CIFAR-10 Challenge

학습 전략은 이 논문을 참고하여 설정하였습니다.

1. Learning rate scheduling
   아래 논문에서는 Learning rate warmup이라 하여 초기 몇 epoch에서는 Learning rate를 linear하게 키우고, 그 이후는 감소시키는 방법을 추천한다고 합니다.그래서 아래의 논문에서는cosine annealing with warm up이라는 lr스케쥴링을 사용하지만, 저는 이와 유사하게 pytorch에서 기본으로 제공하는 도구인 cyclicLR을 사용하였습니다.
2. Data augmentation
   Data augmentation 기법으로는 Randomcrop, horizontal flip을 사용하였고 아래 논문에서 나왔던 MixUp이라는 augmentation 기법을 사용하였습니다.
3. FC-layer
   FC-layer는 분류 하는 layer로써 CNN의 tra 위하여 4096 -> 100 -> 10 으로 설정하였고, 더 빠른 학습을 위하여 softmax 활성화 함수를 마지막에 추가하였습니다.

[He, Tong, et al. "Bag of tricks for image classification with convolutional neural networks." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019.]

```
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

from torch.optim import lr_scheduler

from google.colab import files

import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import random
```

```
batch_size = 32
learning_rate = 2e-3
num_epoch = 200
weight_decay=1e-3
MixUp_choice = 1
MixUp_alpha = 0.4

random_seed=42

torch.manual_seed(random_seed)
torch.cuda.manual_seed(random_seed)
torch.cuda.manual_seed_all(random_seed) # if use multi-GPU
torch.backends.cudnn.deterministic = True
```

```
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
np.random.seed(random_seed)
random.seed(random_seed)
```

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

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to CIFAR10/cifar-10-python.tar.gz
170499072/? [00:01<00:00, 89680892.09it/s]

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

```
# v2
def ComputeAccr(dloader, imodel):
  correct = 0
  total = 0

  with torch.no_grad():
    for j, [imgs, labels] in enumerate(dloader):
      img = Variable(imgs).cuda()
      label = Variable(labels).cuda()

      output = imodel.forward(img)
      _, 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))
  return 100*correct/total
```

cifar-10 augmentation

normalize에 사용한 mean, std 수치는 이곳을 참고하여 사용하였습니다.
reference

```
cifar_train = dset.CIFAR10("CIFAR10/", train=True,
                          transform=transforms.Compose([
                           transforms.RandomCrop(32, padding=4),
                           transforms.RandomHorizontalFlip(),
                           transforms.ToTensor(),
                           transforms.Normalize((0.4914, 0.4822, 0.4465),
                            (0.2023, 0.1994, 0.2010)),
                          ]))
cifar_test = dset.CIFAR10("CIFAR10/",train=False,
                         transform=transforms.Compose([
                           transforms.ToTensor(),
                           transforms.Normalize((0.4914, 0.4822, 0.4465),
                            (0.2023, 0.1994, 0.2010))
                         ]),
```

```
                              target_transform=None,download=True)
```

```
Files already downloaded and verified
```

```
train_loader = torch.utils.data.DataLoader(list(cifar_train)[:],
                                           batch_size=batch_size,
                                           shuffle=True, num_workers=2,# num_workers는 cpu 코어 개수
                                           drop_last=True)
test_loader = torch.utils.data.DataLoader(cifar_test,
                                           batch_size=batch_size,
                                           shuffle=False, num_workers=2,
                                           drop_last=True)
```

