```
# %pip install matplotlib

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 matplotlib.pyplot as plt



# import os
# os.environ["KERAS_BACKEND"] = "plaidml.keras.backend"
# os.environ['KMP_DUPLICATE_LIB_OK']='TRUE'
```

```
# tensor로 변경 후 정규화 작업
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

# CIFAR-10 데이터셋 불러오기
#adagrad
batch_size = 4

# SGD
# batch_size = 32

# adam
# batch_size = 16
#train set 불러오기
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
#train set dataloader
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=2)
#test set 불러오기
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
#test set dataloader
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:12<00:00, 13129965.62it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

```
class DL_MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv_1 = nn.Conv2d(3, 6, 5)
        self.pooling = nn.MaxPool2d(2, 2)
        self.conv_2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
```

```
        x = self.pooling(F.relu(self.conv_1(x)))   # relu 이용
        x = self.pooling(F.relu(self.conv_2(x)))   # relu 이용
        x = torch.flatten(x, 1) # 배치를 제외한 모든 차원을 평탄화(flatten)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


model = DL_MLP()
# model.load_state_dict(torch.load('mlp_cifar10_adagrad.pth'))
# model.load_state_dict(torch.load('mlp_cifar10_sgd.pth'))
# model.load_state_dict(torch.load('mlp_cifar10_adam.pth'))
```

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

model.to(device)
```

```
Using device: cuda:0
DL_MLP(
  (conv_1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pooling): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv_2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```

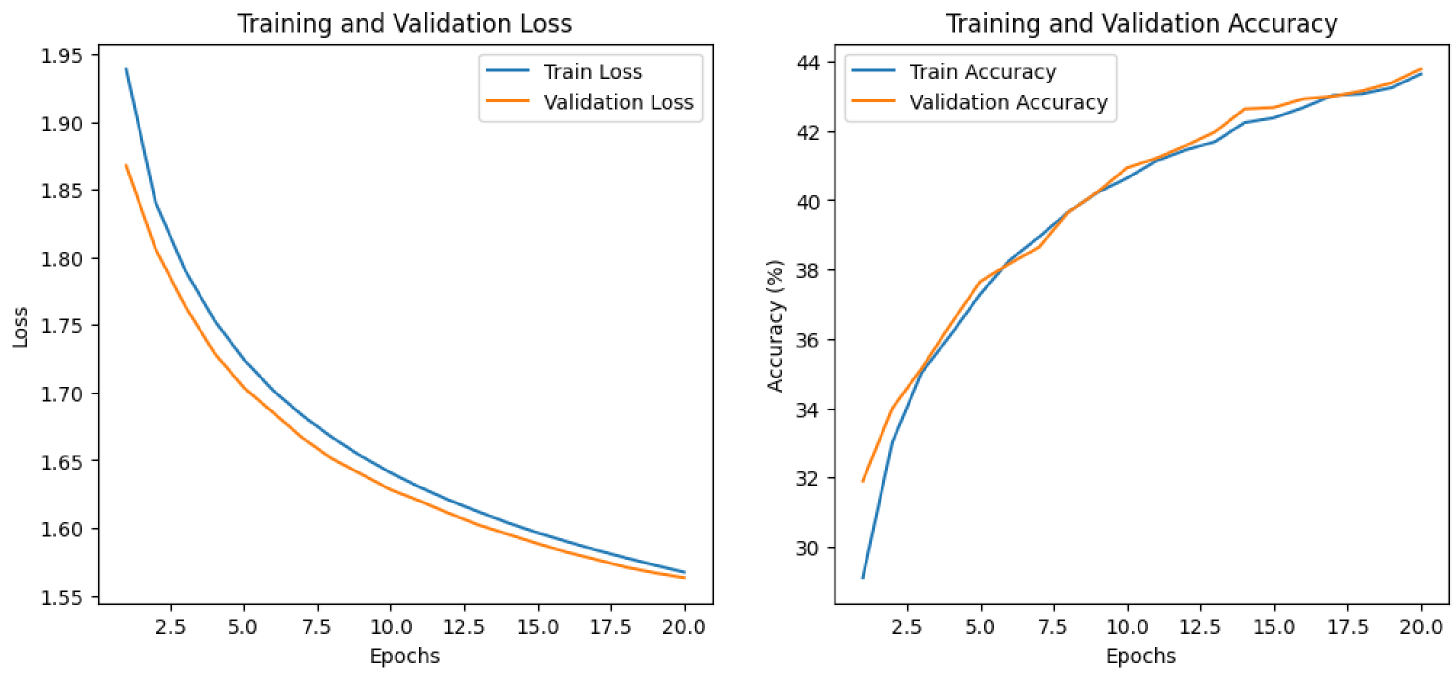
```
criterion = nn.CrossEntropyLoss()

