Projet CLANU 2017-2018: Apprentissage par ordinateur pour la reconnaissance automatique de chiffres manuscripts

O. Bernard, E. Bretin, T. Grenier, T. Redarce et C. Reichert

16 avril 2018

Voici l'énoncé du projet CLANU. Ce projet a pour objectif principal de vous faire concevoir un logiciel en langage `Matlab/C++` illustrant une méthode d'apprentissage par ordinateur spécifique : la regression. Le contexte mathématique est introduit dans cet énoncé et différentes implémentations sont à faire pour répondre aux questions. La qualité des résultats et de l'analyse mathématique d'une part, ainsi que la qualité de la conception informatique d'autre part, seront prises en considération lors des différentes évaluation qui entour ce projet.

1 Introduction

Le but de ce projet est d'implémenter et d'étudier numériquement une méthode d'apprentissage par ordinateur appelée regression. Cette méthode est généralement utilisée en bout de chaine de traitement de méthodes d'intelligence artificelle telles que les réseaux de neronnes. L'étude des réseaux de neuronnes nécessiterait un investissement de votre part trop coûteux dans le cadre du projet clanu mais la découverte des méthodes de regression vous permettera de faire un premier pas dans cet univers fascinant (et utilisé de façon extensive en entreprise) qu'est l'intelligence artificelle.

Dans le contexte du projet CLANU, nous allons vous faire appliquer une méthode dite de regression afin de reconnaitre des chiffres tracés à la main. Ce problème a été grandement résolu dans les années 1998 par le français Yann LeCun qui à l'époque avait proposé une méthode révolutionnaire appelée apprentissage profond (ou deep learning) afin d'apporter une solution fiable et extremement robuste à ce problème. Yann LeCun est actuellement directeur du département intelligence artificelle chez Facebook. Vous allez voir au travers de ce projet que la méthode de regression est une approche relativement simple qui permet d'obtenir des résultats extremement intéressants.

2 Méthode de regression

2.1 Apprentissage par l'exemple

La méthode de regression fait partie des techniques d'apprentissage par ordinateur (en anglais *machine learning*). La plupart des méthodes d'apprentissage par ordinateur sont basées sur deux phases : une étape d'apprentissage et une étape de test. La première étape nécessite la mise en

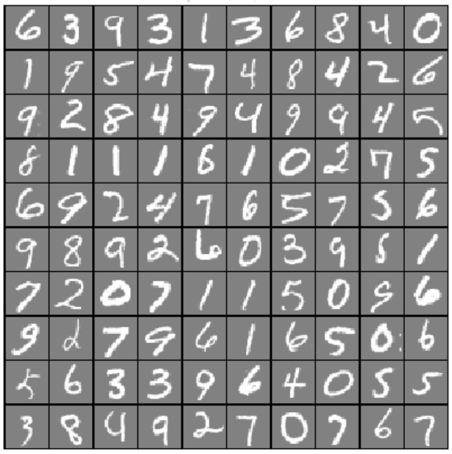
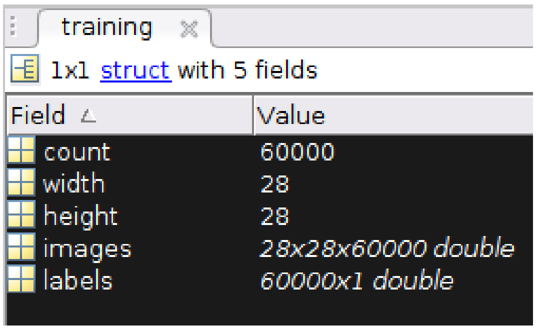
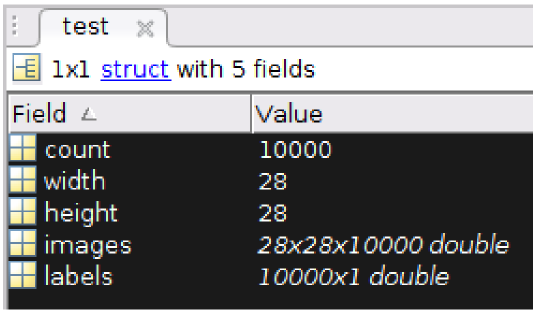


