Évaluation de la bibliothèque `bobodiff`

sur des réseaux de neurones profonds

Test de performance et validation de la différenciation automatique

23 juin 2025

Table des matières

1 Introduction

Après avoir validé le bon fonctionnement de ma bibliothèque `bobodiff` sur un modèle de régression multivariée (avec trois paramètres), j'ai décidé de la tester sur des **réseaux de neurones profonds**. L'objectif est de vérifier si `bobodiff`, développée pour effectuer de la différenciation automatique, est capable de suivre des gradients dans des architectures plus complexes.

2 Réseaux de neurones simples (XOR et make_moons)

2.1 Modèle XOR

Pour débuter, j'ai utilisé un réseau de neurones très simple : un **MLP** (perceptron multicouche) conçu pour résoudre le problème logique XOR. Le modèle utilisé est inspiré du code d'Andrej Karpathy dans son projet académique sur la différenciation automatique.

Le modèle a l'architecture suivante :

```
model = MLP(2, [4, 4, 1])
```

- **Entrée :** vecteurs de 2 scalaires (les bits du XOR)
- **Deux couches cachées :** 4 neurones chacune
- **Sortie :** 1 neurone pour la classification binaire (0 ou 1)

L'objectif est que le modèle apprenne à prédire correctement la sortie du XOR à partir des quatre combinaisons d'entrée : $[0, 0], [0, 1], [1, 0], [1, 1]$. Les poids et biais sont initialisés aléatoirement, puis entraînés via descente de gradient.

Résultats : après 100 époques, la loss devient quasi nulle, et le modèle prédit correctement toutes les sorties.

```
Epoque 0, Loss moyenne: 1.6353
Epoque 40, Loss moyenne: 0.0972
Epoque 80, Loss moyenne: 0.0001
Epoque 100, Loss moyenne: 0.0000
```

Résultats finaux sur XOR :

```
Entree: [0, 0] -> Prediction: 0.000, Attendu: 0
Entree: [0, 1] -> Prediction: 1.000, Attendu: 1
Entree: [1, 0] -> Prediction: 1.000, Attendu: 1
Entree: [1, 1] -> Prediction: 0.000, Attendu: 0
```

Ce test montre que `bobodiff` gère correctement la différenciation dans un MLP simple.

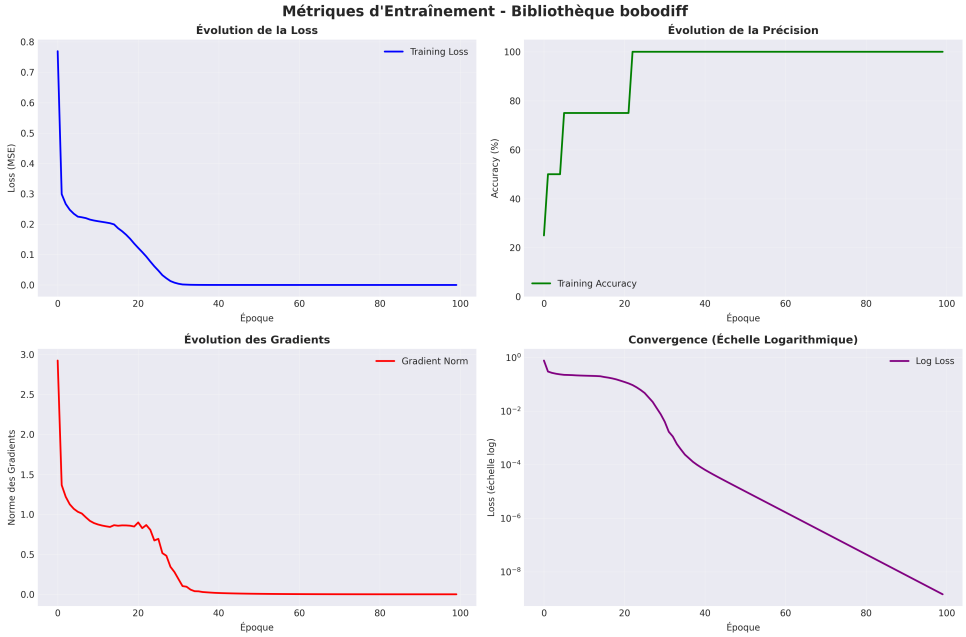

FIGURE 1 – Évolution de la loss pendant l'entraînement du modèle XOR

2.2 Modèle make_moons

J'ai ensuite testé `bobodiff` sur un modèle un peu plus complexe : un **classifieur binaire** appliqué au jeu de données `make_moons`. C'est un problème classique de classification non-linéaire. L'objectif est de séparer deux ensembles de points formant des demi-lunes.

Architecture du modèle :

```
model = MLP(2, [16, 16, 1])
```

- **Entrée :** coordonnées (x, y) d'un point dans le plan 2D
- **Deux couches cachées :** 16 neurones chacune
- **Sortie :** 1 seul neurone (classification binaire : score $> 0 \rightarrow$ classe $+1$, sinon classe -1)

```
Epoch 0:  Loss = 0.9460, Accuracy = 64.00%
Epoch 10: Loss = 0.2363, Accuracy = 90.00%
Epoch 50: Loss = 0.0698, Accuracy = 96.00%
Epoch 90: Loss = 0.0314, Accuracy = 100.00%

Accuracy finale : 100.00%
```

Encore une fois, les performances sont excellentes, prouvant que `bobodiff` sait gérer des réseaux de neurones non linéaires avec ReLU.

