인공지능 hw3

국민대학교 컴퓨터공학부
20163140
이수진

1

<코드해석>

```python
# -*- coding: utf -8 -*-

import torch
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # 커널 정의
        self.conv1 = nn.Conv2d(1, 6, 5)      # 1 * 6 * 5 * 5
        self.conv2 = nn.Conv2d(6, 16, 5)     # 6 * 16 * 5 * 5
        # fully connections 정의
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    # 순전파 함수에 신경망의 구조를 정의
    def forward(self, x):
        # 컨볼루션 연산을 통해 특징 추출 후 풀링 연산하여 subsampling
        # 구조 conv1 - relu - pool - conv2 - relu -  pool - fc1 - relu - fc2 - relu - fc3
        x = F.max_pool2d(F.relu(self.conv1(x)), (2,2))
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        # x를 열의 개수가 self.num_flat_features(x)개이도록 재배열
        # -1 의 의미 : 행의 수를 알 수 없음
        # 1행 배열이 형성됨
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]
        num_features = 1
        for s in size:
            num_features *= s
        return num_features

net = Net()
print("(1)")
print(net)
# (1) 화면 출력 확인 및 의미를 서술

# (2) 정의된 컨볼루션 신경망의 구조 설명 (위의 AlexNet 그림 참고)

params = list(net.parameters())
print("(3)")
print(len(params))
print(params[0].size())

## (3) 화면 출력 확인
#
input = torch.randn(1, 1, 32, 32)
out = net(input)
print("(4)")
print(out)
## (4) 화면 출력 확인

# 오류 역전파를 위해 가중치의 그래디언트 버퍼 초기화
net.zero_grad()
# 역전파
out.backward(torch.randn(1, 10))

# 손실 함수 정의 및 임의의 값들에 대해서 오차 결과 확인
# mse 손실함수 사용
output = net(input)
target = torch.randn(10)
target = target.view(1, -1)
criterion = nn.MSELoss()

# output = 예측값, target = 실제값
loss = criterion(output, target)
print("(5)")
print(loss)
## (5) 화면 출력 확인
```

```
78  net.zero_grad()
79
80  print("(6)")
81  print('conv1.bias.grad before backward')
82  print(net.conv1.bias.grad)
83  # (6) 화면 출력 확인
84
85  loss.backward()
86
87  print("(7)")
88  print('conv1.bias.grad after backward')
89  print(net.conv1.bias.grad)
90  # (7) 화면 출력 확인
91
92  # 스토캐스틱 경사하강법(미래 가중치 = 현재가중치 - 학습률 * 그레이디언트)을 이용하여 가중치 갱신 코드
93  learning_rate = 0.01
94  for f in net.parameters():
95      f.data.sub_(f.grad.data * learning_rate)
96
97
98  # 오류역전파에서 최적화하는 방법
99  import torch.optim as optim
100
101 # torch.optim.SGD를 사용하여 가중치 갱신
102 optimizer = optim.SGD(net.parameters(), lr = 0.01)
103
104 optimizer.zero_grad()
105 output = net(input)
106 loss = criterion(output, target)
107 loss.backward()
108 # 최적화 과정을 수행한다.
109 optimizer.step()
110
```

<출력 결과>

- (1)

```
(1)
Net(
  (conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
  (conv2): 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)
)
```

⇨ 초기화한 신경망의 구조요소들을 의미한다.

⇨ 1 input* 6*5*5 컨볼루션, 6 input * 16*5*5 컨볼루션, input 400-output 120 Fully connection, input 120-output 84 Fully connection, input 84-output 10 Fully connection

- (2)

⇨ Forward함수의 정의를 보면 신경망의 구조를 알 수 있다.

