Machine Learning

Réseau de neurones

ZIDAT Lydia et FAURE Guillaume

Mai 2023

Table des matières

I. Introduction

Dans ce projet, nous avons implémenté un réseau de neurones sous le format de Pytorch organisé en plusieurs modules d'une librairie appelée Module :

- Linear
- Activation
- Loss
- Sequential
- Convolution
- mltools (module utilisé lors des TME permettant de générer des jeux de données)

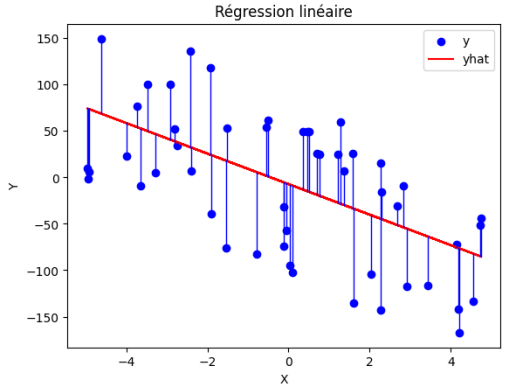
Un dossier appelé Tests regroupe des jupyter-notebooks dans lesquels nous avons réalisé les tests organisés selon les différentes parties de l'énoncé du projet.

Un dossier appelé Data comprend le dataset USPS. Les autres jeux de données ont été importés directement par le biais de la librairie Keras pour MNIST et Fashion-MNIST.

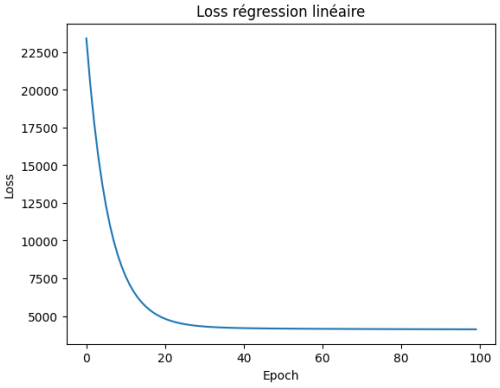
II. Mon premier est linéaire !

Dans cette partie nous avons implémenté le module Linéaire qui va être utilisé tout au long du projet. La principale difficulté de cette partie a été de trouver une initialisation correcte des paramètres W et b. En effet, nous avions initialisé les poids uniformément entre 0 et 1 ce qui entraînait une absence de convergence. Aussi, nous avons par la suite centré la distribution sur 0 ce qui a amélioré la convergence mais celle-ci n'était pas entièrement stable. Nous avons fait quelques recherches et opté pour une initialisation par la formule de Xavier.

Afin de tester ce premier module, nous avons en premier lieu réalisé une régression linéaire. Nous avons initialisé une distribution uniforme de X et avons créé Y = aX + b + bruit.
Nous ne montrerons ici que les résultats pour a = -20 et b = -10.
Nous avons utilisé une couche Linéaire avec un neurone dans cette couche, 100 epochs et un pas de gradient à 1e-4. La Loss utilisée était la MSE-Loss.



(a) Erreur entre la prédiction et valeur réelle



(b) MSE-Loss en fonction du nombre d'epochs

Nous avons ensuite réalisé une classification entre deux distributions équilibrées linéairement séparables avec un peu de bruit à partir de la fonction gen-arti du module mltools.
Pour réaliser cela, la droite séparant les deux distributions est produite à partir de la sortie du module Linéaire. [Figure 2]