FIGURE 1 – Exemple de chiffres manuscripts extraits de la base de données mnist



(a) Variable `training`



(b) Variable `testing`

FIGURE 2 – Organisation interne des deux variables `Matlab` de type structure `training` et `testing`

place d'une base de données à partir de laquel l'algorithme va apprendre à reproduire une tâche spécifique. Dans le cadre de ce projet, la tâche à reproduire est la reconnaissance automatique de chiffres manuscripts (valeurs comprises entre 0 et 9). La seconde étape, le test, permet de quantifier la qualité de la reconnaissance automatique. Pour ce faire, la base de données utilisée est une base de données publique (`http://yann.lecun.com/exdb/mnist/`) nommée mnist (Mixed National Institute of Standards and Technology) largement utilisée au sein de la communauté scientifique internationale afin de continuer à améliorer (si besoin est) la performance des méthodes de reconnaissance de chiffres manuscripts Un exemple de chiffres extraits de cette base de données est fourni dans la figure 1.

A partir du code fourni dans le projet, la base de données est téléchargée automatiquement et placée dans le repertoire *data* à la racine du projet. Une fois téléchargée, l'instruction `load` de `Matlab` permet de lire la base de données et les variables `training` et `testing` sont chargées en mémoire. Ces deux variables sont des structures qui permettent d'accéder à des sous-variables associées. La figure 2 permet d'avoir un aperçu de l'organisation interne de telles structures. A partir de cette figure, nous pouvons voir que la base d'entrainement est constituée de 60000 imagettes (cette information est accessible via l'instruction `training.count`). Chaque chiffre est stocké sous forme d'une imagette de taille 28×28 dont les valeurs sont comprises entre 0 et 1. L'ensemble des imagettes est stocké au sein de la variable `training.images`. A partir de cette variable, il est possible de récupérer la i-ème imagette via la commande `Matlab` suivante : `img = training.images(:,:,i)`. A chaque imagette de la variable `training.images` est associé le chiffre correspondant via la variable `training.label`. Par exemple, si la i-ème imagette de la variable `training.images` correspond au chiffre 3, l'instruction `training.labels(i)` retournera la valeur 3.

2.2 Fonction de décision

La plupart des méthodes d'apprentissage par ordinateur correspondent à des fonctions de décision $h_\Theta(x)$, où Θ est une matrice dont les coefficients sont les inconnues à déterminer et x correspond à un vecteur de données que l'on fournit en entrée de l'algorithme. Dans le cadre du projet CLANU, la méthode de régression étudiée renvoie la probabilité de détection pour chaque chiffre allant de 0 à 9. En particulier, chaque chiffre correspond à une classe c et la fonction de décision peut être modélisée de la façon suivante :

$$h_\Theta(x) = p(y = k|x; \Theta)\,,$$

où le second membre de cette équation correspond à la probabilité que la sortie y appartienne à la classe k connaissant le vecteur d'entrée x et la paramétrisation Θ.

2.3 Phase d'apprentissage

Lors de la phase d'apprentissage, l'ensemble des m imagettes connues (dans le cas de la base de données mnist $m = 60000$) de taille 28×28 est utilisé. Chaque imagette est représentée sous forme d'un vecteur de taille $[1 \times n]$ (avec $n = 785$) composé des valeurs des pixels de chaque ligne de l'imagette mise bout-à-bout plus l'ajout de la valeur 1 en début de vecteur (l'explication d'un tel ajout n'est pas nécessaire pour la compréhension générale du projet). Sous `Matlab`, cette étape est réalisée via les deux lignes d'instruction suivantes :

```
digit = training.images(:,:,i);
x = [1,digit(:)'];
```