⇨ 이 신경망의 구조는 input – conv – relu - pool – conv – relu - pool – fc – relu – fc - relu - fc 이다

⇨ 하나의 사진을 입력
컨볼루션 연산을 통해 1 input => 6개의 특징맵 형성
Relu 연산
풀링 연산을 통해 다운 샘플링
컨볼루션 연산을 통해 6 input특징맵 => 16개의 특징맵 형성
Relu 연산
풀링 연산을 통해 다운 샘플링
형성된 특징맵을 선형으로 변환
400개의 특징input => 120개의 특징들로 fc연산
Relu 연산
120개의 특징 input => 84개의 특징들로 fc연산
Relu 연산
84개의 특징 input => 10개의 output으로 fc연산

- (3)

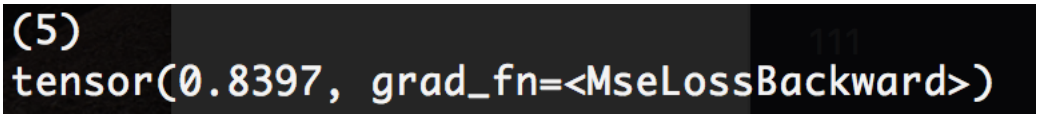
⇨ 모델의 학습가능한 매개변수의 수와 conv1의 weight

여기부터 한번에 컴파일 후 출력 해야 함!!!!!!!!!!

- (4)

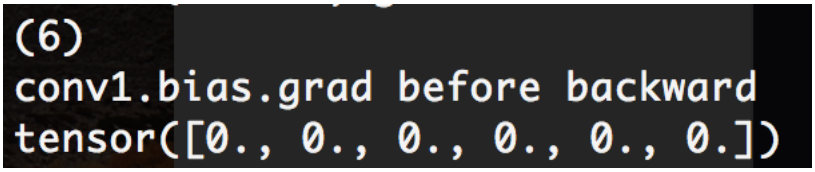
⇨ 신경망에 임의의 32*32 입력한 결과

- (5)



⇨ Loss

- (6)



⇨ 오류역전파 전 0으로 초기화한 conv1의 bias gradient

- (7)

```
(7)
conv1.bias.grad after backward
tensor([-0.0101, -0.0040,  0.0058,  0.0030,  0.0032,  0.0013])
```

