The AI report of lab2

Zhenzhan Zhao,231223999

2024/11/5

Preparations Before the Experiment

Step 1: Complete the environment setup in Jupyter Notebook

```
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import os
import torch
import torch.optim as optim
```

Step 2: Import the experimental dataset

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
data_path = './data'
if not os.path.exists(data_path):
    os.makedirs(data_path)
try:
    dataset = datasets.CIFAR10(root=data_path, train=True, download=True, transform=transform)
except Exception as e:
    print(f"error: {e}")
from torch.utils.data import random_split
```

Begin Experiment

Part1: Individualized Setup

The unique configurations I used is based on my QMUL ID

```
# Set the student's last digit of the ID (replace with your own last digit)
last_digit_of_id = 9  # Example: Replace this with the last digit of your QMUL ID
# Define the split ratio based on QMUL ID
split_ratio = 0.7 if last_digit_of_id <= 4 else 0.8
```

My last digit QMUL ID is 9, so I changed the last_digit_of_id to 9.

Based on the last_digit_of_id, the model uses the following unique configurations:

Dataset Split: The training and validation split ratio is **80%:20%**. This means 80% of the data is used for training, and 20% is used for validation.

Number of Epochs: The number of training epochs is **10 + 9 = 19 epochs**.

Learning Rate: The learning rate is set to 0.001 plus the last digit of the ID multiplied by 0.0001, giving a rate of **0.001 + (9 × 0.0001) = 0.0019**.

Batch Size: The batch size is **32 + 9 = 41**.

```
# Split the dataset
train_size = int(split_ratio * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = random_split(dataset,
[train_size, val_size])
# DataLoaders
from torch.utils.data import DataLoader
batch_size = 32 + last_digit_of_id  # Batch size is 32 + last digit of your QMUL ID
train_loader = DataLoader(train_dataset,
batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset,
batch_size=batch_size, shuffle=False)
print(f"Training on {train_size} images, Validating on {val_size} images.")
# Define the model
model = torch.nn.Sequential(
torch.nn.Flatten(),
torch.nn.Linear(32*32*3, 512),
torch.nn.ReLU(),
torch.nn.Linear(512, 10), # 10 output classes for CIFAR-10
)
# Loss function and optimizer
criterion = torch.nn.CrossEntropyLoss()
# Learning rate based on QMUL ID
learning_rate = 0.001 + (last_digit_of_id * 0.0001)
optimizer = optim.Adam(model.parameters(),
lr=learning_rate)
# Number of epochs based on QMUL ID
num_epochs = 10 + last_digit_of_id
print(f"Training for {num_epochs} epochs with learningrate {learning_rate}.")
```

Part2： Neural Network Architecture and Training

