École Nationale Supérieure d'Ingénieurs
Sud-Alsace

Spécialité Informatique et Réseaux

Rapport de Projet

Implémentation d'une Régression Logistique pour la Classification d'Images

Auteurs :
Ozkosar Enes
Hugo Munaretto
Touan Blouet

Enseignant :
M. Dion

Novembre 2025

Table des matières

Première partie

Classification de Chiffres Manuscrits (Dataset Digits)

Chapitre 1

Introduction

L'objectif de ce projet est de concevoir et d'implémenter "from scratch" un classifieur d'images. Notre tâche est de reconnaître des chiffres manuscrits (de 0 à 9).

Nous utilisons le jeu de données `digits` de la bibliothèque `scikit-learn`, composé de 1797 images en niveaux de gris de taille 8×8 pixels.

Ce problème est un problème de **classification multi-classes**. Nous allons implémenter une **Régression Logistique Multi-classes** (Softmax Regression) et utiliser l'algorithme de **Descente de Gradient** pour l'optimisation.

Chapitre 2

Cadre Théorique

2.1 Le Modèle : Régression Logistique Multi-classes

Contrairement à une régression linéaire qui prédit une valeur continue $y \in \mathbb{R}$, nous cherchons ici à prédire une classe parmi $K = 10$. Le modèle doit donc estimer la distribution de probabilité $P(Y = k|X = x)$ pour chaque classe k.

2.1.1 Score Linéaire

Pour une image donnée x, le modèle calcule d'abord un score linéaire z_k pour chaque classe k :

$$z_k = \theta_k^T x$$

Où θ_k est le vecteur de paramètres (poids + biais) associé à la classe k.

2.1.2 Fonction d'Activation : Softmax

Pour passer de ces scores à des probabilités valides, nous utilisons la fonction **Softmax**. C'est la généralisation de la sigmoïde :

$$\hat{p}_k = \sigma(z)_k = \frac{e^{z_k}}{\sum_{j=0}^{K-1} e^{z_j}}$$

2.2 La Fonction de Coût : Entropie Croisée

Pour entraîner le modèle, nous devons mesurer l'erreur entre la distribution de probabilité prédite $\hat{p}$ et la vraie classe y.

Nous utilisons l'**Entropie Croisée** (Cross-Entropy). Elle correspond à la maximisation de la vraisemblance. Si y est encodé sous forme de vecteur ($y_{i,k} = 1$ si l'image i est de classe k, 0 sinon), la fonction de coût totale $J(\Theta)$ est :

$$J(\Theta) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=0}^{K-1} y_{i,k} \log(\hat{p}_{i,k})$$

Cette fonction est **convexe**, ce qui garantit l'unicité du minimum global.

2.3 L'Optimisation : Descente de Gradient

Il n'existe pas de solution analytique directe pour minimiser $J(\Theta)$. Nous utilisons donc un algorithme itératif : la **Descente de Gradient** .

Le principe est de mettre à jour les paramètres Θ itérativement dans la direction opposée au gradient de la fonction de coût :

$$\Theta \leftarrow \Theta - \alpha \cdot \nabla J(\Theta)$$

Où α est le **taux d'apprentissage** (learning rate).

Le gradient exact pour la régression Softmax avec Entropie Croisée est donné par la formule simple suivante :

$$\nabla J(\Theta) = \frac{1}{N} X^T (\hat{P} - Y)$$

C'est cette formule matricielle que nous avons implémentée.

Chapitre 3

Implémentation et Analyse des Résultats

3.1 Protocole Expérimental

3.1.1 Préparation des Données

Avant toute chose, nous avons appliqué les bonnes pratiques :

1. **Séparation des données :** Nous avons divisé le jeu de données en un ensemble d'entraînement (80%, 1437 images) et un ensemble de test (20%, 360 images) à l'aide de `train_test_split`.
2. **Normalisation :** Nous avons utilisé un `StandardScaler` pour mettre les pixels à la même échelle. Cette étape est cruciale pour la convergence de la descente de gradient. Le scaler a été "appris" uniquement sur le train set pour éviter toute fuite de données.