⇨ 오류역전파 후 conv1의 bias gradient

➔ 신경망을 정의하고 전방전파를 통해 손실을 구하고 오류 역전파를 통해 손실을 전달해 가중치를 갱신하는 신경망을 확인할 수 있는 코드

2

<코드 해석>

```python
import torch
import torchvision
import torchvision.transforms as transforms

# torchvision 데이터셋의 출력(output)은 [0, 1] 범위를 갖는 PILImage 이미지이다.
# 이를 [-1, 1]의 범위로 정규화된 Tensor로 변환
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

# torchvision을 이용해 훈련집합 적재
trainset = torchvision.datasets.CIFAR10(root = './data', train = True, download = True, transform = transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size = 4, shuffle = True, num_workers = 2)

# torchvision을 이용해 테스트집합 적재
testset = torchvision.datasets.CIFAR10(root='./data', train = False, download = True, transform = transform)
testloader = torch.utils.data.DataLoader(testset, batch_size = 4, shuffle = False, num_workers = 2)

classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
# (1) 화면 출력 확인

import matplotlib.pyplot as plt
import numpy as np

# 이미지를 보여주기 위한 함수 정의
def imshow(img):
    img = img/ 2.0 + 0.5      #unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()                # 커맨드라인에서 실행시 추가해주어야 함

# 훈련집합을 무작위로 가져온다
dataiter = iter(trainloader)
images, labels = dataiter.next()

# 가져온 훈련집합을 보여준다
imshow(torchvision.utils.make_grid(images))

print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
# (2) 화면 출력 확인
```

```python
# 컨볼루션 신경망 정의
# 3채널 32 * 32 크기의 사진을 입력받고, 신경망을 통과해 10부류를 수정

import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # 커널 정의
        # 3채널을 입력받을 수 있도록 정의
        self.conv1 = nn.Conv2d(3, 6, 5)
        # 풀링층 정의
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # fully connections 정의
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    # 순전파 함수에 신경망의 구조를 정의
    def forward(self, x):
        # 컨볼루션 연산을 통해 특징 추출 후 풀링 연산하여 subsampling
        # 구조 conv1 - relu - pool - conv2 - relu - pool - fc1 - relu - fc2 - relu - fc3
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net()
import torch.optim as optim

# 손실함수 정의. 교차 엔트로피와 SGD + momentum
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr = 0.001, momentum = 0.9)

# 데이터를 반복해서 신경망에 입력으로 제공하고, 최적화(Optimize)
# 훈련집합을 이용하여 신경망 학습시킴
for epoch in range(2):           # 데이터셋을 여러번 반복한다
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # 데이터 입력
        inputs, labels = data
        # 그레이디언트 버퍼 초기화
        optimizer.zero_grad()

        # input에 대한 신경망의 예측값
        outputs = net(inputs)
        # 손실 계산
        loss = criterion(outputs, labels)
        # 오류 역전파
        loss.backward()
        # 가중치 갱신(최적화)
        optimizer.step()

        # 통계 출력
        running_loss += loss.item()
        if i % 1000 == 999:  # 모든 1000 미니배치들을 출력한다.
            print('[%d, %5d] loss : %.3f' %(epoch + 1, i + 1, running_loss/1000.0))
            running_loss = 0.0

print('Finished Training')
# (3) 화면 출력 확인 및 학습이 되고 있는지 서술
```

```python
# 테스트 집합을 이용하여 학습시킨 신경망 성능 확인

# 테스트 집합을 무작위로 가져온다
dataiter = iter(testloader)
images, labels = dataiter.next()

# 가져온 테스트집합을 보여준다.
imshow(torchvision.utils.make_grid(images))

print('GroundTurth: '+''.join('%5s ' %classes[labels[j]] for j in range(4)))
# (4) 화면 출력 확인

# 출력은 10개 분류 각각에 대한 값으로 나타난다. 어떤 분류에 대해서 더 높은 값이 나타난다는 것은,
# 신경망이 그 이미지가 더 해당 분류에 가깝다고 예측했다는 것이다.
# 따라서, 가장 높은 값을 갖는 인덱스(index)를 결과 예측값으로 뽑는다.
outputs = net(images)
_, predicted = torch.max(outputs, 1)
print('Predicted: '+''.join('%5s ' %classes[predicted[j]] for j in range(4)))
# (5) 화면 출력 확인

# 전체 테스트집합에 적용하여 일반화 성능을 측정한다.
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' %(100 * correct / total))
# (5) 화면 출력 확인 및 일반화 성능 서술
```

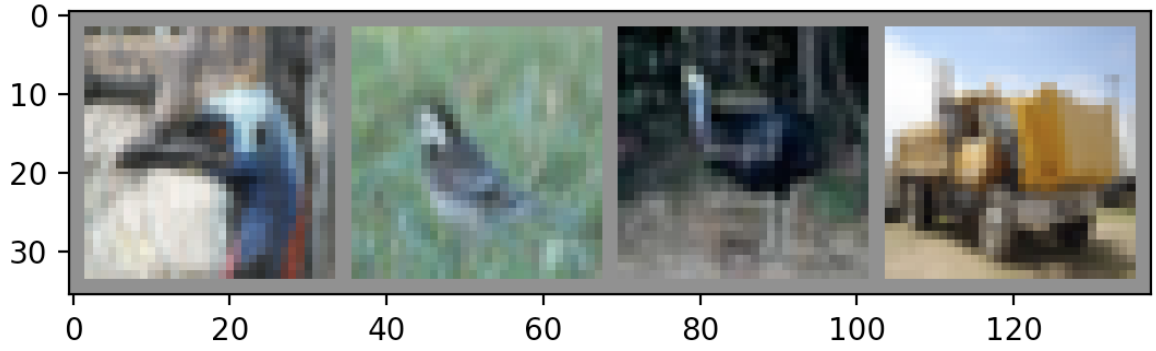
```python
# 각 분류에 대한 일반화 성능 평가
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1

for i in range(10):
    print('Accuracy of %5s : %2d %%' %(classes[i], 100 * class_correct[i]/class_total[i]))
# (7) 화면 출력 확인 및 부류별 분류기의 성능 서술
```

