Sparse Coding Algorithm

Pascal Sitbon

May 3, 2015

Contents

1 Introduction **2**

2 Préliminaires **3**

3 Problème L_1 pénalisé **4**

4 Problèmes des moindres carrés sous contrainte **5**
4.1 Problème Dual 5
4.2 Descente de gradient sous contraintes 5

5 Simulations **6**
5.1 Présentation des données 6
5.2 Problèmes des moindres carrés 6
5.3 Problème L_1 pénalisé 9
5.4 Problème global 9
5.4.1 Représentation des données après ACP 10
5.4.2 Format Sparse 11

6 Conclusion **12**

7 Annexe : Description du code **12**

8 Références **12**

Abstract

Les données massives constituent un domaine de recherche qui ne cesse de croître. La gestion de grosses bases de données nécessite des techniques toujours plus rapides et bien optimisées. Ce document présente les algorithmes couramment appelés *efficient sparse coding*. L'objectif de tels algorithmes est d'apprendre une base de données X, d'en saisir la tendance (*pattern*). Ceci se traduit par la recherche d'une base de vecteurs B dans la quelle on va représenter les vecteurs de X. Les coefficients des éléments de X sur cette nouvelle base sont stockés dans une matrice S. On souhaite par ailleurs maintenir un maximum de coefficients à 0 dans la matrice S. Ce papier étudie le fonctionnement de tels algorithmes. Le problème considéré se résume à un double problème d'optimisation : un problème avec une pénalité de type L_1 pour S et un problème de moindres carrés sous contrainte pour B. Différentes méthodes de résolutions seront présentées dont les méthodes optimales actuelles présentées par Andrew NG. dans *Efficient Sparse Coding Algorithm* Les mots-clés sont : Lagrange Dual, Feature sign Search, Gradient projection, LASSO, Sparse coding algorithm.

1 Introduction

Les algorithmes de type sparse coding sont un nouveau moyen de représenter les données d'une base. Une des méthodes les plus courantes en statistiques pour la réprésentation de données est l'analyse en composante principale (ACP). Cependant il y a une différence fondamentale entre l'ACP et les algorithmes que nous présentons. L'ACP s'intéresse à la réecriture des variables d'une base de données, afin de récupérer des variables non corrélées et qui contiennent la plupart de l'information et qui permettent en théorie de bien classer les individus. Les algorithmes sparses permettent de réecrire non pas les variables mais les points de données afin d'en saisir le pattern. Ces algorithmes sont donc performants sur les base de données représentant des stimulis, c'est à dire des images similaires sur la plupart de l'image sauf à certains endroits caractéristiques. De plus la taille de la nouvelle base de représentation des données obtenue par les algortihmes sparse peut être supérieure à la dimension des éléments de la base de données (nombre de variables), ceci constitue une autre différence avec l'ACP. Un autre avantage des algorithmes sparses est qu'un maximum des coordonnées des vecteurs de l'ancienne base dans la nouvelle seront égaux à zéro, ceci constitue un avantage au niveau stockage de données. J'ai donc sélectionné une base de données sur le site de data science Kaggle représentant des numéros en blanc sur fond noir. Ces images sont similaires sur la plupart des pixels, le nombre blanc représente le stimuli qui caractérise chaque image. Une autre idée de base serait une base de données contenant des

Figure 1: Un exemple de la base de données

radiographies du cerveau avec des tumeurs. Dans un premier temps seront présentés les différents problèmes d'optimisations ainsi que les méthodes théoriques pour les résoudre. J'ai utilisé deux méthodes d'optimisations pour le problème de moindre carrés sous contrainte, la descente de gradient projetée et la résolution du problème dual par un algorithme de Newton. Un nouvel algorithme nommé *feature sign search* extrait du papier d'Andrew Ng [**1**] sera présenté pour le problème de moindres carrés L_1 pénalisé, il sera comparé au Lasso. J'ai ensuite réalisé différentes simulations afin de comparer ces algorithmes sur la base de données présentée ci-dessus.

2 Préliminaires

Cette partie est destinée à présenter le problème sous la forme d'un problème d'optimisation mathématique. Dans toute la suite du rapport, $X \in \mathcal{M}_{k,m}(\mathbb{R})$ représentera la base de données en input. Typiquement chaque colonne de X est un points de données (individu ou image dans notre cas) et k est la dimension du point de données (nombre de pixels de l'image). Par ailleurs m représente le nombre d'individus. $B \in \mathcal{M}_{k,n}(\mathbb{R})$ est la matrice contenant les vecteurs $b_1, .., b_n$ de la nouvelle base, il y a donc n vecteurs de taille k dans la nouvelle bases (on peut avoir $n > k$) $S \in \mathcal{M}_{n,m}(\mathbb{R})$ est la matrice dont les vecteurs s les coefficients des combinaisons linéaires des réprésentations des vecteurs de X sur B. Avec ces notations l'algorithme Sparse est censé déterminer B et S telles que :

