TP1. Les tenseurs en PyTorch.

Frédéric Richard

Cours apprentissage statistique et réseaux de neurones
Master mathématiques appliquées, statistique,
Parcours Data Science.

2025, AMU

1 PyTorch

PyTorch est une librairie très populaire en Python qui permet de faire du machine learning. Cette libraire est très documentée et on en trouve de nombreux tutoriels que l'on pourra consulter en complément de ce cours.

En PyTorch, les données sont généralement stockées dans des tenseurs, qui généralisent les matrices en dimension quelconque. Les tenseurs de PyTorch sont des objets de la class torch.Tensor.

Dans ce TP, l'objectif est de se familiariser avec les tenseurs et leur manipulation. En complément, nous vous invitons à lire le tutoriel (en anglais) Introduction to PyTorch tensors.

L'installation des librairies PyTorch peut se faire comme suit.

[ ]:
```
pip install torch torchvision
```

2 Création et importation de tenseurs.

Il est possible de créer des tenseurs de différentes manières, dont quelques unes sont illustrées ci-dessous.

[1]:
```
from torch import tensor, zeros, ones, rand, manual_seed
from torch import float64, int16

# Création d'un tenseur à la main.
A = tensor([[1, 3, 9], [2, 5, 8]])
print("A =", A)
print(A.shape)

# Création d'un tenseur de dimension 2 de taille (5, 6) contenant des 0
# dont les éléments sont des réels codés sur 64 bits.
B = zeros(5, 6, dtype=float64)
print("B =", B)
print(B.shape)
```

```
# Création d'un tenseur de dimension 3 de taille (2, 7, 8) contenant des 1
# dont les éléments sont des entiers codés sur 16 bits.
C = ones(2, 3, 4, dtype=int16)
print("C =", C)
print(C.shape)

# Création d'un tenseur de dimension 2 de taille (4, 3)
# dont les éléments sont i.i.d. de loi uniforme sur [0, 1].
manual_seed(5)
D = rand(4, 3)
print("D =", D)
print(D.shape)
```

```
A = tensor([[1, 3, 9],
        [2, 5, 8]])
torch.Size([2, 3])
B = tensor([[0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0.]], dtype=torch.float64)
torch.Size([5, 6])
C = tensor([[[1, 1, 1, 1],
         [1, 1, 1, 1],
         [1, 1, 1, 1]],

        [[1, 1, 1, 1],
         [1, 1, 1, 1],
         [1, 1, 1, 1]]], dtype=torch.int16)
torch.Size([2, 3, 4])
D = tensor([[0.8303, 0.1261, 0.9075],
        [0.8199, 0.9201, 0.1166],
        [0.1644, 0.7379, 0.0333],
        [0.9942, 0.6064, 0.5646]])
torch.Size([4, 3])
```

Par ailleurs, on peut convertir un array de numpy en tenseur de la manière suivante.

[2]:
```
from numpy import array
from torch import from_numpy

# Conversion d'un tableau numpy en tenseur.
X_numpy = array([[3, 5], [2, 1]])
print(type(X_numpy))
print(X_numpy)
X_torch = from_numpy(X_numpy)
print(type(X_torch))
print(X_torch)
```

```
<class 'numpy.ndarray'>
[[3 5]
 [2 1]]
<class 'torch.Tensor'>
tensor([[3, 5],
        [2, 1]])
```

Il est également possible d'importer une image dans un tenseur comme dans l'exemple suivant.

[9]:
```
from torchvision.io import read_image

img_torch = read_image("./roses-colorees.jpg")
print(img_torch.shape)
```

```
torch.Size([3, 880, 1353])
```

Notez au passage que l'image comporte trois dimensions, dont la première est réservée aux trois canaux de couleurs (RGB).

Une image préalablement importée pourra être convertie en un tenseur comme suit.

[18]:
```
from imageio.v2 import imread
from torchvision.transforms import ToTensor

img = imread("./roses-colorees.jpg")

img_torch = ToTensor()(img)
print(type(img_torch))
print(img_torch.shape)
```

```
<class 'torch.Tensor'>
torch.Size([3, 880, 1353])
```

3 Opérations élémentaires sur les tenseurs

Il est possible de réaliser des calculs mathématiques élémentaires sur les éléments d'un tenseur. Ces calculs se font élément par élément. En voici quelques exemples.

