Deep Learning Assignment #1 - 이미지 분류

2018062733 컴퓨터소프트웨어학부 윤동빈

1. 코드 설명

```
import torch
import torch.nn as nn
import torch.optim as optim

import torchvision
import torchvision.transforms as transforms
```

딥러닝 학습에 필요한 *PyTorch*, *torchvision* 라이브러리 사용을 위해 import한다.

```
torch.manual_seed(0)
torch.cuda.manual_seed(0)
torch.cuda.manual_seed_all(0)
```

Random Seed 고정하는 과정이다.

```
if torch.cuda.is_available():
  device = torch.device('cuda')
else:
  device = torch.device('cpu')
```

GPU 사용 여부에 따라 device의 값을 변경하는 과정으로, GPU 사용 시 *'cuda'*, 미사용 시 *'cpu'*라는 값을 저장한다.

```
class Classifier(nn.Module):
    # 모델의 코드는 여기서 작성해주세요

    def __init__(self, drop_prob):
        super(Classifier, self).__init__()
        self.linear1 = nn.Linear(32*32*3, 2048) # input size : 32*32*3 (RGB)
        self.linear2 = nn.Linear(2048, 1024)
        self.linear3 = nn.Linear(1024, 512)
        self.linear4 = nn.Linear(512, 10)       # output size : 10

        self.dropout = nn.Dropout(drop_prob)
        self.activation = nn.Sigmoid()

    def forward(self, x):
        z1 = self.linear1(x)
        a1 = self.activation(z1)
        a1 = self.dropout(a1)

        z2 = self.linear2(a1)
        a2 = self.activation(z2)
        a2 = self.dropout(a2)

        z3 = self.linear3(a2)
        a3 = self.activation(z3)
        a3 = self.dropout(a3)

        output = self.linear4(a3)

        return output
```

학습되는 모델에 대한 내용이 정의되는 class로, 생성자 *__init__()*과 forward propagation에 대한 내용이 담긴 *forward()* 메소드로 구성된다.

def __init__(self, drop_prob):

- drop_prob : 모델에 Dropout 적용 시 필요한 Dropout rate으로 layer의 각 노드에서 학습 여부를 결정하는데 영향을 미침
- self.activation = nn.Sigmoid() : Activation function은 Sigmoid function으로 설정

def forward(self, x):

- x : 모델의 input으로, training data 정보가 들어감
- z1, z2 : 각 layer output
- a1, a2 (activation) : z1, z2에 대한 activation function output
- a1, a2 (dropout) : a1, a2에 대한 dropout output으로, 학습 참여 여부 결정

```
if __name__ == "__main__":
    # 학습코드는 모두 여기서 작성해주세요

    train_dataset = torchvision.datasets.CIFAR10(root="CIFAR10/",
                                                 train=True,
                                                 transform=transforms.ToTensor(),
                                                 download=True)
    test_dataset = torchvision.datasets.CIFAR10(root="CIFAR10/",
                                                train=False,
                                                transform=transforms.ToTensor(),
                                                download=True)

    batch_size = 4096

    train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size)

    model = Classifier(0.5).to(device)

    optimizer = optim.Adam(model.parameters(), lr=1e-3, betas=(0.95, 0.999))
    criterion = nn.CrossEntropyLoss()

    epochs = 350
    lmbd = 0.003

    total_batch_num = len(train_dataloader)
```

train_dataset, test_dataset : 학습, 테스트에 필요한 dataset을 저장하는 변수로, CIFAR10 dataset을 불러와 저장하고 있음

