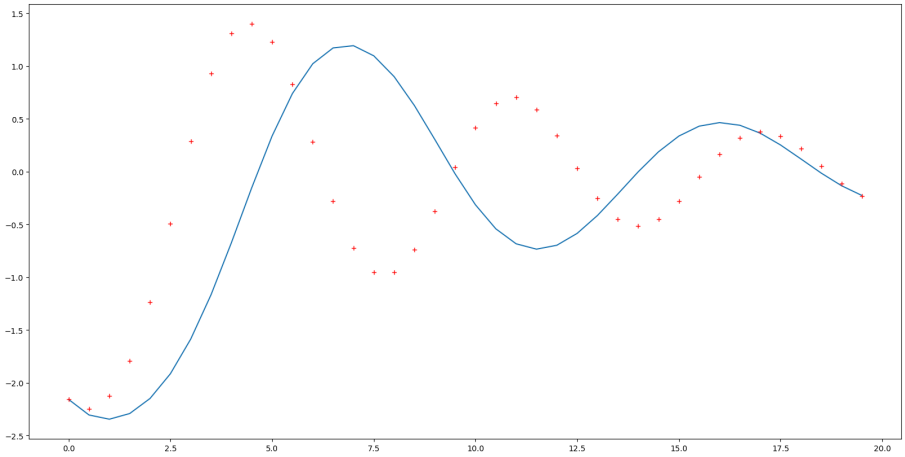


Figure 6: Visualisation des résultats. Bleu = theta réel, rouge = theta prédit avec APHYNITY

Nous pouvons constater qu'APHYNITY prédit bien plus fidèlement le mouvement du pendule que notre forecaster précédent. En effet, APHYNITY est bien plus précis que notre réseau défini précédemment, pour les raisons susmentionnées.

Cependant, nous pouvons constater que bien que l'amplitude et la forme générale des courbes est assez fidèle à la réalité, la fréquence du mouvement prédit par APHYNITY semble plus grande, résultant en le fait qu'APHYNITY prédise 3 mouvements quasi complets du pendule, alors que le pendule réel n'a même pas fait 2 mouvements entiers. Après investigation, cette différence est due au modèle physique utilisé dans le training d'APHYNITY. En effet, ce dernier est incomplet, il manque la partie $-\alpha * p$ pour $\frac{dp}{dt}$, chose qu'il y a dans notre modèle physique. α est le coefficient de frottement, il est donc normal que le pendule réel soit plus lent que la prédiction d'APHYNITY, qui ne tient pas compte de ces ralentissements.