$$\forall i \in \{1, ..., m\},\ X^i \simeq \sum_{j=1}^{n} S_j^i b_j \tag{1}$$

Dans un objectif de parcimonie, on utilisera une distribution à priori sur les S^i qui sera telle que $\mathbb{P}(S^i|B) = \exp(-\beta\phi(S^i))$ où ϕ est une pénalité. La pénalité ici considérée est de type L_1:

$$\phi(S^i) = ||S^i||_1 = \sum_{j=1}^{n} |S_j^i| \tag{2}$$

La pénalité L_1 permet de conserver des coefficients faibles. On suppose par ailleurs que l'erreur $X^i - \sum_{j=1}^{n} S_j^i b_j$ suit une loi normale, de variance fixe (indépendante de i) $\sigma^2 I$. C'est une des hypothèses criticables dans le modèle de Andrew Ng [1] car elle n'est pas justifiée, cependant c'est une hypothèse courante dans la littérature statistiques dans les problèmes de regressions. En assumant un prior uniforme sur les vecteurs de la base, on est ramené au problème de maximisation de log-vraisemblance qui peut s'exprimer comme le problème de minimisation suivant:

$$\min_{(S^i)_{1\leq i\leq m},(b_j)_{1\leq j\leq n}} \frac{1}{2\sigma^2} \sum_{i=1}^{m} ||X^i - \sum_{j=1}^{n} S_j^i b_j||_2^2 + \beta \sum_{i=1}^{n} ||S^i||_1 \tag{3}$$

Il est cependant nécessaire d'imposer une contrainte au problème auquel cas il suffirait d'amener les $||S^i||_1$ à 0 tout en laissant inchangée $\sum_{j=1}^{n} S_j^i b_j$ (par exemple multiplier les vecteurs b_j par 2 et diviser tous les S_j^i par 2). On impose alors la contrainte convexe suivante :

$$\forall j \in \{1, ..., n\}\ ||b_j||_2^2 \leq c \tag{4}$$

Où c est un réel strictement positif. En reformulant le problème avec la notation matricielle introduite en début de préliminaire, le problème s'écrit finalement :

$$\min_{B,S} \frac{1}{2\sigma^2} ||X - BS||_F^2 + \beta \sum_{i=1}^{n} ||S^i||_1 \tag{5}$$

Sous la contrainte

$$||b_j||_2^2 \leq c$$

Où $||.||_F$ est la norme matricielle de Froebenius. Le problème ne peut être résolu simultanément en B et en S car il n'est convexe que en B à S fixé et en S à B fixé. L'idée est alors d'alterner l'optimisation sur B et sur S, et d'utiliser différents algorithmes pour chacun des deux problèmes.

L'optimisation sur S se résume à un problème de moindres carrés L_1 pénalisé et l'optimisation sur B à un problème des moindres carrés sous contraintes. Le papier d'Andrew NG présente une méthode pour l'optimisation sur S nommée *feature sign search*, nous comparerons cette méthode à la méthode du LASSO. L'optimisation de B se fait en résolvant le problème dual. J'ai par ailleurs codé la descente de gradient projeté (en calculant par moi-même le projeté orthognal d'une matrice sur la contrainte convexe du problème (5) pour la norme de Froebenius). Le théorème de projection sur un convexe fermé assurait en effet l'existence de ce projeté orthogonal.

3 Problème L_1 pénalisé

Le problème d'optimisation sur S peut se résumer à un problème L_1 pénalisé s'écrivant:

$$\min_s ||x - Bs||_2^2 + \lambda ||s||_1 \tag{6}$$

La nouvelle méthode présentée dans l'algorithme est le *Feature Sign Search*. Ses performances seront comparées à des méthode plus classiques. Ce permet de maintenir le maximum de coefficients de x égaux à 0 tout en minimisant l'erreur. Cet algorithme se base sur deux conditions d'optimalités sur les coefficients nuls et non nuls de la solution donnée. Elle part d'une solution nulle et les coefficients non nuls seront changés au fur et à mesure de l'algorithme et que si nécessaire (à savoir si la condition 4(b) n'est pas vérifiée dans l'agorithme ci-dessous). L'étape *discrete line search* permet à coup sur de diminuer l'objective value (par convexité de celle-ci [**1**]) L'algorithme s'écrit de la façon suivante :

Algorithm 1 Feature-sign search algorithm

1 Initialize $x := \vec{0}, \theta := \vec{0}$, and *active set* $:= \{\}$, where $\theta_i \in \{-1, 0, 1\}$ denotes $\text{sign}(x_i)$.

