SVD, POD et OMP

Ch. Dossal

Novembre 2008

1 Introduction

Le but de ce TD est de présenter la POD (Proper Orthogonal Decomposition) qui est une application de la SVD (Singular Value Decomposition) et de combiner la POD à un algorithme d'OMP (Orthogonal Matching Pursuit) en compression et débruitage. La POD combinée avec l'OMP peut également être utilisée pour de la détection de motifs.
La svd est ici utilisée pour détecter les motifs les plus récurrents dans une série d'images.
Plus précisément la svd va nous servir à créer un dictionnaire d'imagettes 8×8 à partir d'une image ou d'un jeu d'images de références. Nous utiliserons ensuite ce dictionnaire pour effectuer une compression ou un débruitage d'images plus grandes.

1.1 Rappels sur la SVD

Soit A une application linéaire de $\mathbb{R}^n$ and $\mathbb{R}^p$. la notion de vecteur ou de valeur propre n'a pas de sens car A n'est pas un endomorphisme mais on peut quand même définir des valeurs dites *singulières* et des vecteurs associés. En effet la matrice de Gram A^tA est une matrice carrée $p \times p$ contenant les produits scalaires des colonnes de A. Cette matrice peut être vue comme la matrice d'un endomorphisme qui possède des valeurs et des vecteurs propres. Comme A est une matrice réelle, A^tA est une matrice réelle symétrique positive ainsi ces valeurs propres sont positives et peuvent être notées λ_k^2 avec $\lambda_k \geqslant 0$, pour tout k. Par convention les λ_k sont rangées par ordre décroissant, λ_0 représente donc la valeur propre la plus élevée, etc ...
Si on note les vecteurs propres associés e_k, on a pour tout k, $A^tAe_k = \lambda_k^2 e_k$.
Notons que si $\lambda_k \neq \lambda_i$ alors $\langle e_k, e_i \rangle = 0$. **Les espaces propres de A^tA sont orthogonaux**.
On a donc $\lambda_k = \|Ae_k\|_2$. Les λ_k sont appelés valeurs singulières de A. Si les vecteurs colonnes de A forment une famille libre alors 0 n'est pas une valeur singulière, si les vecteurs colonnes de A forment une famille liée 0 est une valeur singulière. Plus précisément si $p > n$, 0 est valeur singulière d'ordre au moins $p - n$, ainsi le nombre de valeurs singulières non nulles est au plus n.
Les vecteurs $u_k = Ae_k$ jouent un rôle très important. Notons d'abord qu'on retrouve la propriété d'orthogonalité des vecteurs singuliers directement héritée de l'othogonalité des vecteurs e_k: Si $\lambda_k \neq \lambda_i$ alors $\langle u_k, u_i \rangle = 0$.
Si A représente une collection de vecteurs $(a_i)_{i \leqslant p}$ de $\mathbb{R}^n$, le vecteur $u_0 = Ae_0$ est le vecteur qui *représente le mieux statistiquement les vecteurs* $(a_i)_{i \leqslant p}$. Le vecteur u_1 est le vecteur orthogonal à u_0 qui *représente le mieux statistiquement les vecteurs* $(a_i)_{i \leqslant p}$, etc ...
La commade **svd** de matlab calcul pour une matrice A, les valeurs singulières λ_k, les vecteurs e_k et les vecteurs u_k.

2 SVD sur des images

Nous proposons d'établir un dictionnaire *d'imagettes* de références avec la commande **svd**. Pour cela nous allons construire une matrice A dont les vecteurs colonnes correspondent à des *imagettes* $d \times d$ issues d'une image.

1. A l'aide de la fonction **reshape** écrire une fonction **Vectorialiser**

   ```
   function A=Vectorialiser(I,d)
   ```

   qui prend en entrée une matrice I et un entier d et qui renvoie une matrice A dont chaque colonne correspond à une imagette $d \times d$ issue de la partition canonique de A en imagettes $d \times d$.
   Attention : on prendra soin d'enlever sa moyenne à chaque imagette, de manière à ne conserver que des vecteurs de moyenne nulle.

