HPML Assignment 2

Program Outputs

Program C1

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import argparse
import time

class BasicBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, pado
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=
        self.bn2 = nn.BatchNorm2d(out_channels)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=Fals
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out

class ResNet18(nn.Module):
    def __init__(self):
        super(ResNet18, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(64)

        self.layer1 = self._make_layer(64, 64, 2, stride=1)
        self.layer2 = self._make_layer(64, 128, 2, stride=2)
```

```
        self.layer3 = self._make_layer(128, 256, 2, stride=2)
        self.layer4 = self._make_layer(256, 512, 2, stride=2)

        self.linear = nn.Linear(512, 10)

    def _make_layer(self, in_channels, out_channels, blocks, stride):
        layers = []
        layers.append(BasicBlock(in_channels, out_channels, stride))
        for _ in range(1, blocks):
            layers.append(BasicBlock(out_channels, out_channels))
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = F.avg_pool2d(out, 4)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out

def main():
    parser = argparse.ArgumentParser(description='Train ResNet-18 on CIFAR-10')
    parser.add_argument('--use_cuda', action='store_true', help='Use CUDA if available')
    parser.add_argument('--data_path', type=str, default='./data', help='Path to CIFAR-10 da
    parser.add_argument('--num_workers', type=int, default=2, help='Number of data loader wo
    parser.add_argument('--optimizer', type=str, default='sgd', help='Optimizer to use (sgd,
    args = parser.parse_args()

    device = torch.device('cuda' if args.use_cuda and torch.cuda.is_available() else 'cpu')

    # DataLoader for CIFAR-10 dataset
    transform = transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.ToTensor(),
        transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
    ])

    train_dataset = torchvision.datasets.CIFAR10(root=args.data_path, train=True, download=T
    train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True, num_workers=args.

    # Model, loss, and optimizer
    model = ResNet18().to(device)
```

```
criterion = nn.CrossEntropyLoss()

if args.optimizer.lower() == 'sgd':
    optimizer = optim.SGD(model.parameters(), lr=0.1, momentum=0.9, weight_decay=5e-4)
elif args.optimizer.lower() == 'sgd_nesterov':
    optimizer = optim.SGD(model.parameters(), lr=0.1, momentum=0.9, weight_decay=5e-4, n
elif args.optimizer.lower() == 'adagrad':
    optimizer = optim.Adagrad(model.parameters(), lr=0.01, weight_decay=5e-4)
elif args.optimizer.lower() == 'adadelta':
    optimizer = optim.Adadelta(model.parameters(), lr=1.0, weight_decay=5e-4)
elif args.optimizer.lower() == 'adam':
    optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=5e-4)
else:
    raise ValueError('Unsupported optimizer. Use "sgd", "sgd_nesterov", "adagrad", "adad

# Training loop
num_epochs = 5
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    data_loading_time = 0.0
    training_time = 0.0

    # Start timing for data loading
    if device.type == 'cuda':
        torch.cuda.synchronize()
    data_loading_start_time = time.perf_counter()

    for batch_idx, (inputs, targets) in enumerate(train_loader):
        if device.type == 'cuda':
            torch.cuda.synchronize()
        data_loading_end_time = time.perf_counter()
        data_loading_time += data_loading_end_time - data_loading_start_time

        # Start timing for training
        if device.type == 'cuda':
            torch.cuda.synchronize()
        training_start_time = time.perf_counter()

        inputs, targets = inputs.to(device), targets.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)
```

```
            loss.backward()
            optimizer.step()

            if device.type == 'cuda':
                torch.cuda.synchronize()
            training_end_time = time.perf_counter()
            training_time += training_end_time - training_start_time

            running_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()

            if batch_idx % 100 == 0:
                print(f'Epoch [{epoch+1}/{num_epochs}], Batch [{batch_idx+1}/{len(train_loac

            # Reset data loading timer
            if device.type == 'cuda':
                torch.cuda.synchronize()
            data_loading_start_time = time.perf_counter()

        # Total epoch time
        if device.type == 'cuda':
            torch.cuda.synchronize()
        epoch_end_time = time.perf_counter()
        total_epoch_time = data_loading_time + training_time

        print(f'Epoch [{epoch+1}/{num_epochs}] Summary: Data Loading Time: {data_loading_tir

if __name__ == '__main__':
    main()
```

Program C2

```
bash-5.1$ python c2.py --num_workers 2 --optimizer sgd

Files already downloaded and verified
Epoch [1/5], Batch [1/391], Loss: 2.4453, Accuracy: 7.81%
Epoch [1/5], Batch [101/391], Loss: 2.3240, Accuracy: 23.24%
Epoch [1/5], Batch [201/391], Loss: 2.0366, Accuracy: 28.61%
Epoch [1/5], Batch [301/391], Loss: 1.9009, Accuracy: 31.97%
Epoch [1/5] Summary: Data Loading Time: 0.7442s, Training Time: 201.4467s, Total Epoch Time:
Epoch [2/5], Batch [1/391], Loss: 1.3482, Accuracy: 47.66%
Epoch [2/5], Batch [101/391], Loss: 1.4344, Accuracy: 47.25%
Epoch [2/5], Batch [201/391], Loss: 1.3928, Accuracy: 48.81%
Epoch [2/5], Batch [301/391], Loss: 1.3475, Accuracy: 50.94%
```