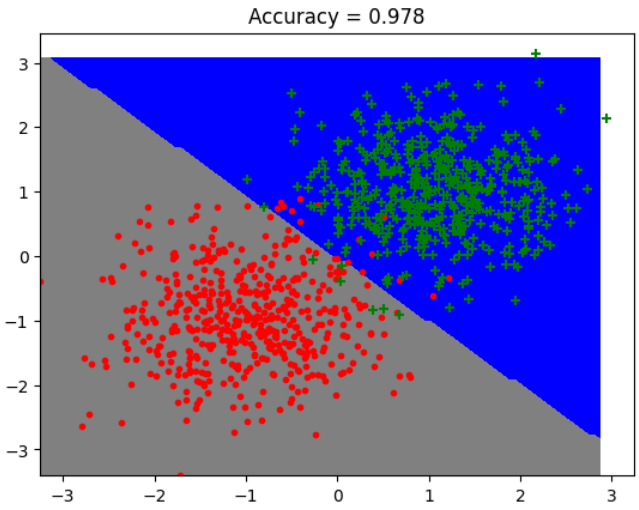
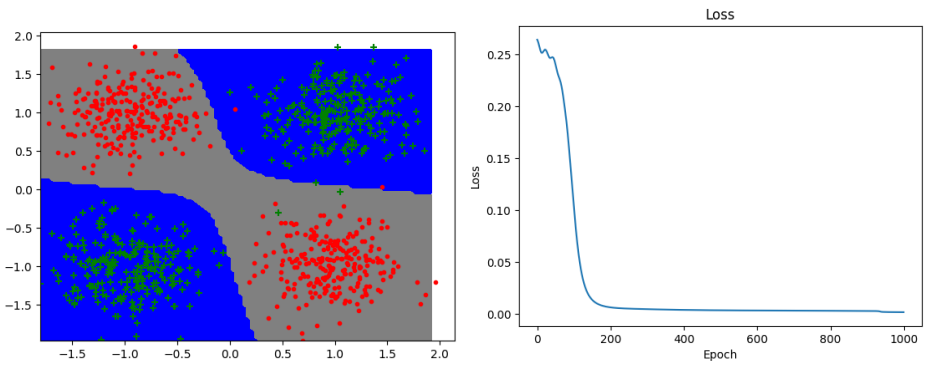


FIGURE 2 – Classification binaire entre deux distributions gaussiennes 2D

III. Mon deuxième est non-linéaire !

Dans cette partie nous avons décidé de nous pencher sur le problème du "ou exclusif" qui est un problème non linéaire. Ainsi, nous avons de nouveau utilisé gen-arti pour créer des distributions gaussiennes centrées sur (-1,-1),(-1,1),(1,-1) et (1,1).
Nous avons utilisé une couche Linéaire de 10 neurones suivi d'une fonction d'activation TanH puis une nouvelle couche Linéaire d'un neurone et d'une fonction d'activation Sigmoïde.

L'accuracy sur le groupe test était de 0.995.



(a) Classification sur données non linéairement séparables

(b) MSE-Loss en fonction du nombre d'epochs

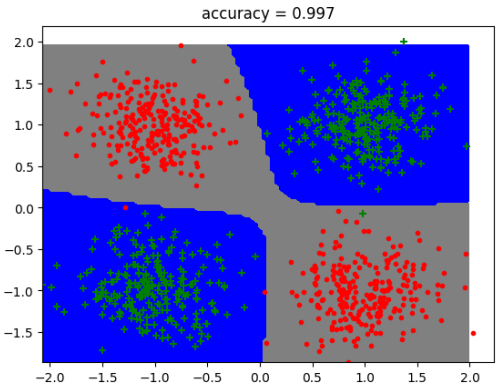
IV. Mon troisième est un encapsulage !

Dans cette partie, nous avons repris le même problème que précédemment à savoir le "ou exclusif". En effet, cette partie permet seulement de mettre en place l'ajout des différents modules et l'automatisation de l'apprentissage.

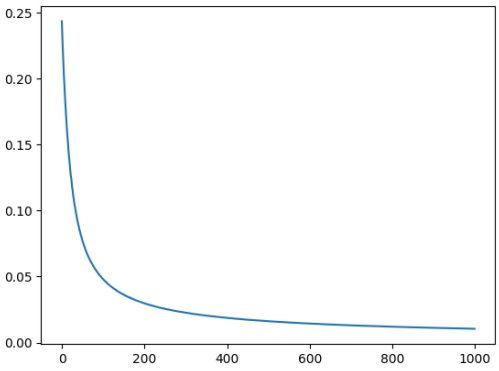
Une différence est à noter, contrairement aux deux premières parties, nous utilisons ici des mini-batchs de taille 200 pour un nombre de datas en train de 1000 soit 1/5ème. Une taille de batch à 1000 correspondrait à la méthode utilisée dans le chapitre précédent et une taille de batch à 1 correspondrait à une mise à jour du gradient de manière stochastique.

