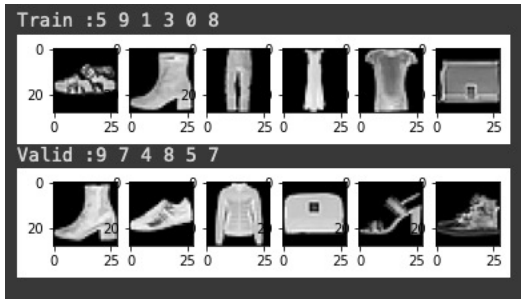
Séance de TP 3 –
Réseaux Convolutifs – GPU et Tensorboard

Sujet

Au cours de cette séance nous allons nous intéresser à l'optimisation d'un modèle de classification de type Réseau Convolutif pour la classification d'images.
Nous utiliserons les datasets MNIST et Fashion-MNIST



- Nous configurerons pytorch pour fonctionner sur GPU et comparerons les temps d'exécution GPU et CPU
- Nous instancierons différentes architectures de CNN sur les bases MNIST et Fashion-MNIST
- Nous configurerons TensorBoard

Séance de TP 3 – Réseaux Convolutifs – GPU et Tensorboard

1- Configurer pytorch sur GPU

Voici les différentes étapes nécessaires pour configurer un apprentissage sur GPU avec Pytorch

1. Tester la présence et identifier le GPU

Selon qu'une GPU est présente ou non, on définit le paramètre device

```
# Test disponibilité du GPU
if torch.cuda.is_available() :
  device = torch.device("cuda:0")
  print(" Proceseur disponible:",device,"device name :",torch.cuda.get_device_name(0))
else:
  device = torch.device("cpu")
  print("Processeur disponible:",device)
```

2. Déclarer un modèle en mémoire GPU

Une fois instancié un modèle sur CPU on le transfert sur la device. Si c'est CPU il ne se passe rien, sinon le modèle est transféré sur GPU

```
my_MLP = MLP(input_features = dim)
my_MLP.to(device)
```

Séance de TP 3 – Réseaux Convolutifs – GPU et Tensorboard

3. Charger les datasets en mémoire GPU

L'entrainement d'un modèle ne peut se faire sur GPU que si les données d'apprentissage et de validation sont présentes sur la mémoire GPU avec le modèle. Pour cela on modifie les boucles d'apprentissage, de validation et de test pour transférer les batch sur la GPU au moment où on les utilise.

Apprentissage

```
for batch, (X, y) in tqdm(enumerate(dataloader)):
  X,y = X.to(device),y.to(device)
  # Compute prediction and loss
  pred = model(X.float())
  loss = loss_fn(pred, y)
  epoch_loss += loss.item()
……
```

Validation

```
with torch.no_grad():
  for X, y in dataloader:
    X, y = X.to(device), y.to(device)
    pred = model(X.float())
    valid_loss += loss_fn(pred, y).item()
……
```

Test

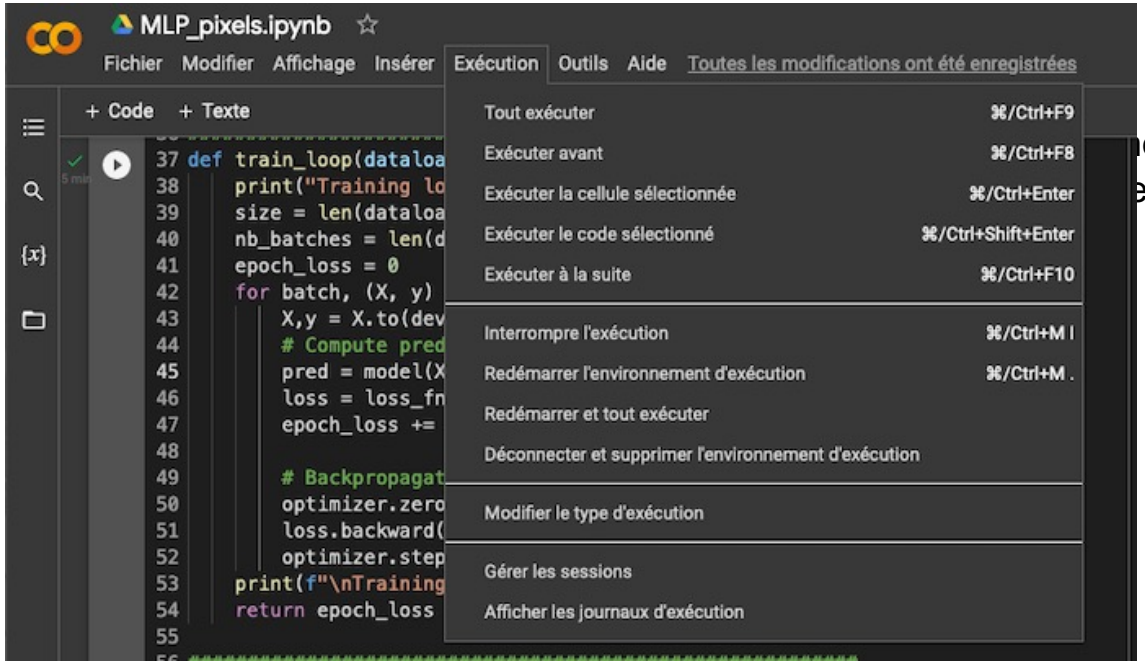
```
for X, y in dataloader:
  X, y = X.to(device), y.to(device)
  pred = model(X.float())
  prob_pred = nn.Softmax(dim=1)(pred) # pas nécessaire pour prédire le meilleur
  y_pred = torch.argmax(prob_pred,dim=1)
  #similaires = np.array(y_pred==y)
  similaires = torch.tensor(y_pred == y)
  #Positifs += np.sum(similaires)
  Positifs += torch.sum(similaires)
return Positifs.cpu().numpy(), Total, y_pred.cpu().numpy()
```