```
Epoch [2/5] Summary: Data Loading Time: 0.7746s, Training Time: 184.8464s, Total Epoch Time:
Epoch [3/5], Batch [1/391], Loss: 1.0872, Accuracy: 64.06%
Epoch [3/5], Batch [101/391], Loss: 1.0842, Accuracy: 60.99%
Epoch [3/5], Batch [201/391], Loss: 1.0535, Accuracy: 62.57%
Epoch [3/5], Batch [301/391], Loss: 1.0470, Accuracy: 62.92%
Epoch [3/5] Summary: Data Loading Time: 0.8489s, Training Time: 194.0466s, Total Epoch Time:
Epoch [4/5], Batch [1/391], Loss: 0.7287, Accuracy: 72.66%
Epoch [4/5], Batch [101/391], Loss: 0.9070, Accuracy: 67.71%
Epoch [4/5], Batch [201/391], Loss: 0.8906, Accuracy: 68.35%
Epoch [4/5], Batch [301/391], Loss: 0.8722, Accuracy: 69.12%
Epoch [4/5] Summary: Data Loading Time: 0.8840s, Training Time: 190.8991s, Total Epoch Time:
Epoch [5/5], Batch [1/391], Loss: 0.8141, Accuracy: 73.44%
Epoch [5/5], Batch [101/391], Loss: 0.7756, Accuracy: 72.79%
Epoch [5/5], Batch [201/391], Loss: 0.7560, Accuracy: 73.34%
Epoch [5/5], Batch [301/391], Loss: 0.7423, Accuracy: 73.95%
Epoch [5/5] Summary: Data Loading Time: 0.8239s, Training Time: 196.3094s, Total Epoch Time:
```

Program C3

```
bash-5.1$ python c3.py

Running with 0 workers...
Data loading time for 0 workers: 10.915s
Running with 4 workers...
Data loading time for 4 workers: 0.7999s
Running with 8 workers...
Data loading time for 8 workers: 0.8983s

Summary of Results:
Number of Workers: 0, Data Loading Time: 10.91s
Number of Workers: 4, Data Loading Time: 0.80s
Number of Workers: 8, Data Loading Time: 0.90s

The best number of workers for runtime performance is: 4

Graph: The graph basically shows that there is diminishing returns with increase in number o
```

Program C4

```
bash-5.1$ python c4.py

Running with 1 worker...
Running with 4 workers...
1 Worker - Data Loading Time: 0.70s, Training Time: 177.81s
4 Workers - Data Loading Time: 0.78s, Training Time: 189.10s
```

```
The data loading times are similar or even slightly decreasing when using 4 workers compared
The benefits of multiple workers might not be noticeable with such a small dataset.
```

Program C5

```
bash-5.1$ python c5.py

Running training with GPU (using 4 workers)...
Running training with CPU (using 4 workers)...

Results over 5 epochs:
Average GPU training time per epoch: 5.31 seconds
Average CPU training time per epoch: 385.19 seconds
Overall speedup factor: 72.51x

Epoch-wise comparison:
Epoch | GPU Time (s) | CPU Time (s) | Speedup
-------------------------------------------------
    1 |         7.80 |       379.64 |   48.69x
    2 |         4.64 |       382.03 |   82.29x
    3 |         4.70 |       398.94 |   84.93x
    4 |         4.79 |       382.11 |   79.77x
    5 |         4.63 |       383.23 |   82.70x
```

Program C6

```
bash-5.1$ python c6.py
Running training with SGD...
Running training with SGD_NESTEROV...
Running training with ADAGRAD...
Running training with ADADELTA...
Running training with ADAM...

Summary (averaged over 5 epochs):
              Training Time    Loss  Accuracy
Optimizer
ADADELTA            15.2735  1.9484    27.778
ADAGRAD             14.9701  1.9013    29.016
ADAM                15.0567  1.7274    36.856
SGD                 15.2735  2.3505    21.024
SGD_NESTEROV        14.8702  2.1111    27.012

Detailed Results:

SGD:
Epoch | Training Time (s) | Loss | Accuracy (%)
```

```
-------------------------------------------------
    1 |            17.35 | 2.5339 |        10.94
    1 |            14.74 | 2.7230 |        17.44
    1 |            14.78 | 2.3524 |        21.67
    1 |            14.74 | 2.1909 |        24.60
    1 |            14.75 | 1.9521 |        30.47

SGD_NESTEROV:
Epoch | Training Time (s) | Loss | Accuracy (%)
-------------------------------------------------
    1 |            15.13 | 2.5006 |         8.59
    1 |            14.82 | 2.5303 |        19.66
    1 |            14.81 | 2.1639 |        25.98
    1 |            14.79 | 1.9974 |        30.05
    1 |            14.80 | 1.3634 |        50.78

