Homework 7 Code & Graphs
Name: Viprav Lipare
Student ID: 801288922
Homework Number: 7
Github Repository: https://github.com/vipravlipare/ECGR-5105-Intro-to-Machine-Learning

Problem 1A

Source Code (1A):

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
import time

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

data_path = "../data-unversioned/p1ch7"

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4823, 0.4468),
                         (0.2470, 0.2435, 0.2616))
])

cifar10_train = datasets.CIFAR10(data_path, train=True, download=True,
transform=transform)
cifar10_test  = datasets.CIFAR10(data_path, train=False, download=True,
transform=transform)

trainX = torch.stack([img for img, _ in cifar10_train]).to(device)
trainY = torch.tensor([label for _, label in cifar10_train],
dtype=torch.long).to(device)

testX = torch.stack([img for img, _ in cifar10_test]).to(device)
testY = torch.tensor([label for _, label in cifar10_test],
dtype=torch.long).to(device)

train_dataset = torch.utils.data.TensorDataset(trainX, trainY)
val_dataset   = torch.utils.data.TensorDataset(testX, testY)
```

```
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64,
shuffle=True)
val_loader   = torch.utils.data.DataLoader(val_dataset, batch_size=1000,
shuffle=False)
class Net(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.act1  = nn.Tanh()
        self.pool1 = nn.MaxPool2d(2)

        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.act2  = nn.Tanh()
        self.pool2 = nn.MaxPool2d(2)

        self.fc1 = nn.Linear(32 * 8 * 8, 64)
        self.act3 = nn.Tanh()

        self.fc2 = nn.Linear(64, 10)

    def forward(self, x):
        x = self.pool1(self.act1(self.conv1(x)))
        x = self.pool2(self.act2(self.conv2(x)))
        x = x.view(x.size(0), -1)
        x = self.act3(self.fc1(x))
        x = self.fc2(x)
        return x

model = Net().to(device)
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=1e-3)

n_epochs = 300
start = time.time()

for epoch in range(n_epochs):
    model.train()
    running_loss = 0.0
    for batch_X, batch_Y in train_loader:
        optimizer.zero_grad()
        outputs = model(batch_X)
        loss = loss_fn(outputs, batch_Y)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * batch_X.size(0)

    epoch_loss = running_loss / len(train_loader.dataset)
    print(f"Epoch {epoch+1}, Loss: {epoch_loss:.4f}")

end = time.time()

print("\nTraining Time =", end - start, "seconds")
print("Final Training Loss =", epoch_loss)

model.eval()
correct = 0
total = 0

with torch.no_grad():
    for batch_X, batch_Y in val_loader:
        outputs = model(batch_X)
        preds = outputs.argmax(dim=1)
        correct += (preds == batch_Y).sum().item()
        total += batch_Y.size(0)