2 From zero coefficients of x, select $i = \arg\max_i \left| \frac{\partial \|y - Ax\|^2}{\partial x_i} \right|$.
  Activate x_i (add i to the *active set*) only if it locally improves the objective, namely:
    If $\frac{\partial \|y - Ax\|^2}{\partial x_i} > \gamma$, then set $\theta_i := -1$, *active set* $:= \{i\} \cup$ *active set*.
    If $\frac{\partial \|y - Ax\|^2}{\partial x_i} < -\gamma$, then set $\theta_i := 1$, *active set* $:= \{i\} \cup$ *active set*.

3 Feature-sign step:
  Let $\hat{A}$ be a submatrix of A that contains only the columns corresponding to the *active set*.
  Let $\hat{x}$ and $\hat{\theta}$ be subvectors of x and θ corresponding to the *active set*.
  Compute the analytical solution to the resulting unconstrained QP ($\text{minimize}_{\hat{x}} \|y - \hat{A}\hat{x}\|^2 + \gamma \hat{\theta}^\top \hat{x}$):
    $\hat{x}_{new} := (\hat{A}^\top \hat{A})^{-1}(\hat{A}^\top y - \gamma \hat{\theta}/2)$,
  Perform a discrete line search on the closed line segment from $\hat{x}$ to $\hat{x}_{new}$:
    Check the objective value at $\hat{x}_{new}$ and all points where any coefficient changes sign.
    Update $\hat{x}$ (and the corresponding entries in x) to the point with the lowest objective value.
  Remove zero coefficients of $\hat{x}$ from the *active set* and update $\theta := \text{sign}(x)$.

4 Check the optimality conditions:
  (a) Optimality condition for nonzero coefficients: $\frac{\partial \|y - Ax\|^2}{\partial x_j} + \gamma \,\text{sign}(x_j) = 0, \forall x_j \neq 0$
    If condition (a) is not satisfied, go to Step 3 (without any new activation); else check condition (b).
  (b) Optimality condition for zero coefficients: $\left| \frac{\partial \|y - Ax\|^2}{\partial x_j} \right| \leq \gamma, \forall x_j = 0$
    If condition (b) is not satisfied, go to Step 2; otherwise return x as the solution.

Figure 2: Algoritme Feature Sign Search extrait de la référence [1]

J'ai comparé cet algorithme à la méthode du Lasso. Cette méthode s'applique à une base de données sur laquelle on souhaite faire une régression régularisée qui vise à éliminer les variables peu pertinentes (i.e. mettre beaucoup de coefficients de s à 0). Il est préfererable d'utiliser une regression Lasso lorsque la matrice B a ses variables corrélées, sinon une régression simple sera en général plus efficace.

4 Problèmes des moindres carrés sous contrainte

4.1 Problème Dual

L'optimisation sur B peut se résumer au problème suivant :

$$\min||X - BS||_F^2 \tag{7}$$

Sous la contrainte (4).
Il s'agit dans cette partie de présenter la méthode de résolution du problème dual de ce problème de moindres carrés sous contrainte. Le Lagrangien de ce problème peut s'écrire comme suit :

$$\mathcal{L}(\lambda, B) = \mathrm{Tr}((X - BS)^T(X - BS)) + \sum_{j=1}^{n} \lambda_j(||b_j||^2 - c) \tag{8}$$

Où les $\lambda_j \geq 0$ sont les variables de Lagrange duales. On maximise alors le dual $\mathcal{D}$:

$$\mathcal{D}(\lambda) = \min_B(\mathcal{L}(\lambda, B)) = \mathrm{Tr}(X^TX) - \mathrm{Tr}(XS^T(SS^T + \Lambda)^{-1}(XS^T)^T) - c\mathrm{Tr}(\Lambda) \tag{9}$$

Où Λ est la matrice diagonale dont les coefficients diagonaux sont les λ_j. $\mathcal{D}$ est obtenu en égalisant le gradient du Lagragien par rapport à B à 0 et en réinjectant l'expression de B obtenue dans le Lagrangien. On optimise alors le dual grâce à la méthode de Newton (algorithme BFGS [2]). Une fois λ obtenu, la base optimale est donnée par l'équation suivante:

