Rapport Implémentation du DAN

Song Mickael
Superviseur : Zhang SiXin

ENSEEIHT & INSA Toulouse - 5ModIA
2023-2024

1 Introduction

L'objectif de ce projet est d'implémenter un Dynamic Artificial Neural Network (DAN) pour modéliser le système linéaire 2D avec une dynamique hamiltonienne périodique.

Pour l'étape de propagation, nous considérons que :

$$x_t = Mx_{t-1} + \eta_t$$

avec $\eta_t \sim \mathcal{N}(\mathbf{0}, \sigma_p \mathbf{I})$

Pour l'étape d'observation, nous considérons que :

$$y_t = Hx_t + \epsilon_t$$

avec $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \sigma_0 \mathbf{I})$

Dans notre cas précis, H = I et

$$M = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

avec $\theta = \pi/100$

Nous avons effectué nos expériences avec des batchs de 256 simulations en parallèle.

2 Data Assimilation network

Notre DAN est composé de 3 réseaux distincts :

— l'analyser a (class ConstructorA) qui assimile une observation

— le propagator b (class ConstructorB) qui propage le modèle

— le procoder c (class ConstructorC) qui estime la distribution

2.1 Pré-entrainement

Nous commençons par pré-entraîner notre procoder, ce qui nous permet d'approximer la distribution initiale $p(x_0)$. Ensuite, la sortie du réseau c est utilisée en entrée du module Gaussien pour initialiser une distribution gaussienne multivariée. Notre objectif est de minimiser la fonction de perte $L_0(q_0^a)$dans ce processus. Le constructeur C est une concatenation d'un module Fully-Connected (6 entrées et 4 sorties) et d'un module Gaussien (μ, Λ) où on applique sur les diagonale de Λ un seuil min-max(-8,8) (voir le code python).

Pour accélerer le calcul du module Gaussien, nous avons effectué des opérations par batch de 256 grâce aux fonctions torch.bmm et torch.triangular solve .

Nous obtenons ces résultats :
INIT a0 mean tensor([3.0001, 3.0000], grad_fn=<SliceBackward0>)
INIT a0 var tensor([7.9684e-05, 1.1324e-04], grad_fn=<SliceBackward0>)
INIT a0 covar tensor([[7.9684e-05, -3.1732e-06], [-3.1732e-06, 1.1337e-04]], grad_fn=<SliceBackward0>)

Une fois que le procoder est pré-entraîné, nous pouvons alors procéder à l'entraînement complet des trois réseaux du DAN.

2.2 Entraînement complet des constructeurs a, b et c

On cherche maintenant à entraîner les réseaux a, b et c. On prend un pas de temps T = 50. Le forward step dans la classe DAN est définit dans le cours comme suit :

— Calcul de la loss et du gradient selon t.

— Entrée initiale : h_0^a

— Entrée : h_{t-1}
— Sorties : $L_t(q_t^b) + L_t(q_t^a)$, h_t^a
— Formules :
  — $h_t^b = b(h_{t-1})$
  — $q_t = c(h_t)$
  — $h_t^a = a(h_t^b, y_t)$
  — $q_t^a = c(h_t^a)$

On minimise au cours de l'entraînement la fonction de perte suivante (full mode) :

$$(1/T) * \sum_{t=1}^{T} \big(L_t(q_t^b) + L_t(q_t^a)\big) + L_0(q_0^a)$$

ou (online mode)

$$L_t(q_t^b) + L_t(q_t^a)$$

2.3 Paramètres et résultats

L'entraînement a été réalisé sur au maximun 130 itérations. Il se déroulait avec des valeurs de t comprises entre 0 et 50 pas de temps. Le jeu de test comprend des valeurs de t entre 0 et 100, afin de voir si le DAN parvient à généraliser ce qu'il a appris en dehors du jeu d'entraînement.