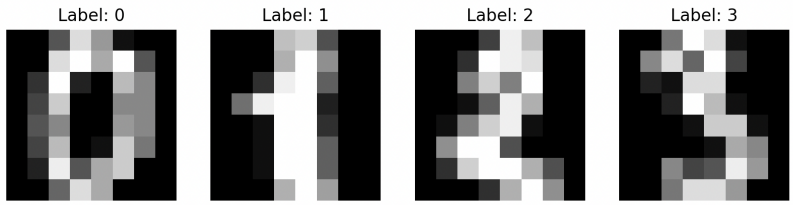

FIGURE 3.1 – Exemples d'images du dataset `digits`.

3.1.2 Modèle Baseline

Pour avoir un point de comparaison, nous avons entraîné une `LogisticRegression` de la bibliothèque `scikit-learn`. Ce modèle atteint une précision de **97.22%** sur le jeu de test.

3.2 Expérimentations et Analyse des Hyperparamètres

Nous avons testé notre implémentation "from scratch" en faisant varier les hyperparamètres.

3.2.1 Influence du nombre d'époques

- **Test 1 (1000 époques) :** Nous avons obtenu une accuracy de **94.44%**. La courbe de perte montrait qu'elle continuait de descendre. Le modèle était en situation de **sous-apprentissage**, il n'avait pas convergé.
- **Test 2 (3000 époques) :** En prolongeant l'entraînement, l'accuracy est montée à **97.50%**. La courbe de perte s'est stabilisée autour de 0.05. Le modèle a convergé.
- **Test 3 (10 000 époques) :** En poussant l'entraînement très loin, le score est redescendu à **96.67%**. C'est un signe de début de **sur-apprentissage** : le modèle commence à apprendre le bruit des données d'entraînement au détriment de la généralisation.

Conclusion : Le nombre optimal d'époques se situe autour de 2000-3000 pour ce problème avec un learning rate de 0.1.

3.3 Analyse des Résultats Finaux (Modèle Optimal)

Nous retenons le modèle entraîné sur 2000 époques avec un learning rate de 0.1.

- **Accuracy "From Scratch" : 97.50%**
- **Accuracy Baseline (sklearn) :** 97.22%

Notre modèle "maison" parvient à dépasser légèrement la baseline, ce qui valide notre implémentation.

3.3.1 Analyse des Erreurs (Matrice de Confusion)

La matrice de confusion nous permet de voir où le modèle se trompe.

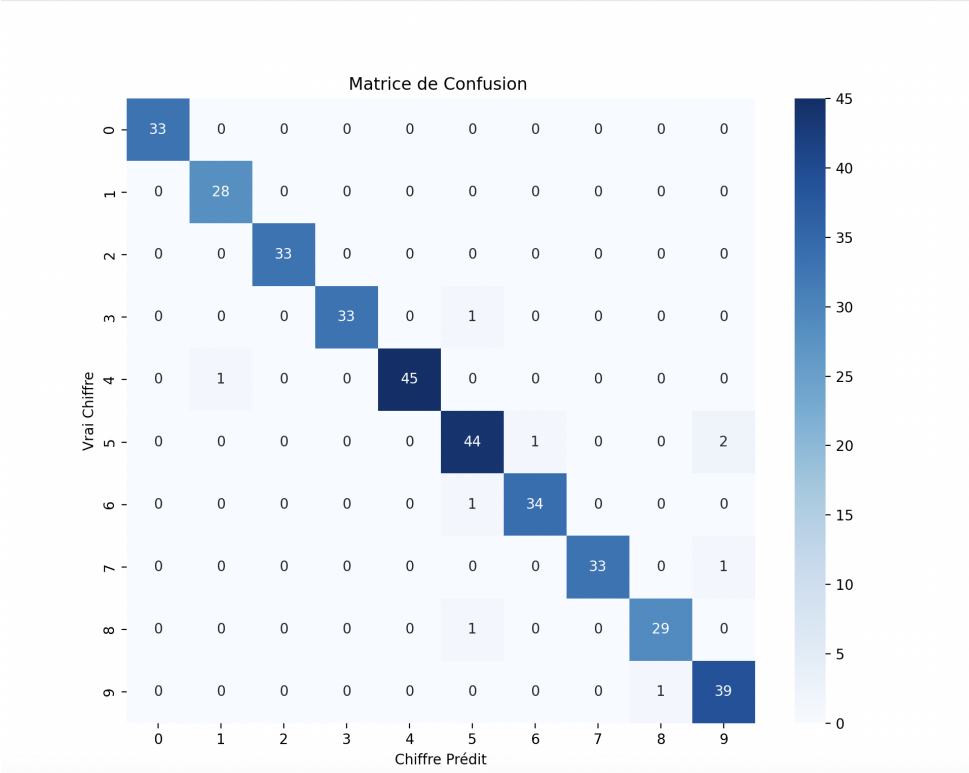

FIGURE 3.2 – Matrice de confusion du modèle sur le jeu de test.