Ainsi, au début du code fourni dans le projet, l'ensemble des données d'apprentissage est stocké au sein d'une matrice X de taille $[m \times n]$. En parallèle de la matrice X, un vecteur y de taille $[m \times 1]$ est créé. Chaque élément de y possède la valeur de la classe de l'imagette correspondante. La classe 0 correspond au chiffre 0, la classe 1 au chiffre 1, ainsi de suite jusqu'à la classe 9 qui correspond au chiffre 9. Ainsi, si la i-ème imagette de la base d'entrainement correspond au chiffre 4, alors la i-ème ligne de la matrice X correspond aux valeurs des pixels de l'imagette et le i-ème élément du vecteur y, nommé $y^{(i)}$ dans la suite de l'énoncé, sera égale à 4.

La matrice Θ est de taille $[10 \times n]$. Il y a donc $10 \times n = 7850$ inconnues à déterminer. Ceci est réalisé lors de la phase d'apprentissage à partir d'une mise en équation qui est développée dans la section 3. ***L'obtention automatique de ces paramètres constitue le coeur de ce projet CLANU.***

2.4 Phase de test

Une fois la matrice Θ connue, la phase de test s'effectue de la façon suivante. Pour une nouvelle imagette, le vecteur x associé de taille $[1 \times n]$ est créé (mise bout-à-bout des lignes de l'imagette avec le rajout de la valeur 1 en début de vecteur). Dans un second temps, la fonction de décision suivante est calculée

$$h_\Theta(x) \;=\; g\left(x\,\Theta^T\right)$$

où $g(z) = \frac{1}{1+e^{-z}}$ correspond à la fonction sigmoïde. Dans ce contexte, la fonction $h_\Theta(x)$ renvoie un vecteur de taille $[1 \times 10]$ ou chaque composante k correspond à la probabilité d'appartenance à la classe k. Classiquement, la probabilité la plus importante permet de déterminer la valeur du chiffre de l'imagette. Si on connait la vérité (la vraie valeur manuscipte) on peut alors mesurer la fiabilité de l'approche.

3 Modélisation mathématique

3.1 Formulation énergétique

Chacune des lignes (il y en a 10) de la matrice Θ, notée Θ_c et de taille $[1 \times n]$, sera déterminée indépendamment les unes des autres en minimisant une énergie J de la forme :

$$J(\Theta_c) = -\frac{1}{m} \sum_{i=1}^{m} \left[y_c^{(i)} \log(h_{\Theta_c}(x^{(i)})) + (1 - y_c^{(i)}) \log(1 - h_{\Theta_c}(x^{(i)})) \right], \tag{1}$$

où

$$h_{\Theta_c}(x^{(i)}) = g\left(x^{(i)}\, \Theta_c^T\right).$$

Le vecteur $x^{(i)}$ est un vecteur de taille $[1 \times n]$ qui correspond à la i-ème ligne de la matrice X et le scalaire $y_c^{(i)}$ est défini par :

$$y_c^{(i)} = \begin{cases} 1 & \text{si } y^{(i)} = c \\ 0 & \text{sinon,} \end{cases}$$

où $y^{(i)}$ correspond à la i-ème case du vecteur y. En d'autres termes, les scalaire $y_c^{(i)}$ s'identifie à 1 si la i^{eme} imagette est associée au chiffre c et 0 si ce n'est pas le cas.