$$B^T = (SS^T + \Lambda)^{-1}(XS^T)^T \tag{10}$$

4.2 Descente de gradient sous contraintes

La descente de gradient est une méthode connue pour trouver les extremas de fonctions de classe $\mathcal{C}^1$ sur un ouver Ω. Cependant l'algorithme tel quel n'est pas adapté à un problème d'optimisation sous contrainte. On cherche ici l'optimum d'une fonction sur un convexe qui est fermé. L'idée pour adapter cet algorithme est de projeter le point retourné par la descente de gradient sur la contraint convexe. Plus précisemment, l'algorithme de descente de gradient simple s'écrit $B_{t+1} = B_t - \eta_t \nabla B(x_t)$ $(*)$, avec une condition d'arrêt sur la norme du gradient du type : "Pour $\epsilon > 0$ tant que $||\nabla L|| > \epsilon$, itérer $(*)$". En effet, si la norme du gradient est faible, on est proche d'un extremum.
Ici ce type de contrainte n'est pas valable car on cherche le maximum sur un fermé, le gradient n'est donc pas forcément nul. On s'intéressera alors à la convergence avec une condition d'arrêt du type: "Pour $\epsilon > 0$, tant que $||B_{t+1} - B_t||_F > \epsilon$, iterer $B_{t+1} = \Pi_{\mathcal{C}}(B_t - \eta_t \nabla L(B_t))$", où $\Pi_{\mathcal{C}}$ est le projecteur orthogonal pour la norme de Froebenius sur la contrainte $\mathcal{C} = \{B \in \mathcal{M}_{k,n}(\mathbb{R}) / \forall j \in \{1, ..., n\},\ ||b_j||_2^2 \leq c\}$. Le projecteur orthogonal pour la norme de Froebenius est défini par :

$$\Pi_{\mathcal{C}}(B) = \mathrm{Argmin}_{B' \in \mathcal{C}}||B - B'||_F \tag{11}$$

Dans notre cas, il suffit de prendre la matrice B' dont les vecteurs colonnes b'_j sont les projetés des vecteurs b_j sur la contrainte $\mathcal{F} = \{b \in \mathbb{R}^k / ||b||_2^2 \leq c\}$. Et géométriquement il on peut voir que le projeté orthogonal b'_j de b_j sur $\mathcal{F}$ est une homothétie de rapport $\frac{\sqrt{c}}{||b||_2}$. En effet la contrainte est d'être dans la boule de centre 0 et de rayon $\sqrt{c}$. Montrons alors que P la matrice dont les colonnes sont les $\Pi_{\mathcal{F}}(b_j) = \frac{\sqrt{c}}{||b_j||_2} b_j$ est solution du problème (11). Remarquons que pour $B' \in \mathcal{C}$, $||B - B'||_F^2 = \sum_{j=1}^n ||b_j - b'j_|_2^2 \geq \sum_{j=1}^n ||b_j - \Pi_{\mathcal{F}}(b_j)||_2^2 = ||B - P||_F^2$. Ainsi on a bien $\Pi_C(B) = P$. (NB: Cette démonstration est personnelle).

5 Simulations

5.1 Présentation des données

La base de données est téléchargeable depuis le site de data science Kaggle (https://www.kaggle.com/c/digit-recognizer/data?train.csv). Elle contient des images de nombres compris entre 0 et 9, 1000 images ont été séléctionnées pour effectuer les simulations ($m = 1000$). Chaque image contient 24*24 pixels, les labels ont été supprimés, afin d'appliquer les algorithmes sparses. Une image se résume alors dans la base de données à un vecteur de $\mathbb{R}^{784}$ (k=784). Les données ont été normalisées avant les simulations. Dans la pratique statistique, les problèmes de régression régularisée L_1

Figure 3: Extrait de la base de données

sont utilisés lorsque la matrice B du problème (6) a ses variables corrélées. Dans notre cas la plupart des pixels d'une image (c'est à dire d'un vecteur de la base X) sont noirs et le sont tout le temps au même endroit sauf au centre (cf figure 3) là ou est écrit le numéro. Si les variables des X sont corrélées on peut imaginer que celle de la matrice B le sont aussi puisque celle-ci est censé reproduire les pattern de la base X. Par ailleurs dans mes simulations j'ai choisi de prendre des valeurs de n telles que $n \geq k$, donc a priori les variables de B seront toujours corrélées. De plus dans les simulations on se rend compte que plus la valeur de n est grande plus les algorithmes sont performants. C'est donc légitime d'essayer ce genre de méthode pour l'optimisation sur S. Lorsque j'ai analysé les corrélations sur X, la plupart des corrélations étaient situées entre 0.98 et 1 sauf pour la correlation entre les variables correspondant aux pixels sur les frontières et celles à l'emplacement des numéros. . Les méthodes utilisées pour résoudre le problème de moindre carrés sous contraintes sont classiques, la méthode du dual est par exemple utilisée pour résoudre le problème d'optimisation posée par les machines à vecteurs supports. La descente de gradient est la méthode la plus répandue en optimisation numérique. Cependant je n'ai pas trouvé d'utilisation du gradient projeté pour ce problème dans la littérature.