<출력 결과>

- (1)

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/ci
far-10-python.tar.gz
Files already downloaded and verified
```

- (2)



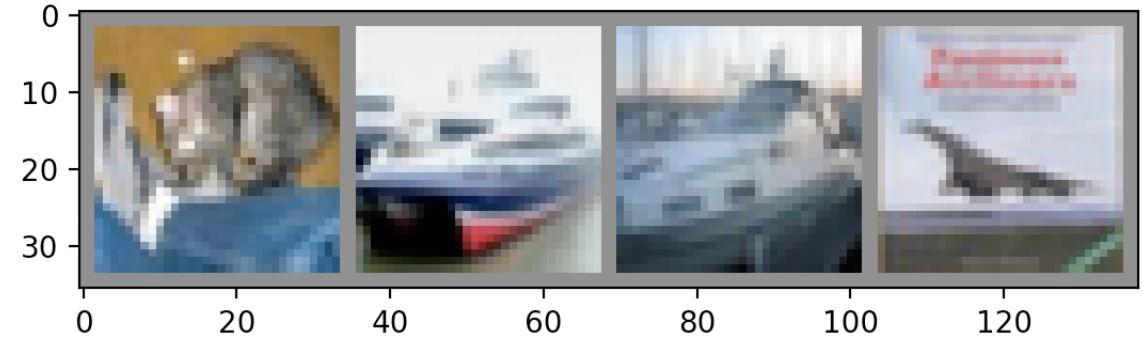

```
bird  bird  bird truck
```

- (3)

```
[1,  1000] loss : 2.298
[1,  2000] loss : 2.155
[1,  3000] loss : 1.936
[1,  4000] loss : 1.795
[1,  5000] loss : 1.719
[1,  6000] loss : 1.668
[1,  7000] loss : 1.613
[1,  8000] loss : 1.583
[1,  9000] loss : 1.499
[1, 10000] loss : 1.497
[1, 11000] loss : 1.465
[1, 12000] loss : 1.478
[2,  1000] loss : 1.380
[2,  2000] loss : 1.393
[2,  3000] loss : 1.384
[2,  4000] loss : 1.342
[2,  5000] loss : 1.331
[2,  6000] loss : 1.334
[2,  7000] loss : 1.334
[2,  8000] loss : 1.315
[2,  9000] loss : 1.316
[2, 10000] loss : 1.300
[2, 11000] loss : 1.297
[2, 12000] loss : 1.287
Finished Training
```

⇨ 적절히 학습되고 있다. Loss값이 점점 줄어들기 때문이다.

- (4)




```
GroundTurth:    cat  ship  ship plane
```

⇨ 학습을 확인하기 위해 가져온 테스트 집합의 실제 분류

- (5)

```
Predicted:    cat  ship   car plane
```

⇨ 정의한 신경망을 통해 예측한 테스트 집합의 예측 분류

- (6)

```
Accuracy of the network on the 10000 test images: 54 %
```

⇨ 전체 테스트 집합을 적용시켜 얻어낸 일반화 성능,

⇨ 일반화 성능 : 54%

- (7)

```
Accuracy of plane : 73 %
Accuracy of   car : 71 %
Accuracy of  bird : 31 %
Accuracy of   cat : 29 %
Accuracy of  deer : 39 %
Accuracy of   dog : 51 %
Accuracy of  frog : 77 %
Accuracy of horse : 63 %
Accuracy of  ship : 63 %
Accuracy of truck : 43 %
```

⇨ 각 부류별 일반화 성능은 Plane 73 %, Car 71%, Bird 31%, Cat 29%, Deer 39%, Dog 51%, Frog 77%, Horse 63%, Ship 63% , Truck 43% 이다.

3

- **# 3-(1)** INPUT – CONV(32 3*3) - CONV(32 3*3) – RELU – POOL - CONV(32 3*3) - CONV(32 3*3) – RELU – POOL - OUTPUT

  <코드>