Nous pouvons de plus remarquer que la fonction d'énergie $J(\Theta_c)$ renvoie bien une valeur scalaire, Θ_c étant un vecteur de taille $[1 \times n]$. De plus, comme la sortie de la fonction g appartient à l'intervalle $[0, 1]$, l'énergie J est positive et atteint sa valeur minimale égale à 0 s'il existe un vecteur Θ_c tel que $x^{(i)}\Theta_c^T = +\infty$ si $y^{(i)} = c$ et $x^{(i)}\Theta_c^T = -\infty$ si $y^{(i)} \neq c$. En partique, ce cas idéal n'arrive jamais, cependant l'idée est bien d'optimiser un vecteur Θ_c de telle sorte que :

— la valeur de $x^{(i)}\Theta_c^T$ soit maximale si $y^{(i)} = c$

— la valeur de $x^{(i)}\Theta_c^T$ soit minimale si $y^{(i)} \neq c$.

Ainsi, la fonction de décision sous-jacente $h_{\Theta_c}(.)$ permettra de reconnaître au mieux l'ensemble des imagettes $x^{(i)}$ associées au chiffre c.

3.2 Minimisation de la fonction d'énergie J et algorithme de descente

La fonction d'énergie $J(\Theta_c)$ est relativement complexe et sa minimisation s'effectue généralement en utilisant un algorithme itératif de descente de gradient dont la forme la plus simple est la méthode du gradient à pas fixe. Plus précisément, cet algorithme consiste à choisir un vecteur initial Θ_c^0 de taille $[1 \times n]$ puis à considérer des vecteurs Θ_c^k définis de façon récursive par :

$$\Theta_c^{k+1} = \Theta_c^k + \tau d_k, \qquad \text{avec } d_k = -\nabla J(\Theta_c^k), \tag{2}$$

où τ représente le pas de descente, d_k la direction de descente et $\nabla J(\Theta_c^k)$ correspond au gradient de la fonction d'énergie J évalué en Θ_c^k.

Une variante de cet algorithme consiste à choisir une nouvelle direction de descente d_k définie de la manière suivante :

$$d_k = \begin{cases} -\nabla J(\Theta_c^k) & \text{si } k = 1 \\ -\nabla J(\Theta_c^k) + \beta_k d_{k-1} & \text{si } k \geq 2 \end{cases} \tag{3}$$

avec

$$\beta_k = \frac{\nabla J(\Theta_c^k) \cdot (\nabla J(\Theta_c^k) - \nabla J(\Theta_c^{k-1}))^T}{\|\nabla J(\Theta_c^{k-1})\|^2}.$$

Cette approche s'avère plus efficace en pratique et est connue sous le nom de méthode du gradient conjugué non linéaire de type Polack-Ribière.

Pour chacun de ces algorithmes, le choix du paramètre τ est déterminant pour le bon fonctionnement de la méthode d'optimisation : une valeur trop grande peut faire diverger l'algortihme alors qu'une valeur trop faible implique un nombre d'itérations extrêmement élevé pour atteindre la convergence de l'algorithme.

Enfin, il est à noter que $\nabla J(\Theta_c)$ est un vecteur de taille $[1 \times n]$ dont la j^{eme} composante peut se mettre sous la forme :

$$(\nabla J(\Theta_c))_j = \frac{\partial}{\partial(\Theta_c)_j} J(\Theta_c),$$

où $(\Theta_c)_j$ correspond à la j^{eme} composante du vecteur Θ_c. L'un des objectifs des questions mathématiques sera de montrer la relation suivante :

$$\frac{\partial}{\partial(\Theta_c)_j} J(\Theta_c) = \frac{1}{m} \sum_{i=1}^{m} \left[x_j^{(i)} . \left(h_{\Theta_c}(x^{(i)}) - y_c^{(i)} \right) \right] . \tag{4}$$

Nous obtiendrons ainsi une expression explicite de gradient de J.

