IFT2015 Structure de donnée
Travail pratique 1

Jonathan Guymont

October 21, 2018

Question 1

(a) Voir `sparse.SparseMatrix`.

(b) La classe `SparseMatrix` encode une matrice dense A de dimension $n \times m$ dans 3 vecteurs (list Python): `rowptr`, `colind` et `data`. Le vecteur `colind` contient les indices des colonnes des valeurs non nulles. Les elements de `colind` et `data` sont données respectivement par les élements en position 2 et 3 des élements de `fromiter`

```
colind = [x[1] for x in fromiter]
data = [x[2] for x in fromiter]
```

Le vecteur `rowptr` est contruit iterativement de la façon suivante:

```
rowptr[0] = 0
for i in np.arange(1, n+1):
    rowptr[i] = rowptr[i-1] + non_zero_counter[i-1]
```

où `non_zero_counter[i-1]` est le nombre d'élements non nuls de la ligne $i - 1$ de la matrice A. Après que la matrice soit encodée, pour acceder a un élement dont la coordonnée est (i, j), on a besoin de 2 choses: le nombre d'élements non nuls dans les lignes precedentes et les colonnes correspondantes au élement non nulle de la ligne i. Le nombre d'elements non nuls dans les ligne precedentes est donne par `rowptr[i]`. Les colonnes correspondantes aux élements non nuls de la ligne i sont données par

```
# colonnes correspondant aux element non nuls de la ligne i
columns = colind[rowptr[i]:rowptr[i+1]]
```

puisque les élements dont les indices sont `rowptr[i],...,rowptr[i+1]-1` sont les élements de la ligne i. Donc si $A[i, j]$ est non nulle, alors j est dans la list `columns`. Si j est egale a `colind[`k`]` pour `rowptr[`i`]` $<= k <$ `rowptr[`i`+1]`, alors $A[i, j]$ = `data[`k`]`. Sinon, l'élement de la position (i, j) est nul, i.e. $A[i, j] = 0$. On peut maintenant retrouver la matrice dense de la facon suivante:

```
dense = [[self[i, j] for j in range(m)] for i in range(n)]
```

(c) La complexité est $O(n)$ ou n est le nombre de ligne de la matrice à encoder. Construire `colind` et `data` requiert nnz opérations de comlexité $O(1)$ (assigner une variable et l'ajouter a une liste). Construire `rowptr` requiert la construction de la liste `non_zero_counter` fait en n opérations de comlexité $O(1)$ et finalement `rowptr` requiert aussi n operations de complexité $O(1)$.

Question 2

Voir la fonction `question2()` dans `main.py`.

Question 3

Soit A une matrice dense de dimension $n \times m$ et supposons qu'on veut l'élement de la position (i, j). Dans le pire cas, le nombre d'elements non nuls de la ligne i est $m - 1$ (en supposant que si le nombre d'élements non nuls est m, $A[i, j] =$

`data[rowptr[i]+j-1]`). L'operarion dominante en terme de complexité de la methode `__getitem__((i, j))` est la recherche de l'indice de de la valeur j dans la liste des colonnes des élements non nuls de la ligne i. La methode `_find_index` effectue une recherche binaire pour trouver l'indice de j (voir `sparse.SparseMatrix._find_index`). Le nombre d'operations d'une recherche binaire est $t(m-1) = t(\lceil m/2 \rceil) + O(1)$. Par un theorem d'algorithmic, si on a la relation $t(n) = lt(n/b) + g(n)$ avec $l = b^0$ et $g(n) \in O(n^0)$, alors $t(n) \in O(n^0 \log n) = O(\log n)$.

Question 4

(a) Voir `sparse.SparseTensor` pour l'interface detaillé. Les vecteurs `colind` et `data` sont construient de la même façon que pour `SparseMatrix` , i.e.

```
colind = [x[2] for x in fromiter]
data = [x[3] for x in fromiter]
```

Le vecteur `rowptr` est construit d'une facon un peu differente. Premièrement, on utilise un indice *flat* pour le tensor: $s = i \cdot n + j$. C'est indice fait en sorte que

$$\texttt{rowptr[}s\texttt{]} = \texttt{rowptr[}s-1\texttt{]} + \#\text{elements de la ligne } j \text{ de l'image } i$$

où $i = 1, ..., 60000$ est l'indice de l'image, $n = 28$ est le nombre de lignes dans une image, et $j = 1, ..., 28$ est l'indice de la ligne. Pour recuperer un élement sachant ces coordonnées (i, j, k), on cherche $\texttt{rowptr[}s\texttt{]} \leq p < \texttt{rowptr[}s+1\texttt{]}$, où $s = i \cdot n + j$, tel que $\texttt{colind[}p\texttt{]} = k$. Si un tel p n'existe pas, on retourn 0, sinon on retourne $\texttt{data[}p\texttt{]}$.

(b) Voir `main.question4b()` .

(c) Oui, le nombre de ligne nulle dans les images MNIST (i.e. dont tous les pixels sont 0) est 495810 et le nombre total de ligne (nombre d'image $\times$ nombre de lignes par image) est 168000. Aussi, les lignes nulles son souvent regroupées ensemble au debut et a la fin des images. Il serait donc possible de rendre le vecteur `rowptr` plus compact. Les vecteurs `data` et `colind` ne peuvent pas être plus compact.

Question 5

(a) Voir `main.question5()` . NOTE: La même implémentation qu'à la question 4 est utilisée.

(b) L'espace occupé par `colind` et `data` est nnz pour les deux. L'espace occupé par `rowptr` est $l \times n$ où l est la dimension 1 du tensor (e.g. le nombre d'images dans notre example) et n est la dimension 2 du tensor (e.g. le nombre de ligne de pixel dans chaque image). L'espace total occupé est donc $2 \cdot nnz + l \cdot n$. L'espace occupé par un tensor dense est $l \cdot n \cdot m$.

(c) L'encodage de Yale n'est pas toujours plus compact. Par exemple, si tous les elements du tensor son non nuls, l'espace occupe par l'encodage de Yale est $2 \cdot l \cdot n \cdot m + l \cdot n$, ce qui est plus de 2 fois l'espace utilisé par le tensor dense.

Question 6

Soit $(i_1, ..., i_k, v)$ les coordonnées d'un élement dans un tensor de dimension k et sa valeur v. Soit $(n_1, ..., n_k)$ les dimensions de chaque coordonnées. Les vecteurs `data` et `colind` seront

```
data = [x[k] for x in fromiter]
colind = [x[k-1] for x in fromiter]
```

et l'indice s sera donné par $\sum_{j=1}^{k-2} i_j \cdot n_j + i_{k-1}$. Le reste de l'implementation est identique a l'implémentation décrite a la question 4 a).