```
# Training Loop
train_losses = []
train_accuracies = []
val_accuracies = []
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    train_accuracy = 100 * correct / total
    print(f"Epoch {epoch+1}/{num_epochs}, Loss:{running_loss:.4f}, Training Accuracy:{train_accuracy:.2f}%")
    # Validation step
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    val_accuracy = 100 * correct / total
    print(f"Validation Accuracy after Epoch {epoch + 1}:{val_accuracy:.2f}%")
    train_losses.append(running_loss)
    train_accuracies.append(train_accuracy)
    val_accuracies.append(val_accuracy)
    import matplotlib.pyplot as plt
```

Model Architecture:

Input Layer:

The input layer flattens each 32×32×3 CIFAR-10 image into a 1D vector of size 3072 (32×32×3) as the input.

Hidden Layer:

The model includes one hidden layer, a fully connected layer with 512 neurons, using the ReLU activation function. This layer introduces non-linearity to enhance the model's ability to learn features.

Output Layer:

The output layer has 10 neurons, corresponding to the 10 classes in the CIFAR-10 dataset. Each output node represents the predicted probability for each class.

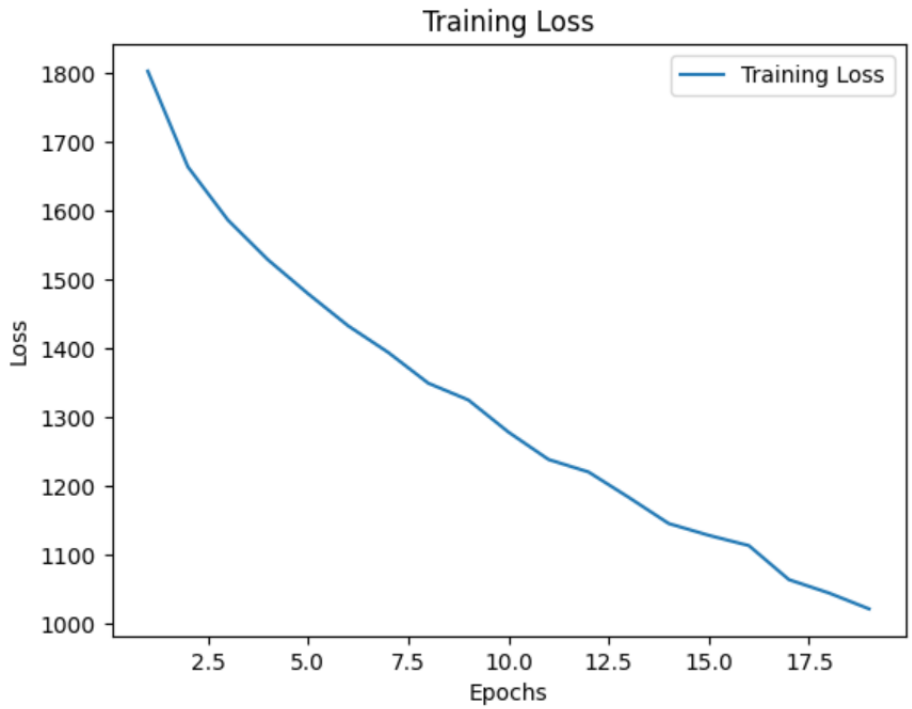
Training Process:

Optimizer:

The model uses the Adam optimizer with a learning rate of **0.0019**. Adam is an optimization algorithm which can dynamically adjust the learning rate for each parameter and accelerate model convergence. By tracking the exponential moving averages of the gradient's first moment (mean) and second moment (variance), Adam automatically adapts to changes in the gradient of each parameter.