batch_size = 4096 : dataset을 mini-batch로 나눠서 학습하는데, mini-batch size를 4096로 설정

train_dataloader : mini-batch에서 학습할 수 있도록, mini-batch size에 맞춰 dataset을 나눠서 저장

model : 모델의 인스턴스로, dropout rate을 0.5으로 설정

optimizer : 최적화 알고리즘 변수로, Adam 알고리즘을 최적화 알고리즘으로 설정하고 β_1=0.95, β_2=0.999로 설정

epochs : 모든 dataset을 모델에 학습하는 횟수로, 350번 학습하도록 설정

lmbd : regularizaion 과정에서 쓰이는 λ 변수로, 0.003로 설정
total_batch_num : 학습하는데 쓰이는 mini-batch용 dataset size로,
학습 과정 중 평균 cost를 구하는데 쓰임

```
for epoch in range(epochs):
  avg_cost = 0
  model.train()

  for b_x, b_y in train_dataloader:
    b_x = b_x.view(-1, 32*32*3).to(device)
    logits = model(b_x) # forward prop
    loss = criterion(logits, b_y.to(device)) # get cost

    reg = model.linear1.weight.pow(2.0).sum()
    reg += model.linear2.weight.pow(2.0).sum()
    reg += model.linear3.weight.pow(2.0).sum()

    loss += lmbd*reg/len(b_x)/len(b_x)/2.

    optimizer.zero_grad()
    loss.backward() # backward prop
    optimizer.step() # update parameters

    avg_cost += loss / total_batch_num # 모든 데이터셋에 대한 cost 값

  print('Epoch : {} / {}, cost : {}'.format(epoch+1, epochs, avg_cost))
```

앞에서 불러온 training data로 모델에 학습을 진행하는 과정으로,
중간에 *print()*를 통해 각 epoch에서의 평균 cost를 학습 중에 확인할 수 있다.

내부 for loop은 train_dataloader에 담긴 dataset을 input(=*b_x*), output(=*b_y*)로 나눈 뒤 모델에 학습을 진행하는 과정을 거친다. **forward propagation**, **back propagation**, **파라미터 업데이트**로 구성되며, back propagation 과정 중에 필요한 loss 값을 구하는 과정과 *loss* 값을 구하는데 적용되는 regularization 과정을 보인다. 최종적으로 *optimizer_step()*으로 파라미터 업데이트까지 처리한다.

optimizer.zero_zero_grad() : gradient 값을 0으로 초기화

2. 실험결과 (직전 결과 기준으로 **변경 사항**은 굵은 글씨로 처리)

[Optimizer=**SGD**, epochs=70, learning rate=1]

```
print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 47.6500%
```

[Optimizer=**Adagrad**, epochs=70, learning rate=1]

```
print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 10.0000%
```

[Optimizer=Adagrad, epochs=70, **learning rate=1e-3**]

```
print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 40.1900%
```

[Optimizer=**RMSprop**, epochs=70, learning rate=1e-3, alpha=0.99]

```
print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 47.7600%
```

[Optimizer=**Adam**, epochs=70, learning rate=1e-3, betas=(0.9, 0.999)]

```
print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 48.9500%
```

[Optimizer=Adam, epochs=70, learning rate=1e-3, betas=(0.9, 0.999), **dropout rate(drop_prob)=0.1**]

```
print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 51.2000%
```

[Optimizer=Adam, **epochs=45**, learning rate=1e-3, betas=(0.9, 0.999), dropout rate(drop_prob)=0.1]

```
print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 50.7500%
```

[Optimizer=Adam, **epochs=60**, learning rate=1e-3, betas=(0.9, 0.999), dropout rate(drop_prob)=0.1]

```
print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 51.3300%
```

[Optimizer=Adam, epochs=60, learning rate=1e-3, betas=(0.9, 0.999), **dropout rate(drop_prob)=0.2**]

```
        print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 51.5900%
```

[Optimizer=Adam, **epochs=70**, learning rate=1e-3, betas=(0.9, 0.999), dropout rate(drop_prob)=0.2]

```
        print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 50.6600%
```

batch_size를 128(default)에서 256으로 변경
(default는 이전 test까지 적용했던 값을 의미합니다.)

