TP

Implémentation de la décomposition de Tucker d'un tenseur.

Olivier Coulaud
CISD - Algorithme numérique

2022-2023

1 Rappel sur la méthode HOSVD

La méthode HOSVD construit pour chaque mode un sous espace de dimension plus petite qui permet d'approcher le tenseur par une représentation de rang faible. La décomposition ou l'approximation de Tucker un tenseur d'ordre d, $\mathcal{X} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ de rang multilinéaire $\mathbf{r} = (r_1, \ldots, r_d)$ est

$$\mathcal{X} \simeq \mathbf{U}^{(1)} \times_1 \ldots \mathbf{U}^{(d)} \times_d \mathcal{S} \tag{1}$$

avec $\mathcal{S} \in \mathbb{R}^{r_1 \times r_2 \times r_3}$ le tenseur de cœur et $\mathbf{U}^{(k)}, \in \mathbb{O}^{n_k \times r_k}$ les facteurs. L'approximation dans l'équation 1 est exacte lorsque le tenseur est de rang $r = (r_1, \ldots, r_d)$, on parle alors de décomposition.

L'algorithme 1 permet de calculer les facteurs et le tenseur de cœur de l'approximation de Tucker de la façon suivante :

Algorithme 1 : $\mathcal{S}, [\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(d)}] =$ tucker_t_hosvd$(\mathcal{X}, \mathbf{r})$ - algorithme HOSVD tronquée.

Input : Tenseur $\mathcal{X}$ et le rang multilinéaire $\mathbf{r} = (r_1, \ldots, r_d)$
Output : L'approximation $[\mathcal{G}, \{\mathbf{U}^{(n)}\}_{n=1}^{d}] =$ tucker_t_hosvd$(\mathcal{X}, \mathbf{r})$
for $k = 1$ **to** d **do**
  construire $\mathbf{X}^{(k)}$ la k-matricisation de $\mathcal{X}$
  Calculer la SVD tronquée à l'ordre r_k de $\mathbf{X}^{(k)} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$
  $\mathbf{U}^{(k)} := U[1 : r_k] := (\mathbf{u}_1, \ldots, \mathbf{u}_{r_k})$
end
Construire : $\mathcal{S} = {\mathbf{U}^{(1)}}^T \times_1 \ldots {\mathbf{U}^{(d)}}^T \times_d \mathcal{X}$
return $\mathcal{S}, [\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(d)}]$

Remarque. *La décomposition est unique à une rotation près. Considérons* $\mathcal{X} \simeq \mathbf{U}^{(1)} \times_1 \ldots \mathbf{U}^{(d)} \times_d \mathcal{S}$ *l'approximation de Tucker de* $\mathcal{X}$ *et si* $\mathbf{V_i} \in \mathbb{O}^{n_i \times n_i}$*, i.e.* ${\mathbf{V_i}}^T\mathbf{V_i} = \mathbf{V_i}{\mathbf{V_i}}^T = I_{n_i}$*, alors* $[\mathbf{V_1} \times_1 \ldots \mathbf{V_d} \times_d \mathcal{S}$*,* $\{\mathbf{U}^{(i)}{\mathbf{V_i}}^T\}_1^d]$ *est aussi une approximation de Tucker de* $\mathcal{X}$*.*

2 Mise en oeuvre

2.1 HOSVD à rang multilinéaire fixé

Écrire en python l'algorithme 1 ci-dessus pour calculer l'approximation par HOSVD d'un tenseur d'ordre d.

Pour valider l'algorithme, on générera un tenseur, $\mathcal{X}_{ref}$, de rang multilinéaire $\mathbf{r}$ et de dimension $\mathbf{n}$ à l'aide de la fonction `build_tensor` du module `tucker_tools.py`.

```
import sys
sys.path.append('/home/coulaud/tensor/tp')
import check_tucker

#ordre du tenseur
d = 3
# dimension du tenseur
n = np.asarray([8,8,8])
# rang du tenseur
r= np.asarray([4,3,2])

# Construction d'un tenseur,  X,  aléatoire d'ordre d et de rang r
X = tucker_tools.build_tensor(n, r)
```

La validation de vos résultats se fera en reconstruisant le tenseur, $\mathcal{X}$, à l'aide de la fonction `tucker_to_tensor` du package tensorly et en calculant la norme $\|\mathcal{X}_{ref} - \mathcal{X}\|/\|\mathcal{X}_{ref}\|$.

