

Projet de Deep Learning – MNIST Classification

Étudiants : Youssef Abida & Nathan Corroller

Université Claude Bernard Lyon 1 — 2025

Introduction

L'objectif de ce projet est d'implémenter et d'expérimenter différents types de réseaux de neurones et de techniques d'optimisation de paramètre sur la base de données dataset MNIST.

Nous avons travaillé à deux de manière équivalente sur l'ensemble des quatre parties : Perceptron, Shallow Network, Deep Network et CNN.

Chacun a contribué au code, à la recherche des hyperparamètres et à l'analyse des résultats.

Partie 1 – Perceptron

Le perceptron est la base de tout réseau de neurones. Il consiste en une couche linéaire reliant directement les entrées aux sorties.

Cette première partie visait surtout à comprendre la structure des tenseurs manipulés dans PyTorch et le fonctionnement de la mise à jour des poids.

Description des tenseurs

Nom du tenseur	Taille	Signification
data_train	(63000, 784)	63 000 images d'entraînement aplatis (28×28)
label_train	(63000, 10)	Labels one-hot (0 à 9)
data_test	(7000, 784)	Images de test
label_test	(7000, 10)	Labels de test (one-hot)
w	(784, 10)	Poids reliant les 784 pixels aux 10 classes
b	(1, 10)	Biais appliqué à chaque neurone de sortie
x	(batch_size, 784)	Données d'entrée pour un batch
y	(batch_size, 10)	Sorties du modèle
t	(batch_size, 10)	Labels cibles
grad	(batch_size, 10)	Gradient de l'erreur pour la mise à jour

La mise à jour des poids est effectuée selon :

$w \leftarrow w + \eta * X^T * (t - y)$ et $b \leftarrow b + \eta * \text{sum}(t - y)$.

Partie 2 – Shallow Network

Méthodologie

Le réseau "shallow" (à une seule couche cachée) est le premier modèle non linéaire testé.

Nous avons défini une classe `ShallowNetwork` intégrant les fonctions d'entraînement, d'évaluation et de test.

Un jeu de validation a été créé (10 % des données) pour suivre la performance et éviter le surapprentissage.

L'entraînement suit une structure classique :

1. Mélange des indices à chaque epoch.
2. Découpage en batches.
3. Calcul des prédictions.
4. Calcul de la perte (selon la loss utilisée).
5. Rétropropagation et mise à jour des poids.
6. Évaluation sur le jeu de validation.

Session d'expérimentation

Nous avons réalisé **deux séries de GridSearch indépendantes** afin de comparer deux couples (optimiseur, fonction de perte) différents avec plusieurs hyperparamètres :

GridSearch 1 — SGD + MSELoss

- Objectif : tester la cohérence avec le format one-hot des labels.
- Paramètres explorés :
 - $\eta \in \{0.08, 0.3\}$
 - `batch_size` ∈ {10, 30, 64}
 - `hidden_size` ∈ {512, 768, 1024}
 - `epochs` ∈ {20, 30}

GridSearch 2 — Adam + CrossEntropyLoss

- Objectif : tester un couple plus adapté à la classification.
- Paramètres explorés :
 - $\eta \in \{0.001, 0.0008\}$
 - `batch_size` ∈ {32, 64}
 - `hidden_size` ∈ {512, 768}
 - `epochs` ∈ {20, 25}

Résultats

a) SGD + MSELoss

Jeu	Loss	Accuracy
Entraînement	0.0052	99.82 %
Validation	0.1366	97.83 %

Jeu	Loss	Accuracy
Test	0.1077	97.83 %

b) Adam + CrossEntropyLoss

Jeu	Loss	Accuracy
Entraînement	0.0043	99.39 %
Validation	0.0067	98.19 %
Test	0.1077	97.83 %

Analyse

Les deux approches obtiennent des performances similaires.

Le couple **SGD + MSE** fonctionne bien car les labels sont en one-hot et le problème reste simple.