Le réseau de neurones est similaire au précédent : nous avons utilisé une couche Linéaire de 10 neurones suivie d'une fonction d'activation TanH puis une nouvelle couche Linéaire d'un neurone et une fonction d'activation Sigmoïde.

Les résultats étaient similaires avec une accuracy de 0.997.



(a) Classification sur données non linéairement séparables



(b) MSE-Loss en fonction du nombre d'epochs

V. Mon quatrième est multi-classe !

Nous avons dû pour cette partie prendre en considération une nouvelle Loss : la cross entropie. Nous avons donc implémenté une classe CElogSM-loss comprenant une activation Softmax passée au logarithme et un coût cross entropique. Nous avons aussi implémenté une fonction d'activation Softmax que nous utilisons pour la forward de la prédiction.

Nous avons utilisé le dataset de Keras MNIST ainsi que le dataset USPS qui correspondent tous les deux à des images en nuances de gris de chiffres de 0 à 9, cependant la taille des images de MNIST est plus grande, 784 pixels contre 256 pixels pour USPS.

Pour le dataset MNIST, nous avons utilisé un réseau comprenant une couche Linéaire(128) → couche d'activation TanH → couche Linéaire(64) → couche d'activation TanH → couche Linéaire(10) → couche d'activation Softmax.
L'accuracy pour MNIST était de 0.957.

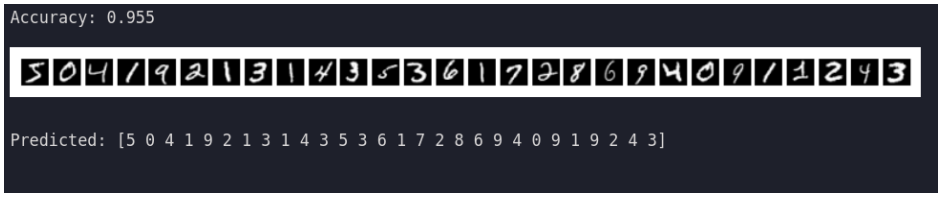


FIGURE 5 – Classification multiclasse à partir d'image de chiffres

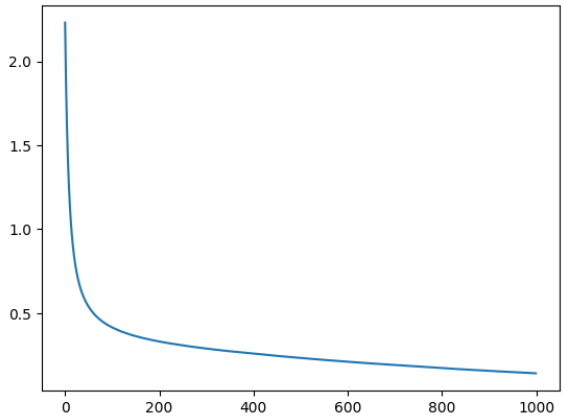


FIGURE 6 – Cross entropy log Loss en fonction du nombre d'epochs

Pour le dataset USPS , nous avons utilisé un réseau comprenant une couche Linéaire(64) → couche d'activation TanH → couche Linéaire(32) → couche d'activation TanH → couche Linéaire(16) → couche d'activation TanH → couche Linéaire(10) → couche d'activation Softmax.

Les résultats en terme d'accuracy à modèle comparable étaient moins bons pour USPS c'est pourquoi le réseau a été modifié. De plus, pour USPS le nombre d'epochs a dû être augmenté et le gradient step a dû être diminué en raison d'une divergence sur la Loss.
L'accuracy était ici de 0.845.