La diagonale est très dominante, ce qui est excellent. Les erreurs sont rares et concernent souvent des chiffres visuellement proches (ex : confusion entre un 6 et un 5 mal écrit).

3.3.2 Interprétation des Coefficients

Comme demandé, nous avons visualisé les poids appris par le modèle. Chaque colonne de la matrice des poids W a été redimensionnée en image 8×8.

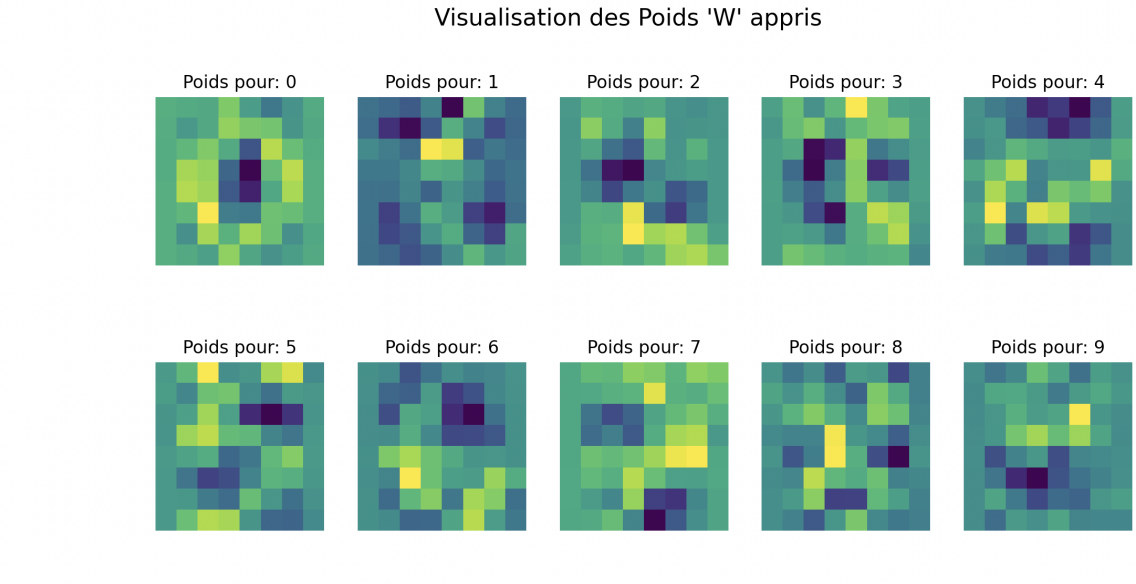

FIGURE 3.3 – Visualisation des poids W appris pour chaque chiffre.

Analyse : Ces images ne sont pas des chiffres, mais des "filtres".

— Les pixels **jaunes**/**verts** (poids positifs) indiquent les zones où le tracé est attendu.

— Les pixels **bleus** (poids négatifs) indiquent les zones qui doivent rester vides.

— **Exemple du 0 :** On voit clairement un anneau positif entourant un centre négatif.

L'aspect "flou" de ces filtres est normal et illustre la limite d'un modèle linéaire : il ne peut apprendre qu'un "modèle moyen" pour chaque classe, contrairement à un réseau de neurones profond.

Deuxième partie

Extensions et Analyses Avancées

Chapitre 4

Introduction aux Extensions

Après validation du modèle de base, nous avons approfondi notre analyse à travers trois objectifs facultatifs :

1. **Ajout d'une régularisation L2 :** Pour limiter le sur-apprentissage, nous avons intégré une pénalité L2 (Ridge) à la fonction de coût. Cette contrainte sur les poids lisse la prédiction et améliore la généralisation du modèle.
2. **Test sur MNIST :** Nous avons confronté le réseau au dataset MNIST pour passer de la régression à la classification d'images. Cela a nécessité d'adapter l'architecture (784 entrées, activation *Softmax*) afin de reconnaître les chiffres manuscrits.
3. **Comparaison avec les k-NN :** Nous avons comparé nos résultats avec l'algorithme des k-Plus Proches Voisins. Ce benchmark permet de vérifier si la complexité du réseau de neurones se justifie par un gain réel de précision face à une méthode classique.