Model

CNN model은 기존 구성과 동일하며 Dropout을 다 제거하였습니다.
FC-layer는 4096 -> 100 -> 10 으로 마지막에 softmax activation을 추가하였습니다.
또한 RELU activation의 변형인 ELU를 사용하였기 때문에 초기 weight을 HE초기화를 진행하였습니다.

```
class CNN(nn.Module):
  def __init__(self):
    super(CNN, self).__init__()
    self.layer = nn.Sequential(
        nn.Conv2d(3,16,3,padding=1),
        nn.ELU(alpha=1.0),
        nn.BatchNorm2d(16),

        nn.Conv2d(16,32,3,padding=1),
        nn.ELU(alpha=1.0),
        nn.BatchNorm2d(32),
        nn.MaxPool2d(2,2),

        nn.Conv2d(32,64,3,padding=1),
        nn.ELU(alpha=1.0),
        nn.BatchNorm2d(64),

        nn.MaxPool2d(2,2)
    )
    self.fc_layer = nn.Sequential(
        nn.Linear(64*8*8,100),
        nn.ELU(alpha=1.0),
        nn.Dropout(0.5),
        nn.BatchNorm1d(100),
        nn.Linear(100,10)
    )
    # Weight initialization
    for m in self.modules():
      if isinstance(m, nn.Conv2d):
        init.kaiming_normal_(m.weight.data)
        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)
    out = nn.functional.log_softmax(out, dim=1)
    return out

model = CNN().cuda()
print(model)
```

Base Line(without MixUp)

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

# # scheduler = lr_scheduler.CyclicLR(optimizer, base_lr=1e-3, max_lr=learning_rate, step_size_up=10,
# #                       step_size_down=None, mode='triangular2',cycle_momentum=False)
# # optimizer = torch.optim.SGD(model.parameters(), lr=0.0001)
# # scheduler = lr_scheduler.OneCycleLR(optimizer, max_lr=0.1,
# #                                                 steps_per_epoch=10, epochs=100)

# losses=[]
# train_acc = []
# val_acc = []

# #model = CNN().cuda()
# Max=0
# for i in range(num_epoch):
#   model.train()
#   print(str(i) + " epochs")
#   for j, [image, label] in enumerate(train_loader):
#     x=Variable(image).cuda()
#     y_=Variable(label).cuda()

#     optimizer.zero_grad() # grad가 누적합으로 계산되기 때문에 0으로 초기화
#     output=model.forward(x) # 순방향 전파
#     loss=loss_func(output,y_) # loss 계산
#     loss.backward() # 역전파
#     optimizer.step()

#   # model training 시각화를 위한 설정
#   model.eval()
#   tmp = ComputeAccr(test_loader,model)
#   val_acc.append(tmp)
#   train_acc.append(ComputeAccr(train_loader,model))
#   print()
#   losses.append(loss)
#   if (Max < tmp) and ( i>9 ): # 최고 성능 모델 저장
#     Max = tmp
#     netname='/content/my_net_'+str(tmp)+'eps'+'.pkl'
#     torch.save(model,netname,)
# #files.download(netname) # 에폭 다 돌렸을 시 최고 성능 모델 local로 저장
```

```
# x = list(range(len(val_acc)))
# plt.plot(x, val_acc)
# plt.plot(x, train_acc)
# plt.show()
```

사용할 learning rate 시각화

```
optimizer = optim.Adam(model.parameters(), lr=learning_rate, #momentum=0.9,
                      weight_decay=weight_decay)
scheduler = lr_scheduler.CyclicLR(optimizer, base_lr=learning_rate/2, max_lr=learning_rate*2, step_size_up
                     step_size_down=None, mode='triangular2',cycle_momentum=False)

lrs=[]
for i in range(200):
    optimizer.step()
    lrs.append(optimizer.param_groups[0]["lr"])
#     print("Factor = ",i," , Learning Rate = ",optimizer.param_groups[0]["lr"])
    scheduler.step()

plt.plot(lrs)
```

MixUp augmentation

사진 두장을 일정 비율로 혼합하여 사용

label 또한 비율로 설정

optimizer는 adam

l2 regulazation

lr_scheduler= CyclicLR

Augmentation = MixUp, Crop, randomHorizantal flip

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate, #momentum=0.9,
                      weight_decay=weight_decay)
scheduler = lr_scheduler.CyclicLR(optimizer, base_lr=learning_rate/2, max_lr=learning_rate*2, step_size_up
                     step_size_down=None, mode='triangular2',cycle_momentum=False)

def mixup_data(x, y, alpha=1.0, use_cuda=True):
    '''Returns mixed inputs, pairs of targets, and lambda'''
    if alpha > 0:
        lam = np.random.beta(alpha, alpha)
    else:
        lam = 1

    batch_size = x.size()[0]
    if use_cuda:
        index = torch.randperm(batch_size).cuda()
    else:
        index = torch.randperm(batch_size)