4 Liste des questions liées au module MA2

4.1 Modélisation mathématique

1. Lors de la phase de test, l'imagette fournit en entrée de la fonction de décision $h_\Theta(x)$ correspond au chiffre manuscript 3. Dans l'hypothèse où cette imagette est correctement détectée, quelle composante du vecteur $h_\Theta(x)$ possèdera la plus grande valeur ? Est ce que cette valeur sera forcément égale à 1 ?
2. Montrez que la dérivée de la fonction sigmoïde $g(z) = \frac{1}{1+e^{-z}}$ peut s'écrire sous la forme suivante :

$$g'(z) = g(z)\,(1 - g(z))$$

3. En déduire la relation suivante :

$$\frac{\partial}{\partial(\Theta_c)_j}\, h_{\Theta_c}(x^{(i)}) = x_j^{(i)}\, h_{\Theta_c}(x^{(i)})\, \left(1 - h_{\Theta_c}(x^{(i)})\right)$$

   où $x_j^{(i)}$ correspond à la j-ème composante du vecteur $x^{(i)}$. ***On rappelle la relation suivante pour deux fonctions* $\mathbf{u}$ *et* $\mathbf{v}$ *:*** $(\mathbf{u} \circ \mathbf{v})' = (\mathbf{u}' \circ \mathbf{v}).\mathbf{v}'$

4. En déduire l'expression suivante de la dérivée de la fonction J :

$$\frac{\partial}{\partial(\Theta_c)_j}\, J(\Theta_c) = \frac{1}{m} \sum_{i=1}^{m} \left[x_j^{(i)}.\left(h_{\Theta_c}(x^{(i)}) - y_c^{(i)}\right)\right]$$

4.2 Implémentation `Matlab`

Vous devez télécharger le projet `Matlab` présent sous moodle (GE-3/Mathématiques 2). Le projet est structuré de la façon suivante :

`data`	Dossier utilisé pour stocker la base de données mnist
[∗] `exercise`	Dossier contenant des scripts pour la prise en main du formalisme mathématique
[∗] `+lrc`	Dossier contenant les fichiers relatifs à la méthode de régression
`param`	Dossier utilisé pour stocker la matrice Θ sous forme d'un fichier '.mat'
[∗] `scripts`	Dossier contenant les scripts principaux du projet
`+tools`	Dossier contenant des fonctions d'usage générale
`+visu`	Dossier contenant des fonctions utilisées pour des aspects de visualisation avancés

∗ indique que le dossier contient des fichiers à modifier.

Il est a noté que les fonctions présentent dans un dossier dont le nom commence par le symbole `+` sont utilisable au travers du nom du dossier associé. Par exemple, la fonction `sigmoid` présente dans le dossier `+lrc` peut être appelée au sein d'un script afin de calculer la valeur de la fonction sigmoïde pour une donnée z passée en argument d'entrée via l'instruction suivante :

```
val = lrc.sigmoid(z)
```

1. Compléter le code du script `script_ex1.m` présent dans le dossier `exercise` afin de calculer la valeur de l'energie $J(\Theta_c)$ (equation 1) pour $c = 1$ à partir de la base de données mnist et d'un vecteur de paramètre Θ_c pré-calculé. Si vous avez bien programmé le calcul de l'énergie, vous devez obtenir la valeur 2.3824 pour le cas étudié.
2. Compléter le code du script `script_ex2.m` présent dans le dossier `exercise` afin de calculer le vecteur gradient $\nabla J(\Theta_c)$ suivant pour $c = 1$ à partir de la base de données mnist et d'un vecteur de paramètre pré-calculé :

$$\nabla J(\Theta_c) = \left[\frac{\partial}{\partial(\Theta_c)_1} J(\Theta_c), \frac{\partial}{\partial(\Theta_c)_2} J(\Theta_c), \cdots, \frac{\partial}{\partial(\Theta_c)_n} J(\Theta_c)\right] \quad \in \quad \mathbb{R}^{1\times n}$$

