Reconnaissance de chiffres manuscrits

Loïc Abbès Daniel Déchelotte

6 juillet 2001

Table des matières

Table des figures

1 Recherche des composantes principales

1.1 Lecture, centrage et normalisation des données

La lecture et le centrage des données sont réalisés dans le début du script `KL.m`. Compte-tenu de la taille conséquente du fichier de test, nous ne prenons que les `NbKL = 4096` premières images pour réaliser le traitement. Le fichier de test possède un en-tête de 4 entiers de 32 bits puis des images carrées de `taille` 28 pixels de côté. Après avoir sauté l'en-tête, on lit donc le fichier par blocs de 28×28 octets. Les données obtenues sont rangées en colonne dans une matrice que l'on transpose. `images` récupère l'intégralité des images lues, `NbKL` lignes de `taille*taille` colonnes.

```
img_fic=fopen('train-images-idx3-ubyte','rb','ieee-be');

entete = fread(img_fic, 4, 'int32');
if entete(1) != 2051
  disp(sprintf('[Attention] Nombre magique : %d != %d.',
               entete(1), 2051));
endif
disp(sprintf('Le fichier contient %d images %d*%d.',
             entete(2), entete(3), entete(4)));
taille = entete(3);

images = fread(img_fic, [taille * taille, NbKL])';

fclose(img_fic);
```

Le fichier `affichage.m`, ajouté en 2005, fournit des fonctions pour afficher des vecteurs comme des images. Ces fonctions gèrent la reconstitution en 2D des images qui ont été linéarisées en 1D, ainsi que l'affichage de plusieurs d'entre elles en grille sur une seule image (plus grande, naturellement). Le code est donné en appendix.

Tous ces chiffres seront utilisés pour la recherche des composantes principales. En revanche, les observations se limiteront aux 100 premiers chiffres, présentés à la figure 1, page 3.

Afin d'appliquer une analyse en composantes principales sur ces données, il faut les centrer et les normer :

```
moyenne = mean(images);
variance = std(images) + epsilon;
images_norm = (images - repmat(moyenne, NbKL, 1)) ./
              repmat(variance, NbKL, 1);
```

`epsilon` est chargé d'éviter une division par 0 dans le cas où un pixel d'une position donnée serait de la même couleur dans toutes les images lues.

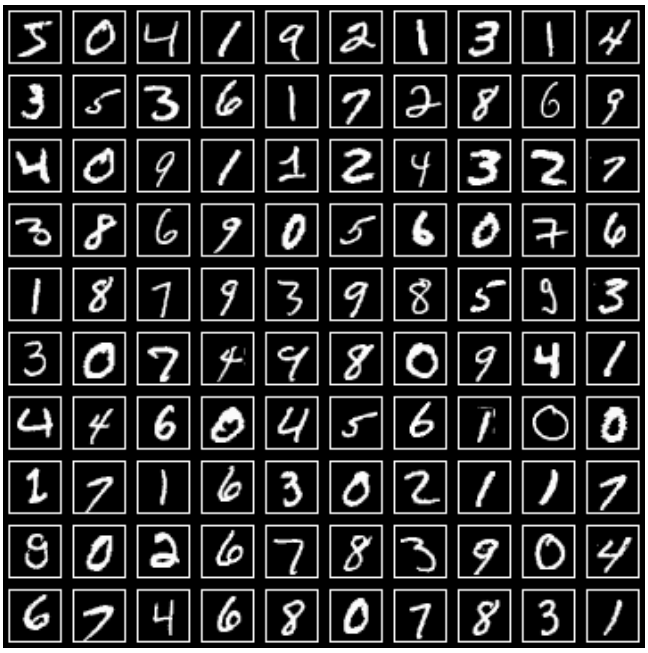


FIG. 1 – Les 100 premiers chiffres

1.2 Recherche des valeurs propres

Pour l'analyse en composantes principales, il faut calculer les valeurs propres de la matrice de covariance des données centrées normées, et l'on range les valeurs propres obtenues. On garde précieusement les indices (renvoyés par `sort` et stockés dans `indices`) des valeurs propres avant rangement afin de ranger les vecteurs propres de la même façon que les valeurs propres ont été rangées.

```
[vep, vap] = eig(cov(images_norm));
[vap_rangees, indices] = sort(-diag(vap));
```