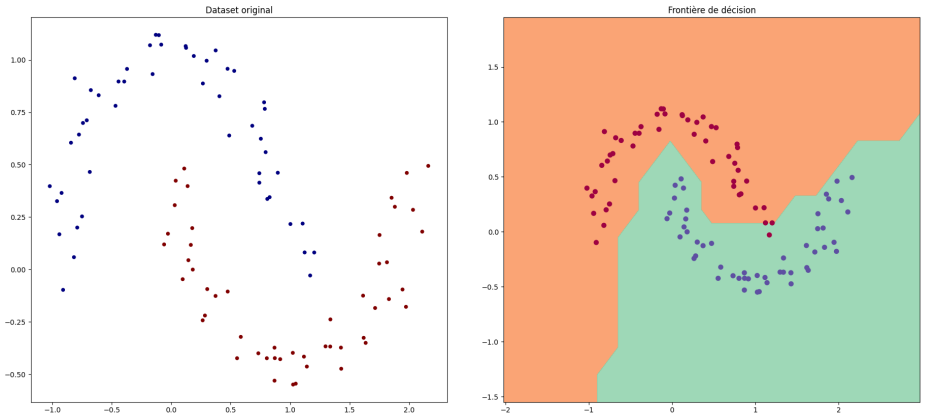

FIGURE 2 – Représentation des demi-lunes séparées par le classifieur binaire

3 Réseau dense sur MNIST

Pour aller plus loin, j'ai testé ma bibliothèque sur un vrai cas d'usage du deep learning : la **classification d'images manuscrites** avec MNIST. Le code utilisé provient du site de formation du CNRS, et le dataset est une référence dans le domaine. Il a été popularisé par **Yann LeCun**, l'un des pionniers du deep learning.

3.1 Description du dataset

— **Images :** 28×28 pixels (niveaux de gris)
— **Classes :** chiffres de 0 à 9 (10 classes)
— **Entraînement :** 60 000 images
— **Test :** 10 000 images

3.2 Architecture du réseau utilisé

TABLE 1 – Architecture du réseau dense pour MNIST

Couche	Type	Entrée	Neurones	Paramètres calculés
0	Dense	784 (28×28)	128	784×128 + 128 = **100 480**
1	Dense	128	64	128×64 + 64 = **8 256**
2	Dense	64	32	64×32 + 32 = **2 112**
3	Dense	32	10	32×10 + 10 = **330**

Sortie : 10 neurones (un score par chiffre)

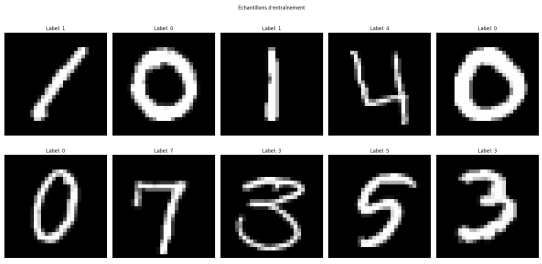

FIGURE 3 – Exemple d'image MNIST : Donnée d'entraînement

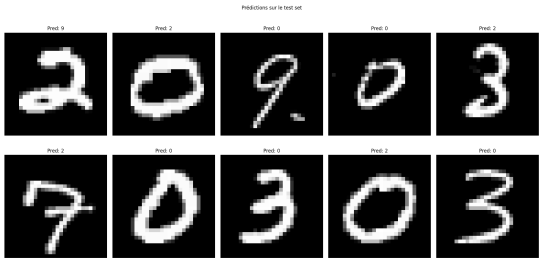

FIGURE 4 – Prédiction du modèle après entraînement

Résultat avec bobodiff :
Malheureusement, les performances sont très faibles : sur 10 sorties, le modèle n'en prédit correctement que 2. Soit une précision d'environ **20%**, bien en-dessous de ce qu'on attend (typiquement 97–99% avec PyTorch ou TensorFlow).

4 Analyse des limitations

Après analyse, plusieurs raisons expliquent ces mauvais résultats :

1. **Limites de performance :** à partir de 400 à 500 entrées, `bobodiff` plante souvent, notamment à cause de la récursivité dans son moteur d'autodiff.
2. **Pas d'optimisation bas niveau :** `bobodiff` est 100% en Python. Contrairement à des frameworks comme PyTorch (optimisés en C++/CUDA), il ne supporte pas bien les multiplications matricielles massives.
3. **Pas de Cython :** aucune accélération via Cython ou Numba n'a encore été intégrée dans la version actuelle.

5 Conclusion

Les tests montrent que ma bibliothèque `bobodiff` fonctionne très bien sur des modèles simples à intermédiaires comme le XOR ou `make_moons`. Elle gère bien les gradients, les structures MLP, et la classification binaire.

— Gestion correcte des gradients sur MLP simples
— Support des fonctions d'activation non-linéaires (ReLU)
— Classification binaire efficace
— Convergence stable sur petits datasets

Cependant, pour des modèles complexes comme ceux utilisés sur MNIST, elle atteint ses limites. Cela est dû aux lourdes opérations vectorielles et à l'absence d'optimisations natives.

Prochaine étape : intégrer des accélérations avec Cython ou Numba, ou transformer certaines parties critiques en C/C++ pour rendre `bobodiff` compétitif sur des tâches de deep learning à grande échelle.