2.2 HOSVD à précision fixée

Adapter l'algorithme 1 lorsque la précision est donnée, ϵ, en entrée à la place du rang multilinéaire $\mathbf{r}$. Pour cela, il faut trouver l'entier r_k (ligne 1 de l'algorithme 1) pour que l'erreur entre la $k-$matricisation et son approximation SVD soit plus petite que $\delta = \epsilon\|\mathcal{X}\|\sqrt{d}$; i.e., avec les notations du cours $\|\mathbf{X}^{(k)} - \mathcal{T}_k(\mathbf{X})\|| = \sqrt{\sigma_{r+1}^2 + \ldots + \sigma_n^2} \leq \delta$.
On vérifiera que :

$$\|\mathcal{X} - (\mathbf{U}^1, \ldots, \mathbf{U}^d) \times \mathcal{S}\||/\|\mathcal{X}\| \leq \epsilon$$

En plus du tenseur généré de manière aléatoire, on utilisera le tenseur sur les données climatiques utilisée en cours. Pour cela, on chargera le tenseur comme suit

```
import sys
sys.path.append('/home/coulaud/tensor/tp')
import check_tucker

X = tucker_tools.load_data()
```

2.3 Approche ST-HOSVD à précision fixé

On considère la méthode ≪ sequential truncated HOSD ≫ qui consiste à calculer de manière itérative les facteurs et le tenseur de cœur de l'approximation de Yucker. L'algorithme est :

Algorithme 2 : $\mathcal{S}, [\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(d)}] = \text{tucker\_st\_hosvd}(\mathcal{X}, \epsilon)$

Input : Tenseur $\mathcal{X}$ et la précision ϵ

Output : L'approximation $[\mathcal{G}, \mathbf{U}^{(n)}]$

$\mathcal{S} := \mathcal{X}$

for $k = 1$ **to** d **do**

- Construire $\mathbf{S}^{(k)}$ la k-matricisation de $\mathcal{X}$
- Calculer la SVD de $\mathbf{X}^{(k)} = \mathbf{U\Sigma V}^T$
- Déterminer r_k tel que $\|\mathbf{X}^{(k)} - \mathbf{U}_{r_k}\mathbf{\Sigma}_{r_k}\mathbf{V}_{r_k}{}^T\| \leq \epsilon\|\mathcal{X}\|/\sqrt{d}$ où $\mathbf{U}_{r_k} := U[1 : r_k]$
- $\mathbf{U}^{(k)} := \mathbf{U}_{r_k} = (\mathbf{u}_1, \ldots, \mathbf{u}_{r_k})$
- Construire : $\mathcal{S} = \mathbf{U}_{r_k}^{(k)T} \times_k \mathcal{S}$

end

A Implémenter et valider cet algorithme.

1. Regarder l'influence de l'ordre de la boucle sur l'erreur finale $\|\mathcal{X} - (\mathbf{U}^1, \ldots, \mathbf{U}^d) \times \mathcal{S}\|$.

B Pour des tenseurs avec des dimensions très différentes ($n_1 >> n_d$ et $n_d >> n_1$) regarder l'impact sur le temps de calcul. Expliquer le choix de l'ordre des boucles.

D Comparer les deux méthodes en termes de résultats numériques, de rapidité, complexité, ...

3 Aide phyton

Le travail en python nécessite les packages suivants :
— `h5py` pour lire les données au format hdf5
— `Pillow` pour filtrer les images
— `numpy` (nd-array, svd, ...) et `scipy` (gaussian_filter)
— `tensorly` pour la manipulation des tenseurs
`conda install -c tensorly tensorly` ou `pip install -U tensorly`

Sur PlaFRIM vous pouvez directement utiliser l'environnement python via :

```
export PATH=/home/coulaud/tensor/anaconda3/bin/:$PATH
```

et le code et les données sont dans le répertoire : `/home/coulaud/tensor/tp`. Le plus simple est de positionner la variable PYTHONPATH ou dans le script mettre

```
import sys
sys.path.append('/home/coulaud/tensor/tp')
```

Opérations sur les tenseurs. Toutes les opérations sur les tenseurs doivent se faire à l'aide du package tensorly. Quelques opérations en tensorly :

1. Pour utiliser tensorly

```
import tensorly as tl
from tensorly import tenalg as tla
```

2. `unfold` correpond à la k-matricisation d'un tenseur $\mathcal{A}$
$\mathcal{A}^{(k)} \longleftrightarrow$ `tl.unfold(A, k)`
3. `mode_dot` correpond au produit sur le k-mode $\mathcal{A}$
$\mathbf{U} \times_2 \mathcal{C} \longleftrightarrow$ `core = tla.mode_dot(C, U, 2)`
4. `multi_mode_dot` correspond au produit d'une matrice sur chaque mode du tenseur
$(\mathbf{U}, \mathbf{V}, \mathbf{W}) \times \mathcal{C} \longleftrightarrow$ `core = tla.multi_mode_dot(C, [U, V, W])`
5. `tucker_to_tensor` reconstruit le tenseur dense à partir de la décomposition de Tucker
$\mathcal{X} = (\mathbf{U}, \mathbf{V}, \mathbf{W}) \times \mathcal{C} \longleftrightarrow$ `X = tl.tucker_tensor.tucker_to_tensor((C, [U, V, W]))`

hdf5 Pour lire les données hdf5, :

```
import h5py

file = '/home/cisd-tensor/tp/minst.h5'
with h5py.File(file, 'r') as hf:
      train = hf.get('train')
      X_tr = train.get('data')[:]
      y_tr = train.get('target')[:]
      test = hf.get('test')
      X_te = test.get('data')[:]
      y_te = test.get('target')[:]
```