TD : Classification d'images MNIST avec des réseaux de neurones convolutionnels (CNN) en PyTorch

Master GBM - Apprentissage statistique

April 3, 2025

1 Introduction

Dans ce TD, nous allons implémenter un réseau de neurones convolutif (CNN) pour classifier les chiffres manuscrits de la base de données MNIST. Cette base contient 60 000 images d'entraînement et 10 000 images de test, chacune représentant un chiffre manuscrit de 0 à 9 en niveaux de gris de taille 28×28 pixels.

2 Préparation des données

```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torchvision
from torchvision import transforms
from torch.utils.data import DataLoader

# D finition des transformations
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))  # Normalisation entre -1
    et 1
])

# Chargement des donn es MNIST
train_dataset = torchvision.datasets.MNIST(root='./data', train=
    True,
                                           transform=transform,
    download=True)
test_dataset = torchvision.datasets.MNIST(root='./data', train=
    False,
                                          transform=transform,
    download=True)

# Cr ation des dataloaders
```

```
batch_size = 64
train_loader = DataLoader(train_dataset, batch_size=batch_size,
    shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size,
    shuffle=False)

# Affichage de quelques exemples
examples = iter(train_loader)
samples, labels = next(examples)
plt.figure(figsize=(10, 5))
for i in range(9):
    plt.subplot(3, 3, i+1)
    plt.imshow(samples[i][0], cmap='gray')
    plt.title(f"Chiffre: {labels[i]}")
    plt.axis('off')
plt.tight_layout()
plt.show()

print("Forme des donn es d'entra nement:", samples.shape)
print("Nombre de classes:", len(train_dataset.classes))
```

3 Construction d'un CNN simple

```
import torch.nn as nn
import torch.nn.functional as F

class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        # Premi re couche de convolution + pooling
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1,
    padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        # Deuxi me couche de convolution
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1,
    padding=1)

        # Couches enti rement connect es
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        # Premi re couche conv + activation + pooling
        x = self.pool(F.relu(self.conv1(x)))

        # Deuxi me couche conv + activation + pooling
        x = self.pool(F.relu(self.conv2(x)))

        # Aplatissement des caract ristiques
        x = x.view(-1, 64 * 7 * 7)

        # Couche enti rement connect e avec dropout
        x = F.relu(self.fc1(x))
```

```
31
32          # Couche de sortie
33          x = self.fc2(x)
34          return x
35
36  # Initialisation du mod le
37  device = torch.device("cuda:0" if torch.cuda.is_available() else "
        cpu")
38  model = SimpleCNN().to(device)
39  print(model)
```

4 Entraînement du modèle

```
1   import torch.optim as optim
2   import time
3
4   # D finition de la fonction de perte et de l'optimiseur
5   criterion = nn.CrossEntropyLoss()
6   optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
7
8   # Fonction pour calculer la pr cision
9   def calculate_accuracy(model, data_loader, device):
10      model.eval()
11      correct = 0
12      total = 0
13      with torch.no_grad():
14          for inputs, labels in data_loader:
15              inputs, labels = inputs.to(device), labels.to(device)
16              outputs = model(inputs)
17              _, predicted = torch.max(outputs.data, 1)
18              total += labels.size(0)
19              correct += (predicted == labels).sum().item()
20      return 100 * correct / total
21
22  # Entra nement du mod le
23  num_epochs = 10
24  train_losses = []
25  train_accuracies = []
26  val_accuracies = []
27
28  start_time = time.time()
29  for epoch in range(num_epochs):
30      model.train()
31      running_loss = 0.0
32      for i, (inputs, labels) in enumerate(train_loader):
33          inputs, labels = inputs.to(device), labels.to(device)
34
35          # Mise    z ro des gradients
36          optimizer.zero_grad()
37
38          # Forward + backward + optimize
39          outputs = model(inputs)
40          loss = criterion(outputs, labels)
41          loss.backward()
42          optimizer.step()
```

```
43
44          running_loss += loss.item()
45
46          if (i+1) % 100 == 0:
47              print(f'Epoch [{epoch+1}/{num_epochs}], Batch [{i+1}/{
    len(train_loader)}], Loss: {loss.item():.4f}')
48
49      # Calcul de la perte moyenne sur l' poque
50      epoch_loss = running_loss / len(train_loader)
51      train_losses.append(epoch_loss)
52
53      # Calcul de la pr cision sur les donn es d'entra nement et
    de validation
54      train_acc = calculate_accuracy(model, train_loader, device)
55      val_acc = calculate_accuracy(model, test_loader, device)
56      train_accuracies.append(train_acc)
57      val_accuracies.append(val_acc)
58
59      print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f},
     Train Acc: {train_acc:.2f}%, Val Acc: {val_acc:.2f}%')
60
61  print(f'Temps d\'entra nement: {time.time() - start_time:.2f}
    secondes')
62
63  # Visualisation de l'apprentissage
64  plt.figure(figsize=(12, 4))
65
66  # Courbe de perte
67  plt.subplot(1, 2, 1)
68  plt.plot(train_losses)
69  plt.title('Perte d\'entra nement')
70  plt.xlabel(' poque ')
71  plt.ylabel('Perte')
72
73  # Courbe de pr cision
74  plt.subplot(1, 2, 2)
75  plt.plot(train_accuracies, label='Entra nement')
76  plt.plot(val_accuracies, label='Validation')
77  plt.title('Pr cision du mod le')
78  plt.xlabel(' poque ')
79  plt.ylabel('Pr cision (%)')
80  plt.legend()
81
82  plt.tight_layout()
83  plt.show()
```

