Tenseurs

Vidéo ■ partie 15. Tenseurs

Un tenseur est un tableau à plusieurs dimensions, qui généralise la notion de matrice et de vecteur et permet de faire les calculs dans les réseaux de neurones.

1. Tenseurs (avec *numpy*)

1.1. Qu'est ce qu'un tenseur ?

Un tenseur est un tableau de nombres à plusieurs dimensions. Voici des exemples de tenseurs :

- un vecteur V est un tenseur de dimension 1 (c'est un tableau à une seule dimension),
- une matrice M est un tenseur de dimension 2 (c'est un tableau à deux dimensions),
- un 3-tenseur T est un tableau à 3 dimensions.

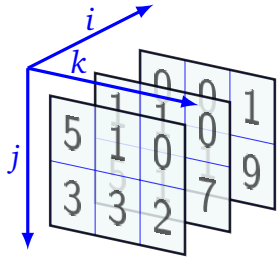
Voici un exemple pour chacune des situations avec leur définition *numpy*.

$$V = \begin{pmatrix} 5 \\ 7 \\ 8 \end{pmatrix}$$

```
V = np.array([5, 7, 8])
```

$$M = \begin{pmatrix} 6 & 0 & 2 \\ 3 & 2 & 4 \end{pmatrix}$$

```
M = np.array([[6, 0, 2],
              [3, 2, 4]])
```




```
T = np.array([ [[5, 1, 0],
                [3, 3, 2]],
               [[1, 1, 0],
                [5, 1, 7]],
               [[0, 0, 1],
                [8, 1, 9]] ])
```

On accède aux éléments en utilisant les coordonnées :

- $V[i]$,
- $M[i,j]$ (ou bien $M[i][j]$),
- $T[i,j,k]$ (ou bien $T[i][j][k]$).

Plus généralement un ***n*-tenseur** T est un tableau de nombres ayant n dimensions. On accèdent à ses éléments en précisant leurs n coordonnées $T[i_1, i_2, \ldots, i_n]$.

Il n'est pas très difficile de définir des n-tenseurs, ce sont juste des tableaux de nombres. Par contre, il est plus difficile de visualiser un tenseur à partir de la dimension 4. Une matrice peut-être vue comme une liste de vecteurs. De même, un 3-tenseur est une liste de matrices, un 4-tenseur une liste de 3-tenseurs…

Une autre façon de voir un 4-tenseur est de le présenter comme une matrice dans laquelle chaque élément est en fait une matrice :

$$T = \begin{pmatrix} M_{1,1} & M_{1,2} & \cdots \\ M_{2,1} & M_{2,2} & \cdots \\ \vdots & \vdots & \vdots \end{pmatrix}.$$

Remarque.

- La notion de tenseur avec *tensorflow* est beaucoup plus sophistiquée que celle de tenseur avec *numpy*. Nous en aurons un aperçu dans la section suivante.
- Il existe une définition plus générale (et beaucoup plus compliquée) de tenseur en mathématiques. Pour ceux qui connaissent les espaces vectoriels, la différence entre un tenseur mathématique et un tenseur comme présenté ici est la même qu'entre une application linéaire et une matrice.

1.2. Vocabulaire

Il est important de maîtriser le vocabulaire lié aux tenseurs et de ne pas confondre les notions.

- ***Dimension.***
  - C'est le nombre d'indices qu'il faut utiliser pour accéder à un élément. Chaque indice correspond à un ***axe***.
  - Un vecteur est de dimension 1, une matrice de dimension 2 (le premier axe correspond par exemple aux lignes et le second axe aux colonnes), un n-tenseur est de dimension n car on accède à un élément $T[i_1, i_2, \ldots, i_n]$ en utilisant n indices.
  - Le nom anglais est *rank* (qui n'a rien à voir avec le rang d'une matrice).
  - La commande est `T.ndim`.
- ***Taille.***
  - C'est la liste des longueurs de chaque axe.
  - Un vecteur de longueur 3 a pour taille (3) que l'on note plutôt $(3,)$ (pour signifier que c'est bien une liste). Une matrice 3×4 a pour taille $(3, 4)$. Un tenseur de taille $(3, 5, 7, 11)$ est un tenseur de dimension 4, le premier indice i_1 varie de 1 à 3 (ou plutôt de 0 à 2 avec *Python*), le second indice i_2 varie de 1 à 5…
  - Le nom anglais est *shape*.
  - La commande est `T.shape`.
- ***Nombre d'éléments.***
  - C'est le nombre total d'éléments nécessaire pour définir un tenseur. Cela correspond au nombre d'emplacements qu'il faut réserver dans la mémoire pour stocker le tenseur. Le nombre d'éléments d'un tenseur est le produit $\ell_1 \times \cdots \times \ell_n$ des longueurs de la taille $(\ell_1, \ldots, \ell_n)$ du tenseur.
  - Un vecteur de longueur 3 possède 3 éléments, une matrice 3×4 en a 12 et un 4-tenseur de taille $(3, 5, 7, 11)$ en a $3 \times 5 \times 7 \times 11$.
  - Le nom anglais est *size*.
  - La commande est `T.size`.