   avec $n = 785$ et où l'expression $\frac{\partial}{\partial(\Theta_c)_j} J(\Theta_c)$ correspond à l'equation (4). Si vous avez bien programmé le calcul du gradient, le produit scalaire $\nabla J.\nabla J^T$ doit vous retourner la valeur 0.5859 pour le cas étudié.
3. En vous inspirant de vos codes précedents, compléter la fonction `lrCostFunction.m`. Cette fonction reçoit en paramètre d'entrée un vecteur $\Theta_c \in \mathbb{R}^{[1\times n]}$, une matrice $X \in \mathbb{R}^{[m\times n]}$, un vecteur $y \in \mathbb{R}^{[m\times 1]}$ et renvoie la valeur de la fonction d'énergie $J(\Theta_c)$ ainsi que le vecteur de gradient $\nabla J(\Theta_c) \in \mathbb{R}^{[1\times n]}$ associé.
4. Une fois la fonction `lrCostFunction.m` codée, vous devez exécuter et analyser les trois fichiers présents dans le dossier `scripts` dans l'ordre suivant :

`scriptTrainingLRC.m`	script qui permet d'exécuter la phase d'entrainement
`scriptTestingLRC.m`	script qui permet d'exécuter la phase de tests
`scriptTestingLRC_gui.m`	script qui permet de visualiser les résultats

5. Les fichiers `scriptTrainingLRC.m` et `scriptTestingLRC.m` affichent une mesure de précision (*accuracy* en anglais) calculée à partir de la base de données d'entrainement (pour le fichier `scriptTrainingLRC.m`) et de la base de données de test (pour le fichier `scriptTestingLRC.m`). En analysant le code, expliquer à quoi correspond cette métrique. Dans le cas idéal (reconnaissance exacte de l'ensemble des chiffres manuscripts), quelle valeur sera affichée pour chacun des deux fichiers ?
6. La technique de descente de gradient codée au sein de la fonction `gradient_descent.m` correspond à la méthode à pas fixe dont l'équation de mise à jour du vecteur Θ_c est donnée par l'équation (2). En vous inspriant de ce code, créer une nouvelle fonction nommée `conjugate_gradient.m` qui reçoit en entrée les mêmes paramètres, met à jour le vecteur Θ_c via la méthode du gradient conjugué (équation (3)) et renvoie en sortie les mêmes paramètres que ceux de la fonction `gradient_descent.m`
7. Les deux méthodes de descente de gradient (pas fixe et conjugué) sont principalement sensibles à deux paramètres de réglage qui sont le pas de descente τ et le nombre d'itération maximum utilisé lors de la phase d'apprentissage.
   — Coder un nouveau script nommé `scriptInfluenceMaxIteration.m` au sein du dossier `scripts` afin de tracer sur un même graphique l'évolution de la mesure de précision calculée à partir de la ***base de données de test*** via les deux méthodes de descente de gradient pour un nombre d'iterations variant de 20 à 200 avec 10 points de mesures (on prendra une valeur fixe du pas de descente τ égale à 1 pour l'ensemble des experiences). Commenter les résultats ainsi obtenus et conclure sur la valeur du nombre d'itérations à retenir.
   — Coder un nouveau script nommé `scriptInfluenceTau.m` au sein du dossier `scripts` afin de tracer sur un même graphique l'évolution de la mesure de précision calculée à partir de la ***base de données de test*** via les deux méthodes de descente de gradient pour une

valeur de pas de descente τ variant de 0.01 à 2.5 avec 10 points de mesures (on prendra une valeur fixe d'itération maximum égale à 40 pour l'ensemble des experiences). Commenter les résultats ainsi obtenus et conclure sur la valeur de τ à utiliser.

5 Liste des questions liées au module IF2

Les notations mathématiques (Θ, τ, ...) et représentations des données (X, y, ...) sont rigoureusement les mêmes que celles décrites précédemment.

