

## ▼ 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 논문에서는 cosine annealing with warm up이라는 lr 스케줄링을 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.](#)

```
1 import sys
2 print(sys.version_info)

sys.version_info(major=3, minor=7, micro=10, releaselevel='final', serial=0)

1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 import torch.nn.init as init
5 import torchvision.datasets as dset
6 import torchvision.transforms as transforms
7 from torch.utils.data import DataLoader
8 from torch.autograd import Variable
9
10 from torch.optim import lr_scheduler
11
12 from google.colab import files
13
14 import matplotlib.pyplot as plt
15 %matplotlib inline
16 import numpy as np
17 import random

1 batch_size = 32
2 learning_rate = 2e-3
3 num_epoch = 200
4 weight_decay=1e-3
```

```

5 MixUp_choice = 1
6 MixUp_alpha = 0.4
7
8 random_seed=42
9
10 torch.manual_seed(random_seed)
11 torch.cuda.manual_seed(random_seed)
12 torch.cuda.manual_seed_all(random_seed) # if use multi-GPU
13 torch.backends.cudnn.deterministic = True
14 torch.backends.cudnn.benchmark = False
15 np.random.seed(random_seed)
16 random.seed(random_seed)

```

```

1 cifar_train = dset.CIFAR10("CIFAR10/", train=True, transform=transforms.ToTensor(),
2                             target_transform=None, download=True)
3 cifar_test = dset.CIFAR10("CIFAR10/", train=False, transform=transforms.ToTensor(),
4                             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

```

1 # v2
2 def ComputeAccr(dloader, imodel):
3     correct = 0
4     total = 0
5
6     with torch.no_grad():
7         for j, [imgs, labels] in enumerate(dloader):
8             img = Variable(imgs).cuda()
9             label = Variable(labels).cuda()
10
11             output = imodel.forward(img)
12             _, output_index = torch.max(output, 1)
13
14             total += label.size(0)
15             correct += (output_index == label).sum().float()
16 print("Accuracy of Test Data: {}".format(100*correct/total))
17 return 100*correct/total

```

## ▼ cifar-10 augmentation

normalize에 사용한 mean, std 수치는 이곳을 참고하여 사용하였습니다.

[reference](#)

```

1 cifar_train = dset.CIFAR10("CIFAR10/", train=True,
2                             transform=transforms.Compose([
3                                 transforms.RandomCrop(32, padding=4),
4                                 transforms.RandomHorizontalFlip(),
5                                 transforms.ToTensor(),
6                                 transforms.Normalize([0.4914, 0.4822, 0.4465],

```

```

6         transforms.Normalize((0.4914, 0.4822, 0.4465),
7                               (0.2023, 0.1994, 0.2010)),
8     ]))
9 cifar_test = dset.CIFAR10("CIFAR10/",train=False,
10                           transform=transforms.Compose([
11                               transforms.ToTensor(),
12                               transforms.Normalize((0.4914, 0.4822, 0.4465),
13                                                     (0.2023, 0.1994, 0.2010))
14                           ]),
15                           target_transform=None,download=True)

```

Files already downloaded and verified

```

1 train_loader = torch.utils.data.DataLoader(list(cifar_train)[:],
2                                               batch_size=batch_size,
3                                               shuffle=True, num_workers=2,# num_workers는 cpu 코어 개수
4                                               drop_last=True)
5 test_loader = torch.utils.data.DataLoader(cifar_test,
6                                             batch_size=batch_size,
7                                             shuffle=False, num_workers=2,
8                                             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.

```

1 class CNN(nn.Module):
2     def __init__(self):
3         super(CNN, self).__init__()
4         self.layer = nn.Sequential(
5             nn.Conv2d(3,16,3,padding=1),
6             nn.ELU(alpha=1.0),
7             nn.BatchNorm2d(16),
8
9             nn.Conv2d(16,32,3,padding=1),
10            nn.ELU(alpha=1.0),
11            nn.BatchNorm2d(32),
12            nn.MaxPool2d(2,2),
13
14            nn.Conv2d(32,64,3,padding=1),
15            nn.ELU(alpha=1.0),
16            nn.BatchNorm2d(64),
17
18            nn.MaxPool2d(2,2)
19        )
20        self.fc_layer = nn.Sequential(
21            nn.Linear(64*8*8,100),
22            nn.ELU(alpha=1.0),
23            nn.Dropout(0.5),
24            nn.BatchNorm1d(100),
25            nn.Linear(100,10)
26        )