1.3 Création de la base ordonnée de vecteurs propres

Une fois les valeurs propres rangées, on souhaite obtenir la base de vecteurs propres associée à ces valeurs propres. Cela revient à permuter les colonnes de la matrice de vecteurs propres initiaux :

```
permut = zeros(taille * taille);

for i=1:taille * taille,
  permut(indices(i), i) = 1;
```

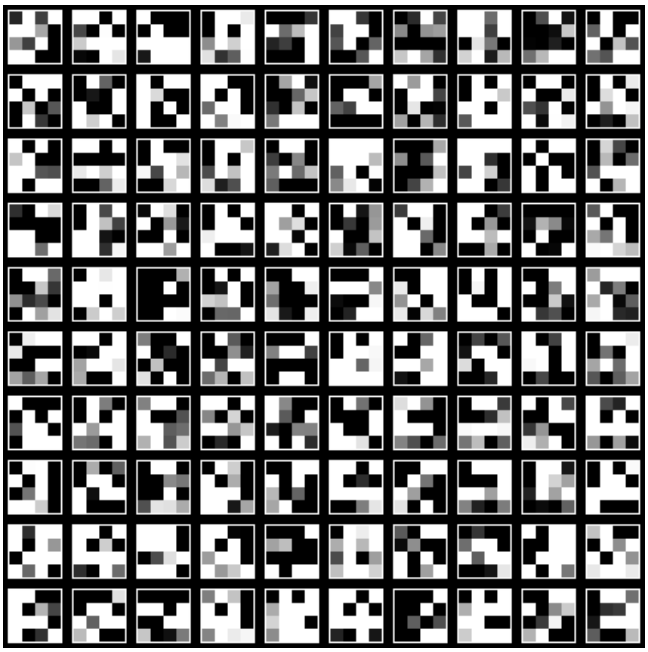


FIG. 2 – Projection des 100 chiffres sur les 16 composantes principales

```
endfor
```

La base de projection des images est maintenant obtenue par :

```
base_proj = vep * permut;
```

2 Premières applications

2.1 Projection des 100 premiers chiffres

On va projeter les images sur les `DimProj = 16` premiers vecteurs de la base ainsi obtenue. On obtient 16 valeurs, comprises entre -1 et 1, que l'on « réorganise » en petites images de 4×4. On multiplie par 128 puis on ajoute 128 pour obtenir une dynamique de 0 à 256.

Dans le script suivant, on affiche les 100 premières images originales (figure 1, page 3) ainsi que leur projection (figure 2, page 4).

```
saveimage("chiffres.ppm",Affiche_matrice(images,10),"ppm");

transformee = images_norm * base_proj;
saveimage("projetes.ppm", Affiche_matrice(128 +
```

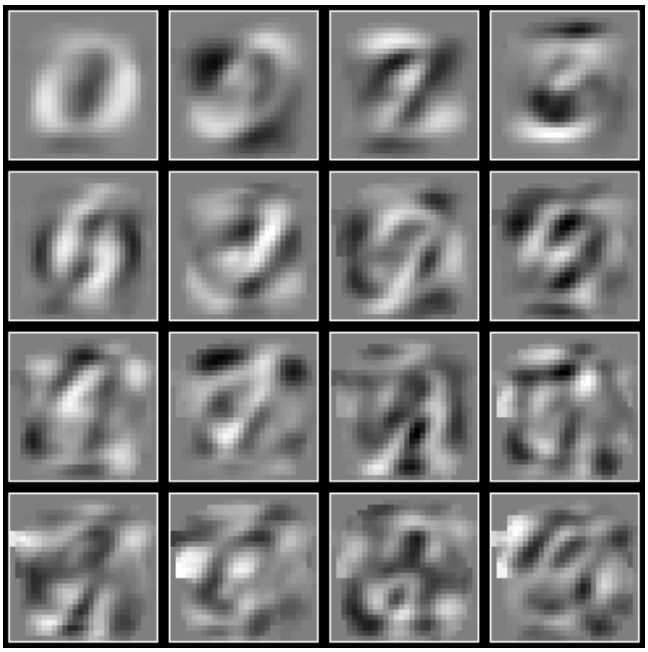


FIG. 3 – Affichage des vecteurs propres comme des images

```
    128 * transformee, 10, 10, 8, DimProj), "ppm");
```