5.2 Problèmes des moindres carrés

Nous allons ici comparer la méthode de descente de gradient projetée et la méthode du dual sur notre base de données. La matrice X contient les images, S est donnée (ayant des coefficients pris aléatoirement). On résoud alors le problème (7) par descente de gradient projetée et par la méthode du dual. Voici un graphique représentant la valeur de $||X - BS||_F^2 = \mathrm{Tr}(X - BS)^t(X - BS)$ en fonction du nombre d'itérations pour les 2 méthodes d'optimisation. L'algorithme résolvant le problème dual permet d'atteindre une erreur très faible en peu d'itérations comparé à la descente de gradient. La descente de gradient présente cependant un avantage dans le sens ou chaque itération est peu coûteuse en termes de calculs comparé à l'autre algorithme.

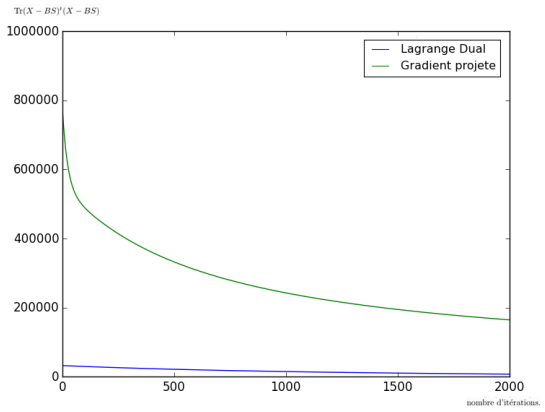


Figure 4: Graphique représentant l'erreur en fonction du nombre d'itérations

En effet chaque itération dans la résolution du problème dual est faite avec l'algorithme de type Newton qui nécessite le calcul du gradient l'approximation de la hessienne du dual. Ces calculs sont couteux car ils font intervenir des inverses de matrices de dimension n^2, ainsi que des produits matriciels comme le montrent les formules suivantes:

$$\frac{\partial D(\lambda)}{\partial \lambda_i} = ||XS^T(SS^T + \Lambda)^{-1}e_i||^2 - c \tag{12}$$

$$\frac{\partial^2 D(\lambda)}{\partial \lambda_i \partial \lambda_j} = -2((SS^T + \Lambda)^{-1}(XS^T)^T XS^T(SS^T + \Lambda)^{-1})_{ij}(SS^T + \Lambda)^{-1}_{ij} \tag{13}$$

Le calcul du gradient dans la descente de gradient est bien moins coûteux, en notant $L(B) = ||X - BS||_F^2$ on a:

$$\nabla_B(L) = -2(X - BS)S^T \tag{14}$$

On préferera néamoins utiliser l'algorithme du dual car finalement plus rapide et fournissant un meilleur résultat que dans la descente de gradient projetée. La principale diffuculté dans l'utilisation de la descente de gradient projetée réside dans le choix du pas de la descente de gradient. Pour obtenir des pas efficaces permettant aux algorithmes de converger, j'ai fait des simulations en faisant varier le pas sur des valeurs espacées de 0.1 entre 0 et 1, puis par des espaces 0.001, et ainsi de suite (environ 10 fois) jusqu'à obtenir un algorithme convergent et faisaint diminuer l'erreur, comme c'est le cas sur la figure 3. Le paramètre c doit aussi être optimisé, sachant qu'à partir d'une certaine valeurs de c la descente de gradient projeté est une descente de gradient simple. Plus la contrainte sur c est importante plus il est dur d'atteindre une *ojective value faible* et une convergence rapide. Voici un graphique représentant l'objective value atteind par la descente de gradient projetée en fonction de $c = 10^{-17}...10^{12}$ pour 2000 d'itérations dans la descentes de gradient et $n = 200$. On observe sur la figure 4 que la contrainte joue a un impact fort sur la solution trouvée par la descente de gradient. La valeur objectif atteinte décroit lorsque c augmente. C'est en accord avec la théorie dans le sens où minimiser sur un espace plus grand permet de trouver un meilleur minimum. Par ailleurs on remarque qu'à partir de l'abcisse 17 sur le graphqiue, la descente de gradient projetée est alors une descente de gradient simple. Les paramètres du problème d'optimisation font qu'à partir de cette valeur de c la contrainte n'a plus d'influence sur la solution trouvée. Le choix de la valeur de c peut se faire à partir de ce graphique, en attribuant un importance égale à la valeur de l'objective

