

PyTorch

-

Bibliothèque de « Deep Learning »

# Résumé des ingrédients du « Deep Learning »

1) Grande base de données étiquetées

# Résumé des ingrédients du « Deep Learning »

1) Grande base de données étiquetées

2) « Bonne » architecture de réseau de neurones profond

- ▶ « Perceptron » multicouche, Réseau de neurones à convolution, Transformer
- ▶ Optimisation par descente de gradient stochastique (AdamW, etc.)

# Résumé des ingrédients du « Deep Learning »

1) Grande base de données étiquetées

2) « Bonne » architecture de réseau de neurones profond

- ▶ « Perceptron » multicouche, Réseau de neurones à convolution, Transformer
- ▶ Optimisation par descente de gradient stochastique (AdamW, etc.)

3) Grande capacité de calculs en parallèle (GPU)

# Résumé des ingrédients du « Deep Learning »

1) Grande base de données étiquetées

2) « Bonne » architecture de réseau de neurones profond

- ▶ « Perceptron » multicouche, Réseau de neurones à convolution, Transformer
- ▶ Optimisation par descente de gradient stochastique (AdamW, etc.)

3) Grande capacité de calculs en parallèle (GPU)

→ calculs sur GPU de manière transparente

**Besoins**

# Résumé des ingrédients du « Deep Learning »

1) Grande base de données étiquetées

2) « Bonne » architecture de réseau de neurones profond

- ▶ « Perceptron » multicouche, Réseau de neurones à convolution, Transformer
- ▶ Optimisation par descente de gradient **stochastique** (AdamW, etc.)

3) Grande capacité de calculs en parallèle (GPU)

Besoins

- calculs sur GPU de manière transparente
- génération de minibatches sur CPUs

# Résumé des ingrédients du « Deep Learning »

1) Grande base de données étiquetées

2) « Bonne » architecture de réseau de neurones profond

- « Perceptron » multicouche, Réseau de neurones à convolution, Transformer
- Optimisation par descente de **gradient** stochastique (AdamW, etc.)

3) Grande capacité de calculs en parallèle (GPU)

## Besoins

- calculs sur GPU de manière transparente
- génération de minibatches sur CPUs
- « Autograd » : calcul automatique du gradient (rétropropagation)

# Résumé des ingrédients du « Deep Learning »

1) Grande base de données étiquetées

2) « Bonne » architecture de réseau de neurones profond

- ▶ « Perceptron » multicouche, Réseau de neurones à convolution, Transformer
- ▶ Optimisation par descente de gradient stochastique (AdamW, etc.)

3) Grande capacité de calculs en parallèle (GPU)

## Besoins

- calculs sur GPU de manière transparente
- génération de minibatches sur CPUs
- « Autograd » : calcul automatique du gradient (rétropropagation)
- couches classiques (FC, ReLU, Convolution, Attention, etc.)



# Bibliothèques



# PyTorch

Numpy sur GPU

```
# Create a numpy array.  
x = np.array([[1, 2], [3, 4]])  
  
# Convert the numpy array to a torch tensor.  
y = torch.from_numpy(x)  
  
# Convert the torch tensor to a numpy array.  
z = y.numpy()
```

# PyTorch

Numpy sur GPU

```
# Create a numpy array.  
x = np.array([[1, 2], [3, 4]])  
  
# Convert the numpy array to a torch tensor.  
y = torch.from_numpy(x)  
  
# Convert the torch tensor to a numpy array.  
z = y.numpy()
```

Autograd

```
# Create tensors.  
x = torch.tensor(1., requires_grad=True)  
w = torch.tensor(2., requires_grad=True)  
b = torch.tensor(3., requires_grad=True)  
  
# Build a computational graph.  
y = w * x + b      # y = 2 * x + 3  
  
# Compute gradients.  
y.backward()  
  
# Print out the gradients.  
print(x.grad)      # x.grad = 2  
print(w.grad)      # w.grad = 1  
print(b.grad)      # b.grad = 1
```

# PyTorch : Chargement de données (« Dataloader »)

## Définition du Dataset

Objet dont la méthode `def __getitem__(self, idx):` doit charger et renvoyer la donnée numéro idx (et son étiquette)

# PyTorch : Chargement de données (« Dataloader »)

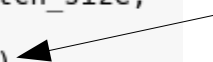
## Définition du Dataset

Objet dont la méthode `def __getitem__(self, idx):` doit charger et renvoyer la donnée numéro idx (et son étiquette)

## Définition du Dataloader

```
train_loader = t.utils.data.DataLoader(dataset=train_set,  
                                       batch_size=batch_size,  
                                       shuffle=True,  
                                       num_workers=2)
```

Nombre de processus qui vont préparer des minibatches en parallèle sur CPU(s).