```
def __init__(self):
    super(Net, self).__init__()
    # 커널 정의
    # 3채널을 입력받을 수 있도록 적의
    self.conv1 = nn.Conv2d(3, 32, 3)     # 3 * 32 * 3 * 3
    # 풀링층 정의
    self.pool = nn.MaxPool2d(2, 2)
    self.conv2 = nn.Conv2d(32, 32, 3)    # 32 * 32 * 3 * 3
    # fully connections 정의
    self.fc1 = nn.Linear(32* 5 * 5, 10)

# 순전파 함수에 신경망의 구조를 정의
def forward(self, x):
    # 컨볼루션 연산을 통해 특징 추출 후 풀링 연산하여 subsampling
    # 구조 conv1 - conv2 - relu - pool - conv2 - conv2 - relu - pool - fc - output
    x = self.pool(F.relu(self.conv2(self.conv1(x))))
    x = self.pool(F.relu(self.conv2(self.conv2(x))))
    x = x.view(-1, 32*5*5)

    x = self.fc1(x)
    return x
```

  ⇨ 특징맵의 channel수 * (특징맵의 크기)변화는 3 * (32 * 32) => 32 * (30 * 30) => 32 * (28 * 28) => 32 * (14 * 14) => 32 * (12 * 12) => 32 * (10 * 10) => 32 * (5 * 5) => 10 * (1 * 1) 이 된다.

  ⇨ 위의 변화는 이와 같은 연산을 통해 변화하게 된다
  3 * (32 * 32) conv에 의해=> 32 * (30 * 30) conv에 의해=> 32 * (28 * 28) 풀링 연산에 의해=> 32 * (14 * 14) conv에 의해=> 32 * (12 * 12) conv에 의해=> 32 * (10 * 10) 풀링 연산에 의해=> 32 * (5 * 5) 을 fully connection연산이 가능하도록 재배열 => 10 * (1 * 1)이 된다

  ⇨ 컨볼루션연산과 풀링연산의 반복을 통해 얻은 특징맵에 fully connection 연산을 통해 output을 얻어낸다.

<실행 결과>

```
[1,  1000] loss : 2.128
[1,  2000] loss : 1.811
[1,  3000] loss : 1.645
[1,  4000] loss : 1.559
[1,  5000] loss : 1.448
[1,  6000] loss : 1.409
[1,  7000] loss : 1.388
[1,  8000] loss : 1.357
[1,  9000] loss : 1.331
[1, 10000] loss : 1.308
[1, 11000] loss : 1.258
[1, 12000] loss : 1.233
[2,  1000] loss : 1.233
[2,  2000] loss : 1.209
[2,  3000] loss : 1.225
[2,  4000] loss : 1.195
[2,  5000] loss : 1.199
[2,  6000] loss : 1.188
[2,  7000] loss : 1.163
[2,  8000] loss : 1.193
[2,  9000] loss : 1.184
[2, 10000] loss : 1.162
[2, 11000] loss : 1.174
[2, 12000] loss : 1.170
Finished Training
Accuracy of the network on the 10000 test images: 58 %
Accuracy of plane : 49 %
Accuracy of   car : 82 %
Accuracy of  bird : 38 %
Accuracy of   cat : 26 %
Accuracy of  deer : 43 %
Accuracy of   dog : 71 %
Accuracy of  frog : 84 %
Accuracy of horse : 65 %
Accuracy of  ship : 62 %
Accuracy of truck : 62 %
```

⇨ 학습이 올바르게 되었다.

⇨ 일반화 성능 58 % 로 2번의 신경망보다 더 높은 일반화 성능을 갖는다.

- **# 3-(2)** 2번 문제의 신경망에 Adam 최적화

<코드>