ADAGRAD:
Epoch | Training Time (s) | Loss | Accuracy (%)
-------------------------------------------------
    1 |            15.41 | 2.3631 |         8.59
    1 |            14.85 | 2.0576 |        24.95
    1 |            14.86 | 1.8760 |        30.85
    1 |            14.85 | 1.7731 |        34.60
    1 |            14.88 | 1.4369 |        46.09

ADADELTA:
Epoch | Training Time (s) | Loss | Accuracy (%)
-------------------------------------------------
    1 |            15.63 | 2.3647 |        10.16
    1 |            15.19 | 2.0376 |        25.40
    1 |            15.15 | 1.8836 |        30.14
    1 |            15.19 | 1.7792 |        34.13
    1 |            15.20 | 1.6769 |        39.06

ADAM:
Epoch | Training Time (s) | Loss | Accuracy (%)
-------------------------------------------------
    1 |            15.41 | 2.4420 |         8.59
    1 |            14.93 | 1.8183 |        31.90
    1 |            14.98 | 1.6571 |        38.13
    1 |            14.97 | 1.5526 |        42.38
    1 |            15.00 | 1.1672 |        63.28
```

Program C7

```
bash-5.1$ python c7.py --use_cuda --num_workers 4 --optimizer sgd
Files already downloaded and verified
Epoch [1/5], Batch [1/391], Loss: 2.3068, Accuracy: 8.59%
Epoch [1/5], Batch [101/391], Loss: 2.2537, Accuracy: 14.02%
Epoch [1/5], Batch [201/391], Loss: 2.1501, Accuracy: 18.73%
Epoch [1/5], Batch [301/391], Loss: 2.0567, Accuracy: 22.46%
Epoch [1/5] Summary: Data Loading Time: 0.7678s, Training Time: 3.1920s, Total Epoch Time: 3
Epoch [2/5], Batch [1/391], Loss: 1.7660, Accuracy: 31.25%
Epoch [2/5], Batch [101/391], Loss: 1.6879, Accuracy: 37.23%
Epoch [2/5], Batch [201/391], Loss: 1.6605, Accuracy: 38.36%
Epoch [2/5], Batch [301/391], Loss: 1.6239, Accuracy: 39.80%
Epoch [2/5] Summary: Data Loading Time: 0.8277s, Training Time: 2.4362s, Total Epoch Time: 3
Epoch [3/5], Batch [1/391], Loss: 1.4339, Accuracy: 42.19%
Epoch [3/5], Batch [101/391], Loss: 1.4382, Accuracy: 47.80%
Epoch [3/5], Batch [201/391], Loss: 1.4222, Accuracy: 48.47%
Epoch [3/5], Batch [301/391], Loss: 1.3870, Accuracy: 49.65%
Epoch [3/5] Summary: Data Loading Time: 0.7970s, Training Time: 2.5720s, Total Epoch Time: 3
Epoch [4/5], Batch [1/391], Loss: 1.3065, Accuracy: 50.78%
Epoch [4/5], Batch [101/391], Loss: 1.2422, Accuracy: 55.32%
Epoch [4/5], Batch [201/391], Loss: 1.2311, Accuracy: 56.01%
Epoch [4/5], Batch [301/391], Loss: 1.1983, Accuracy: 57.30%
Epoch [4/5] Summary: Data Loading Time: 0.8031s, Training Time: 2.8857s, Total Epoch Time: 3
Epoch [5/5], Batch [1/391], Loss: 1.0690, Accuracy: 57.81%
Epoch [5/5], Batch [101/391], Loss: 1.0938, Accuracy: 61.68%
Epoch [5/5], Batch [201/391], Loss: 1.0752, Accuracy: 62.27%
Epoch [5/5], Batch [301/391], Loss: 1.0434, Accuracy: 63.37%
Epoch [5/5] Summary: Data Loading Time: 0.7971s, Training Time: 2.4515s, Total Epoch Time: 3
```

Answers of Theroetical Questions

Question 1

How many convolutional layers are in the ResNet-18 model?

Answer 1

- 1 initial convolutional layer
- 8 Basic Blocks (2 in each of the 4 layers) * 2 conv layers per block = 16 conv layers
- 3 shortcut convolutional layers (in the first block of layers 2, 3, and 4)
- Total of 20 convolutional layers

Question 2

What is the input dimension of the last linear layer?

Answer 2

```
self.linear = nn.Linear(512, 10)
```

Input dimension is 512

Question 3

How many trainable parameters and how many gradients in the ResNet-18 model that you build (please show both the answer and the code that you use to count them), when using SGD optimizer?

Answer 3

- Total parameters: 11173962
- Trainable parameters: 11173962
- Total gradients: 11173962

Refer file q3.py for code

Question 4

How many trainable parameters and how many gradients in the ResNet-18 model that you build (please show only the answer), when using ADAM optimizer?

Answer 4

- Total parameters: 11173962
- Trainable parameters: 11173962
- Total gradients: 11173962

Refer file q4.py for code