FIGURE 7 – Classification multiclasse à partir d'image de chiffres

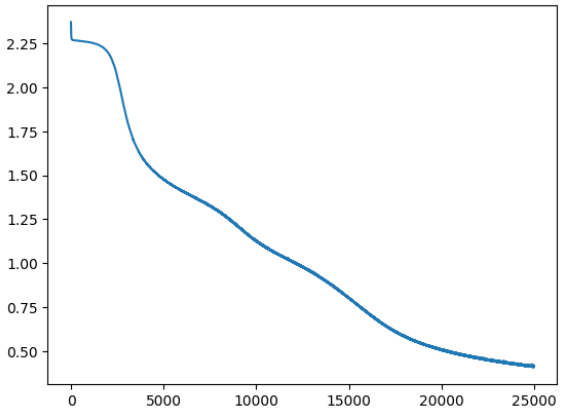


FIGURE 8 – Cross entropy log Loss en fonction du nombre d'epochs

VI. Mon cinquième se compresse !

Dans cette section, nous avons décidé de nous pencher sur deux problématiques. La première est celle de la visualisation des images reconstruites après une forte compression. La deuxième est de visualiser les représentations obtenues dans un espace 2D par le biais de la t-SNE.

Nous avons commencé par la reconstruction d'images compressées, l'objectif étant de réaliser sur l'image la compression la plus importante possible et d'observer la reconstruction la plus proche de l'image originale .
Nous avons testé différents réseaux avec une compression de plus en plus importante et en comparant deux Loss : la MSE-Loss et la BCE-Loss.
Afin de juger de l'efficacité de la reconstruction et de comparer les différents modèles, une MSE a été réalisée entre l'image initiale et l'image reconstruite.

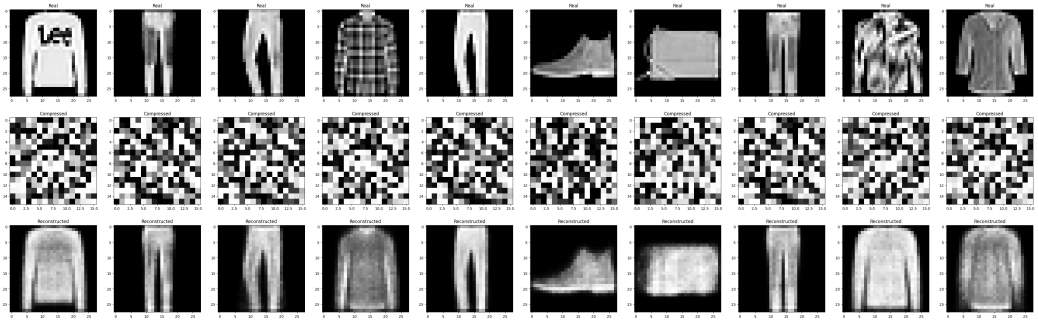


FIGURE 9 – Train : images, images compressées et images reconstruites

.

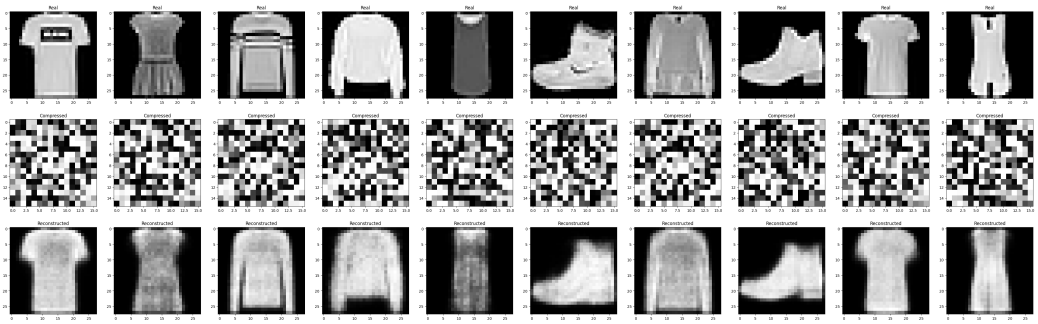


FIGURE 10 – Test : images, images compressées et images reconstruites

Premièrement, on peut voir sur la figure 11 comme attendu qu'à taille de compression égale, la BCE-Loss permet une meilleure reconstruction que la MSE-Loss. En effet, la MSE entre l'image et l'image reconstruite est significativement plus petite pour la BCE-Loss que pour la MSE-Loss : test de Wilcoxon p-value 0.03125

Deuxièmement, ici aussi de manière attendue, la MSE semble inversement proportionnelle à la taille de la compression en pixel, en effet plus la compression est importante moins bien l'image est reconstruite.