5.1 Prise en main du projet

A partir du canevas `MNIST_LR` disponible sur moodle IF2 et à l'aide du fichier **README.md** dans le répertoire du projet :

1. Compiler le projet en mode *"Release"*.
2. Exécuter le programme `mnist_train_lrgd` avec les fichiers `data/mnist_train.csv` et `data/mnist_test.csv` puis les valeurs de τ et du nombre d'itérations établis à la suite des conclusions des deux dernières questions de mathématiques. Ce programme effectue l'apprentissage sur le fichier `data/mnist_train.csv` avec la descente de gradient simple (non conjuguée).
3. Vérifer à partir du code de `mnist_train_lrgd.cpp` que le fichier `data/mnist_test.csv` n'est pas utilisé pour l'entrainement (il sera utilisé pour la partie suivante).
4. Dans le rapport vous donnerez la ligne de commande utilisée et un exemple d'exécution puis le temps nécessaire pour le calcul des coefficients de la matrice Θ que vous pourrez comparez au temps mis avec Matlab.

5.2 Premiers développements : fonction Accuracy

1. Passer le projet en mode *"Debug"* pour pouvoir mettre des *breakpoints* dans le code (relire le fichier **README.md**).
2. Exécuter de nouveau le programme `mnist_train_lrgd` et comparer les temps d'exécution avec le mode *"Release"*.
3. Vérifier que les *breakpoints* sont fonctionnels. Typiquement vous observerez le contenu de `argv[2]`, puis vous vérifierez si les contenus de *test_X* et *test_y* sont cohérents.
4. Ajouter aux fichiers `clanu_functions.h` et `clanu_functions.cpp` la fonction `Accuracy` qui calcule et retourne le taux de chiffres bien reconnu sur le nombre de chiffres testés. Cette fonction prend en paramètre la matrice Θ, la matrice *test_X* des images à tester (de forme équivalente à X) et le vecteur *test_y* des valeurs des chiffres de chaque lignes de *test_X*. Les informations de taille des matrices seront aussi passées à cette fonction.
5. On souhaite, lors de la phase d'entrainement, voir s'afficher toutes les 5 itérations de k la valeur de *Accuracy* calculée à partir du contenu du fichier `data/mnist_test.csv` et des coefficients Θ de l'itération en cours (Θ^k). Adapter en conséquence le main de `mnist_train_lrgd.cpp`.

5.3 Seconds développements : fonction de sauvegarde et lecture de la matrice Θ

1. Il s'agit maintenant d'ajouter aux fichiers `clanu_functions.h` et `clanu_functions.cpp` les fonctions permettant de sauvegarder et lire les coefficients de la matrice Θ. Ces fonctions prendront en entrée les arguments suivants : le nom du fichier, la matrice Θ et les tailles de la matrice. Pour les deux fonctions, la matrice Θ **devra déjà être allouée en mémoire** avant l'appel de ces fonctions.
2. Modifier le main du fichier `mnist_train_lrgd.cpp` de manière à sauvegarder la matrice Θ permettant d'obtenir la meilleure Accuracy sur les données de `data/mnist_test.csv`. Il faudra aussi que le nom du fichier de sauvegarde de Θ soit passé en paramètre au programme (comme le sont les fichiers csv et les valeurs de τ et *max_it*).

3. Dans le rapport vous donnerez la nouvelle ligne d'appel à votre programme, la meilleure valeur d'Accuracy sauvegardée et comment vous l'avez obtenue (paramètres utilisés τ et max_it et méthode de recherche des paramètres) ainsi que le format de sauvegarde de la matrice Θ que vous avez utilisé.

5.4 Troisième développement : programme de test `mnist_test.cpp`

En vous appuyant sur les fonctions précédentes :

1. Modifier le fichier `mnist_test.cpp` de manière à lire une matrice Θ puis un fichier cvs de la même forme que `data/mnist_test.csv` ou `data/mnist_train.csv`. Ce programme doit afficher pour chaque ligne du fichier csv la valeur du chiffre obtenu par votre algorithme, la bonne valeur et si la prédiction est bonne ou non. Vous compléterez l'affichage par la précision de la détection pour chacun des 10 chiffres (*"accuracy"*) par chiffre).
2. Dans le rapport vous donnerez les précisions par chiffre obtenu avec votre meilleur matrice Θ pour le jeu d'entrainement `data/mnist_train.csv` et de test `data/mnist_test.csv`.