Voici un tableau récapitulatif selon des valeurs de deep de 1,3 et 5 :

TABLE 1 – Résultats d'entraînement pour différentes valeurs de Deep

Valeur de Deep	Iteration	RMSE_b	RMSE_a	LOGPDF_b	LOGPDF_a	LOSS
1	1	4.166758	4.180621	-36229.633	-184992.793	221222.426
	130 (fin)	0.564910	2.563824	-2.130805	-6.925254	9.056059
3	1	4.019430	4.175898	-2001467.903	-215832.141	2217300.043
	70 (fin)	2.393785	2.730856	-2.665123	-5.092683	7.757806
5	1	3.771260	4.174352	-499482.756	-177558.788	677041.543
	130 (fin)	0.509589	2.693341	-1.140506	-5.960700	7.101206

On voit sur la figure 1 trajectoire prédite par le DAN est très proche de la trajectoire réelle : le réseau a bien assimilé le fonctionnement du modèle et sait correctement intégrer les observations.

On voit sur la figure 2 que les performances du modèle sur le test est excellent. Mais il est biaisé par le fait que les observations sont parfaitement sur la trajectoire réelle et qu'il n'y a l'air de ne pas avoir de bruit sur ces observations.

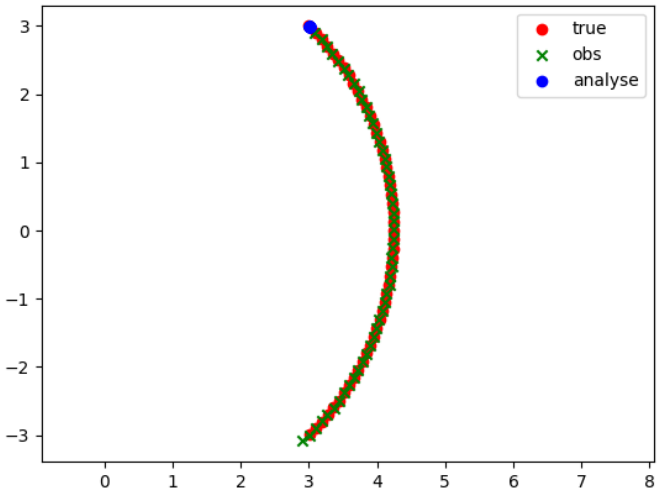

FIGURE 1 – Prédictions du DAN sur un échantillon de l'entraînement

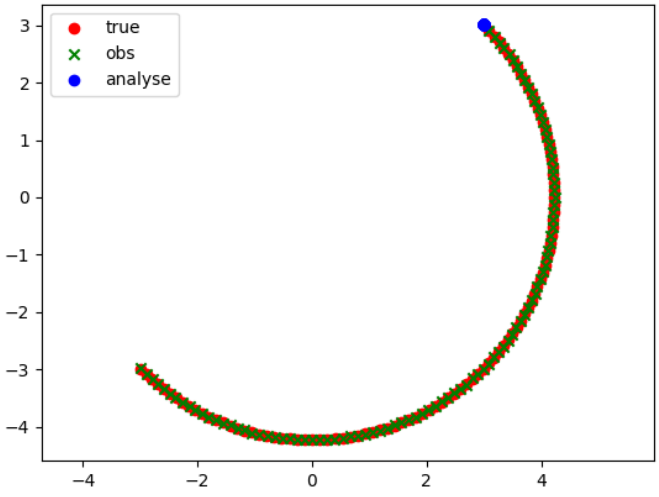

FIGURE 2 – Prédictions du DAN sur un échantillon test

3 Conclusion

Ce projet nous a permis d'explorer plus en détail l'implémentation et le fonctionnement d'un DAN.

Cependant les observations ne présentent aucun bruit (cela est implémenté dans la classe ConstructorObs), ce qui nous permet pas de conclure complètement sur l'efficacité du réseau à assimiler des observations et à généraliser.

Je n'ai pas eu le temps de tester les capacités du réseau avec une expérience plus complexe et non-linéaire (modèle de Lorentz 96).