Projet Deep Learning 2025 : Génération d'Images avec un Réseau Adversaire Génératif (GAN)

Cardi Julien, Ferroni Sandro et Moyo Kamdem Auren

1er décembre 2024

Table des matières

1 Introduction

Dans le cadre de notre cursus scolaire et de notre majeure SCIA-G, axée sur l'intelligence artificielle, nous avons eu l'opportunité de suivre des cours de *Deep Learning*. Ces cours visent à nous apprendre les concepts fondamentaux et avancés des réseaux de neurones, les architectures modernes, ainsi que leurs applications pratiques dans divers domaines, afin de nous préparer à développer des solutions basées sur l'apprentissage profond.

L'objectif principal de ce projet était d'implémenter un réseau antagoniste génératif (*Generative Adversarial Network*, GAN) en utilisant PyTorch, et de tester son efficacité sur différents ensembles de données de notre choix.

Le GAN a été choisi parmi les architectures avancées proposées pour son importance dans le domaine de la génération d'images. Les principaux défis incluaient l'implémentation de l'architecture, l'entraînement sur différents ensembles de données (MNIST, FashionMNIST, CIFAR10) et l'optimisation des hyperparamètres pour obtenir des résultats cohérents et visuellement plausibles.

En effet, pour ce projet, nous avons fait le choix de tester notre modèle sur différents ensembles de données précités, que nous détaillerons ultérieurement.

Dans ce rapport, nous commencerons par expliquer le fonctionnement des GANs, puis nous présenterons les ensembles de données utilisés et le travail effectué dessus. Nous détaillerons ensuite l'implémentation de notre code, ainsi que les résultats obtenus sur différents tests. Enfin, nous conclurons en explicitant les simplifications apportées et les limites de notre projet.

Notre projet est intégralement disponible sur notre GitHub : `https://github.com/CirSandro/DeepLearning-GAN`

2 Architecture choisie : GAN

Pour notre projet, nous avions le choix de l'architecture à implémenter et avons décidé de nous porter sur les réseaux adversaires génératifs (*Generative Adversarial Networks*, GAN). Ils ont été introduits par plusieurs membres du Département d'informatique et de recherche opérationnelle de l'Université de Montréal en 2014.

Un GAN est composé de deux réseaux neuronaux entraînés simultanément :

- Un **discriminateur**, chargé de différencier les échantillons réels des échantillons générés.
- Un **générateur**, qui tente de produire des échantillons réalistes pour tromper le discriminateur.

L'objectif est d'atteindre un équilibre où le générateur produit des données qui sont statistiquement indiscernables des données réelles.

Pour ce projet :

- Le **discriminateur** utilise une architecture basée sur des activations *Maxout* et des couches avec *Dropout* pour améliorer la généralisation.
- Le **générateur** produit des images normalisées entre $[-1, 1]$ en utilisant des couches $LeakyReLU$ suivies d'une activation *Tanh*.

Nous allons ainsi tester nos modèles sur différents ensembles de données.

3 Ensembles de données

Initialement, nous avons fait le choix de tester notre modèle sur **MNIST**, qui est un dataset d'images de chiffres manuscrits (28x28 en niveaux de gris). Cet ensemble de données est composé de 60 000 images d'entraînement et de 10 000 images de test.

Comme ce dataset reste assez simple, nous avons décidé d'utiliser 2 autres datasets avec des niveaux de difficulté différents.

FashionMNIST représente des images d'articles de mode (28x28 en niveaux de gris), assez similaires à celles de MNIST mais avec des formes plus difficiles à générer. La taille de ce dataset est similaire à celle de MNIST.

Le dernier dataset utilisé est **CIFAR-10**, qui représente des images RGB (rouge, vert, bleu) de taille 32x32. Ce dataset comprend 10 types d'images, telles que des bateaux, des avions, ou encore des oiseaux. Il est composé de 50 000 données d'entraînement et de 10 000 de test.