# PyTorch : Chargement de données (« Dataloader »)

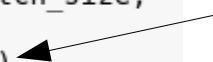
## Définition du Dataset

Objet dont la méthode `def __getitem__(self, idx):` doit charger et renvoyer la donnée numéro idx (et son étiquette)

## Définition du Dataloader

```
train_loader = t.utils.data.DataLoader(dataset=train_set,  
                                       batch_size=batch_size,  
                                       shuffle=True,  
                                       num_workers=2)
```

Nombre de processus qui vont préparer des minibatches en parallèle sur CPU(s).



## Définition du GPU

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

# PyTorch : Chargement de données (« Dataloader »)

## Définition du Dataset

Objet dont la méthode `def __getitem__(self, idx):` doit charger et renvoyer la donnée numéro idx (et son étiquette)

## Définition du Dataloader

```
train_loader = t.utils.data.DataLoader(dataset=train_set,  
                                       batch_size=batch_size,  
                                       shuffle=True,  
                                       num_workers=2)
```

Nombre de processus qui vont préparer des minibatches en parallèle sur CPU(s).

## Définition du GPU

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

## Boucle principale d'apprentissage

```
for epoch in range(num_epochs):  
    for i, (images, labels) in enumerate(train_loader):  
        images = images.to(device)  
        labels = labels.to(device)  
        ...
```

Mise à disposition d'un minibatch

Transfert du minibatch au GPU

# TensorBoard

-

## Outil de visualisation



# Visualisations au cours d'un apprentissage

Lors d'un apprentissage, il est indispensable de réaliser de nombreux affichages, à minima :

- Coût d'apprentissage
- Coût/Performances de validation
- Pas d'apprentissage
- Exemples de résultats

# Visualisations au cours d'un apprentissage

Lors d'un apprentissage, il est indispensable de réaliser de nombreux affichages, à minima :

- Coût d'apprentissage
- Coût/Performances de validation
- Pas d'apprentissage
- Exemples de résultats

Il est important de pouvoir :

- Visualiser ces affichages **au cours** de l'apprentissage
- Sauvegarder ces affichages

# Visualisations au cours d'un apprentissage

Lors d'un apprentissage, il est indispensable de réaliser de nombreux affichages, à minima :

- Coût d'apprentissage
- Coût/Performances de validation
- Pas d'apprentissage
- Exemples de résultats

Il est important de pouvoir :

- Visualiser ces affichages **au cours** de l'apprentissage
- Sauvegarder ces affichages

En pratique, on lance souvent plusieurs apprentissages simultanément (par exemple avec différents valeurs d'hyperparamètres, ou différentes variantes d'architectures) **sur un serveur à distance**.

# TensorBoard

À mettre dans son script d'entraînement :

Instanciation d'un objet TensorBoard (crée un fichier log dans ./runs/)

```
from torch.utils.tensorboard import SummaryWriter  
writer = SummaryWriter()
```

# TensorBoard

À mettre dans son script d'entraînement :

Instanciation d'un objet TensorBoard (crée un fichier log dans ./runs/)

```
from torch.utils.tensorboard import SummaryWriter  
writer = SummaryWriter()
```

Ajout d'un point à une courbe

```
writer.add_scalar("Loss/train", loss, epoch)
```

Nom courbe

Valeur ordonnée

Valeur abscisse

# TensorBoard : accès aux logs

Lancer TensorBoard depuis un terminal :

```
tensorboard --logdir=runs
```

# TensorBoard : accès aux logs

Lancer TensorBoard depuis un terminal :

```
tensorboard --logdir=runs
```

Dans un navigateur aller à :

<http://localhost:6006/>