```
optimizer = optim.Adam(net.parameters(), lr = 0.001, betas=(0.9, 0.999), eps =1e-3)
```

⇨ 신경망을 최적화하는 optimizer를 Adam을 정의한다.

⇨ 기본 하이퍼 매개변수를 사용

<실행결과>

```
[1,  1000] loss : 2.040
[1,  2000] loss : 1.745
[1,  3000] loss : 1.648
[1,  4000] loss : 1.539
[1,  5000] loss : 1.530
[1,  6000] loss : 1.463
[1,  7000] loss : 1.435
[1,  8000] loss : 1.417
[1,  9000] loss : 1.367
[1, 10000] loss : 1.356
[1, 11000] loss : 1.346
[1, 12000] loss : 1.336
[2,  1000] loss : 1.267
[2,  2000] loss : 1.255
[2,  3000] loss : 1.238
[2,  4000] loss : 1.265
[2,  5000] loss : 1.244
[2,  6000] loss : 1.238
[2,  7000] loss : 1.258
[2,  8000] loss : 1.247
[2,  9000] loss : 1.218
[2, 10000] loss : 1.213
[2, 11000] loss : 1.219
[2, 12000] loss : 1.233
Finished Training
```

```
Accuracy of the network on the 10000 test images: 56 %
Accuracy of plane : 56 %
Accuracy of   car : 65 %
Accuracy of  bird : 35 %
Accuracy of   cat : 48 %
Accuracy of  deer : 35 %
Accuracy of   dog : 51 %
Accuracy of  frog : 80 %
Accuracy of horse : 58 %
Accuracy of  ship : 75 %
Accuracy of truck : 56 %
```

⇨ 학습이 올바르게 되었다.

⇨ 2번 최적화 함수인 SGD + momentum 보다 조금 더 좋은 일반화 성능을 갖는다.

- **# 3-(3)** 데이터 확대 방법들 중 하나를 적용한 후, 2번 문제의 신경망 학습

<코드>

```
transform2 =transforms.Compose([transforms.RandomHorizontalFlip( p = 0.5 ),
    transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
trainset+=torchvision.datasets.CIFAR10(root = './data', train = True, download = True,
    transform = transform2)
```

⇨ 데이터 확대 기법 중 반전을 사용하였다.

⇨ Transform2 를 추가로 정의하여 반전된 훈련데이터들을 trainset에 추가하였다.

<실행결과>

```
[1,  1000] loss : 2.293
[1,  2000] loss : 2.115
[1,  3000] loss : 1.935
[1,  4000] loss : 1.799
[1,  5000] loss : 1.705
[1,  6000] loss : 1.639
[1,  7000] loss : 1.586
[1,  8000] loss : 1.537
[1,  9000] loss : 1.503
[1, 10000] loss : 1.529
[1, 11000] loss : 1.489
[1, 12000] loss : 1.452
[1, 13000] loss : 1.409
[1, 14000] loss : 1.411
[1, 15000] loss : 1.393
[1, 16000] loss : 1.362
[1, 17000] loss : 1.365
[1, 18000] loss : 1.343
[1, 19000] loss : 1.320
[1, 20000] loss : 1.292
[1, 21000] loss : 1.251
[1, 22000] loss : 1.282
[1, 23000] loss : 1.260
[1, 24000] loss : 1.247
[1, 25000] loss : 1.224
```

```
[2,  1000] loss : 1.207
[2,  2000] loss : 1.228
[2,  3000] loss : 1.202
[2,  4000] loss : 1.192
[2,  5000] loss : 1.193
[2,  6000] loss : 1.168
[2,  7000] loss : 1.162
[2,  8000] loss : 1.162
[2,  9000] loss : 1.163
[2, 10000] loss : 1.137
[2, 11000] loss : 1.164
[2, 12000] loss : 1.129
[2, 13000] loss : 1.130
[2, 14000] loss : 1.083
[2, 15000] loss : 1.147
[2, 16000] loss : 1.131
[2, 17000] loss : 1.098
[2, 18000] loss : 1.094
[2, 19000] loss : 1.090
[2, 20000] loss : 1.118
[2, 21000] loss : 1.118
[2, 22000] loss : 1.087
[2, 23000] loss : 1.118
[2, 24000] loss : 1.067
[2, 25000] loss : 1.054
Finished Training
```