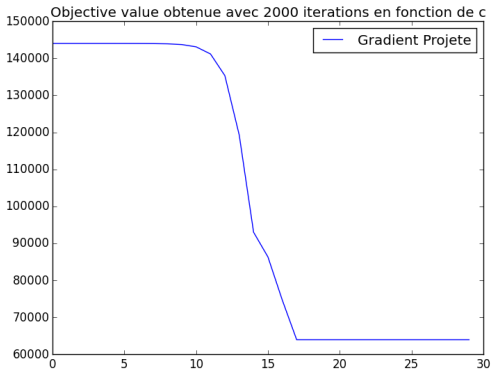


Figure 5: *Objective value* atteinte par la descente de gradient projetée (DGP) en fonction de la contrainte c [1]

value et à la valeur des vecteurs de la base B, on peut choisir la valeur de c correspondant à l'abcisse sur la figure 4. On observe également que lorsque l'on diminue la valeur de n (à savoir le nombre de vecteurs de la nouvelle base), il est dur d'obtenir une objective value faible. C'est en accord avec l'intuition, plus j'ai de vecteurs à dispositions pour B plus j'aurai de moyen de représenter ma base de données. Par ailleurs, plus n est grand plus algorithmes sont couteux en terme d'opérations.

Expérience	Objective value	Temps d'execution (s)
DGP, $n = 1000$, 2000 itérations	160 000	207
DGP, $n = 1200$, 2000 itérations	140 000	325
Dual, $n = 1000$, 2000 itérations	20 587	17
Dual, $n = 1200$, 2000 itérations	0.01	108

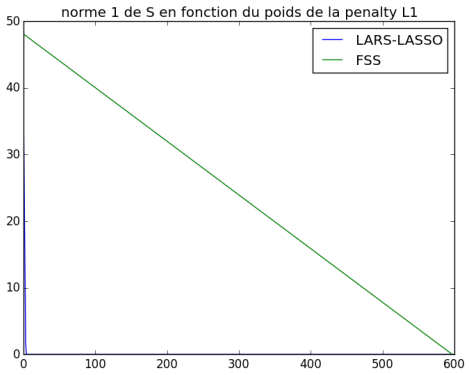
Dans la suite j'ai décidé de redimensionner la base d'image au format $12 * 12$ grâce au package PIL. La raison principale est que les algorithmes résolvant le problème L1 pénalisé sont trop longs à résoudre le programme d'optimisation. Voici les résultats obtenus en conservant à peu près une valeur équivalente de $\frac{k}{n}$ sur la base avec les images au nouveau format :

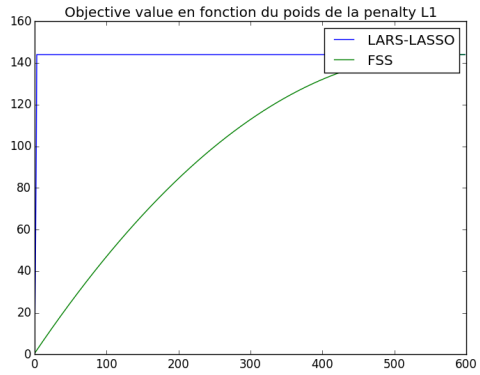
Expérience	Objective value	Temps d'execution (s)
DGP, $n = 200$, 2000 itérations	77 319	16
DGP, $n = 400$, 2000 itérations	74 716	26
Dual, $n = 200$, 2000 itérations	73 954	0.53
Dual, $n = 400$, 2000 itérations	55 858	2.35

On constate comme dans les expériences précedentes qu'augmenter n permet d'atteindre une meilleure *objective value*, mais cela au détend d'un temps d'exécution plus long. Par ailleurs, les algorithmes atteignent une meilleure objective value lorsque la base n'est pas resizé tout choses équivalentes par ailleurs. Ceci est dû au fait que l'on perd de l'information lorsque l'on redimensionne les données, et qu'il est plus difficile d'apprendre le pattern d'une base avec moins d'informations dessus. Comme nous l'avions supposé en 5.1, on observe dans les simulations que la matrice des covariances de B présentent énormément de valeurs très proches de 1. On remarque par ailleurs, que les résultats entre DGP et Dual sont plus similaires que dans le cas où la base n'est pas reformatée, même si la méthode duale reste toujours plus performante.

5.3 Problème L_1 pénalisé

Dans cette partie, on s'intéresse aux simulations relatives à l'optimisation sur S, on se réduira dans cette sous-partie à la résolution du problème (6) c'est à dire pour une image seulement, afin de voir la performance sur un seul problème L_1 pénalisé, les simulations sur la base entière seront faires en 5.4. On résoud alors le problème (6) pour une image x, une matrice B est prise égale à celle renvoyée par l'algorithme dual afin de conserver les features telles que la corrélation entre les variables. Nous allons alors comparer les méthodes Lars-Lasso, Feature Sign Search (FSS). Par ailleurs pour faire tourner ces algorithmes j'ai resize les images au format 12*12 pour pouvoir faire plus de simulations, ces simulations étant trop longues sur les images 28*28 (supérieure à 5 heures pour $m = 1$ et $k = 28 * 28$). La figure ci-dessous représente l'objective value du problème (6) en fonction du paramètre de parcimonie λ, pour une image x donéee. Sur les figures les abcisses i sont les valeurs de $\lambda_i = 0.01 * i, i = 1..600$. Comme le calcul le montre, on observe que sur la figure ci-dessous, si le coefficient λ tend vers l'infini la norme 1 de s (cf problème (6)) va tendre vers 0. J'ai par ailleurs testé les algorithmes avec une matrice B prise aléatoire (et donc à priori ayant ses variables non corrélées), les algorithmes sont plus lent et moins efficace, et ils ne convergent pas forcément sur certaines expériences.