Quel que soit le dataset, nos données ont été normalisées dans l'intervalle $[-1, 1]$ et redimensionnées à la taille désirée.

4 Implémentation

Le projet a été structuré comme une bibliothèque Python avec les composants suivants :

- **data/dataset.py** : Chargement et prétraitement des ensembles de données. C'est ici que nos données sont normalisées.
- **models/discriminator.py** : Implémentation du réseau discriminateur avec comme couches *Linear*, *LeakyReLU*, *Dropout*, *Maxout*, *Dropout*, *Linear* et *Sigmoid*.
- **models/generator.py** : Implémentation du réseau générateur avec 3 couches *Linear* entrecoupées de 2 couches *LeakyReLU* et une couche *Tanh*.
- **utils/image_utils.py** : Fonctions pour la visualisation des images générées et des courbes de pertes de nos 2 modèles. Ces images sont enregistrées dans un dossier */results*.
- **main.py** : Script principal pour l'entraînement.
- **config.py** : Fichier de configuration pour ajuster les paramètres du modèle.
- **Makefile** : Fichier permettant de télécharger les bibliothèques nécessaires et de lancer notre modèle avec le dataset de notre choix.

Pour notre entraînement, nous alternons 5 entraînements du discriminateur et un entraînement du générateur, dans l'objectif que, pour chaque itération du générateur, le discriminateur soit suffisamment robuste pour détecter la différence entre les images réelles et celles générées à l'aide de bruit.

Nos pertes sont calculées à l'aide de *BCELoss*, et les poids des modèles sont mis à jour grâce aux optimiseurs *Adam*.

De plus, l'objectif de cet algorithme est de minimiser $\log(1 - D(G(z)))$. Cependant, au tout début de l'entraînement, il y a des risques de saturation. Ainsi, pour les 5 premières époques, nous cherchons plutôt à maximiser $\log D(G(z))$ pour notre générateur.

5 Expériences et résultats

5.1 Configuration

Dans l'objectif de trouver les meilleurs résultats possibles, après avoir implémenté nos algorithmes, nous avons fait varier les hyperparamètres ci-dessous pour chaque dataset.

- **Taille de batch** : 16, 36, 64
- **Nombre d'époques** : 20, 50, 100
- **Taux d'apprentissage** : 10^{-4}, 10^{-5}
- **Latent space** : 100

Malheureusement, nous n'avons pas pu tester l'intégralité des combinaisons possibles car le code est assez long à tourner.

5.2 Résultats obtenus

Pour chaque dataset, voici les résultats les plus performants que nous avons pu obtenir avec les hyperparamètres spécifiés.

Pour **MNIST** : [batch=32, epoch=100, lr=10^{-4}, latent=100] - figure 1 et 2

Pour **FashionMNIST** : [batch=32, epoch=100, lr=10^{-4}, latent=100] - figure 3 et 4