Voici des exemples que l'on a déjà rencontrés :

- Une image 28×28 en niveaux de gris est représentée par un tenseur de taille $(28, 28)$. Sa dimension est 2 et son nombre d'éléments est $784 = 28 \times 28$.
- Une image 32×32 en couleurs est représentée par un tenseur de taille $(32, 32, 3)$. Sa dimension est 3 et son nombre d'éléments est $3072 = 32 \times 32 \times 3$.
- Les données d'apprentissage de la base CIFAR-10 sont composées de 50 000 images couleurs de taille 32×32. L'ensemble des données est donc représenté par un 4-tenseur de taille $(50\,000, 32, 32, 3)$. Un pixel étant codé par un entier sur un octet, il faut réserver $50\,000 \times 32 \times 32 \times 3 = 153.6$ Mo en mémoire.

1.3. Opérations sur les tenseurs

Les tenseurs se comportent comme les vecteurs et les tableaux *numpy* vus dans les chapitres « Python : numpy et matplotlib avec une variable » et « Python : numpy et matplotlib avec deux variables ».
Nous reprenons les exemples définis plus haut.

Opérations sur tous les éléments.

- `V + 3` ajoute 3 à chaque élément de `V`.
- `A ** 2` calcule le carré de chaque élément de `A`.
- `T - 1` retranche 1 à chaque élément de `T`.

Opérations élément par élément entre tenseurs de même taille.

- `A * AA` calcule chaque produit $a_{ij} \times a'_{ij}$ (ici en dimension 2) et n'a rien à voir avec le produit de deux matrices.
- `T + TT` calcule chaque somme $T_{ijk} + T'_{ijk}$ (ici en dimensions 3).

Fonctions sur un tenseur.

Certaines fonction agissent élément par élément d'autre pas.

- `V.mean()` renvoie la moyenne de tous les éléments.
- `A.sum()` renvoie la somme de tous les éléments.
- `np.sqrt(T)` renvoie un tenseur, où chaque élément est obtenu par racine carrée.

1.4. Changer de taille

Changer la taille d'un tenseur tout en conservant le nombre d'éléments est un opération fréquente.

Aplatissement. C'est la mise sous la forme d'un vecteur (tenseur de dimension 1). L'instruction est `T.flatten()` qui pour notre exemple renvoie le tenseur de taille $(18,)$:

```
[5 1 0 3 3 2 1 1 0 5 1 7 0 0 1 8 1 9]
```

Changement de taille. La méthode `reshape()` transforme un tenseur en changeant sa taille. Par exemple notre tenseur `T` est de taille $(3,2,3)$ et possède 18 éléments.

- `T1 = np.reshape(T,(2,9))` renvoie un tenseur de taille $(2,9)$ contenant les éléments de `T` réorganisés.

```
[[5 1 0 3 3 2 1 1 0]
 [5 1 7 0 0 1 8 1 9]]
```

- `T2 = np.reshape(T,(2,3,3))` renvoie un tenseur de taille $(2,3,3)$.

```
[[[5 1 0]
  [3 3 2]
  [1 1 0]]
 [[5 1 7]
  [0 0 1]
  [8 1 9]]]
```

- `T3 = np.reshape(T,(9,-1))` renvoie un tenseur de taille $(9,2)$. Le « -1 » indique à *Python* qu'il doit deviner tout seul la taille à calculer (sachant qu'il y a 18 éléments une taille $(9,x)$ impose $x = 2$).

```
[[5 1]
 [0 3]
 [3 2]
 [1 1]
 [0 5]
 [1 7]
 [0 0]
 [1 8]
 [1 9]]
```

Regrouper deux tenseurs. Soient

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \qquad B = \begin{pmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}$$

définis par

```
A = np.arange(1,7).reshape((2,3))
B = np.arange(7,13).reshape((2,3))
```

Voici deux commandes illustrant le regroupement de A et B :

```
C1 = np.concatenate((A, B), axis=0)
C2 = np.concatenate((A, B), axis=1)
```

$$C_1 = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix} \qquad C_2 = \begin{pmatrix} 1 & 2 & 3 & 7 & 8 & 9 \\ 4 & 5 & 6 & 10 & 11 & 12 \end{pmatrix}$$

De façon plus générale, toutes les opérations dont vous pourriez avoir besoin sont disponibles, mais il faudra parfois se creuser la tête pour comprendre les mécanismes associés dès que la dimension est plus grande que 3.

2. Tenseurs (avec *tensorflow*/*keras*)

Avec *tensorflow*/*keras* les tenseurs sont des objets beaucoup plus puissants que les tableaux *numpy*. Il est en effet possible de manipuler des tenseurs sans définir leurs éléments. Nous ne voyons ici que quelques fonctionnalités, les plus simples.