J'observe cependant un phénomène étrange, c'est que la plage de variation de l'objective value (avant qu'elle ne stagne) n'est pas la même pour l'algorithme FSS que pour l'algorithme Lasso. En d'autres termes termes la courbe bleu sur les 2 graphiques a ses prinicipales variations sur des valeurs beaucoup plus faibles de λ et que la courbe bleue. J'ai donc tracé un graphique centrée sur de plus faibles valeurs de λ (sur la figure 6 les abcisses i correspondent aux valeurs de $\lambda_i = 0.0001 * i, i = 1..600$). Je peux cependant estimer sur chacun des graphiques un valeur de λ notée λ^* qui attribue une importance égale à l'objective value atteinte et à la valeur de la norme 1 de s. Sur le graphique de FSS on peut choisir $i = 250$ qui correspond à $\lambda^*_{FSS} = 2.5$ et sur la figure 6 on peut choisir $i = 300$ qui correspond à $\lambda^*_{lasso} = 0.03$.

$\lambda = \lambda^*$	Objective value	Temps d'execution (s)	norme 1
FSS, $n = 200$	25.6	0.02	27.9
LASSO, $n = 200$	74.5	0.001	14

5.4 Problème global

On s'intéresse dans cette partie à la résolution du problème global (5), k est fixé égale à 144 sauf On alterne l'optimisation sur S et sur B en choisissant les méthodes pour S et pour B. Voici un

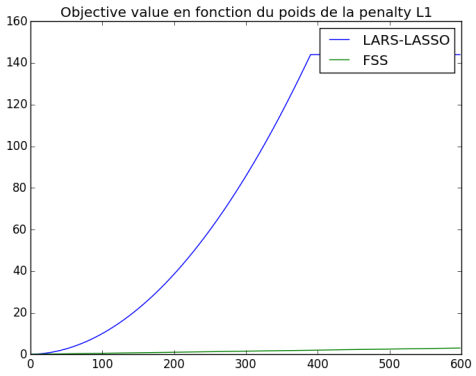


Figure 6: Objective value en fonction deλ [1]

tableau récapitulant l'objective value obtenue par les différents duo d'algorithmes. J'ai décider de selectionner une petite partie de la base de données pour les simulations par soucis de temps.

5.4.1 Représentation des données après ACP

J'ai réalisé plusieurs ACP sur la base d'images, en conservant à chaque fois un nombre différents d'axes. J'ai ensuite projeté mes images sur les nouveaux axes grâce à la fonction *pca.transform()* puis je les ai replacé dans l'espace initial avec la fonction *pca.inverse-transform()*. On remar-

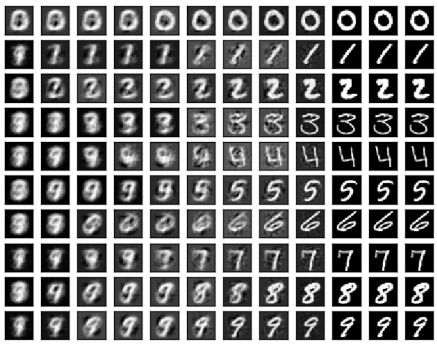
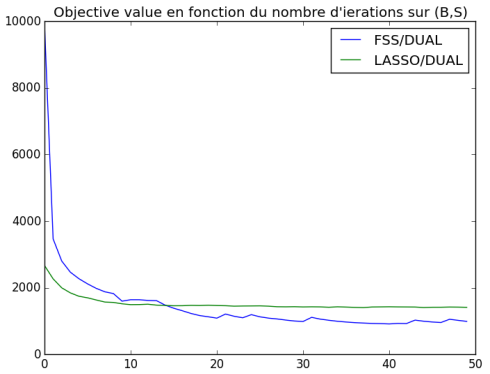


Figure 7: Données après ACP pour un nombre d'axes conservé $n = 1, 2, 3, 4, 5, 10, 20, 30, 60, 100, 150, 200,\ k = 784$

que qu'à partir de 100 axes l'information apportée par les nouveaux axes est negligeable. Une extension intéressante serait de comparer un algorithme d'apprentissage type réseaux de neurones (dont l'objectif est d'apprendre à dire le numéro présent sur une image) sur la base de données projetée sur les axes d'une des ACP, puis de faire le même travail avec la représentation Sparse. La comparaison montrerait alors laquelle des représentations est la plus pertinente pour le problème d'apprentissage sur ces images.

