université PARIS-SACLAY

MODULE 1 : STRUCTURE DE DONNÉES PYTORCH

Agro–IODAA–Semestre 1

Vincent Guigue

Calcul matriciel

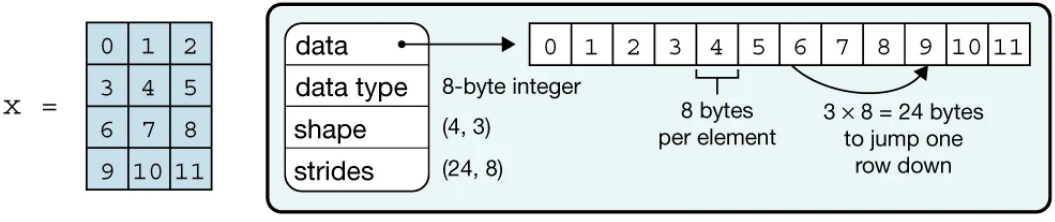
Matrices & philosophie générale

Pourquoi utiliser des matrices?

- Matrice = idéale pour stocker un ensemble de données
  (e.g. ligne=individu, colonne=descripteur)
- Type des valeurs de base
  - Les matrices sont typées (bon à savoir)
  - `torch.int16`, `torch.int16` $\Rightarrow$ Le plus souvent, on cherche à gagner de la place!
- Philosophie = **opérateurs de haut niveau** sur les matrices $\neq$ boucles
- Toutes les structures sont `torch`
  - Passage plus facile sur GPU
  - Calcul de gradient

Quelques fonctions pour créer des matrices

```
1  import torch
2  # creer une matrice de 4 lignes & 3 colonnes
3  cst = torch.tensor([[3.1, 2.7, 1.9], [1.6, 0.0, 9.8], [1.7, 8.1, 5.
4  zeros = torch.zeros(4, 3)
5  ones = torch.ones(4, 3)
6  x = torch.rand(4, 3)
```

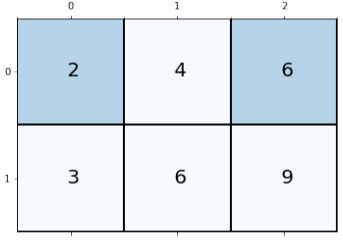



```
1  zeros_like_x = torch.zeros_like(x)
```

Accéder aux valeurs

Penser aux motifs plutôt qu'aux boucles:

```
1      x = torch.tensor([[2, 4, 6], [3, 6, 9]])
2      print(x[0][0], " or ", x[0,0])  # note that each cell is a tensor i
3      print(x[:,0])                   # all rows, first column
4      print(x[0,[0,2]])               # first row, column 0 and 2
5
6      # general syntax on rows or column  start:end (excluded):step
7      print(x[0,0:2], " or ", x[0,:2])    # upper bound
8      print(x[0,1:3], " or ", x[0,1:])    # lower bound
9      print(x[0,0:3:2], " or ", x[0,::2]) # step = 2
```



Calcul entre matrices

De nombreux opéprateurs disponibles:

$+, -, *, /, **$

```
1  # avec des scalaires
2  x = torch.tensor([[2, 4, 6], [3, 6, 9]])
3  y = x+2 # [[4, 6, 8], [5, 8, 11]]
4
5  # entre matrice (terme a terme)
6  z = x * y # [[8, 24, 48], [15, 48, 99]]
7
8  # produit matriciel
9  a = x @ torch.tensor([[1], [2], [3]])
```

Inplace (ou pas)

Selon les situations, pour gagner de la place:

- Opération classique & stockage dans un nouveau vecteur

```
1  b = torch.sin(a)
2  c = a+b
```

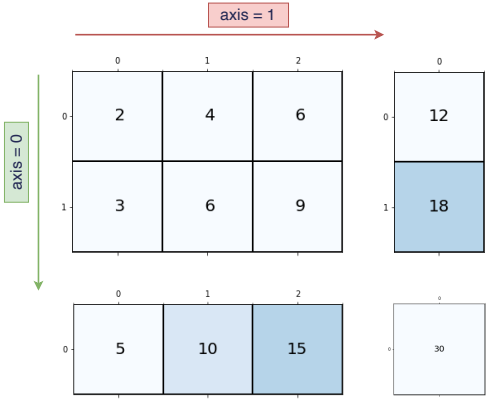
- Opération *inline* & modification de l'argument

```
1  torch.sin_(a) # _ => modification de l'argument (a)
2  a.add_(b)     # => modification de l'objet a
```

Calcul agrégatifs dans matrice

```python
x   = torch.tensor([[2, 4, 6], [3, 6, 9]])
li  = x.sum(1) # ou x.sum(axis=1)
co  = x.sum(0)
tot = x.sum()
# somme des colonnes 0 et 2
li  = x[:,::2].sum(1)
```

- somme, moyenne, min, max, ...



squeeze / unsqueeze

- Ligne de données = 1d; Image = 2d; corpus d'image = 3d
- Donner une image en entrée du système ⇒ `unsqueeze`
  Dimension homogène à un batch
- Récupérer (& analyser) une sortie intermédiaire ⇒ `squeeze`
  batch ⇒ individu

```
1  a = torch.rand(3, 226, 226)
2  b = a.unsqueeze(0)
3
4  print(a.shape) # torch.Size([3, 226, 226])
5  print(b.shape) # torch.Size([1, 3, 226, 226])
```