[Optimizer=Adam, epochs=60, learning rate=1e-3, betas=(0.9, 0.999), dropout rate(drop_prob)=0.2, **batch_size=256**]

```
        print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 52.2500%
```

[Optimizer=Adam, **epochs=70**, learning rate=1e-3, betas=(0.9, 0.999), dropout rate(drop_prob)=0.2, batch_size=256]

```
        print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 52.5000%
```

[Optimizer=Adam, **epochs=85**, learning rate=1e-3, betas=(0.9, 0.999), dropout rate(drop_prob)=0.2, batch_size=256]

```
        print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 52.7200%
```

[Optimizer=Adam, **epochs=100**, learning rate=1e-3, betas=(0.9, 0.999), dropout rate(drop_prob)=0.2, batch_size=256]

```
        print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 52.8500%
```

[Optimizer=Adam, **epochs=120**, learning rate=1e-3, betas=(0.9, 0.999), dropout rate(drop_prob)=0.2, **batch_size=512**]

```
print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 53.4600%
```

[Optimizer=Adam, epochs=120, learning rate=1e-3, betas=(0.9, 0.999), **dropout rate(drop_prob)=0.3**, batch_size=512]

```
print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 53.6800%
```

[Optimizer=Adam, **epochs=140**, learning rate=1e-3, betas=(0.9, 0.999), dropout rate(drop_prob)=0.3, **batch_size=1024**]

```
print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 54.3200%
```

[Optimizer=Adam, **epochs=180**, learning rate=1e-3, betas=(0.9, 0.999), dropout rate(drop_prob)=0.3, batch_size=1024]

```
print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 54.6900%
```

[Optimizer=Adam, **epochs=250**, learning rate=1e-3, betas=(0.9, 0.999), dropout rate(drop_prob)=0.3, **batch_size=2048**]

```
print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 55.5200%
```

2^{nd} layer output과 3^{rd} layer input dimension을 128(default)에서 256으로 변경

[Optimizer=Adam, epochs=250, learning rate=1e-3, betas=(0.9, 0.999), dropout rate(drop_prob)=0.3, **batch_size=4096**,
2^{nd} layer output & 3^{rd} layer input 차원=256]

```
print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 55.9000%
```

1st layer output과 2nd layer input dimension을 256(default)에서 512으로 변경

[Optimizer=Adam, epochs=250, learning rate=1e-3, betas=(0.9, 0.999),
dropout rate(drop_prob)=0.3, batch_size=4096,
1st layer output & 2nd layer input 차원=512,
2nd layer output & 3rd layer input 차원=256]

```
        print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 56.4700%
```

[Optimizer=Adam, epochs=250, learning rate=1e-3, betas=(0.9, 0.999),
dropout rate(drop_prob)=0.3, batch_size=4096,
1st layer output & 2nd layer input 차원=512,
2nd layer output & 3rd layer input 차원=512]

```
        print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 56.6400%
```

[Optimizer=Adam, epochs=250, learning rate=1e-3, betas=(0.9, 0.999),
dropout rate(drop_prob)=0.3, batch_size=4096,
1st layer output & 2nd layer input 차원=1024,
2nd layer output & 3rd layer input 차원=512]

```
        print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 57.1300%
```

[Optimizer=Adam, **epochs=300**, learning rate=1e-3, betas=(0.9, 0.999),
dropout rate(drop_prob)=0.3, batch_size=4096,
1st layer output & 2nd layer input 차원=1024,
2nd layer output & 3rd layer input 차원=512]

```
        print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 57.7600%
```

4^{th} layer 추가 및 변경된 dimension은 다음과 같음
- 1^{st} layer output & 2^{nd} layer input 차원=2048
- 2^{nd} layer output & 3^{rd} layer input 차원=1024
- 3^{rd} layer output & 4^{th} layer input 차원=512

Adam optimizer algorithm $\beta_1 = 0.9$(default)에서 $\beta_1 = 0.95$로 변경

[Optimizer=Adam, **epochs=350**, learning rate=1e-3, betas=(0.9, 0.999),
dropout rate(drop_prob)=0.5, batch_size=4096,
1^{st} layer output & 2^{nd} layer input 차원=2048,
2^{nd} layer output & 3^{rd} layer input 차원=1024,
3^{rd} layer output & 4^{th} layer input 차원=512,
$\beta_1 = 0.95$]

```
        print("Accuracy on test set : {:.4f}%".format(100 * correct / total))

Files already downloaded and verified
Accuracy on test set : 58.6700%
```

▶ 최종 accuracy : **58.67%**