```
Accuracy of the network on the 10000 test images: 57 %
Accuracy of plane : 64 %
Accuracy of   car : 68 %
Accuracy of  bird : 49 %
Accuracy of   cat : 22 %
Accuracy of  deer : 41 %
Accuracy of   dog : 82 %
Accuracy of  frog : 53 %
Accuracy of horse : 61 %
Accuracy of  ship : 73 %
Accuracy of truck : 61 %
```

⇨ 추가한 훈련집합을 이용해 적절히 학습되었다.

⇨ 일반화 성능 57%로 훈련집합의 확대에 의해 2번보다 더 높은 일반화 성능을 갖는다.

- **# 3-(4)** 2번 문제의 신경망에 CONV층마다 배치정규화를 적용

<코드>

```
self.conv2_bn = nn.BatchNorm2d(16)
```

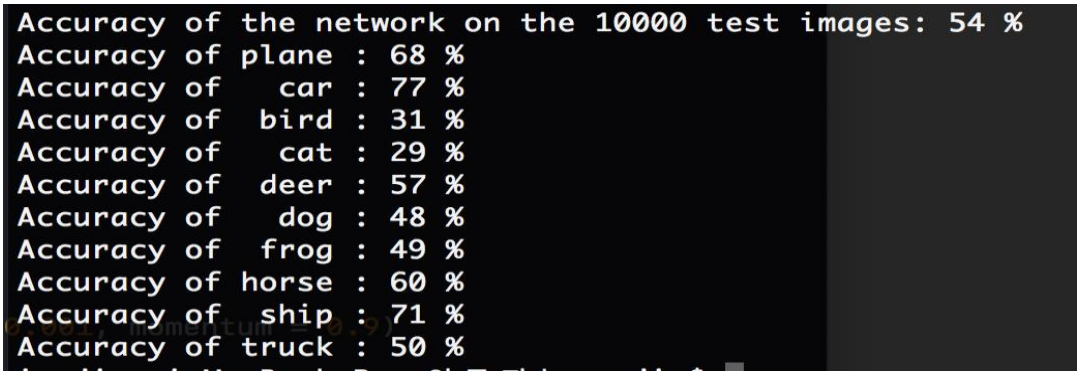
⇨ 신경망 생성자에 배치정규화 함수를 정의한다.

```
x = self.pool(F.relu(self.conv1_bn(self.conv1(x))))
x = self.pool(F.relu(self.conv2_bn(self.conv2(x))))
```

⇨ 신경망의 Forward과정에 conv층 마다 배치정규화를 적용시킨다.

<실행결과>

```
[1,  1000] loss : 2.213
[1,  2000] loss : 1.959
[1,  3000] loss : 1.830
[1,  4000] loss : 1.774
[1,  5000] loss : 1.686
[1,  6000] loss : 1.656
[1,  7000] loss : 1.575
[1,  8000] loss : 1.589
[1,  9000] loss : 1.544
[1, 10000] loss : 1.476
[1, 11000] loss : 1.492
[1, 12000] loss : 1.464
[2,  1000] loss : 1.464
[2,  2000] loss : 1.394
[2,  3000] loss : 1.403
[2,  4000] loss : 1.404
[2,  5000] loss : 1.368
[2,  6000] loss : 1.363
[2,  7000] loss : 1.304
[2,  8000] loss : 1.340
[2,  9000] loss : 1.352
[2, 10000] loss : 1.302
[2, 11000] loss : 1.313
[2, 12000] loss : 1.300
Finished Training
```



⇨ 적절히 학습되었다.

⇨ 일반화 성능으로 보아 2번과 유사한 일반화 성능을 갖는다.

⇨ 배치 정규화를 통해 신경망의 그레이디언트 흐름이 개선되고, 높은 학습률을 적용시켜 수렴속도를 높일 수 있으며 초기화에 대한 의존도가 감소하게 된다.

- **# 3-(5)** 2번 문제의 신경망에 로그우도 손실함수를 적용

<코드>

