Deep Learning:

Lab2:

REALISE PAR:

DAMIATI KAOUTAR

Part 1: CNN Classifier:

jupyter lab2_part1 Dernière Sauvegarde : mercredi dernier à 13:57 (modifié)

Logout

File Edit View Insert Cell Kernel Widgets Help Non fiable Python 3 (ipykernel)

Exécuter Code

Entrée [1]:

```python
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.optim as optim
```

Entrée [3]:

```python
# Transformer les données
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# Téléchargement des données d'entraînement
trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)

# Téléchargement des données de test
testset = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=False)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data\MNIST\raw\train-images-idx3-ubyte.gz

100%|████████████████████████████████████████| 9912422/9912422 [00:23<00:00, 413328.20it/s]

Extracting ./data\MNIST\raw\train-images-idx3-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data\MNIST\raw\train-labels-idx1-ubyte.gz

100%|██████████████████████████████████████████| 28881/28881 [00:00<00:00, 656423.27it/s]

Extracting ./data\MNIST\raw\train-labels-idx1-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data\MNIST\raw\t10k-images-idx3-ubyte.gz

100%|███████████████████████████████████████| 1648877/1648877 [00:01<00:00, 1196987.58it/s]

Extracting ./data\MNIST\raw\t10k-images-idx3-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data\MNIST\raw\t10k-labels-idx1-ubyte.gz

100%|███████████████████████████████████████████| 4542/4542 [00:00<00:00, 4618310.00it/s]

Extracting ./data\MNIST\raw\t10k-labels-idx1-ubyte.gz to ./data\MNIST\raw
```

Entrée [11]:
```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3)
        self.conv2 = nn.Conv2d(32, 64, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 5 * 5, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        print("Input shape:", x.shape)
        x = self.pool(nn.functional.relu(self.conv1(x)))
        print("Shape after conv1 and pool:", x.shape)
        x = self.pool(nn.functional.relu(self.conv2(x)))
        print("Shape after conv2 and pool:", x.shape)
        x = x.view(-1, 64 * 5 * 5)
        print("Shape after flattening:", x.shape)
        x = nn.functional.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = CNN()
```

Entrée [12]:
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Entrée [13]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device)

num_epochs = 5

for epoch in range(num_epochs):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data[0].to(device), data[1].to(device)

        optimizer.zero_grad()

        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if i % 100 == 99:
            print(f"Epoch [{epoch + 1}/{num_epochs}], Batch [{i + 1}/{len(trainloader)}], Loss: {running_loss / 100:.3f}")
            running_loss = 0.0
```

```
Input shape: torch.Size([64, 1, 28, 28])
Shape after conv1 and pool: torch.Size([64, 32, 13, 13])
Shape after conv2 and pool: torch.Size([64, 64, 5, 5])
Shape after flattening: torch.Size([64, 1600])
Input shape: torch.Size([64, 1, 28, 28])
Shape after conv1 and pool: torch.Size([64, 32, 13, 13])
Shape after conv2 and pool: torch.Size([64, 64, 5, 5])
Shape after flattening: torch.Size([64, 1600])
Input shape: torch.Size([64, 1, 28, 28])
Shape after conv1 and pool: torch.Size([64, 32, 13, 13])
Shape after conv2 and pool: torch.Size([64, 64, 5, 5])
Shape after flattening: torch.Size([64, 1600])
Input shape: torch.Size([64, 1, 28, 28])
Shape after conv1 and pool: torch.Size([64, 32, 13, 13])
Shape after conv2 and pool: torch.Size([64, 64, 5, 5])
Shape after flattening: torch.Size([64, 1600])
Input shape: torch.Size([64, 1, 28, 28])
Shape after conv1 and pool: torch.Size([64, 32, 13, 13])
Shape after conv2 and pool: torch.Size([64, 64, 5, 5])
```

Entrée [14]:
```python
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data[0].to(device), data[1].to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = correct / total
print(f"Accuracy on test set: {100 * accuracy:.2f}%")
```