5.5 Quatrième développement : Descente de Gradient Conjuguée

Il s'agit d'écrire le fichier `mnist_train_lrCgd.cpp` qui implémente la descente de gradient conjuguée comme décrit précédemment.

1. Donner l'algorithme d'entrainement implémenté dans `mnist_train_lrgd.cpp` sous forme de pseudo code et d'équations mathématique. Montrer que cela est conforme à la description mathématique. Justifier le choix algorithmique fait pour le calcul de $\nabla J(\Theta_c)$.
2. En vous inspirant du fichier `mnist_train_lrgd.cpp`, compléter le fichier `mnist_train_lrCgd.cpp` qui implémente la descente de gradient conjuguée.
3. Dans le rapport, montrer comment vous avez validé votre implémentation puis comparer les performances et précisions des deux méthodes de descente de gradient. Conclure.

6 Évaluations et rendus

6.1 Organisation

Ce projet rentre dans le cadre des modules IF2 et MA2 et s'effectue en binôme d'étudiants **du même groupe** (et un monôme en cas d'effectif impair). Il sera évaluer collectivement et individuellement pour les modules MA2 et IF2.

Les dates clées de ce projet :

1. Les binômes sont à établir par les étudiants **sous le module IF2 de moodle avant le 16 mars minuit**. Tous les étudiants doivent avoir un numéro de binôme CLANU pour cette date.
2. L'évaluation individuelle sur la partie mathématique est prévue fin avril.
3. L'évaluation individuelle sur la partie informatique est prévue mi juin (lors du DS IF2).
4. Le rapport est à remettre sur moodle **au plus tard le 3 juin minuit**.

Attention, le rendu se faisant obligatoirement sous *moodle*, aucun délai supplémentaire ne sera accordé. Les retards et autres "problèmes" de soumission sous moodle ne seront pas acceptés.

6.2 Evaluation individuelle de Mathématiques : MA2

Une évaluation individuelle de la partie mathématique aura lieu au retour des vacances d'avril. Cette évaluation s'effectuera sur moodle et prendra la forme d'un TP noté sous matlab basé sur les éléments étudiés au cours des questions de la partie Mathématiques du projet.

6.3 Evaluation individuelle d'Informatique : IF2

L'évaluation individuelle informatique aura lieu pendant la semaine de DS. Cette évaluation s'effectuera sur papier lors du DS IF2. Les questions seront basées sur les éléments étudiés au cours des questions de la partie informatique du projet. Cette évaluation a pour but d'évaluer les capacités en programmation acquises lors de l'apprentissage par projet en s'appuyant sur les outils du projet. Les documents seront autorisés.

6.4 Rapport commun aux modules IF2 et MA2

Chaque binôme déposera sur *moodle* module IF2 le rapport du projet au format pdf.

La structure du rapport sera la suivante :

1. une introduction présentant l'étude et l'organisation de votre rapport.
2. une partie *Analyse Numérique* dans laquelle les réponses aux questions mathématiques seront données ainsi que quelques résultats pertinents.
3. une partie *Informatique* dans laquelle les réponses aux questions informatiques seront données ainsi que quelques résultats pertinents.
4. une conclusion résumant le travail réalisé et les résultats obtenus, ainsi que l'organisation du projet et la répartition du travail dans le binôme.
5. au besoin, une partie bibliographie.

Afin d'éviter certaines dérives du travail en binôme, il vous est demandé de préciser le partage des tâches dans le rapport et l'auteur des sources (cf. conclusion). Pensez à votre responsabilité et à l'éthique par rapport au plagiat.