2.2 Affichage des vecteurs propres

Un vecteur propre a la même dimension qu'une image[1]. On décide donc d'afficher les 16 premiers vecteurs comme des images, après les avoir multipliés par 128[2] et recentrés en 128. On obtient les « formes-propres » de la figure 3.

```
saveimage("propres.ppm", Affiche_matrice(128 +
          1024 * base_proj', 4, 4, 3), "ppm");
```

On obtient des formes qui sont particulièrement discriminantes pour la classification des chiffres manuscrits[3]. On retrouve un contour s'approchant d'un zéro, plusieurs diagonales, et l'on semble parfois reconnaître ici un 3, là un 7, ou un 4, un 6… Si ces formes sont correctement pondérées, on devrait ainsi

[1] À savoir `taille*taille`, mais le terme « dimension » peut être également pris dans le sens de la physique : un vecteur propre est *homogène* à une image.

[2] Comme le lecteur attentif l'aura remarqué, une multiplication par 1024 a été utilisée dans le script de façon à augmenter la dynamique entre le noir (0) et le blanc (255).

[3] Note de janvier 2005 : cette phrase est absolument fausse. À la place, on obtient des formes qui sont particulièrement *fréquentes* dans les chiffres manuscrits.

pouvoir reconstituer tous les chiffres. Nous allons maintenant nous intéresser aux chiffres moyens et à leur projeté.

2.3 Étude des moyennes pour chacun des 10 chiffres

La classe des chiffres à traiter est renseignée dans le fichier des étiquettes. Ce fichier possède un en-tête de 2 entiers de 32 bits, puis chaque étiquette est de taille 1 octet. On lit `NbKL` étiquettes en tout, correspondant aux `NbKL` images que l'on a lues :

```
etiq_fic=fopen('train-labels-idx1-ubyte','rb','ieee-be');
entete = fread(etiq_fic, 2, 'int32');
if entete(1) != 2049
  disp(sprintf('[Attention] Nombre magique : %d != %d.',
               entete(1), 2053));
endif

etiquettes = fread(etiq_fic, [NbKL, 1], 'uchar');

fclose(etiq_fic);
```

On trie ensuite les étiquettes, et l'on récupère les transitions d'un chiffre à l'autre (`etiq(i) < etiq(i + 1)`) dans le tableau trié `etiq`. Comme `ind` contient les indices des étiquettes avant le tri, on se sert de lui et des transitions pour délimiter les plages d'indices pour chaque chiffre.

```
[etiq, ind] = sort(etiquettes);

debut(1) = 1;
for i = 1:NbKL - 1,
  if etiq(i) < etiq(i + 1)
    fin(etiq(i) + 1) = i;
    debut(etiq(i + 1) + 1) = i + 1;
  endif
endfor
fin(10) = NbKL;
```

On peut maintenant accéder à des ensembles de chiffres par leur valeur. On calcule alors le chiffre moyen pour chaque classe et on le projette sur la base de vecteurs propres.

```
moy_reel = zeros(10, taille * taille);
moy_proj = zeros(10, DimProj);

for chif = 1:10,
  indices = ind(debut(chif) : fin(chif));
  moy_reel(chif, :) = mean(images(indices, :));
```