Analysis of the Plots:

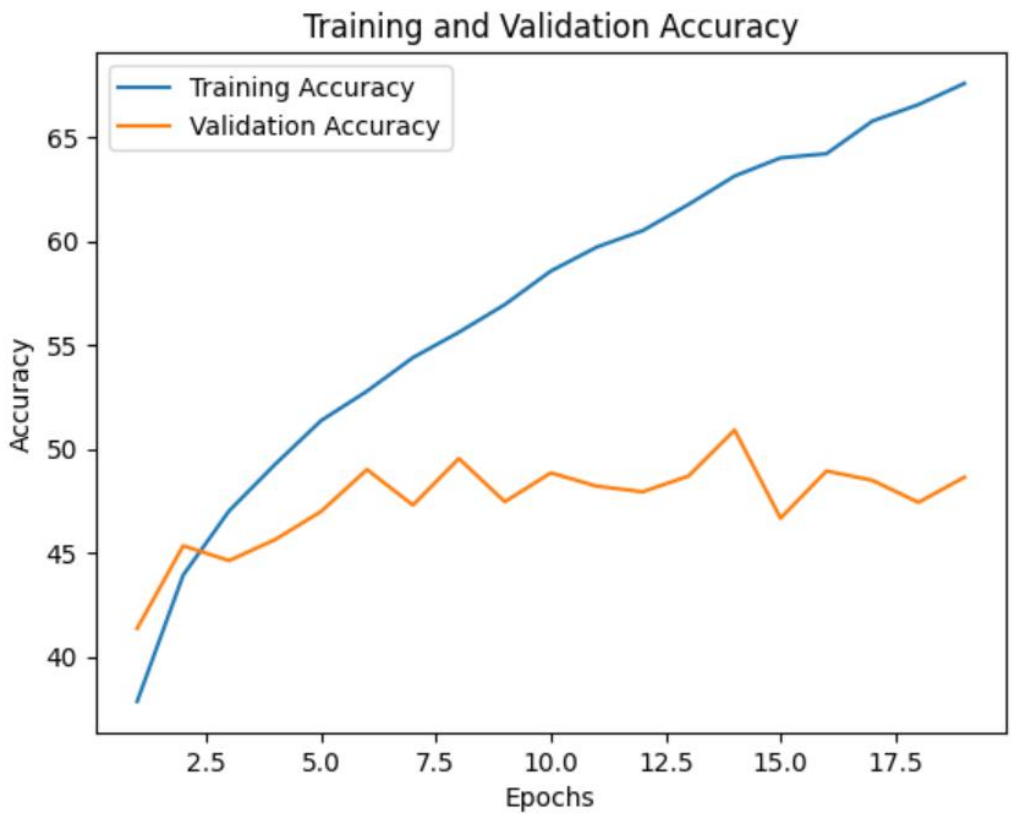
Ⅰ、Training Loss Curve:



In this plot, the training loss decreases gradually as the number of epochs increases. This indicates that the model is progressively learning the features in the training data, and the loss consistently declines throughout the training process, showing improvement in the model's performance on the training set. The loss reduction is especially significant in the initial epochs and then gradually levels off.

This trend suggests that the model is effectively minimizing the loss function during training, and with the dynamic learning rate adjustment of the Adam optimizer, it can achieve stable convergence.

II、Training and Validation Accuracy Curve:



In this plot, we can see that the training accuracy (blue curve) steadily increases over the training process, starting from a low value and eventually approaching around 70%. This indicates that the model is continually learning patterns and features in the training data, improving its performance.

However, the validation accuracy (orange curve) rises quickly in the first few epochs to around 50%, but then fluctuates without significant improvement. This suggests signs of overfitting, as the model performs well on the training data but struggles to generalize to the validation data, indicating that it is overly fitted to the specific patterns in the training set and does not adapt well to new data.

Part3: Results Analysis