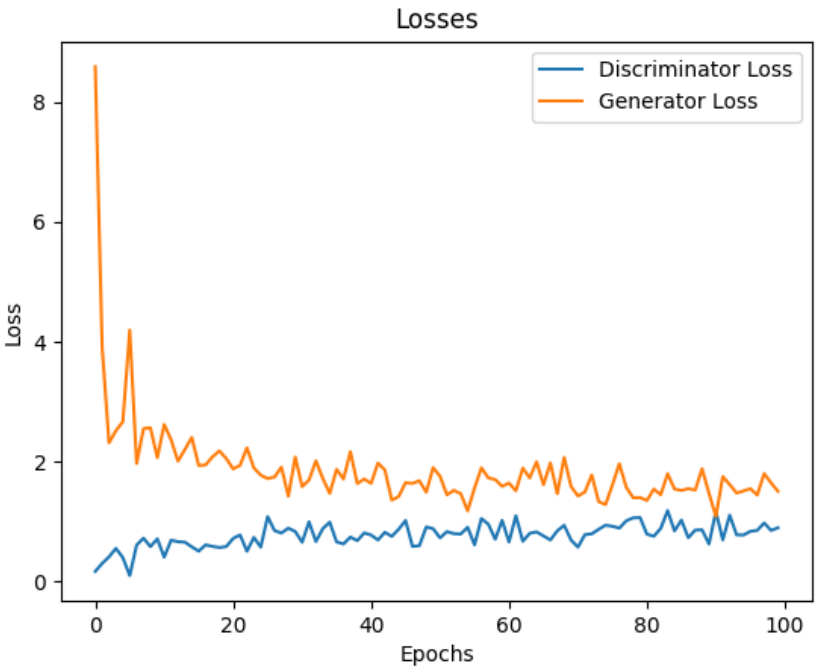

FIGURE 1 – Courbe de perte de nos modèles sur MNIST

FIGURE 2 – Images générées à la 100ème époque sur MNIST

Pour **CIFAR10** : [batch=32, epoch=100, lr=10^{-4}, latent=100] - figure 5 et 6

- **MNIST et FashionMNIST** : Le générateur produit des images reconnaissables et assez similaires à celles du dataset. Les courbes de pertes montrent une convergence stable.
- **CIFAR10** : La tâche est plus difficile en raison de la complexité des images, même si certaines formes commencent à être reconnaissables.

Les résultats (images générées et courbes de pertes) sont enregistrés dans le dossier `results`.

6 Simplifications et limitations

Pour tenir compte des contraintes de temps et de calcul :

- La taille des datasets ne peut être augmentée, cela prendrait beaucoup trop de temps d'exécution.
- La résolution des images a été standardisée à 28x28 (MNIST, FashionMNIST) et 32x32 (CIFAR10).
- Le nombre d'époques a été limité, ainsi nous n'obtenons pas les résultats les plus performants mais nous sommes limités par nos ordinateurs.
- Avec de meilleurs ordinateurs et plus de temps, nous aurions pu laisser tourner nos algorithmes avec plusieurs hyperparamètres possibles afin d'obtenir les plus performants.

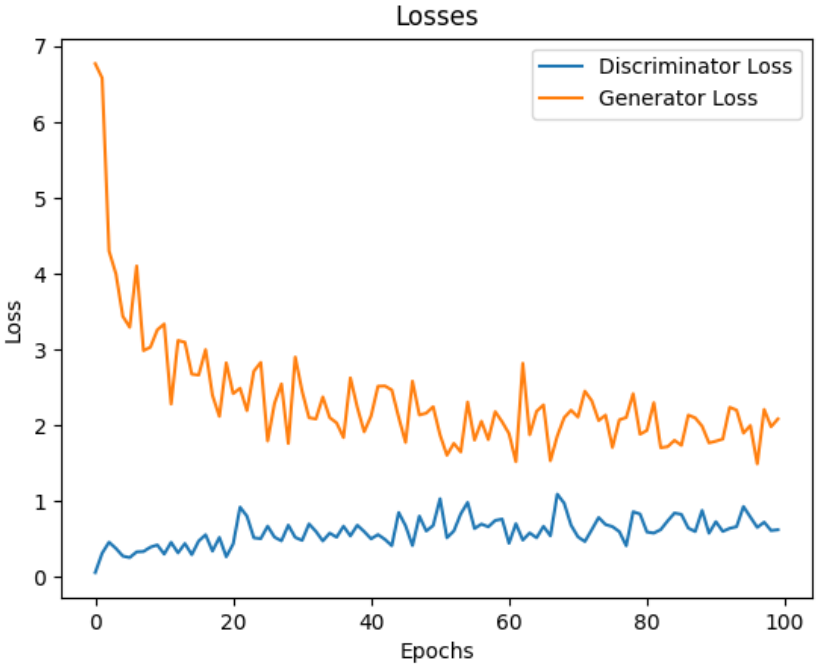

FIGURE 3 – Courbe de perte de nos modèles sur FashionMNIST

FIGURE 4 – Images générées à la 100ème époque sur FashionMNIST

7 Conclusion

En résumé, ce projet nous a permis d'appliquer les connaissances acquises dans le cadre du cours de Deep Learning, plus spécifiquement sur l'implémentation des réseaux adversaires génératifs (GAN). Nous avons exploré des techniques avancées telles que l'optimisation de modèles complexes, la gestion des hyperparamètres, et l'évaluation des performances sur plusieurs ensembles de données comme MNIST, FashionMNIST et CIFAR10.