2. A l'aide la fonction **svd** et de la fonction **Vectorialiser**, écrire une fonction **Dictionnaire**

   ```
   function D=Dictionnaire(I,d,n)
   ```

   qui prend en entrée une matrice I et deux entiers d et n et qui renvoie une matrice D ayant $d \times d$ lignes et n colonnes. Les colonnes de D doivent correspondre aux n imagettes $d \times d$ qui représentent le mieux la collecion d'imagettes associée à la matrice I.
   Quelques remarques :

   - Les vecteurs u_k correspondent aux vecteurs colonnes de la première matrice (appelée U dans le code matlab) fournie par le commande *svd* appliquée à la matrice A.
   - On ne considèrera que les $n-1$ premiers vecteurs u_k fournis par **svd** et on ajoutera un vecteur colonne constant à la matrice D.
   - On renormalisera tous les vecteurs colonnes de D de manière à ce que leurs normes euclidiennes soient toutes égales à 1. Ce dernier point est surtout utile en vu du Matching Pursuit.

3. Ecrire un script qui permet de visualiser les premières imagettes ainsi obtenues.

4. Faire des tests sur différentes images et comparer les imagettes de références ainsi obtenues. On pourra utiliser la commande **ReadImage** pour générer des images.

5. A quoi vous font penser ces imagettes ?

3 OMP pour la POD

Nous allons avoir besoin d'un programme réalisant un (OMP) Orthogonal Matching Pursuit pour exploiter ce dictionnaire d'imagette.

6. En reprenant le TD consacré au MP et à l'OMP écrire un programme OMP

   ```
   X=OMP(y,D,nb)
   ```

prenant en entrée un vecteur y taille n, une matrice D de taille $n \times p$, un entier nb et renvoyant le vecteur X de taille p obtenu après nb itérations d'OMP.

7. A l'aide le fonction précédente et de la fonction **reshape** écrire une fonction **PODImagette**

   ```
   function Rec=PODImagette(I,D,nb)
   ```

   qui prend en entrée une matrice I représentant une imagette $d \times d$, un dictionnaire D et un entier nb et qui renvoie une reconstruction Rec par OMP sur le dictionnaire D de I en nb étapes.

8. Faire des tests sur différentes imagettes.

9. L'OMP utilise t-il souvent les mêmes vecteurs ?

10. Ecrire une fonction **POD**

    ```
    function Rec=POD(I,D,nb)
    ```

    qui prend en entrée

    - une matrice I représentant une image,
    - une matrice D représentant un dictionnaire D d'imagettes
    - et un entier nb déterminant le nombre d'imagettes utilisées pour reconstruire chaque imagette de I par OMP.

    et qui renvoie une reconstruction Rec de I en effectuant un OMP sur chaque imagette.

11. Faire différents tests avec différents dictionnaires construits à partir de I ou pas. Mesurer la qualité de reconstruction en terme de psnr. Faire varier la taille du dictionnaire et le nombre de vecteurs choisis par l'algorithme d'OMP.

12. Pour un nombre de paramètres équivalent, comparer l'approximation obtenue à une approximation non linéaire en ondelettes (Ondelettes de daubechies 4 par exemple ou ondelettes biorthogonales) en terme de psnr.

4 Débruitage avec la POD et l'OMP

On peut également combiner la POD et l'OMP pour effectuer un débruitage. Pour cela on construit un dictionnaire sur une image ou un jeu d'images non bruitées et on effectue une approximation d'une image bruitée dans la famille d'imagettes obtenue par la POD.

13. Avec la fonction **POD**, faire différents essais de débruitage, en choisissant de construire D à l'aide de l'image recherchée ou non. Mesurer les qualités des débruitages ainsi obtenus en terme de psnr.

14. On constate que cette méthode de débruitage est surtout efficace à fort niveau de bruit, quand on conserve un nombre de vecteurs limité. On voit alors apparaitre les bords des blocks $d \times d$.
    Pour eviter ces effets et améliorer la qualité du débruitage, écrire une fonction **DebruitPODTI** de débruitage par POD et OMP utilisant l'invariance par translation vue sur les TD en ondelettes.

```
function Rec=DebruitPODTI(I,D,nb,TI)
```

où TI désigne le nombre de translations considérées sur chaque dimension. Pour un nombre de translations donné, comment choisir les translations ?

15. Comparer cette nouvelle approche avec la précédente.