```
Input shape: torch.Size([64, 1, 28, 28])
Shape after conv1 and pool: torch.Size([64, 32, 13, 13])
Shape after conv2 and pool: torch.Size([64, 64, 5, 5])
Shape after flattening: torch.Size([64, 1600])
Input shape: torch.Size([64, 1, 28, 28])
Shape after conv1 and pool: torch.Size([64, 32, 13, 13])
Shape after conv2 and pool: torch.Size([64, 64, 5, 5])
Shape after flattening: torch.Size([64, 1600])
Input shape: torch.Size([64, 1, 28, 28])
Shape after conv1 and pool: torch.Size([64, 32, 13, 13])
Shape after conv2 and pool: torch.Size([64, 64, 5, 5])
Shape after flattening: torch.Size([64, 1600])
Input shape: torch.Size([64, 1, 28, 28])
Shape after conv1 and pool: torch.Size([64, 32, 13, 13])
Shape after conv2 and pool: torch.Size([64, 64, 5, 5])
Shape after flattening: torch.Size([64, 1600])
Input shape: torch.Size([64, 1, 28, 28])
Shape after conv1 and pool: torch.Size([64, 32, 13, 13])
Shape after conv2 and pool: torch.Size([64, 64, 5, 5])
```

Part 2: Vision Transformer (VIT) :

Entrée [9]:
```python
# importing the libraries
import torchvision.transforms as transforms
from torchvision.datasets import MNIST
from torch.utils.data import DataLoader
import torch
from torchvision.models.detection import fasterrcnn_resnet50_fpn
import torch.optim as optim
import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
```

Entrée [10]:
```python
# transformations to be applied on images
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# defining the training and testing set
trainset = MNIST('./data', download=True, train=True, transform=transform)
testset = MNIST('./data', download=True, train=False, transform=transform)

# defining trainloader and testloader
trainloader = DataLoader(trainset, batch_size=64, shuffle=True)
testloader = DataLoader(testset, batch_size=64, shuffle=False)
```

Entrée [11]:

```
from torchvision.models.detection import FasterRCNN
from torchvision.models.detection.rpn import AnchorGenerator

# Charger le modèle Faster R-CNN sans les poids pré-entraînés
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=False)

# Modifier la couche de sortie pour correspondre au nombre de classes dans MNIST
num_classes = 10  # MNIST a 10 classes
# Obtenir le nombre d'entrées de la couche de classification
in_features = model.roi_heads.box_predictor.cls_score.in_features
# Modifier la couche de classification pour le nombre de classes souhaité
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

# Vous pouvez ensuite charger les poids pré-entraînés si nécessaire avec :
# model.load_state_dict(torch.load('votre_modele.pth'))

# Vérifier si CUDA est disponible et déplacer le modèle sur le GPU si possible
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

Out[11]:

```
FasterRCNN(
  (transform): GeneralizedRCNNTransform(
      Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      Resize(min_size=(800,), max_size=1333, mode='bilinear')
  )
  (backbone): BackboneWithFPN(
    (body): IntermediateLayerGetter(
      (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
      (bn1): FrozenBatchNorm2d(64, eps=1e-05)
      (relu): ReLU(inplace=True)
      (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
      (layer1): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(64, eps=1e-05)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(64, eps=1e-05)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(256, eps=1e-05)
```

Entrée [19]:

```
import torch.nn as nn
import os
import tensorflow as tf
optimizer = optim.SGD(model.parameters(), lr=0.005, momentum=0.9)
# Définir une fonction de perte (entropie croisée) pour la classification
criterion = nn.CrossEntropyLoss()

# Boucle d'entraînement
num_epochs = 10
for epoch in range(num_epochs):
    for images, targets in trainloader:
        images = list(image.to(device) for image in images)

        # Traiter les cibles pour correspondre aux attentes du modèle Faster R-CNN
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]
        for t in targets:
            t['labels'] = t.pop('class_labels')
            t['boxes'] = t.pop('bounding_box')

        optimizer.zero_grad()
        # Obtenir les prédictions du modèle
        loss_dict = model(images, targets)

        # Calculer la perte de classification
        classification_loss = criterion(loss_dict['class_logits'], loss_dict['labels'])

        # Calculer la perte totale
        loss = classification_loss
        loss.backward()
        optimizer.step()

    # Affichage de la perte à chaque époque ou autre métrique souhaitée
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item()}")
```

```
: pip install tensorflow
```

```
: model.eval()
total_correct = 0
total_images = 0

for images, labels in testloader:
    images = list(image.to(device) for image in images)
    labels = [{k: v.to(device) for k, v in t.items()} for t in labels]

    with torch.no_grad():
        outputs = model(images)
        # Gérer les sorties du modèle comme requis pour l'évaluation

accuracy = total_correct / total_images
print(f"Accuracy on test set: {accuracy * 100:.2f}%")
```