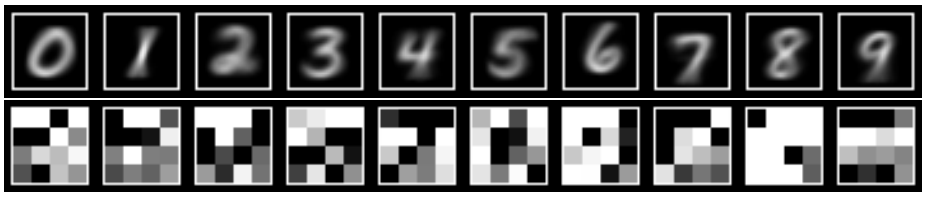


FIG. 4 – Chiffres moyens et leur projeté

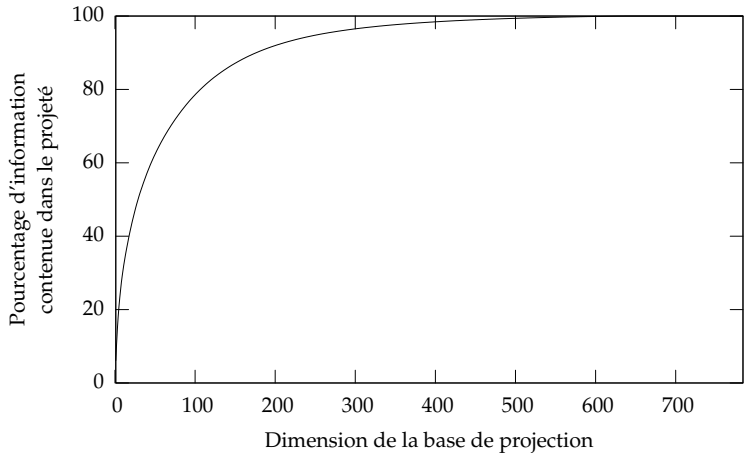
```
  moy_transf = mean(images_norm(indices, :)) * base_proj;
  moy_proj(chif, :) = moy_transf(1:DimProj);
endfor

saveimage("chif_moy.ppm", Affiche_matrice(moy_reel,
                          10, 1, 1), "ppm");
saveimage("proj_moy.ppm", Affiche_matrice(128 +
          128 * moy_proj, 10, 1, 8), "ppm");
```

Le traitement précédent permet d'obtenir une « base » des projections des chiffres moyens. Cette base permet visuellement de réaliser une classifications des chiffres : on compare la projection d'un chiffre à celles de tous les chiffres moyens. Le diagnostic se fait par ressemblance des profils de projection, le tout avec une priorité donnée aux premières composantes (lues de gauche à droite et de haut en bas sur les graphes de projection).

2.4 Problèmes à résoudre avant de classifier

L'œil humain ne peut pas comparer de façon fiable deux niveaux de gris. En revanche, un ordinateur le peut. Mais le facteur humain intervient immédiatement pour le choix de la distance entre deux images projetées : faut-il considérer les différences en valeur absolue, ou au carré ou à un ordre supérieur ? Comment pondérer les différentes composantes ? Par ailleurs, rien n'oblige à utiliser une classification du type « plus proche voisin ». On peut, comme NIST le fait, utiliser un réseau de neurones.

Pour le réseau de neurones aussi (et même *a fortiori*), la réduction du nombre de composantes est extrêmement intéressante, voire nécessaire, afin que le réseau ait une taille et un temps d'apprentissage raisonnables. C'est la détermination du nombre de composantes principales à conserver qui est au cœur de la dernière partie de ce projet.



Vecteurs conservés	Information conservée
16	38,3 %
29	50,2 %
36	54,8 %
100	78,6 %

FIG. 5 – Dimension de l'espace de projection et quantité d'information

3 Détermination de la taille de l'espace de projection

3.1 Quantité d'information conservée

Un premier élément de réponse est la quantité d'information que contiennent les 16 composantes principales. On trouve 38,3 %. De façon générale, le (court) script Octave suivant calcule le pourcentage d'information conservée en fonction du nombre de vecteurs sur lesquels on projette. On obtient la courbe présentée à la figure 5.

```
plot(100 * cumsum(vap_rangees) ./ sum(vap_rangees),";;");
axis([1,taille * taille, 0, 100]);
xlabel("Dimension de la base de projection");
ylabel('[]{\\shortstack{Pourcentage d\'information\\\\
                        contenue dans le éprojet}}');
```