[3]:
```
from torch import tensor
from torch import sqrt, exp

x = tensor([[1., 2., 3.], [5., 6., 0.]])

x2 = x ** 2
xs = sqrt(x)
xe = exp(x)

print("x =", x)
print("x2 =", x2)
```

```
print("xs =", xs)
print("xe =", xe)
```

```
x = tensor([[1., 2., 3.],
        [5., 6., 0.]])
x2 = tensor([[ 1.,  4.,  9.],
        [25., 36.,  0.]])
xs = tensor([[1.0000, 1.4142, 1.7321],
        [2.2361, 2.4495, 0.0000]])
xe = tensor([[  2.7183,   7.3891,  20.0855],
        [148.4132, 403.4288,   1.0000]])
```

Questions

1. Que produisent les méthodes ** 2, sqrt et exp ?
2. Compléter les opérations ci-dessus en calculant le tenseur dont les termes valent le cosinus des termes de x.

PyTorch prend en charge toutes les opérations arithmétiques élémentaires (addition, soustraction, multiplication, division) sur les tenseurs. Ces opérations peuvent être effectuées élément par élément lorsque les formes des tenseurs sont compatibles. En voici quelques exemples.

[4]:
```
from torch import tensor

# Création de deux tenseurs
x = tensor([[1, 2, 3], [5, 6, 0]])
y = tensor([[4, 5, 6], [2, 1, 4]])
print("x =", x)
print("y =", y)

# Addition élément par élément
somme = x + y
print("x + y =", somme)

# Multiplication élément par élément
multi = x * y
print("x * y =", multi)

# Division élément par élément
divis = x / y
print("x / y =", divis)
```

```
x = tensor([[1, 2, 3],
        [5, 6, 0]])
y = tensor([[4, 5, 6],
        [2, 1, 4]])
x + y = tensor([[5, 7, 9],
        [7, 7, 4]])
x * y = tensor([[ 4, 10, 18],
```

```
        [10,  6,  0]])
x / y = tensor([[0.2500, 0.4000, 0.5000],
        [2.5000, 6.0000, 0.0000]])
```

En principe, ces opérations se réalisent sur des tenseurs de même taille. Toutefois, il est possible de les réaliser sur des tenseurs de tailles différentes. PyTorch interprète alors les opérations en faisant du "broadcasting". Pour commencer à se familiariser avec le broadcasting, analysons les exemples suivants.

[5]:
```
from torch import tensor

# Création de deux tenseurs
x = tensor([[1, 2, 3], [5, 6, 0]])
y = tensor([4, 5, 6])
print("x =", x)
print("taille: ", x.shape)
print("y =", y)
print("taille: ", y.shape)

somme = x + y
print("x + y", somme)

multi = x * y
print("x * y =", multi)

divis = x / y
print("x / y =", divis)
```

```
x = tensor([[1, 2, 3],
        [5, 6, 0]])
taille:  torch.Size([2, 3])
y = tensor([4, 5, 6])
taille:  torch.Size([3])
x + y tensor([[ 5,  7,  9],
        [ 9, 11,  6]])
x * y = tensor([[ 4, 10, 18],
        [20, 30,  0]])
x / y = tensor([[0.2500, 0.4000, 0.5000],
        [1.2500, 1.2000, 0.0000]])
```

Question

Analyser ces exemples et interpréter le broadcasting utilisé pour réaliser ces opérations.

Pour une compréhension des règles du broadcasting, on pourra consulter Broadcasting semantics ou Broadcastinh.

3.1 5. Statistiques élémentaires sur les tenseurs.

Des commandes de pytorch permettent de faire des calculs de statistiques élémentaires sur les élements d'un tenseur. Ces calculs peuvent se faire selon des dimensions.

Certaines de ces opérations sont illustrées sur l'exemple suivant.

[32]:
```
from torch import tensor, sum, mean

X = tensor([[1., 2., 3.], [4., 5., 6.]])

y1 = sum(X)
print("y1 = ", y1)

y2 = sum(X, dim=0)
print("y2 = ", y2)

y3 = sum(X, dim=1)
print("y3 = :", y3)

ym = mean(X)
print("ym = ", ym)
```

```
y1 =  tensor(21.)
y2 =  tensor([5., 7., 9.])
y3 = : tensor([ 6., 15.])
ym =  tensor(3.5000)
```

Questions

1. Quelles sont les différences entre y1, y2 et y3 ? A quoi sert l'option dim ?
2. A quoi correspond ym ?
3. Compléter ces opérations en calculant

- le minimum de X selon la première dimension,
- le maximum de X selon la seconde dimension,
- la moyenne de X selon la première dimension,
- l'écart-type de tous les éléments de X.

