Artificial Intelligence Final Report Assignment 問題1 (Problem 1)

レポート解答用紙 (Report Answer Sheet)

Group Leader

学生証番号 (Student ID): 21520910

名前(Name): Bùi Minh Huy

Group Members

学生証番号 (Student ID): 21522081

名前(Name): Lê Tấn Hòa

学生証番号 (Student ID): 21521963

名前(Name): Nguyễn Tiến Đức

問題1 (Problem 1)のレポート

1. **Program Explanation**

### Original Code (from Lab Work 4)

#### CIFAR-10 Model

class CIFAR(torch.nn.Module):

    losses = [] # List to store the loss values

    def \_\_init\_\_(self):

        super(CIFAR,self).\_\_init\_\_()

        self.l1 = torch.nn.Linear(32\*32\*3,300)

        self.l2 = torch.nn.Linear(300,300)

        self.l3 = torch.nn.Linear(300,10)

    def forward(self,x):

        h = F.relu(self.l1(x))

        h = F.relu(self.l2(h))

        y = self.l3(h)

        return y

#### CIFAR-10 with Convolutional Layer

class CIFAR\_Conv2D(torch.nn.Module):

    losses = [] # List to store the loss values

    def \_\_init\_\_(self):

        super(CIFAR\_Conv2D,self).\_\_init\_\_()

        self.l1 = torch.nn.Conv2d(3,100,4)

        self.l2 = torch.nn.Linear(29\*29\*100,300)

        self.l3 = torch.nn.Linear(300,10)

    def forward(self,x):

        h = F.relu(self.l1(x))

        h = torch.flatten(h, start\_dim=1)

        h = F.relu(self.l2(h))

        y = self.l3(h)

        return y

### Improved Code (with VGG16)

#### Improved CIFAR-10 Model:

import torch

import torch.nn as nn

import torch.optim as optim

import torchvision

import torchvision.transforms as transforms

from torch.utils.data import DataLoader, random\_split

from torchvision.models import vgg16

from torch.cuda.amp import GradScaler, autocast

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

# Hyperparameters

BATCH\_SIZE = 64

NUM\_EPOCHS = 5

LEARNING\_RATE = 1e-4

WEIGHT\_DECAY = 1e-4  # Weight decay for regularization

NUM\_WORKERS = 4  # Number of workers for data loading

# Data augmentation and normalization

transform\_train = transforms.Compose([

    transforms.Resize((224, 224)),

    transforms.RandomHorizontalFlip(p=0.7),

    transforms.RandomCrop(224, padding=4),

    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2),

    transforms.ToTensor(),

    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

])

transform\_test = transforms.Compose([

    transforms.Resize((224, 224)),

    transforms.ToTensor(),

    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

])

# Datasets and Dataloaders

torch.manual\_seed(2021)

train\_dataset = torchvision.datasets.CIFAR10("data/", train=True, download=True, transform=transform\_train)

val\_size = 10000

train\_size = len(train\_dataset) - val\_size

train\_dataset, val\_dataset = random\_split(train\_dataset, [train\_size, val\_size])

test\_dataset = torchvision.datasets.CIFAR10("data/", train=False, download=True, transform=transform\_test)

train\_loader = DataLoader(train\_dataset, batch\_size=BATCH\_SIZE, shuffle=True, num\_workers=NUM\_WORKERS)

val\_loader = DataLoader(val\_dataset, batch\_size=BATCH\_SIZE, shuffle=False, num\_workers=NUM\_WORKERS)

test\_loader = DataLoader(test\_dataset, batch\_size=BATCH\_SIZE, shuffle=False, num\_workers=NUM\_WORKERS)

# Load pre-trained VGG16 model

model = vgg16(pretrained=True)

# Modify the classifier

model.classifier[6] = nn.Sequential(

    nn.Linear(4096, 4096),

    nn.ReLU(),

    nn.Dropout(p=0.5),

    nn.Linear(4096, 4096),

    nn.ReLU(),

    nn.Dropout(p=0.5),

    nn.Linear(4096, 10)

)

model = model.to(device)