# Set 1: 첫 Hyperparameters
# Learning Rate: 0.001
# Optimizer: Adagrad
# Batch Size: 4
# Set 2: SGD와 모멘텀 사용한 하이퍼파라미터 변경
# Learning Rate: 0.01,momentum=0.9
# Optimizer: SGD with Momentum
# Batch Size: 32
# Set 3: 최신 옵티마이저 adam 사용한 하이퍼파라미터 변경
# Learning Rate: 0.0001
# Optimizer: Adam
# Batch Size: 16

optimizer = optim.Adagrad(model.parameters(), lr=0.001)
# optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
# optimizer = optim.Adam(model.parameters(), lr=0.0001)
```

```
# AUC 계산 함수
def cal_AUC(model, dataloader,device):
    correct = 0
    total = 0
    model.eval() # eval 모드 설정
    with torch.no_grad():
        for data in dataloader:
            images, labels = data[0].to(device), data[1].to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    return 100 * correct / total
# val loss 계산 함수
```

```
def validation_loss(model, validloader, criterion, device):
    total_loss = 0.0
    total_samples = 0
    model.eval()
    with torch.no_grad():
        for data in validloader:
            images, labels = data[0].to(device), data[1].to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            total_loss += loss.item() * images.size(0)
            total_samples += images.size(0)
    return total_loss / total_samples

# 모델 훈련
num_epochs = 20
train_losses, val_losses = [], []
train_accuracies, val_accuracies = [], []

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data[0].to(device), data[1].to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        mini_batch_loss = loss.item()
        if(i %1000==0):
          print(f'Epoch {epoch + 1}, Mini-batch {i+1000}, Loss: {mini_batch_loss:.4f}')

        # AUC 계산
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    train_accuracy = 100 * correct / total
    train_accuracies.append(train_accuracy)
    train_loss = running_loss / len(trainloader)
    train_losses.append(train_loss)

    # 계산 후 val loss와 AUC 기록
    val_loss = validation_loss(model, testloader, criterion, device)
    val_losses.append(val_loss)
    val_accuracy = cal_AUC(model, testloader,device)
    val_accuracies.append(val_accuracy)

    print(f'Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Train Accuracy: {train_accuracy:.2f}%, Validation L