accuracy = correct / total
print("Evaluation Accuracy =", accuracy)
# Homework 6-3A FNN Results
print("Homework 6-3A - FNN Results")
print("Training Time = 626.65065574646 seconds")
print("Final Training Loss = 0.49717244245529174")
print("Evaluation Accuracy = 0.4871")
```

Results (1A):

Training Time = 1710.6581213474274 seconds
Final Training Loss = 0.5592639330291748

Evaluation Accuracy = 0.7074

Homework 6-3A - FNN Results
Training Time = 626.65065574646 seconds
Final Training Loss = 0.49717244245529174
Evaluation Accuracy = 0.4871

Problem 1B

Source Code (1B):

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
import time

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

data_path = "../data-unversioned/p1ch7"

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4823, 0.4468),
                         (0.2470, 0.2435, 0.2616))
])

cifar10_train = datasets.CIFAR10(data_path, train=True, download=True,
transform=transform)
cifar10_test  = datasets.CIFAR10(data_path, train=False, download=True,
transform=transform)

trainX = torch.stack([img for img, _ in cifar10_train]).to(device)
trainY = torch.tensor([label for _, label in cifar10_train],
dtype=torch.long).to(device)

testX = torch.stack([img for img, _ in cifar10_test]).to(device)
testY = torch.tensor([label for _, label in cifar10_test],
dtype=torch.long).to(device)
```

```
train_dataset = torch.utils.data.TensorDataset(trainX, trainY)
val_dataset   = torch.utils.data.TensorDataset(testX, testY)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64,
shuffle=True)
val_loader   = torch.utils.data.DataLoader(val_dataset, batch_size=1000,
shuffle=False)
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.act1  = nn.Tanh()
        self.pool1 = nn.MaxPool2d(2)

        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.act2  = nn.Tanh()
        self.pool2 = nn.MaxPool2d(2)

        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.act3  = nn.Tanh()
        self.pool3 = nn.MaxPool2d(2)

        self.fc1 = nn.Linear(64 * 4 * 4, 64)
        self.act4 = nn.Tanh()
        self.fc2 = nn.Linear(64, 10)

    def forward(self, x):
        x = self.pool1(self.act1(self.conv1(x)))
        x = self.pool2(self.act2(self.conv2(x)))
        x = self.pool3(self.act3(self.conv3(x)))

        x = x.view(x.size(0), -1)

        x = self.act4(self.fc1(x))
        x = self.fc2(x)

        return x
model = Net().to(device)
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=1e-3)
```

```python
n_epochs = 300
start = time.time()

for epoch in range(n_epochs):
    model.train()
    running_loss = 0.0

    for batch_X, batch_Y in train_loader:
        optimizer.zero_grad()
        outputs = model(batch_X)
        loss = loss_fn(outputs, batch_Y)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * batch_X.size(0)

    epoch_loss = running_loss / len(train_loader.dataset)
    print(f"Epoch {epoch+1}, Loss: {epoch_loss:.4f}")

end = time.time()

print("\nTraining Time =", end - start, "seconds")
print("Final Training Loss =", epoch_loss)

model.eval()
correct = 0
total = 0

with torch.no_grad():
    for batch_X, batch_Y in val_loader:
        outputs = model(batch_X)
        preds = outputs.argmax(dim=1)
        correct += (preds == batch_Y).sum().item()
        total += batch_Y.size(0)

accuracy = correct / total
print("Evaluation Accuracy =", accuracy)
# Homework 6-3B FNN Results
print("Homework 6-3B - FNN Results")
print("Training Time = 1167.7478566169739 seconds")
print("Final Training Loss = 0.015094490163475275")
print("Evaluation Accuracy = 0.4552")
```

Results (1B):

Training Time = 1944.0796592235565 seconds
Final Training Loss = 0.5673410249710082
Evaluation Accuracy = 0.7333

Homework 6-3B - FNN Results
Training Time = 1167.7478566169739 seconds
Final Training Loss = 0.015094490163475275
Evaluation Accuracy = 0.4552

Problem 2A

Source Code (2A):

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import time

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

data_path = "../data-unversioned/p1ch7"
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4823, 0.4468),
                         (0.2470, 0.2435, 0.2616))
])

cifar10_train = datasets.CIFAR10(data_path, train=True, download=True,
transform=transform)
cifar10_test  = datasets.CIFAR10(data_path, train=False, download=True,
transform=transform)
```

```
trainX = torch.stack([img for img, _ in cifar10_train]).to(device)
trainY = torch.tensor([label for _, label in cifar10_train],
dtype=torch.long).to(device)

testX = torch.stack([img for img, _ in cifar10_test]).to(device)
testY = torch.tensor([label for _, label in cifar10_test],
dtype=torch.long).to(device)

train_dataset = torch.utils.data.TensorDataset(trainX, trainY)
val_dataset   = torch.utils.data.TensorDataset(testX, testY)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64,
shuffle=True)
val_loader   = torch.utils.data.DataLoader(val_dataset, batch_size=1000,
shuffle=False)
class NetRes(nn.Module):
    def __init__(self, n_chans1=32):
        super().__init__()
        self.n_chans1 = n_chans1
        self.conv1 = nn.Conv2d(3, n_chans1, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(n_chans1, n_chans1 // 2, kernel_size=3,
padding=1)
        self.conv3 = nn.Conv2d(n_chans1 // 2, n_chans1 // 2,
kernel_size=3, padding=1)
        self.fc1 = nn.Linear(4 * 4 * (n_chans1 // 2), 32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        out = F.max_pool2d(torch.relu(self.conv1(x)), 2)
        out = F.max_pool2d(torch.relu(self.conv2(out)), 2)
        out1 = out
        out = F.max_pool2d(torch.relu(self.conv3(out)) + out1, 2)