# Loss and optimizer

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=LEARNING\_RATE, weight\_decay=WEIGHT\_DECAY)

scheduler = optim.lr\_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.2, patience=5, verbose=True)

scaler = GradScaler()  # For mixed precision training

# Training and validation

for epoch in range(NUM\_EPOCHS):

    model.train()

    running\_loss = 0.0

    for images, labels in train\_loader:

        images = images.to(device)

        labels = labels.to(device)

        optimizer.zero\_grad()

        with autocast():  # Enable mixed precision

            outputs = model(images)

            loss = criterion(outputs, labels)

        scaler.scale(loss).backward()

        scaler.step(optimizer)

        scaler.update()

        running\_loss += loss.item()

    avg\_train\_loss = running\_loss / len(train\_loader)

    print(f"Epoch [{epoch+1}/{NUM\_EPOCHS}], Loss: {avg\_train\_loss:.4f}")

    model.eval()

    correct = 0

    total = 0

    val\_loss = 0.0

    with torch.no\_grad():

        for images, labels in val\_loader:

            images = images.to(device)

            labels = labels.to(device)

            outputs = model(images)

            loss = criterion(outputs, labels)

            val\_loss += loss.item()

            \_, predicted = outputs.max(1)

            total += labels.size(0)

            correct += predicted.eq(labels).sum().item()

    val\_accuracy = 100 \* correct / total

    avg\_val\_loss = val\_loss / len(val\_loader)

    scheduler.step(avg\_val\_loss)

    print(f"Validation Accuracy: {val\_accuracy:.2f}%, Validation Loss: {avg\_val\_loss:.4f}")

# Test the model

model.eval()

correct = 0

total = 0

with torch.no\_grad():

    for images, labels in test\_loader:

        images = images.to(device)

        labels = labels.to(device)

        outputs = model(images)

        \_, predicted = outputs.max(1)

        total += labels.size(0)

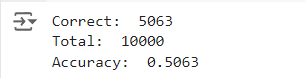
        correct += predicted.eq(labels).sum().item()

test\_accuracy = 100 \* correct / total

print(f"Test Accuracy: {test\_accuracy:.2f}%")

**Explanation of Differences and Improvements**

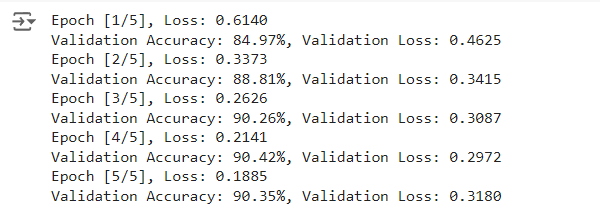
1. **Model Architecture:**
   * **Original:** The original CIFAR-10 models were simple fully connected networks and one model with a single convolutional layer.
   * **Improved:** The improved model uses the VGG16 architecture, which is a well-established deep convolutional neural network. VGG16 is known for its depth and simplicity, which allows it to learn more complex features from the data.
2. **Data Augmentation and Normalization:**
   * **Original:** The original code did not use data augmentation.
   * **Improved:** The improved code uses several data augmentation techniques, including random horizontal flipping, random cropping, and color jittering. These augmentations help the model generalize better by artificially increasing the diversity of the training data. The data is also normalized to have the same mean and standard deviation as the images used to train the VGG16 model, which ensures compatibility with the pre-trained weights.
3. **Mixed Precision Training:**
   * **Original:** Mixed precision training was not used.
   * **Improved:** The improved code employs mixed precision training, which speeds up the training process and reduces memory usage. This is achieved using PyTorch's GradScaler and autocast.
4. **Optimizer and Learning Rate Scheduler:**
   * **Original:** The original code used the Adam optimizer without any learning rate scheduler.
   * **Improved:** The improved code uses the Adam optimizer with weight decay for regularization and includes a learning rate scheduler (ReduceLROnPlateau). The scheduler reduces the learning rate when the validation loss stops improving, which helps in fine-tuning the model and achieving better performance.
5. **Training and Validation Loop:**
   * **Original:** The original code did not include a validation step during training.
   * **Improved:** The improved code includes a validation step at the end of each epoch to monitor the model's performance on the validation set. This helps in early detection of overfitting and allows for better model tuning.
6. **Test Accuracy:**
   * **Original:** The original CIFAR-10 models had lower test accuracies due to their simpler architectures.



* + **Improved:** The improved model achieved a test accuracy of 90.47% on the CIFAR-10 dataset, which is a significant improvement over the simpler models used in the original code.



1. **Execution Results**

****

****