```
Files already downloaded and verified
Training on 40000 images, Validating on 10000 images.
Training for 19 epochs with learningrate 0.0019000000000
Epoch 1/19, Loss:1802.9607, Training Accuracy:37.86%
Validation Accuracy after Epoch 1:41.37%
Epoch 2/19, Loss:1664.0840, Training Accuracy:43.93%
Validation Accuracy after Epoch 2:45.34%
Epoch 3/19, Loss:1586.7465, Training Accuracy:47.02%
Validation Accuracy after Epoch 3:44.63%
Epoch 4/19, Loss:1528.9895, Training Accuracy:49.26%
Validation Accuracy after Epoch 4:45.64%
Epoch 5/19, Loss:1479.7299, Training Accuracy:51.36%
Validation Accuracy after Epoch 5:46.99%
Epoch 6/19, Loss:1433.0917, Training Accuracy:52.78%
Validation Accuracy after Epoch 6:49.01%
Epoch 7/19, Loss:1394.4958, Training Accuracy:54.40%
Validation Accuracy after Epoch 7:47.30%
Epoch 8/19, Loss:1349.7717, Training Accuracy:55.62%
Validation Accuracy after Epoch 8:49.54%
Epoch 9/19, Loss:1325.4311, Training Accuracy:56.95%
Validation Accuracy after Epoch 9:47.46%
Epoch 10/19, Loss:1278.7782, Training Accuracy:58.56%
Validation Accuracy after Epoch 10:48.85%
Epoch 11/19, Loss:1238.9936, Training Accuracy:59.71%
Validation Accuracy after Epoch 11:48.21%
Epoch 12/19, Loss:1220.7718, Training Accuracy:60.51%
Validation Accuracy after Epoch 12:47.94%
Epoch 13/19, Loss:1184.0588, Training Accuracy:61.77%
Validation Accuracy after Epoch 13:48.70%
Epoch 14/19, Loss:1145.9514, Training Accuracy:63.13%
Validation Accuracy after Epoch 14:50.91%
Epoch 15/19, Loss:1128.8244, Training Accuracy:64.00%
Validation Accuracy after Epoch 15:46.67%
Epoch 16/19, Loss:1113.9395, Training Accuracy:64.20%
Validation Accuracy after Epoch 16:48.94%
Epoch 17/19, Loss:1064.4599, Training Accuracy:65.78%
Validation Accuracy after Epoch 17:48.49%
Epoch 18/19, Loss:1045.0318, Training Accuracy:66.57%
Validation Accuracy after Epoch 18:47.43%
Epoch 19/19, Loss:1021.9561, Training Accuracy:67.58%
Validation Accuracy after Epoch 19:48.64%
```

1、The analysis of the training and validation performance

The training loss consistently decreases across epochs, indicating that the model is learning and optimizing its parameters effectively on the training set.

Training accuracy steadily increases throughout the training process, reaching close to 70% by the final epoch. This shows that the model is increasingly fitting to the training data, capturing patterns and features effectively.

The validation accuracy initially improves in the first few epochs, reaching around 50%, but it begins to fluctuate and does not show consistent improvement in later epochs. This suggests that while the model learns to some extent from the training data, its ability to generalize to unseen data is limited.

The gap between training and validation accuracy grows larger as training progresses. This growing disparity is a key indicator of **overfitting**, where the model performs well on the training data but struggles on new data (validation set).

2、The model shows signs of overfitting

1：Training accuracy steadily increases, eventually reaching close to 70%

2：Validation accuracy initially improves but fluctuates after reaching around 50%, without further significant improvement.

3：The gap between training and validation accuracy widens as training progresses, showing that while the model's performance on the training data improves, its performance on new data does not improve accordingly.

Part4: Concept Verification

1: Overfitting issue

In machine learning, overfitting occurs when an algorithm fits too closely or even exactly to its training data, resulting in a model that can't make accurate predictions or conclusions from any data other than the training data.

Overfitting defeats the purpose of the machine learning model. Generalization of a model to new data is ultimately what allows us to use machine learning algorithms every day to make predictions and classify data.

When machine learning algorithms are constructed, they leverage a sample dataset to train the model. However, when the model trains for too long on sample data or when the model is too complex, it can start to learn the "noise," or irrelevant information, within the dataset. When the model memorizes the noise and fits too closely to the training set, the model becomes "overfitted," and it is unable to generalize well to new data. If a model cannot generalize well to new data, then it will not be able to perform the classification or prediction tasks that it was intended for.

2: How to solve overfitting

Early stopping: This method seeks to pause training before the model starts learning the noise within the model. This approach risks halting the training process too soon, leading to the opposite problem of underfitting. Finding the "sweet spot" between underfitting and overfitting is the goal here.

Data augmentation: While it is better to inject clean, relevant data into your training data, sometimes noisy data is added to make a model more stable.

Feature selection: When you build a model, you'll have several parameters or features that are used to predict a given outcome, but many times, these features can be redundant to others. Feature selection is the process of identifying the most important ones within the training data and then eliminating the irrelevant or redundant ones. This is commonly mistaken for dimensionality reduction, but it is different. However, both methods help to simplify your model to establish the dominant trend in the data.

3: What is the role of loss function

A loss function is a type of objective function, which in the context of data science refers to any function whose minimization or maximization represents the objective of model training. The term "loss function," which is usually synonymous with cost function or error function, refers specifically to situations where minimization is the training objective for a machine learning model.

A loss function tracks the degree of error in an artificial intelligence (AI) model's output. It does so by quantifying the difference ("loss") between a predicted value—that is, the model's output—for a given input and the actual value or ground truth. If a model's predictions are accurate, the loss is small. If its predictions are inaccurate, the loss is large.

The fundamental goal of machine learning is to train models to output good predictions. Loss functions enable us to define and pursue that goal mathematically. During training, models "learn" to output better predictions by adjusting parameters in a way that reduces loss. A machine learning model has been sufficiently trained when loss has been minimized below some predetermined threshold.

4: Two representative loss functions.

Regression:

Loss functions measure errors in predictions involving continuous values. Though they most intuitively apply to models that directly estimate quantifiable concepts such as price, age, size or time, regression loss has a wide range of applications. For example, a regression loss function can be used to optimize an image model whose task entails estimating the color value of individual pixels.

Classification:

Loss functions measure errors in predictions involving discrete values, such as the category a data point belongs to or if an email is spam or not. Types of classification loss can be further subdivided into those suitable for binary classification and those suitable for multi-class classification.