5 Prédictions et visualisation

```
1  # Faire des pr dictions sur quelques exemples
2  model.eval()
3  with torch.no_grad():
4      # R cup ration d'un batch de test
5      dataiter = iter(test_loader)
6      images, labels = next(dataiter)
7
```

```
    # S lection des 9 premi res images
    images = images[:9].to(device)
    labels = labels[:9]

    # Pr dictions
    outputs = model(images)
    _, predicted = torch.max(outputs, 1)

    # Affichage des images et des pr dictions
    plt.figure(figsize=(12, 9))
    for i in range(9):
        plt.subplot(3, 3, i+1)
        plt.imshow(images[i][0].cpu().numpy(), cmap='gray')
        plt.title(f"Pr dit: {predicted[i].item()}, R el: {labels[
    i].item()}")
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# Sauvegarde du mod le
torch.save(model.state_dict(), 'mnist_cnn.pth')
print("Mod le sauvegard  avec succ s!")
```

6 Exercices

6.1 Modification de l'architecture

Ajoutez une couche de convolution supplémentaire et observez l'impact sur les performances.

```
class EnhancedCNN(nn.Module):
    def __init__(self):
        super(EnhancedCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
    # Nouvelle couche
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(128 * 3 * 3, 128)
        self.fc2 = nn.Linear(128, 10)


    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 128 * 3 * 3)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

6.2 Expérimentation avec les hyperparamètres

Modifiez le taux d'apprentissage, la taille des batchs ou la fonction d'activation et comparez les résultats.

```
# Exemple de modification des hyperparam tres
learning_rates = [0.001, 0.01, 0.1]
batch_sizes = [32, 64, 128]
optimizers = [
    optim.SGD(model.parameters(), lr=0.01, momentum=0.9),
    optim.Adam(model.parameters(), lr=0.001),
    optim.RMSprop(model.parameters(), lr=0.001)
]

# Vous pouvez cr er une boucle pour tester diff rentes
    combinaisons
```

6.3 Augmentation des données

Implémentez des techniques d'augmentation de données pour améliorer la robustesse du modèle.

```
# Transformations avec augmentation de donn es
transform_augmented = transforms.Compose([
    transforms.RandomRotation(10),
    transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# Cr ation d'un nouveau dataset avec augmentation
train_dataset_augmented = torchvision.datasets.MNIST(
    root='./data', train=True, transform=transform_augmented,
    download=True)
train_loader_augmented = DataLoader(
    train_dataset_augmented, batch_size=batch_size, shuffle=True)
```

6.4 Comparaison avec un réseau entièrement connecté

Implémentez un réseau de neurones entièrement connecté (sans couches convolutives) et comparez ses performances avec le CNN.

```
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 10)

    def forward(self, x):
        x = x.view(-1, 28 * 28)  # Aplatissement de l'image
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
```

```
14          x = self.fc3(x)
15          return x
```