5.4.2 Format Sparse

J'ai effectuer les différents duo d'algorithmes possibles sur la base de données. Les figures suivantes montrent l'evolution de *l'objective value* gobale (équation (5)) et la norme 1 de la matrice S en fonction du nombre d'itérations sur le couple (B, S) (chaque itération consiste en une itération pour S et une pour B). Par ailleurs, ici la plage de variation de Lasso et Fss en fonction de λ est la même pour le Lasso et Feature Sign Search (j'ai obtenu $\lambda^* = 5$). Les simulations suivantes sont donc faites à λ fixé et égal à λ^*. Par ailleurs dans les deux graphiques suivants et dans le tableau $n = k$.



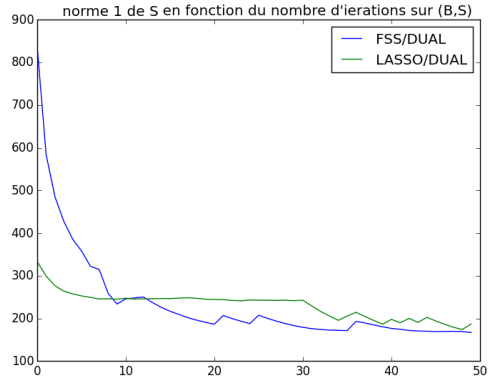


Figure 8: Images de la base B obtenue avec le duo FSS/DUAL, $n = 160 > k = 144$

Expérience $n = 144 = k$	Objective value	Temps d'execution (s)
FSS/DUAL	883	618
LASSO/DUAL	1 040	18
FSS/DGP	5525	4785
LASSO/DGP	6260	128

C'est finalement le duo FSS/Dual qui est le plus efficace pour résoudre le problème (5) avec notre base de données. On remarque sur les figures ci-dessus que les algorihmes convergent (environ

au bout de 50 itérations) et la norme 1 de S diminue bien avec le nombre d'itérations et converge elle aussi. Néanmoins le temps d'execution du duo FSS/DUAL est cepenant plus long que celui du duo LASSO/DUAL. La figure 8 réprésente les images (pour le cas $n = 160 > k = 144$) que la matrice B contient.

6 Conclusion

Les algorithmes de type sparse sont très utilisés en data science, nous avons vu ici quelques méthodes de résolution des problèmes d'optimisations posés par ces algorithmes. Le duo d'algorithme *feature sign search*/Dual s'avère être le plus performant sur ma base de données. Cependant il aurait été intéressant de faire des simulations sur un plus grand nombre d'images et pour plein de valeurs de n, ce qui a été impossible dans le temps imparti. Une extension intéressante serait de comparer la vitesse d'algorithmes d'apprentissage sur nos données dans les formats simples avec ou sans ACP et au format sparse.

7 Annexe : Description du code

Les codes sont séparés en deux parties. Certains sont des méthodes les autres des simulatios.
Les fichiers méthodes sont les suivants :

- *feature-sign-search-l1.py* est le fichier contenant l'algorithme feature sign search et son application à une base de données.
- *gradient-proj.py* est le fichier contenant l'algorithme de gradient projeté.
- *new-learning-basis.py* est le fichier contenant l'algorithme du dual.
- *Lasso.py* est le fichier contenant l'application du Lasso à une base de données

Les fichiers simulations sont les suivants:

- *acp-grap.py* contient le fichier faisant l'acp et la reconstitution des images.
- *resizing-savingimages.py* contient le fichier qui redimensionne les images et qui sauvearde le résultat.
- **IMPORTANT** *base-extraction* contient le fichier faisant l'algorithme Dual/Fss sur la base redimensionnée, et qui permet d'obtenir la figure 8. Vous pouvez lancer ces codes, l'optimisation se lancera et les graphiques s'afficheront.
- *simulation-optim-BS-dgp-fss-lasso.py* et *simulation-optim-BS-dual-fss-lasso.py* sont d'autres fichiers de simulations qui m'ont servi à faire mes simulations. Vous pouvez lancer ces codes, l'optimisation se lancera et les graphiques s'afficheront.
- *simulation-optim-B.py simulation-optim-S.py* contiennent les simulations sur les problemes indépendants.

8 Références

[1] Honglak Lee Alexis Battle Rajat Raina Andrew Y. Ng. Efficient sparse coding algorithms
[2] Y. Censor and S. A. Zenios. Parallel Optimization: Theory, Algorithms and Applications. 1997
[3] Cours d'optimisation de l'ENSAE Paris Tech.