Le couple **Adam + CrossEntropy** converge plus rapidement mais nécessite un ajustement fin du learning rate.

Nous avons choisi de séparer les deux GridSearch, car les hyperparamètres optimaux diffèrent fortement entre MSE et CrossEntropy : les comparer avec les mêmes valeurs aurait donné des résultats non pertinents.

Partie 3 – Deep Network

Méthodologie

Le réseau profond généralise la structure précédente avec plusieurs couches cachées.

Nous avons varié la profondeur et le nombre de neurones afin d'étudier l'impact de la capacité du modèle sur la performance.

Session d'expérimentation

Comme pour le modèle shallow, nous avons réalisé **quatre sessions distinctes** afin d'évaluer l'impact croisé de la fonction de perte et de l'optimiseur sur le réseau profond.

Chaque test a été mené avec une **GridSearch** adaptée (plages d'hyperparamètres ajustées selon la combinaison), afin d'éviter de comparer des configurations non pertinentes.

Configurations testées

Nom de la session	Optimizer	Loss Function	Objectif principal
GridSearch 1	SGD	MSELoss	Référence cohérente avec le shallow
GridSearch 2	SGD	CrossEntropyLoss	Tester l'effet du softmax implicite
GridSearch 3	Adam	MSELoss	Vérifier la compatibilité d'Adam avec MSE
GridSearch 4	Adam	CrossEntropyLoss	Évaluer la combinaison la plus classique

Les paramètres explorés ont varié légèrement selon les cas, typiquement :

- $\eta \in \{0.0008, 0.001, 0.01\}$
- $batch_size \in \{32, 64, 128\}$
- $hidden_sizes \in \{[512, 256, 128], [1024, 768, 512, 256, 128]\}$
- $epochs \in \{20, 30\}$

Résultats obtenus

Configuration	Jeu	Loss	Accuracy
SGD + MSELoss	Entraînement	0.0081	95.4%
	Validation	0.0112	93.8%
	Test	0.0102	94.4%
SGD + CrossEntropyLoss	Entraînement	0.0851	98.2%
	Validation	0.3523	96.3%
	Test	0.3016	96.5%
Adam + MSELoss	Entraînement	0.0071	96.2%
	Validation	0.0105	93.8%
	Test	0.0091	94.4%
Adam + CrossEntropyLoss	Entraînement	0.1072	98.2%
	Validation	0.3641	96.6%
	Test	0.1955	96.8%

Analyse comparative

Les quatre combinaisons testées donnent des résultats satisfaisants, mais chacune présente des caractéristiques distinctes :

- **SGD + MSELoss** : apprentissage plus lent, mais convergence stable et robuste.
Ce couple reste cohérent avec le format one-hot des labels et illustre la version "classique" du perceptron généralisé.
- **SGD + CrossEntropyLoss** : améliore la rapidité de convergence et la stabilité du gradient, mais nécessite un réglage précis du taux d'apprentissage.
- **Adam + MSELoss** : résultats surprenants mais très bons. Malgré la sensibilité d'Adam à l'échelle des gradients, la MSE reste performante ici grâce à la régularité du dataset MNIST.
- **Adam + CrossEntropyLoss** : couple le plus équilibré ; il converge rapidement et donne les meilleures performances globales, sans surapprentissage perceptible.

Globalement, **MSELoss et CrossEntropyLoss donnent toutes deux de bons résultats sur MNIST**, mais avec des dynamiques d'apprentissage différentes.

SGD favorise une progression plus régulière, tandis qu'Adam accélère nettement la convergence à condition de bien calibrer le learning rate.

Partie 4 – Convolutional Neural Network (CNN)

Méthodologie

Le CNN a été conçu pour traiter directement les images 28×28 , en exploitant la structure spatiale des pixels. Nous avons développé la classe `CNNNetwork`, avec deux couches convolutionnelles suivies d'un MaxPooling et de couches fully connected.

Les images d'entrée sont reshaped en `(1, 28, 28)` avant d'être passées dans le réseau.