Notre projet pourrait toutefois être amélioré de diverses manières. Par exemple, nous aurions pu tester davantage de changer les hyperparamètres comme la taille du batch ou le taux d'apprentissage, et augmenter la durée d'entraînement pour obtenir des résultats plus fins. Cependant, les limitations matérielles et temporelles ont restreint nos possibilités d'expérimentations.

Malgré ces contraintes, nous avons renforcé notre compréhension des GAN et des défis associés à leur entraînement, notamment la gestion de la saturation des fonctions de perte et l'équilibre entre le générateur et le discriminateur. Ce qui a permis d'obtenir des résultats plus satisfaisants, particulièrement pour MNIST.

Ce projet constitue une solide base pour de futures explorations dans le domaine des réseaux génératifs, et nous a permis de développer des compétences clés dans l'implémentation et l'optimisation de modèles de Deep Learning. Il représente une étape importante dans notre maîtrise des outils de l'intelligence artificielle et dans notre capacité à résoudre des problématiques concrètes, en particulier dans le domaine de la génération d'images.

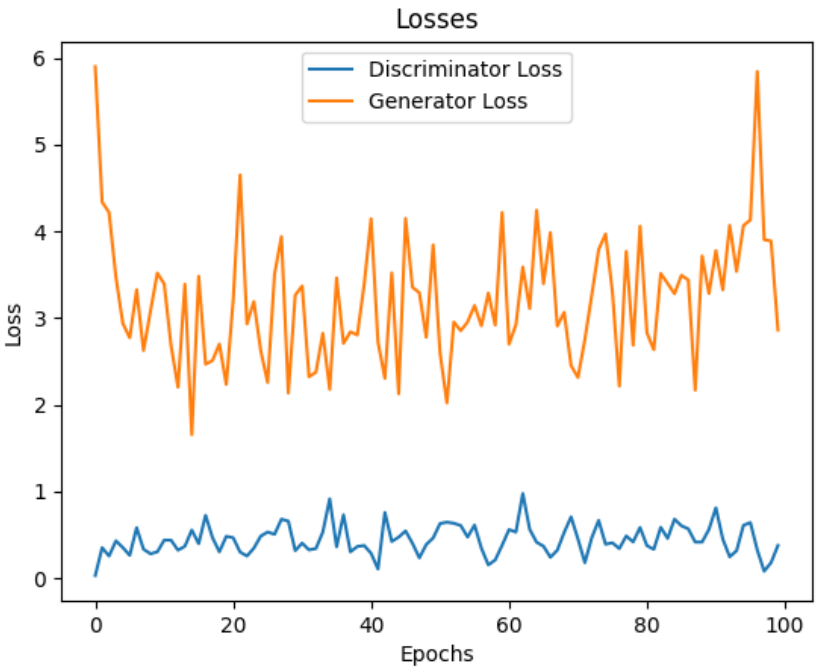

FIGURE 5 – Courbe de perte de nos modèles sur CIFAR10

FIGURE 6 – Images générées à la 100$^{\text{ème}}$ époque sur CIFAR10

8 Références

— *Generative Adversarial Nets*, `https://arxiv.org/pdf/1406.2661`
— Datasets :
  — MNIST : `http://yann.lecun.com/exdb/mnist/`
  — FashionMNIST : `https://github.com/zalandoresearch/fashion-mnist`
  — CIFAR10 : `https://www.cs.toronto.edu/~kriz/cifar.html`
— Github : `https://github.com/CirSandro/DeepLearning-GAN`