Couche 1	Taille compression en pixel	Mean MSE	Std MSE
MSE Loss	400	4,1	1,18
	256	4,55	1,31
	100	4,92	1,61
	64	4,67	1,33
	16	5,37	1,41
BCE Loss	400	3,81	1,15
	256	3,87	1,13
	100	4,52	1,31
	64	4,53	1,25
	16	5,26	1,41

FIGURE 11 – MSE en fonction du niveau de compression et de la Loss

Nous nous sommes ensuite intéressés à observer les représentations obtenues dans un espace 2D par le biais de la t-SNE.

Nous avons d'abord observé la représentation avec t-SNE sur les images de Train , les images compressées et les images reconstruites.
On peut voir sur la figure 12 des images originales que les points de même couleur, correspondant à une classe de vêtements, se regroupent dans le même espace formant des clusters.
De plus, les habits qui se ressemblent se retrouvent dans le même espace du graphique. Ainsi, les bottes/baskets/sandales se retrouvent dans le même espace (à gauche sur la figure 12), les pulls/tops/chemises (à droite sur la figure 12), les pantalons sont en bas du graphe et les sacs qui ont une forme assez différente sont en haut du graphe.

Sur la figure 13 des images de Train compressées , nous observons que le clustering selon la classe d'habits est plutôt bien conservé. De même, les classes d'habits se ressemblant sont proches les unes des autres, par exemple, les bottes/baskets/sandales se retrouvent cette fois à droite du graphique.

Sur la figure 14 des images reconstruites, on peut observer que le clustering selon la classe d'habit est plutôt bien conservé. De même, les classes d'habits se ressemblant sont proches les unes des autres.
Aussi, la représentation t-SNE des images reconstruites est très proche de celle des images initiales.

Concernant les images Test, la représentation avec la t-SNE conserve comme précédemment les caractéristiques de clustering et de regroupement des classes qui se ressemblent.
Cependant, nous observons une différence : la t-SNE des images Test reconstruites ne ressemble pas à la t-SNE des images Test originales mais plutôt à celle des images compressées contrairement à ce que l'on avait pu voir avec les images Train. Cela est probablement du à une mauvaise généralisation de notre modèle qui reconstruit moins bien les images Tests que les images Train.