Séance de TP 3 – Réseaux Convolutifs – GPU et Tensorboard

4. Configurer Colab pour demander l'accès à une GPU disponible

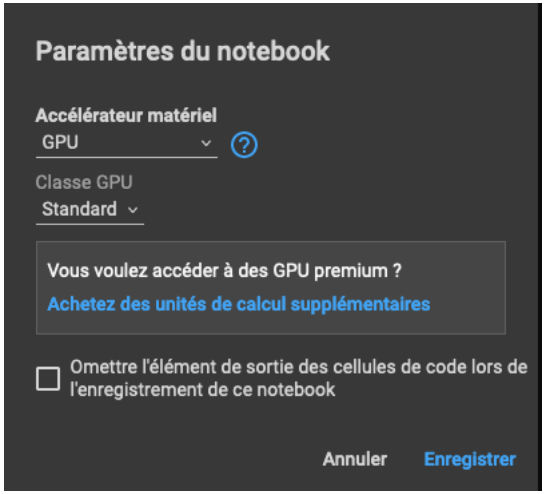
Descendre dans le menu Exécution

puis sélectionner « Modifier le type d'exécution»



oisir l'accélérateur

el



- Lancer l'apprentissage d'un MLP sur GPU pour la meilleure configuration obtenue lors de la séance précédente et relever le nombre de batch traités par seconde et le comparer à la version CPU

Séance de TP 3 – Réseaux Convolutifs – GPU et Tensorboard

2- Instancier un premier réseau convolutif pour la reconnaissance des chiffres de la base NIST

```
class CNN(nn.Module):
  def __init__(self):
      super(CNN, self).__init__()
      # premier bloc conv ReLU Pooling à 16 canaux
      self.conv1 = nn.Sequential(
      nn.Conv2d(in_channels=1,out_channels=16,kernel_size=3,stride=1,padding=1),nn.ReLU(),nn.MaxPool2
      d(kernel_size=2))
      # second bloc conv ReLU Pooling à 32 canaux sur carte de taille 7 x 7 car 2 MaxPooling
      self.conv2 = nn.Sequential(nn.Conv2d(16, 32, 3, 1, 1),nn.ReLU(),nn.MaxPool2d(2),nn.Flatten())
      # fully connected layer, output 10 classes
      self.last_layer = nn.Linear(32 * 7 * 7, 10)
        def forward(self, x):
      x = self.conv1(x)
      x = self.conv2(x)
      return output
```

- ✓ Relever les performances et le temps d'apprentissage sur GPU par rapport au meilleur MLP obtenu lors de la séance précédente

Séance de TP 3 – Réseaux Convolutifs – GPU et Tensorboard

3- Utilisation de TensorBoard

Tensorboard est un outil de visualisation en ligne initialement conçu pour TensorFlow, il a été ensuite intégré à Pytorch. Il permet de visualiser les variables au cours du déroulement du programme.

Son instanciation est réalisée par les lignes suivantes

```
from torch.utils.tensorboard import SummaryWriter

# Initialisation du writer de TensorBoard:
writer = SummaryWriter()

train_epoch_loss = train_loop(trainloader, my_MLP, my_loss, my_optimizer,device)
# On ajoute la loss moyenne d'entrainement de l'epoch aux logs Tensorboard:
writer.add_scalar('Loss/train', train_epoch_loss, t)

valid_epoch_loss = valid_loop(validloader, my_MLP, my_loss,device)
# On ajoute la loss moyenne de validation de l'epoch aux logs Tensorboard:
writer.add_scalar('Loss/valid', valid_epoch_loss, t)
```

Séance de TP 3 –
Réseaux Convolutifs – GPU et Tensorboard

3- Utilisation de TensorBoard

- Utiliser Tensorboard pour visualiser les différentes informations logguées durant l'entraînement précédent
  - ✓ Courbes de loss en entraînement et validation (onglet « scalars »)
  - ✓ Echantillons d'images du dataset (onglet « images »)
  - ✓ Graph de l'architecture du modèle (onglet « graphs »). Quelles sont les dimensions d'entrée de la dernière couche du modèle ?
  - ✓ Projection des données dans la représentation apprise par le modèle (onglet « projector »). Que voyez vous ? A quoi cela correspond-il ?
  - ✓ Onglet « Hparams » : contient un tableau les différentes expériences effectuées avec les hyperparamètres et les métriques associées

- Modifiez le code pour logguer dans TensorBoard la précision sur le jeu de validation obtenue au fil de l'entraînement

Séance de TP 3 – Réseaux Convolutifs – GPU et Tensorboard

4- Envisager la reconnaissance des images du dataset FashionMNIST

La dataset FashionMNIST comporte des images de taille 28 x 28 pixels en niveaux de gris qui se distribuent sur 10 classes. L'ensemble d'apprentissage comporte 60 000 exemples et l'ensemble de test 10 000.
Les datasets d'apprentisage et de test sont chargés avec les instructions suivantes:

```
from torchvision.datasets import FashionMNIST

dataset = FashionMNIST(os.getcwd(), train = True, download=True, transform=transforms.ToTensor())

test_dataset = FashionMNIST(os.getcwd(), train = False, download=True, transform=transforms.ToTensor())
```

- Envisager différentes architectures de réseaux convolutifs pour attendre les meilleures performances possibles sur la base FashionMNIST. (utilisez l'onglet « Hparams » de TensorBoard pour lister les résultats facilement)