# torch.save(model.state_dict(), 'mlp_cifar10_adagrad.pth')
```

```
Epoch 17, Mini-batch 8000, Loss: 1.3376
Epoch 17, Mini-batch 9000, Loss: 2.1824
Epoch 17, Mini-batch 10000, Loss: 1.4878
Epoch 17, Mini-batch 11000, Loss: 1.7521
Epoch 17, Mini-batch 12000, Loss: 1.7100
Epoch 17, Mini-batch 13000, Loss: 2.3514
Epoch 17/20, Train Loss: 1.5836, Train Accuracy: 43.02%, Validation Loss: 1.5766, Validation Accuracy: 42.99%
Epoch 18, Mini-batch 1000, Loss: 1.7860
Epoch 18, Mini-batch 2000, Loss: 1.6554
Epoch 18, Mini-batch 3000, Loss: 1.7322
Epoch 18, Mini-batch 4000, Loss: 2.3016
Epoch 18, Mini-batch 5000, Loss: 1.6653
Epoch 18, Mini-batch 6000, Loss: 1.7626
Epoch 18, Mini-batch 7000, Loss: 0.8717
Epoch 18, Mini-batch 8000, Loss: 1.4787
Epoch 18, Mini-batch 9000, Loss: 1.1820
Epoch 18, Mini-batch 10000, Loss: 1.6680
Epoch 18, Mini-batch 11000, Loss: 1.1784
Epoch 18, Mini-batch 12000, Loss: 2.8027
Epoch 18, Mini-batch 13000, Loss: 2.4743
Epoch 18/20, Train Loss: 1.5778, Train Accuracy: 43.07%, Validation Loss: 1.5710, Validation Accuracy: 43.15%
Epoch 19, Mini-batch 1000, Loss: 1.6074
Epoch 19, Mini-batch 2000, Loss: 1.2130
Epoch 19, Mini-batch 3000, Loss: 1.1482
Epoch 19, Mini-batch 4000, Loss: 1.9169
Epoch 19, Mini-batch 5000, Loss: 1.9159
Epoch 19, Mini-batch 6000, Loss: 1.9030
Epoch 19, Mini-batch 7000, Loss: 1.5755
Epoch 19, Mini-batch 8000, Loss: 1.2720
Epoch 19, Mini-batch 9000, Loss: 1.9866
Epoch 19, Mini-batch 10000, Loss: 2.1061
Epoch 19, Mini-batch 11000, Loss: 2.1489
Epoch 19, Mini-batch 12000, Loss: 1.2842
Epoch 19, Mini-batch 13000, Loss: 1.0330
Epoch 19/20, Train Loss: 1.5724, Train Accuracy: 43.24%, Validation Loss: 1.5668, Validation Accuracy: 43.38%
Epoch 20, Mini-batch 1000, Loss: 1.1740
Epoch 20, Mini-batch 2000, Loss: 1.5859
Epoch 20, Mini-batch 3000, Loss: 1.8476
Epoch 20, Mini-batch 4000, Loss: 1.2961
Epoch 20, Mini-batch 5000, Loss: 1.8601
Epoch 20, Mini-batch 6000, Loss: 1.3358
Epoch 20, Mini-batch 7000, Loss: 1.5555
Epoch 20, Mini-batch 8000, Loss: 1.2067
Epoch 20, Mini-batch 9000, Loss: 1.8071
Epoch 20, Mini-batch 10000, Loss: 1.4378
Epoch 20, Mini-batch 11000, Loss: 1.8550
Epoch 20, Mini-batch 12000, Loss: 1.9212
Epoch 20, Mini-batch 13000, Loss: 1.2949
Epoch 20/20, Train Loss: 1.5673, Train Accuracy: 43.63%, Validation Loss: 1.5632, Validation Accuracy: 43.78%
```

```python
# train 과 val loss 와 auc plot
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(1, num_epochs+1), train_losses, label='Train Loss')
plt.plot(range(1, num_epochs+1), val_losses, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(range(1, num_epochs+1), train_accuracies, label='Train Accuracy')
plt.plot(range(1, num_epochs+1), val_accuracies, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy (%)')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.show()
```




```
import matplotlib.pyplot as plt
import numpy as np

# 레이블과 함께 이미지 출력
def imshow(img, labels):
    img = img / 2 + 0.5
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.title(' '.join(f'{classes[labels[j]]}' for j in range(len(labels))))
    plt.show()

# 6개 랜덤 훈련된 이미지
dataiter = iter(trainloader)
images, labels = [], []
for _ in range(6):
    img, label = next(dataiter)
    images.append(img[0])
    labels.append(label[0])


# grid에 이미지를 리스트로 변환
images_grid = torchvision.utils.make_grid(images)
imshow(images_grid, labels)
```