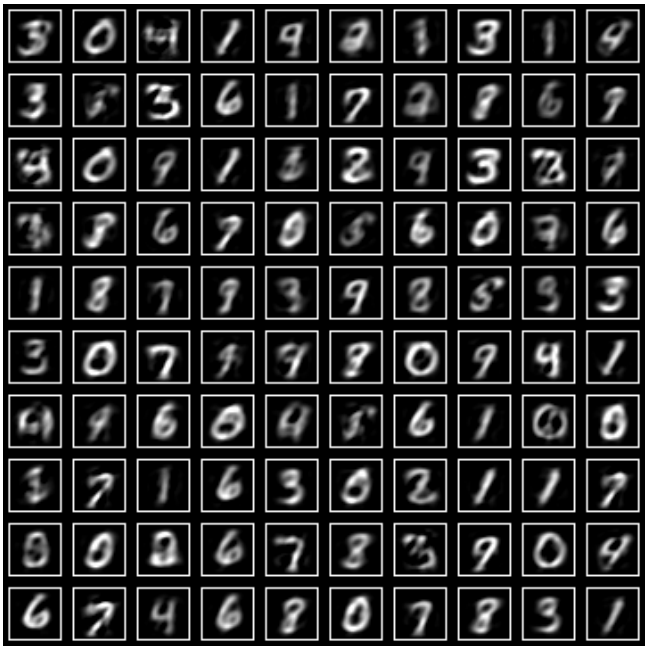


FIG. 6 – Reconstitution des chiffres à partir des 16 composantes principales

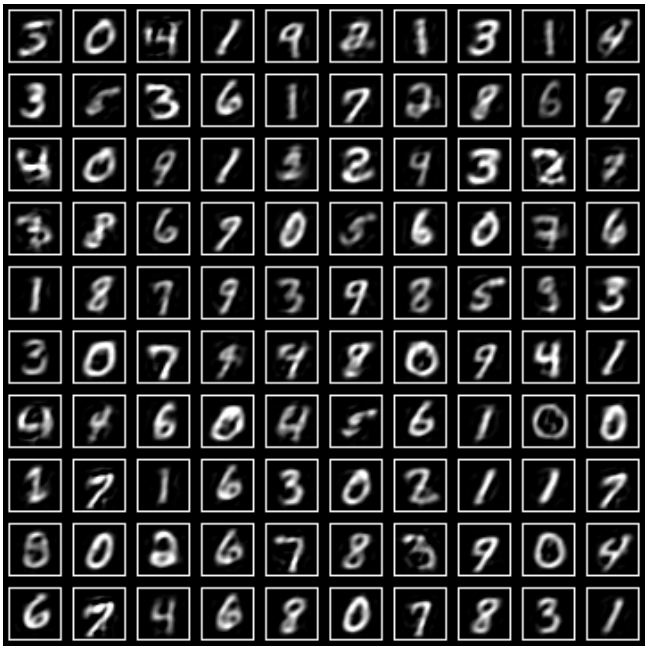


FIG. 7 – Reconstitution des chiffres à partir des 36 composantes principales

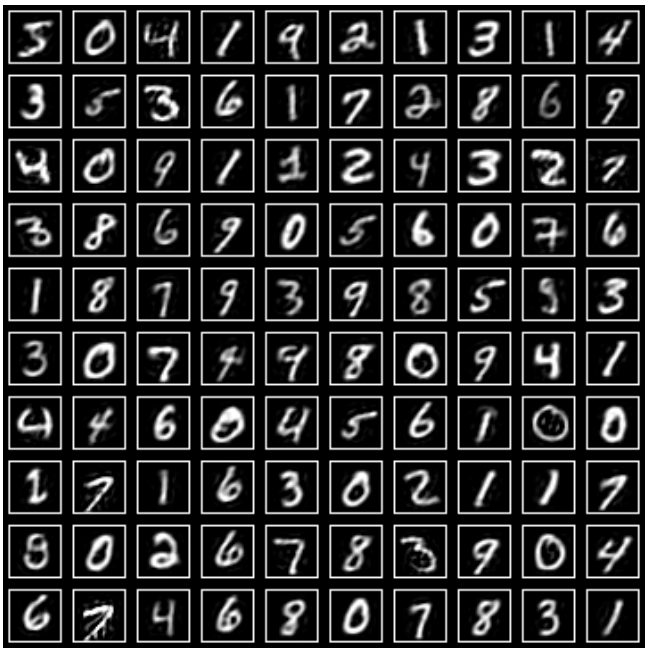


FIG. 8 – Reconstitution des chiffres à partir des 100 composantes principales

3.2 Reconstruction à partir des projetés

On considère qu'un réseau de neurones ne pourra faire mieux que le système visuel de l'homme. Cette assertion paraît raisonable : après tout, l'œil et le cerveau humains ne constituent-ils pas un réseau de neurones particulièrement entrainé et efficace ? Ainsi, plutôt que de discuter *a priori* sur la quantité d'information contenue dans la projection, on va construire des images à partir des projetés et tenter de reconnaître les chiffres.

Voici le script Octave qui permet de reconstruire et d'afficher les images à partir des projetés.

```
for DimProj = [16, 36, 100],
  saveimage(sprintf("recons_%d.ppm", DimProj),
   Affiche_matrice(repmat(variance, NbKL, 1) .*
    (base_proj * [transformee(:,1:DimProj),
                  zeros(NbKL, taille*taille - DimProj)]')'
                   + repmat(moyenne, NbKL, 1), 10),"ppm");
endfor
```

Pour un espace de projection de dimension, respectivement, 16, 36 et 100, on obtient, pour les 100 chiffres présentés page 3, les figures 6, 7 et 8.

On constate que ni 16, ni 36 composantes principales ne suffisent pour pou-

voir reconnaître tous les chiffres sans ambiguïté. En revanche, avec 100 composantes, on reconnaît les chiffres aussi bien (aussi mal, pour certains) que sur la planche originale.