```

```

26     /
27     # Weight initialization
28     for m in self.modules():
29         if isinstance(m, nn.Conv2d):
30             init.kaiming_normal_(m.weight.data)
31             m.bias.data.fill_(0)
32         if isinstance(m, nn.Linear):
33             init.kaiming_normal_(m.weight.data)
34             m.bias.data.fill_(0)
35     def forward(self, x):
36         out = self.layer(x)
37
38         out = out.view(batch_size,-1)
39         out = self.fc_layer(out)
40         out = nn.functional.log_softmax(out, dim=1)
41         return out
42
43 model = CNN().cuda()
44 print(model)

```

## ▼ Base Line(without MixUp)

```

1 # loss_func = nn.CrossEntropyLoss()
2 # optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
3
4 # # scheduler = lr_scheduler.CyclicLR(optimizer, base_lr=1e-3, max_lr=learning_rate, step_size_up=10,
5 # #                                     step_size_down=None, mode='triangular2',cycle_momentum=False)
6 # # optimizer = torch.optim.SGD(model.parameters(), lr=0.0001)
7 # # scheduler = lr_scheduler.OneCycleLR(optimizer, max_lr=0.1,
8 # #                                     steps_per_epoch=10, epochs=100)
9
10 # losses=[]
11 # train_acc = []
12 # val_acc = []
13
14 # #model = CNN().cuda()
15 # Max=0
16 # for i in range(num_epoch):
17 #     model.train()
18 #     print(str(i) + " epochs")
19 #     for j, [image, label] in enumerate(train_loader):
20 #         x=Variable(image).cuda()
21 #         y_=Variable(label).cuda()
22
23 #         optimizer.zero_grad() # grad가 누적함으로 계산되기 때문에 0으로 초기화
24 #         output=model.forward(x) # 순방향 전파
25 #         loss=loss_func(output,y_) # loss 계산
26 #         loss.backward() # 역전파
27 #         optimizer.step()
28
29 # # model training 시각화를 위한 설정
30 # model.eval()
31 # tmp = ComputeAccr(test_loader,model)
32 # val_acc.append(tmp)
33 # train_acc.append(ComputeAccr(train_loader,model))

```

```

33 # train_acc.append(computer_acc(train_loader,model))
34 # print()
35 # losses.append(loss)
36 # if (Max < tmp) and ( i>9 ): # 최고 성능 모델 저장
37 #     Max = tmp
38 #     netname='/_content/my_net_'+str(tmp)+'eps'+'.pkl'
39 #     torch.save(model,netname,)
40 # #files.download(netname) # 에폭 다 돌렸을 시 최고 성능 모델 local로 저장

1 # x = list(range(len(val_acc)))
2 # plt.plot(x, val_acc)
3 # plt.plot(x, train_acc)
4 # plt.show()

```

## ▼ 사용할 learning rate 시각화

```

1 optimizer = optim.Adam(model.parameters(), lr=learning_rate, #momentum=0.9,
2                             weight_decay=weight_decay)
3 scheduler = lr_scheduler.CyclicLR(optimizer, base_lr=learning_rate/2, max_lr=learning_rate*2, step_size_up=
4                             step_size_down=None, mode='triangular2',cycle_momentum=False)
5
6 lrs=[]
7 for i in range(200):
8     optimizer.step()
9     lrs.append(optimizer.param_groups[0]["lr"])
10 #     print("Factor = ",i," , Learning Rate = ",optimizer.param_groups[0]["lr"])
11     scheduler.step()
12
13 plt.plot(lrs)

```

## ▼ MixUp augmentation

사진 두장을 일정 비율로 혼합하여 사용

label 또한 비율로 설정

optimizer는 adam

l2 regularization

lr\_scheduler= CyclicLR

Augmentation = MixUp, Crop, randomHorizontal flip

```

1 criterion = nn.CrossEntropyLoss()
2 optimizer = optim.Adam(model.parameters(), lr=learning_rate, #momentum=0.9,
3                             weight_decay=weight_decay)
4 scheduler = lr_scheduler.CyclicLR(optimizer, base_lr=learning_rate/2, max_lr=learning_rate*2, step_size_up=
5                             step_size_down=None, mode='triangular2',cycle_momentum=False)
6
7 def mixup_data(x, y, alpha=1.0, use_cuda=True):
8     '''Returns mixed inputs, pairs of targets, and lambda'''
9     if alpha > 0:
10         lam = np.random.beta(alpha, alpha)
11     else:

```

```

12     lam = 1
13
14     batch_size = x.size()[0]
15     if use_cuda:
16         index = torch.randperm(batch_size).cuda()
17     else:
18         index = torch.randperm(batch_size)
19
20     mixed_x = lam * x + (1 - lam) * x[index, :]
21     y_a, y_b = y, y[index]
22     return mixed_x, y_a, y_b, lam
23
24
25 def mixup_criterion(criterion, pred, y_a, y_b, lam): # MixUp augmentation에서의 lossfunction으로 실제 lab
26     return lam * criterion(pred, y_a) + (1 - lam) * criterion(pred, y_b)

```

```

1 use_cuda = False
2 Max=0
3 losses=[]
4 train_acc = []
5 val_acc = []
6 best_net = []
7
8 for i in range(num_epoch):
9     model.train()
10    print(str(i) + " epochs")
11    for j, [image, label] in enumerate(train_loader):
12        choice = np.random.rand()
13        x=Variable(image).cuda()
14        y_=Variable(label).cuda()
15        if choice < MixUp_choice: # if use mixup
16            x, targets_a, targets_b, lam = mixup_data(x, y_,
17                                                    MixUp_alpha, use_cuda)
18            x, targets_a, targets_b = map(Variable, (x,
19                                                    targets_a, targets_b))
20            outputs = model(x)
21            loss = mixup_criterion(criterion, outputs, targets_a, targets_b, lam)
22            _, predicted = torch.max(outputs.data, 1)
23
24            optimizer.zero_grad()
25            loss.backward()
26            optimizer.step()
27        else: # else
28            optimizer.zero_grad() # grad가 누적함으로 계산되기 때문에 0으로 초기화
29            output=model.forward(x) # 순방향 전파
30            loss=criterion(output,y_) # loss 계산
31            loss.backward() # 역전파
32            optimizer.step()
33
34    model.eval()
35    tmp = ComputeAccr(test_loader,model)
36    val_acc.append(tmp)
37    train_acc.append(ComputeAccr(train_loader,model))
38    print()
39    losses.append(loss)
40    if (Max < tmp) and ( i>9 ):

```

```

41     Max = tmp
42     netname='/content/my_bestNet'+'.pkl'
43     torch.save(model,netname,)

1 criterion = nn.CrossEntropyLoss()
2 optimizer = optim.Adam(model.parameters(), lr=learning_rate, #momentum=0.9,
3                             weight_decay=weight_decay)
4 scheduler = lr_scheduler.CyclicLR(optimizer, base_lr=learning_rate/2, max_lr=learning_rate*2, step_size_up=
5                             step_size_down=None, mode='triangular2',cycle_momentum=False)
6 for i in range(num_epoch):
7     model.train()
8     print(str(i) + " epochs")
9     for j, [image, label] in enumerate(train_loader):
10         choice = np.random.rand()
11         x=Variable(image).cuda()
12         y_=Variable(label).cuda()
13         if choice < MixUp_choice: # if use mixup
14             x, targets_a, targets_b, lam = mixup_data(x, y_,
15                                                         MixUp_alpha, use_cuda)
16             x, targets_a, targets_b = map(Variable, (x,
17                                                         targets_a, targets_b))
18             outputs = model(x)
19             loss = mixup_criterion(criterion, outputs, targets_a, targets_b, lam)
20             _, predicted = torch.max(outputs.data, 1)
21
22             optimizer.zero_grad()
23             loss.backward()
24             optimizer.step()
25         else: # else
26             optimizer.zero_grad() # grad가 누적함으로 계산되기 때문에 0으로 초기화
27             output=model.forward(x) # 순방향 전파
28             loss=criterion(output,y_) # loss 계산
29             loss.backward() # 역전파
30             optimizer.step()
31
32     model.eval()
33     tmp = ComputeAccr(test_loader,model)
34     val_acc.append(tmp)
35     train_acc.append(ComputeAccr(train_loader,model))
36     print()
37     losses.append(loss)
38     if (Max < tmp) and ( i>9 ):
39         Max = tmp
40         netname='/content/my_bestNet'+'.pkl'
41         torch.save(model,netname,)
42     files.download(netname)

```

## ▼ Visualization

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

```

1 x = list(range(len(val_acc)))
2

```

```
2
3 plt.plot(x, val_acc)
4 plt.plot(x, train_acc)
5 plt.show()
```

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

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

✓ 0초 오후 3:35에 완료됨

● ✕

reCAPTCHA 서비스에 연결할 수 없습니다. 인터넷 연결을 확인한 후 페이지를 새로고침하여 reCAPTCHA 보안문자를 다시 로드하세요.