        out = out.view(-1, 4 * 4 * (self.n_chans1 // 2))
        out = torch.relu(self.fc1(out))
        out = self.fc2(out)
        return out

model = NetRes().to(device)
loss_fn = nn.CrossEntropyLoss()
```

```python
optimizer = optim.SGD(model.parameters(), lr=1e-3)
n_epochs = 300
start = time.time()

for epoch in range(n_epochs):
    model.train()
    running_loss = 0.0

    for batch_X, batch_Y in train_loader:
        batch_X, batch_Y = batch_X.to(device), batch_Y.to(device)

        optimizer.zero_grad()
        outputs = model(batch_X)
        loss = loss_fn(outputs, batch_Y)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * batch_X.size(0)

    epoch_loss = running_loss / len(train_loader.dataset)
    print(f"Epoch {epoch+1}, Loss: {epoch_loss:.4f}")

end = time.time()
print("\nTraining Time =", end - start, "seconds")
print("Final Training Loss =", epoch_loss)

model.eval()
correct = 0
total = 0

with torch.no_grad():
    for batch_X, batch_Y in val_loader:
        batch_X, batch_Y = batch_X.to(device), batch_Y.to(device)
        outputs = model(batch_X)
        preds = outputs.argmax(dim=1)
        correct += (preds == batch_Y).sum().item()
        total += batch_Y.size(0)