Concluons par le gain (en terme de compression de l'information) que la décomposition KL permet. En ne conservant que 16 composantes, on a vu que l'on conservait 38,3 % de l'information, et l'on compressait par un ratio de $\frac{16}{784} \approx 2{,}0\%$; cependant avec aussi peu de composantes on ne reconnaît que les caractères bien écrits. Avec 36 composantes, c'était 54,8 % de l'information qui était conservée pour un ratio de compression de $\frac{36}{784} \approx 4{,}6\%$. Pour 100 composantes enfin, on conservait suffisamment d'information pour rendre tous les chiffres reconnaissables, alors que l'on compressait tout de même avec un ratio de $\frac{100}{784} \approx 12{,}8\%$.

C'est ce très bon rapport « information conservée / place mémoire occupée » qui explique que NIST a utilisé une transformation KL avant de soumettre les données issues de la projection à un traitement par réseau de neurones.

A Fonction de préparation à l'affichage

```
function foo = mon_reshape(mat, n_lig, n_col)
  foo = reshape(mat, n_col, n_lig)';
endfunction

function foo = Affiche_matrice(donnees, larg_grille,
                               haut_grille,
                               zoom, dim_proj)
  if nargin < 3                          # Par defaut:
    haut_grille = larg_grille;           # grille carree
  endif
  if nargin < 4                          # Par defaut:
    zoom = 1;                            # pas de zoom
  endif
  if nargin < 5                          # Par default:
    dim_proj = size(donnees, 2);         # affiche toutes
  endif                                  # les colonnes
  taille_image = sqrt(dim_proj);
  ## Cadre blanc de 1px, et marge noire de 3px
  w = taille_image * zoom + 8;
  foo = zeros(haut_grille * w, larg_grille * w);
  for l = 0 : haut_grille - 1,
    for c = 0 : larg_grille - 1,
      n = l * larg_grille + c;
      foo(l * w + [1:w-6] + 3, c * w + [1:w - 6] + 3) =
        255 * ones(w - 6);
```

```
      foo(l * w + [1:w-8] + 4, c * w + [1:w - 8] + 4) =
        zoom_matrice(mon_reshape(donnees(n+1,1:dim_proj),
                     taille_image, taille_image), zoom);
    endfor
  endfor
endfunction
```