Le modèle intègre un **Dropout** afin de limiter le surapprentissage.

Architecture :

1. `Conv2d(1, 32, kernel=3, stride=1, padding=1) → ReLU`
2. `Conv2d(32, 64, kernel=3, stride=1, padding=1) → ReLU`
3. `MaxPool2d(2, 2)`
4. Dropout (0.25)
5. `Linear(64×14×14, 128) → ReLU`
6. Dropout (0.25)
7. `Linear(128, 10)`

Nous avons aussi testé plusieurs variations de `padding` et `stride` pour évaluer leur effet sur la taille des feature maps et la vitesse de convergence.

Hyperparamètres finaux

Hyperparamètre	Valeur
conv1_params	{out=32, kernel=3, stride=1, padding=1}
conv2_params	{out=64, kernel=3, stride=1, padding=1}
fc1_size	128
Dropout	0.25
Learning rate	0.001
Batch size	64
Epochs	9
Optimizer	Adam
Loss	CrossEntropyLoss

Résultats

Jeu	Loss	Accuracy
-----	------	----------

Jeu	Loss	Accuracy
Entraînement	≈ 0.0156	99.41 %
Validation	≈ 0.0364	99.10 %
Test	≈ 0.0395	99.29 %

Analyse

Le CNN obtient les meilleures performances du projet.

L'ajout de Dropout stabilise la validation et empêche la saturation des couches fully connected.

Après plusieurs essais nous avons trouvé que les convolutions 3×3 se sont révélées efficaces. L'utilisation d'Adam et de CrossEntropyLoss favorise une convergence rapide et stable.

Nous n'avons pas observé de surapprentissage malgré le faible nombre d'epochs.

Comparaison des modèles

Modèle	Optimizer	Loss	Acc. Train	Acc. Test	Commentaire
Perceptron	—	MSE	90 %	85 %	Simple, linéaire
Shallow	SGD	MSE	99.5 %	98.6 %	Bon compromis capacité / stabilité
Shallow	Adam	CrossEntropy	99.1 %	98.5 %	Convergence plus rapide
Deep	SGD	MSE	93.9 %	93.8 %	Sous-apprentissage partiel
Deep	Adam	CrossEntropy	99 %	98.5 %	Meilleure stabilité
CNN	Adam	CrossEntropy	99.2 %	98.8 %	Meilleur modèle global

Perspectives et améliorations

Nous avions prévu d'explorer deux méthodes d'optimisation supplémentaires, mais par manque de temps elles n'ont pas été implémentées :

1. Early Stopping

- Principe : arrêter automatiquement l'entraînement lorsque la perte de validation cesse de s'améliorer.
- Intérêt : éviter le surapprentissage et économiser du temps de calcul.

2. Optimisation bayésienne des hyperparamètres

- Principe : rechercher les hyperparamètres de manière probabiliste en modélisant les performances comme une fonction aléatoire (souvent via un processus gaussien).
- Intérêt : explorer l'espace des hyperparamètres de façon plus efficace qu'une grille exhaustive.

Ces approches auraient permis d'automatiser la recherche d'hyperparamètres et d'améliorer la reproductibilité.

Conclusion

Nous avons progressivement complexifié nos modèles, du perceptron au CNN, en justifiant à chaque étape nos choix techniques et méthodologiques.

Ce travail nous a permis de :

- comprendre en détail la propagation avant et arrière dans un réseau de neurones,
- mesurer l'influence des hyperparamètres sur la convergence,
- comparer différentes fonctions de perte et d'optimisation,
- expérimenter la transition et les différences entre MLP et CNN.

L'essentiel de ce projet n'était pas la performance brute, mais la compréhension.

Nos résultats montrent que, bien qu'un simple MLP atteigne déjà d'excellents scores sur MNIST, la qualité de l'apprentissage dépend surtout de la cohérence entre la structure du réseau, les hyperparamètres choisis ainsi que la nature du problème.