3.2 6. Manipulations de la forme des tenseurs.

Ci-dessous sont illustrés quelques opérations essentielles que l'on peut opérer sur les tenseurs.

[46]:
```
from torch import rand, tensor
from torch import reshape, cat, permute, squeeze, unsqueeze

# Manipulation 1.

U = tensor([[1, 2], [3, 4], [5, 6]])
Ur = reshape(U, (2, 3))
```

```
print("U =", U)
print("Ur =", Ur)

# Manipulation 2.

X = tensor([[1, 2], [3, 4]])
Y = tensor([[5, 6], [7, 8]])
Z1 = cat((X, Y), dim=0)
Z2 = cat((X, Y), dim=1)

print("X = ", X)
print("Y = ", Y)
print("Z1 = ", Z1)
print("Z2 = ", Z2)


# Manipulation 3.

V = rand(2, 3, 5)
Vp = permute(V, (2, 0, 1))

print(V.shape)
print(Vp.shape)


# Manipulation 4.

W = rand(3)
Ws = unsqueeze(W, dim=0)
Wu = squeeze(W, dim=0)

print("W =", W)
print(W.shape)
print("Ws =", Ws)
print(Ws.shape)
print("Wu =", Wu)
print(Wu.shape)
```

```
U = tensor([[1, 2],
        [3, 4],
        [5, 6]])
Ur = tensor([[1, 2, 3],
        [4, 5, 6]])
X =  tensor([[1, 2],
        [3, 4]])
Y =  tensor([[5, 6],
```

```
        [7, 8]])
Z1 =  tensor([[1, 2],
        [3, 4],
        [5, 6],
        [7, 8]])
Z2 =  tensor([[1, 2, 5, 6],
        [3, 4, 7, 8]])
torch.Size([2, 3, 5])
torch.Size([5, 2, 3])
W = tensor([0.0803, 0.3134, 0.1028])
torch.Size([3])
Ws = tensor([[0.0803, 0.3134, 0.1028]])
torch.Size([1, 3])
Wu = tensor([0.0803, 0.3134, 0.1028])
torch.Size([3])
```

Questions

1. A quoi correspondent les manipulations de forme 1, 2, 3 et 4 ?
2. Quel est le rôle de l'option dim dans les méthodes cat, squeeze et unsqueeze ?

Exercice

1. Importer l'image couleur "roses-colorees.jpg" dans un tenseur.
2. Mettre les canaux couleurs de cette image dans la dernière dimension du tenseur.
3. Convertir l'image en array de numpy.
4. Afficher l'image avec la commande imshow de la méthode pyplot de matplot lib.
5. Calculer le niveau de gris moyen dans chaque canal de couleur.
6. Calculer l'écart-type des niveaux de gris dans chaque canal de couleur.

4 Simulation et estimation dans un modèle linéaire.

On se place dans le cadre d'un modèle linéaire

$$Y_i = \theta_0 + \sum_{j=1}^{p} \theta_j x_{ij} + \epsilon_i, i = 1, \cdots, n,$$

où les θ_j sont des paramètres de régression, les x_{ij} des variables de régression et les ϵ_i des variables aléatoires i.i.d. de loi gaussienne centrée de variance σ^2. Ce modèle peut également s'écrire sous une forme matricielle

$$Y = X\theta + \epsilon,$$

où Y et ϵ sont des vecteurs de taille n, X une matrice de design de taille $n \times (p+1)$ dont la première colonne vaut est une colonne de 1 et θ est un vecteur de taille $p+1$.

Exercice.

1. Ecrire une fonction en python qui, étant donnés n et p, génère aléatoirement une matrice de design X et un vecteur de paramètres θ sous la forme de tenseur PyTorch.

2. A l'aide de la fonction précédente, générer une matrice X et un vecteur θ pour $p = 1000$ et $n = 10000$. Puis, pour $\sigma^2 = 1$, générer un jeu d'observations aléatoires Y à partir de X et θ.

3. A l'aide de la commande *solve* du module **torch.linalg**, faire l'estimation $\hat{\theta}$ des paramètres θ par maximum de vraisemblance.

4. Comparer l'estimation $\hat{\theta}$ à la vraie valeur des paramètres θ avec la norme euclidienne et la norme 1.