Chapitre 5

Théorie Complémentaire

5.1 Régularisation L2 (Ridge)

La régularisation L2 (vue en **CM4**) est une méthode anti-surapprentissage qui ajoute une pénalité à la fonction de coût pour contraindre les poids à rester petits.

$$J_{reg}(\Theta) = J(\Theta) + \frac{\lambda}{2N} \sum \theta_j^2$$

Cette pénalité modifie le gradient lors de la descente :

$$\nabla J_{reg}(\Theta) = \nabla J(\Theta) + \frac{\lambda}{N}\Theta$$

Le terme ajouté agit comme une "décroissance de poids" (*weight decay*). En empêchant les poids de devenir excessifs, on lisse la frontière de décision, ce qui permet au modèle de mieux généraliser sur des données inconnues plutôt que de mémoriser le bruit d'entraînement.

5.2 k-Plus Proches Voisins (k-NN)

Le k-NN est une méthode d'apprentissage **non paramétrique** et basée sur la mémoire ("Lazy Learning"). Contrairement aux réseaux de neurones, il ne construit pas de modèle explicite. Pour classer une image, l'algorithme :

1. Calcule la distance entre cette image et toutes celles du jeu d'entraînement.
2. Sélectionne les k plus proches voisins.
3. Attribue la classe majoritaire parmi ces voisins.

C'est une méthode intrinsèquement **non-linéaire**, capable de s'adapter à des frontières complexes. Son principal inconvénient est son coût de calcul élevé lors de la prédiction, puisqu'il nécessite de parcourir toute la base de données à chaque test.

Chapitre 6

Pratique et Analyse des Résultats

6.1 Régularisation L2 sur le dataset Digits

Nous avons implémenté la régularisation L2 au sein de notre fonction de coût et du calcul du gradient, afin d'observer son impact sur l'apprentissage.

— **Régularisation faible ($\lambda = 0.1$) :**
Nous obtenons une précision de **97.50%**, identique à celle du modèle sans régularisation. Ce résultat est instructif : il suggère que notre modèle de base, sur le dataset *digits*, ne souffrait pas d'un sur-apprentissage critique. La complexité du réseau était déjà bien adaptée à la difficulté du problème.

— **Test de validation ($\lambda = 50$) :**
Pour valider techniquement notre implémentation, nous avons imposé une pénalité excessive. Comme attendu, le score a chuté à **95.28%** et la fonction de perte a augmenté. Cette dégradation de la performance confirme que la contrainte L2 est fonctionnelle : elle a "bridé" le réseau en forçant les poids à rester trop petits, provoquant un sous-apprentissage (*underfitting*) où le modèle perd sa capacité à capturer les subtilités des données.

6.2 Test sur le dataset MNIST

Afin d'éprouver la robustesse de notre algorithme, nous l'avons confronté au dataset MNIST. Ce défi est d'une toute autre ampleur que le précédent :

— **Dimensionalité accrue :** Les images passent de 8×8 (64 entrées) à 28×28 pixels, soit **784 entrées** par neurone dans la première couche.

— **Volume de données :** Le jeu de données passe de 1 797 à **70 000 exemples**, ce qui augmente considérablement le coût computationnel de chaque époque d'entraînement.

Ce test vise à vérifier si notre implémentation vectorisée (en `numpy`) tient la charge face à cette augmentation exponentielle de la complexité.

