CIFAR10 인식 정확도 챌린지

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.init as init
import torchvision.datasets as dset
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torch.autograd import Variable
import matplotlib.pyplot as plt

# (8) learning rate decay
from torch.optim import lr_scheduler

batch_size=32
learning_rate=0.001
num_epoch=10
```

CIFAR10 train, test dataset 가져오기

```
cifar_train=dset.CIFAR10("CIFAR10/",train=True, transform=transforms.ToTensor(), target_transform=None, downloa
cifar_test=dset.CIFAR10("CIFAR10/",train=False, transform=transforms.ToTensor(), target_transform=None, downloa

# (2) data augmentation
#cifar_train=dset.CIFAR10("CIFAR10/", train=True,
#                         transform=transforms.Compose([
#                            transforms.Scale(36),
#                            transforms.CenterCrop(32),
#                            transforms.RandomHorizontalFlip(),
#                            transforms.Lambda(lambda x: x.rotate(90)),
#                            transforms.ToTensor()
#                         ]))

# (4) Data Normalization
#cifar_train=dset.CIFAR10("CIFAR10/", train=True,
#                         transform=transforms.Compose([
#                            transforms.ToTensor(),
#                            transforms.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5)),
#                         ])
#                        , target_transform=None, download=False)
#cifar_test=dset.CIFAR10("CIFAR10/", train=False,
#                         transform=transforms.Compose([
#                            transforms.ToTensor(),
#                            transforms.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5)),
#                         ])
#                         , target_transform=None, download=False)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to CIFAR10/cifar-10-python.tar.gz
170499072/? [00:55<00:00, 3053444.82it/s]

Extracting CIFAR10/cifar-10-python.tar.gz to CIFAR10/
Files already downloaded and verified
```

성능 측정 함수

```
def ComputeAccr(dloader, imodel):
    correct=0
    total=0

    for j, [imgs,labels] in enumerate(dloader): # batch_size 만큼
        img = Variable(imgs, volatile = True).cuda() # x
        #label=Variable(labels)    # y
        label=Variable(labels).cuda()
        #.cuda(): GPU에 로드되기 위함, 만약 CPU로 설정되어 있다면 에러남

        output=imodel.forward(img)#forward prop.
        _, output_index=torch.max(output,1)

        total+=label.size(0)
        correct+=(output_index==label).sum().float()
    print("Accuracy of Test Data: {}".format(100*correct/total))
```

인식 정확도 높이기

```
# === 3. 데이터 로드 함수 ===
train_loader=torch.utils.data.DataLoader(list(cifar_train)[:], batch_size=batch_size, shuffle=True, num_workers
test_loader=torch.utils.data.DataLoader(cifar_test, batch_size=batch_size, shuffle=False, num_workers=2, drop_

# === 4. 모델선언 ===
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.layer=nn.Sequential(
            nn.Conv2d(3,16,3,padding=1),
            nn.ReLU(),
            nn.Dropout2d(0.2),   # (2) drop out
            nn.BatchNorm2d(16),  # (5) Batch normalization
            nn.Conv2d(16,32,3,padding=1),
            nn.ReLU(),
            nn.Dropout2d(0.2),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(2,2),
            nn.Conv2d(32,64,3,padding=1),
            nn.ReLU(),
            nn.Dropout2d(0.2),
            nn.BatchNorm2d(64),
            nn.MaxPool2d(2,2)
        )
        self.fc_layer=nn.Sequential(
            nn.Linear(64*8*8, 100),
            nn.ReLU(),
            nn.Dropout2d(0.2),
            nn.BatchNorm1d(100),
            nn.Linear(100,10)
        )

         # (3) weight initialization
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                init.kaiming_normal(m.weight.data)  #ReLU일때
                m.bias.data.fill_(0)
```

```
                m.bias.data.fill_(0)
            if isinstance(m, nn.Linear):
                init.kaiming_normal(m.weight.data)
                m.bias.data.fill_(0)


    def forward(self, x):
        out=self.layer(x)
        out=out.view(batch_size, -1)
        out=self.fc_layer(out)

        return out
model=CNN().cuda()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:36: UserWarning: nn.init.kaiming_normal is
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:39: UserWarning: nn.init.kaiming_normal is
```

```
# === 5. loss, optimizer ===
loss_func = nn.CrossEntropyLoss()
#optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
optimizer=torch.optim.Adam(model.parameters(),lr=learning_rate) #(6)Adam optimizer

schedule = lr_scheduler.StepLR(optimizer, step_size=100, gamma=0.2)

# == 6. 학습 ===
model.train()
for i in range(num_epoch):
    for j,[image,label] in enumerate(train_loader):
        x = Variable(image).cuda()
        y_ = Variable(label).cuda()

        optimizer.zero_grad()
        output = model.forward(x)
        loss = loss_func(output, y_)
        loss.backward()
        optimizer.step()

        if j%1000 == 0:
            print(j,loss)
```

```
0 tensor(2.9207, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(1.6460, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(1.4006, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(1.6907, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(1.3007, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(1.3600, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.9713, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(1.1956, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(1.0173, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.7792, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(1.0126, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(1.2258, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(1.1031, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.8146, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(1.0036, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.8597, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.6517, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.8537, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.6476, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.7225, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.7025, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.6646, device='cuda:0', grad_fn=<NllLossBackward>)
```

```
0 tensor(0.7293, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.9683, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.4406, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.7554, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.7289, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.8738, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.7486, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.5964, device='cuda:0', grad_fn=<NllLossBackward>)
```

```
model.eval()
ComputeAccr(test_loader, model)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: UserWarning: volatile was removed and no

Accuracy of Test Data: 75.79126739501953
```

```
# 학습된 파라미터 저장
netname = 'my_net_BEST.pkl'
torch.save(model, netname, )
```

```
# 저장된 파라미터 로드
netname = 'my_net_BEST.pkl'
model = torch.load(netname)

# 성능 확인
model.eval()
ComputeAccr(test_loader, model)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: UserWarning: volatile was removed and no

Accuracy of Test Data: 76.9831771850586
```