accuracy = correct / total
print("Evaluation Accuracy =", accuracy)
```

Results (2A):

Training Time = 2188.1524357795715 seconds
Final Training Loss = 0.6781243886947632
Evaluation Accuracy = 0.6853

Problem 2B

Source Code (2B):

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import time
import datetime

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

data_path = "../data-unversioned/p1ch7"
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4823, 0.4468),
                         (0.2470, 0.2435, 0.2616))
])

cifar10_train = datasets.CIFAR10(data_path, train=True, download=True,
transform=transform)
cifar10_test  = datasets.CIFAR10(data_path, train=False, download=True,
transform=transform)

trainX = torch.stack([img for img, _ in cifar10_train]).to(device)
trainY = torch.tensor([label for _, label in cifar10_train],
dtype=torch.long).to(device)

testX = torch.stack([img for img, _ in cifar10_test]).to(device)
testY = torch.tensor([label for _, label in cifar10_test],
dtype=torch.long).to(device)
```

```
train_dataset = torch.utils.data.TensorDataset(trainX, trainY)
val_dataset   = torch.utils.data.TensorDataset(testX, testY)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64,
shuffle=True)
val_loader   = torch.utils.data.DataLoader(val_dataset, batch_size=1000,
shuffle=False)
class NetRes(nn.Module):
    def __init__(self, n_chans1=32):
        super().__init__()
        self.n_chans1 = n_chans1
        self.conv1 = nn.Conv2d(3, n_chans1, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(n_chans1, n_chans1 // 2, kernel_size=3,
padding=1)
        self.conv3 = nn.Conv2d(n_chans1 // 2, n_chans1 // 2,
kernel_size=3, padding=1)
        self.fc1 = nn.Linear(4 * 4 * (n_chans1 // 2), 32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        out = F.max_pool2d(torch.relu(self.conv1(x)), 2)
        out = F.max_pool2d(torch.relu(self.conv2(out)), 2)
        out1 = out
        out = F.max_pool2d(torch.relu(self.conv3(out)) + out1, 2)

        out = out.view(out.size(0), -1)
        out = torch.relu(self.fc1(out))
        out = self.fc2(out)
        return out

model = NetRes().to(device)
class NetDropout(nn.Module):
    def __init__(self, n_chans1=32):
        super().__init__()
        self.n_chans1 = n_chans1
        self.conv1 = nn.Conv2d(3, n_chans1, kernel_size=3, padding=1)
        self.conv1_dropout = nn.Dropout2d(p=0.3)
        self.conv2 = nn.Conv2d(n_chans1, n_chans1 // 2, kernel_size=3,
padding=1)
        self.conv2_dropout = nn.Dropout2d(p=0.3)
        self.fc1 = nn.Linear(8 * 8 * (n_chans1 // 2), 32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        out = F.max_pool2d(torch.tanh(self.conv1(x)), 2)
        out = self.conv1_dropout(out)
        out = F.max_pool2d(torch.tanh(self.conv2(out)), 2)
        out = self.conv2_dropout(out)
        out = out.view(out.size(0), -1)
        out = torch.tanh(self.fc1(out))
        out = self.fc2(out)
        return out

model = NetDropout().to(device)
class NetBatchNorm(nn.Module):
    def __init__(self, n_chans1=32):
        super().__init__()
        self.n_chans1 = n_chans1
        self.conv1 = nn.Conv2d(3, n_chans1, kernel_size=3, padding=1)
        self.conv1_batchnorm = nn.BatchNorm2d(num_features=n_chans1)
        self.conv2 = nn.Conv2d(n_chans1, n_chans1 // 2, kernel_size=3,
padding=1)
        self.conv2_batchnorm = nn.BatchNorm2d(num_features=n_chans1 // 2)
        self.fc1 = nn.Linear(8 * 8 * (n_chans1 // 2), 32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        out = self.conv1_batchnorm(self.conv1(x))
        out = F.max_pool2d(torch.tanh(out), 2)
        out = self.conv2_batchnorm(self.conv2(out))
        out = F.max_pool2d(torch.tanh(out), 2)
        out = out.view(out.size(0), -1)
        out = torch.tanh(self.fc1(out))
        out = self.fc2(out)
        return out

model = NetBatchNorm().to(device)
def train_model(model, train_loader, val_loader, n_epochs=300, lr=1e-3,
weight_decay=0.0):
    model = model.to(device)
    loss_fn = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=lr,
weight_decay=weight_decay)
    start = time.time()

    for epoch in range(1, n_epochs + 1):
        model.train()
        running_loss = 0.0

        for batch_X, batch_Y in train_loader:
            batch_X, batch_Y = batch_X.to(device), batch_Y.to(device)
            optimizer.zero_grad()
            outputs = model(batch_X)
            loss = loss_fn(outputs, batch_Y)
            loss.backward()
            optimizer.step()
            running_loss += loss.item() * batch_X.size(0)

        epoch_loss = running_loss / len(train_loader.dataset)
        if epoch == 1 or epoch % 10 == 0:
            print(f"Epoch {epoch}, Loss: {epoch_loss:.4f}")

    end = time.time()
    print("\nTraining Time =", end - start, "seconds")
    print("Final Training Loss =", epoch_loss)

    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for batch_X, batch_Y in val_loader:
            batch_X, batch_Y = batch_X.to(device), batch_Y.to(device)
            outputs = model(batch_X)
            preds = outputs.argmax(dim=1)
            correct += (preds == batch_Y).sum().item()
            total += batch_Y.size(0)
    accuracy = correct / total
    print("Evaluation Accuracy =", accuracy)
    return epoch_loss, accuracy
print("\n ResNet with Weight Decay (lambda=0.001)")
```

```
train_model(NetRes(), train_loader, val_loader, weight_decay=0.001)

print("\n ResNet-10 with Dropout (p=0.3)")
train_model(NetDropout(), train_loader, val_loader)

print("\n ResNet-10 with Batch Normalization")
train_model(NetBatchNorm(), train_loader, val_loader)
```

Results (2B):

ResNet with Weight Decay (lambda=0.001)
Training Time = 2230.9503643512726 seconds
Final Training Loss = 0.7093406404495239
Evaluation Accuracy = 0.7079

ResNet-10 with Dropout (p=0.3)
Training Time = 2274.358279466629 seconds
Final Training Loss = 1.0689404953575135
Evaluation Accuracy = 0.6555

ResNet-10 with Batch Normalization
Training Time = 3162.8528480529785 seconds
Final Training Loss = 0.5697642152786255
Evaluation Accuracy = 0.6912