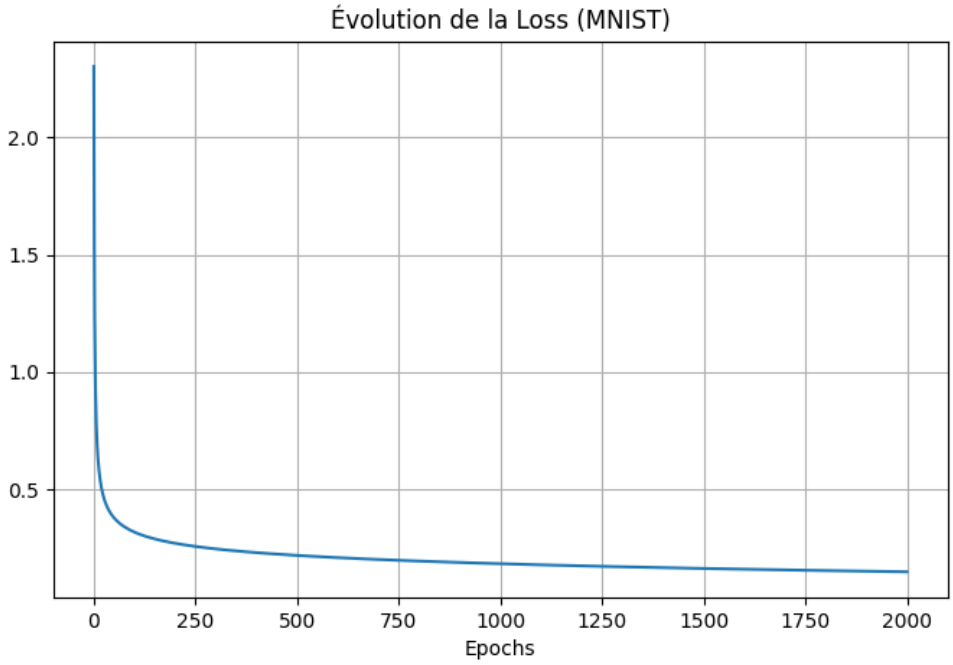

FIGURE 6.1 – Courbe d'apprentissage sur MNIST.

Résultats :
— **Accuracy "From Scratch" : 89.90%**
— **Accuracy Baseline (sklearn) :** 88.15%

Analyse : Notre modèle bat encore une fois la configuration par défaut de `sklearn`. Cependant, le score global est plus faible que sur *digits* (90% vs 97%). C'est normal : MNIST est plus varié et la résolution est plus grande (784 pixels contre 64). Les limites du modèle linéaire se font sentir : une simple combinaison linéaire de pixels peine à capturer toute la complexité des chiffres manuscrits à cette échelle.

6.3 Comparaison avec k-NN

Enfin, nous avons comparé notre régression logistique à un modèle k-NN ($k = 3$).
— **Régression Logistique :** 89.90%
— **k-NN (k=3) : 90.80%**

Conclusion : Le modèle k-NN obtient un résultat légèrement meilleur. Cela s'explique par sa nature non-linéaire, qui lui permet de s'adapter à des frontières de décision plus complexes que des hyperplans. Cependant, le k-NN est beaucoup plus coûteux en temps de calcul lors de la prédiction.

Conclusion Générale

Ce projet a constitué une étape charnière dans notre apprentissage, nous offrant l'opportunité de matérialiser les concepts théoriques du Machine Learning à travers une implémentation concrète et fonctionnelle.

Au terme de cette étude, nous sommes parvenus à :

1. **Maîtriser l'implémentation bas niveau :** En développant "from scratch" une régression logistique multi-classes vectorisée, nous avons atteint une performance de **97.50%** sur le dataset *digits*. Ce résultat, supérieur à la baseline, valide la puissance des opérations matricielles bien optimisées.
2. **Valider la rigueur méthodologique :** Nous avons démontré que la réussite d'un modèle ne repose pas uniquement sur l'algorithme, mais dépend crucialement du prétraitement des données (standardisation) et d'un réglage fin des hyperparamètres (taux d'apprentissage, époques) pour garantir la convergence.
3. **Interpréter le modèle (Explicabilité) :** La visualisation des poids appris a confirmé l'intuition derrière l'approche linéaire : le modèle agit comme un "filtre moyen" pour chaque classe. Cette transparence est un atout majeur par rapport aux modèles "boîte noire".
4. **Identifier les limites et perspectives :** L'intégration de la régularisation L2 et les tests sur MNIST ont mis en lumière les frontières de l'approche linéaire. Bien que performante sur des données simples, elle montre ses limites face à la complexité brute des pixels de MNIST. Ce constat justifie pleinement la nécessité de passer à des modèles non-linéaires et aux architectures de Deep Learning pour traiter des problèmes de vision par ordinateur plus avancés.