```

```
    mixed_x = lam * x + (1 - lam) * x[index, :]
    y_a, y_b = y, y[index]
    return mixed_x, y_a, y_b, lam


def mixup_criterion(criterion, pred, y_a, y_b, lam):  # MixUp augmentation에서의 lossfunction으로 실제 labe
    return lam * criterion(pred, y_a) + (1 - lam) * criterion(pred, y_b)
```

```
use_cuda = False
Max=0
losses=[]
train_acc = []
val_acc = []
best_net = []

for i in range(num_epoch):
  model.train()
  print(str(i) + " epochs")
  for j, [image, label] in enumerate(train_loader):
    choice = np.random.rand()
    x=Variable(image).cuda()
    y_=Variable(label).cuda()
    if choice <MixUp_choice: # if use mixup
      x, targets_a, targets_b, lam = mixup_data(x, y_,
                                        MixUp_alpha, use_cuda)
      x, targets_a, targets_b = map(Variable, (x,
                                        targets_a, targets_b))
      outputs = model(x)
      loss = mixup_criterion(criterion, outputs, targets_a, targets_b, lam)
      _, predicted = torch.max(outputs.data, 1)

      optimizer.zero_grad()
      loss.backward()
      optimizer.step()
    else:  # else
      optimizer.zero_grad() # grad가 누적합으로 계산되기 때문에 0으로 초기화
      output=model.forward(x) # 순방향 전파
      loss=criterion(output,y_) # loss 계산
      loss.backward() # 역전파
      optimizer.step()

  model.eval()
  tmp = ComputeAccr(test_loader,model)
  val_acc.append(tmp)
  train_acc.append(ComputeAccr(train_loader,model))
  print()
  losses.append(loss)
  if (Max < tmp) and ( i>9 ):
    Max = tmp
    netname='/content/my_bestNet'+'.pkl'
    torch.save(model,netname,)
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate, #momentum=0.9,
                       weight_decay=weight_decay)
scheduler = lr_scheduler.CyclicLR(optimizer, base_lr=learning_rate/2, max_lr=learning_rate*2, step_size_up=
```

```
                      step_size_down=None, mode='triangular2',cycle_momentum=False)
for i in range(num_epoch):
  model.train()
  print(str(i) + " epochs")
  for j, [image, label] in enumerate(train_loader):
    choice = np.random.rand()
    x=Variable(image).cuda()
    y_=Variable(label).cuda()
    if choice <MixUp_choice: # if use mixup
      x, targets_a, targets_b, lam = mixup_data(x, y_,
                                         MixUp_alpha, use_cuda)
      x, targets_a, targets_b = map(Variable, (x,
                                         targets_a, targets_b))
      outputs = model(x)
      loss = mixup_criterion(criterion, outputs, targets_a, targets_b, lam)
      _, predicted = torch.max(outputs.data, 1)

      optimizer.zero_grad()
      loss.backward()
      optimizer.step()
    else:  # else
      optimizer.zero_grad() # grad가 누적합으로 계산되기 때문에 0으로 초기화
      output=model.forward(x) # 순방향 전파
      loss=criterion(output,y_) # loss 계산
      loss.backward() # 역전파
      optimizer.step()

  model.eval()
  tmp = ComputeAccr(test_loader,model)
  val_acc.append(tmp)
  train_acc.append(ComputeAccr(train_loader,model))
  print()
  losses.append(loss)
  if (Max < tmp) and ( i>9 ):
    Max = tmp
    netname='/content/my_bestNet'+'.pkl'
    torch.save(model,netname,)
files.download(netname)
```

Visualization

모델의 training을 시각화 하여 학습이 어떻게 진행되는지 각 epoch당 train,test accuracy 그래프로 확인하였습니다.

```
x = list(range(len(val_acc)))

plt.plot(x,val_acc)
plt.plot(x, train_acc)
plt.show()
```

```
netname ='/content/main2.pkl'
eval_model=torch.load(netname)
ComputeAccr(test_loader,eval_model)
```

```
Accuracy of Test Data: 78.57572174072266
tensor(78.5757, device='cuda:0')
```