FIGURE 12 – t-SNE sur les images Train

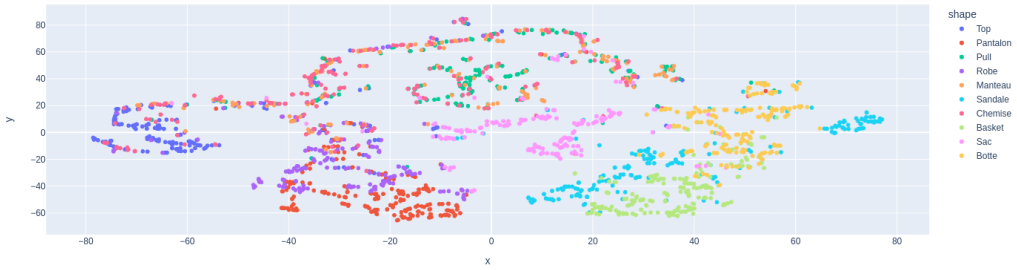


FIGURE 13 – t-SNE sur les images Train compressées

FIGURE 14 – t-SNE sur les images Train reconstruites

FIGURE 15 – t-SNE sur les images Test

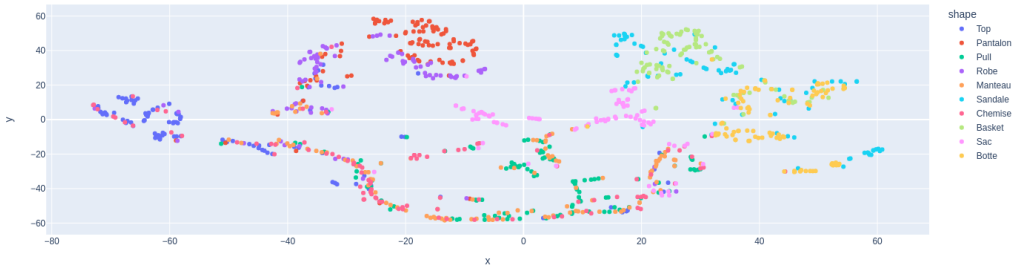


FIGURE 16 – t-SNE sur les images Test compressées

FIGURE 17 – t-SNE sur les images Test reconstruites

VII. Mon sixème se convole !

Nous avons implémenté les différentes classes Conv1D, MaxPool1D, Flatten et ReLU.

Ne sachant pas si les librairies permettant de réaliser de manière performante les convolutions étaient autorisées dans ce projet, nous avons utilisé uniquement les fonctions standards de numpy ainsi que des boucles, ce qui augmente grandement le temps d'apprentissage. Afin de limiter les boucles, la méthode backward_update_gradient a été laissée vide dans la classe et a été implémentée dans la méthode backward_delta.

Ici, nous avons utilisé la base USPS des chiffres pour laquelle nous avions de moins bonnes performances avec le réseau de neurones multiclasse afin d'observer une potentielle amélioration de la prédiction.

Nous avons utilisé le réseau recommandé dans l'énoncé à savoir : Conv1D(3,1,32) → MaxPool1D(2,2) → Flatten() → Linear(4064,100) → ReLU() → Linear(100,10) → Softmax().

Nous avons réduit le dataset de Train à 500 images en raison d'une durée d'entraînement trop longue, malgré cela, l'accuracy en Test a été améliorée comparativement au réseau sans convolution.
L'accuracy était de 0.900.
Par ailleurs, la courbe de la Loss montre que le pas de mise à jour du gradient est légèrement trop grand cependant, cela a été réalisé volontairement afin de limiter le temps de convergence pour pouvoir sortir des résultats, bien entendu dans l'absolu nous aurions choisi un pas de gradient plus petit.

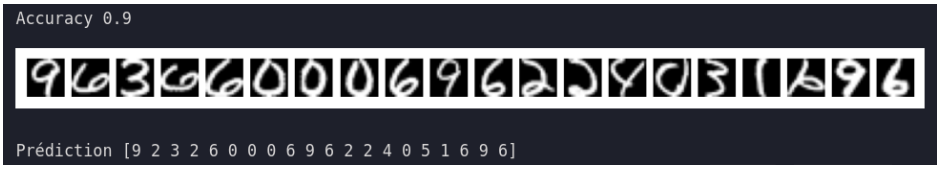


FIGURE 18 – Réseau convolutionnel : classification multiclasse à partir d'image de chiffres

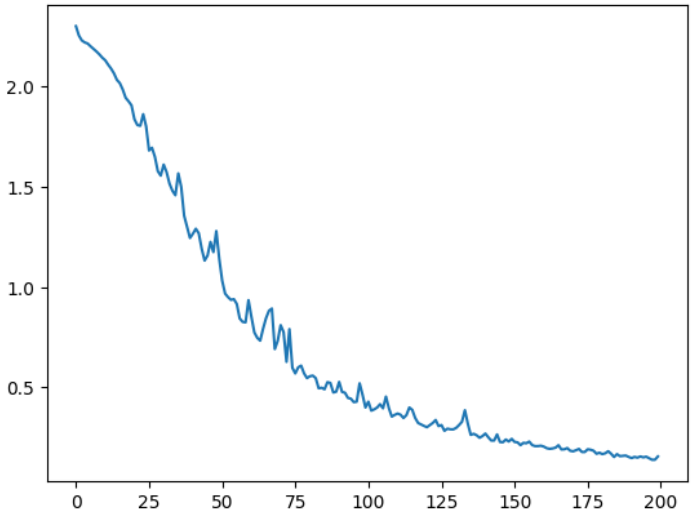


FIGURE 19 – Cross entropy log Loss en fonction du nombre d'epochs