2.1. Une case vide

Avec *Python* on ne peut pas écrire l'opération

```
y = x + 2
```

si on n'a pas auparavant affecté une valeur à la variable `x`.
Ce que l'on voudrait, c'est pouvoir manipuler `y` comme une variable ordinaire. Un autre point de vue est de dire que l'on veut définir la fonction $x \mapsto y = x + 2$. Ainsi, on ne pourrait préciser qu'en fin de programme que la variable, ici x, valait en fait 7 et donc que tout ce qui concernait y l'était finalement pour $y = 9$.
Voici comment faire avec *keras*.

```
from tensorflow.keras import backend as K
x = K.placeholder(shape=(1,))
y = x + 2
```

```
f = K.function([x],[y])
print(f([7]))
```

La fonction `placeholder()` réserve un emplacement mémoire pour un tenseur nommé x (ici un vecteur de longueur 1, donc un nombre), que l'on peut manipuler ensuite comme une variable. On définit ensuite la fonction $f : x \mapsto y = x + 2$. Ce n'est qu'à la fin, lorsque l'on calcule $f(7)$ que l'emplacement mémoire de `x` est affectée à la valeur `[7]` et que le calcul $y = x + 2$ est vraiment effectué.

Voici un autre autre exemple qui correspond au calcul $A \times X$ où A est une matrice 3×3 et X un vecteur de taille 3.

```
import numpy as np

A = K.placeholder(shape=(3,3))
X = K.placeholder(shape=(3,1))
Y = K.dot(A,X)
f = K.function([A, X], [Y])

Aval = np.arange(1,10).reshape((3,3))
Xval = np.arange(1,4).reshape((3,1))

Yval = f([Aval, Xval])
```

2.2. Gradient

On termine par une fonction de *keras* particulièrement importante pour ce cours : la fonction gradient.
Une variable.
Voici comment calculer la dérivée de la fonction $x \mapsto y = x^2$ en $x_0 = 3$.

```
x = K.constant([3])
y = x**2
grad = K.gradients(y, x)
print("Dérivée :", K.eval(grad[0]))
```

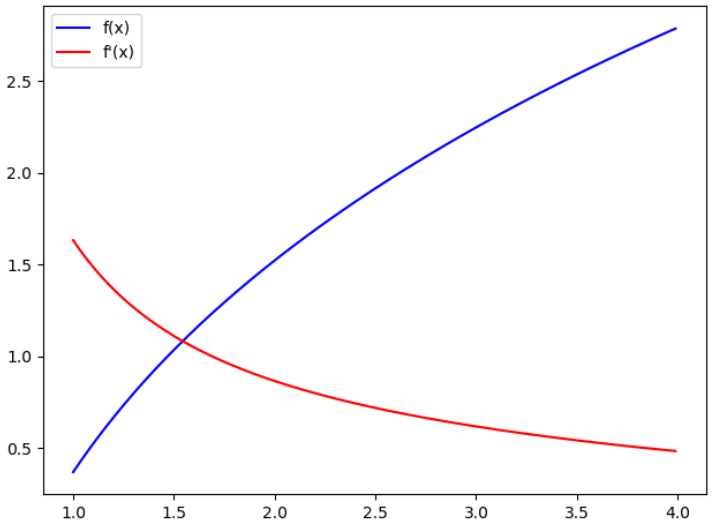
Voici comment calculer la dérivée de la fonction définie par $f(x) = 2\ln(x) + \exp(-x)$ en plusieurs points.

```
def f(x):
    return 2*K.log(x) + K.exp(-x)

X = K.arange(1,4,0.5)
Y = f(X)
grad = K.gradients(Y, X)

print("Point x :", K.eval(X))
print("Valeur y=f(x) :", K.eval(Y))
print("Dérivée f'(x) :", K.eval(grad[0]))
```

Ici les abscisses x sont les éléments de la liste $[1., 1.5, 2., 2.5, 3., 3.5]$, les valeurs calculées $y = f(x)$ sont $[0.367, 1.034, \ldots]$. Et on obtient la liste des dérivées $f'(x)$: $[1.632, 1.110, \ldots]$.
Avec plus de points, on trace le graphe de f et sa dérivée f' sur l'intervalle $[1, 4]$.



Deux variables.
Voici comment calculer le gradient de $f(x,y) = xy^2$ en $(x_0, y_0) = (2,3)$.

```
x = K.constant([2])
y = K.constant([3])
z = x * (y**2)
grad = K.gradients(z, [x,y])
dxZ = K.eval(grad[0])
dyZ = K.eval(grad[1])
print("Gradient :", dxZ, dyZ)
```

On peut ainsi tracer le gradient en chaque point du plan. Ci-dessous le gradient et les lignes de niveau de la fonction $f(x,y) = x^2 - y^2$ (un point-selle).