```
criterion = nn.NLLLoss()
loss = criterion(outputs, labels)
```

⇨ 로그우도만 적용시켜보았다.

<실행결과>

```
[1,  1000] loss : nan
[1,  2000] loss : nan
[1,  3000] loss : nan
[1,  4000] loss : nan
[1,  5000] loss : nan
[1,  6000] loss : nan
[1,  7000] loss : nan
[1,  8000] loss : nan
[1,  9000] loss : nan
[1, 10000] loss : nan
[1, 11000] loss : nan
[1, 12000] loss : nan
[2,  1000] loss : nan
[2,  2000] loss : nan
[2,  3000] loss : nan
[2,  4000] loss : nan
[2,  5000] loss : nan
[2,  6000] loss : nan
[2,  7000] loss : nan
[2,  8000] loss : nan
[2,  9000] loss : nan
[2, 10000] loss : nan
[2, 11000] loss : nan
[2, 12000] loss : nan
Finished Training
```

```
Accuracy of the network on the 10000 test images: 10 %
Accuracy of plane : 100 %
Accuracy of   car :  0 %
Accuracy of  bird :  0 %
Accuracy of   cat :  0 %
Accuracy of  deer :  0 %
Accuracy of   dog :  0 %
Accuracy of  frog :  0 %
Accuracy of horse :  0 %
Accuracy of  ship :  0 %
Accuracy of truck :  0 %
```

⇨ Loss 가 nan 값과 10 %의 낮은 일반화 성능이 나왔다.

그래서 최댓값이 아닌 값을 억제하여 0에 가깝게 만들어주는 softmax 중 효율적인 LogSoftmax함수와 함께 사용하였다.

<코드>

```
criterion = nn.NLLLoss()
m = nn.LogSoftmax()
loss = criterion(m(outputs), labels)
```

⇨ Logsoftmax함수를 정의하여 로그우도를 통해 손실을 구하기 전에 logsoftmax를 통해 최댓값이 아닌 값을 억제해준다.

<실행결과>

```
[1,  1000] loss : 2.287
[1,  2000] loss : 2.082
[1,  3000] loss : 1.870
[1,  4000] loss : 1.770
[1,  5000] loss : 1.661
[1,  6000] loss : 1.627
[1,  7000] loss : 1.576
[1,  8000] loss : 1.551
[1,  9000] loss : 1.504
[1, 10000] loss : 1.511
[1, 11000] loss : 1.472
[1, 12000] loss : 1.476
[2,  1000] loss : 1.404
[2,  2000] loss : 1.413
[2,  3000] loss : 1.410
[2,  4000] loss : 1.333
[2,  5000] loss : 1.355
[2,  6000] loss : 1.335
[2,  7000] loss : 1.336
[2,  8000] loss : 1.348
[2,  9000] loss : 1.288
[2, 10000] loss : 1.272
[2, 11000] loss : 1.267
[2, 12000] loss : 1.297
Finished Training
```

```
Accuracy of the network on the 10000 test images: 55 %
Accuracy of plane : 54 %
Accuracy of   car : 82 %
Accuracy of  bird : 30 %
Accuracy of   cat : 26 %
Accuracy of  deer : 42 %
Accuracy of   dog : 44 %
Accuracy of  frog : 79 %
Accuracy of horse : 64 %
Accuracy of  ship : 77 %
Accuracy of truck : 54 %
```

⇨ 적절히 학습되었다.

⇨ 일반화 성능 55%로 2번과 유사한 일반화성능을 갖는다

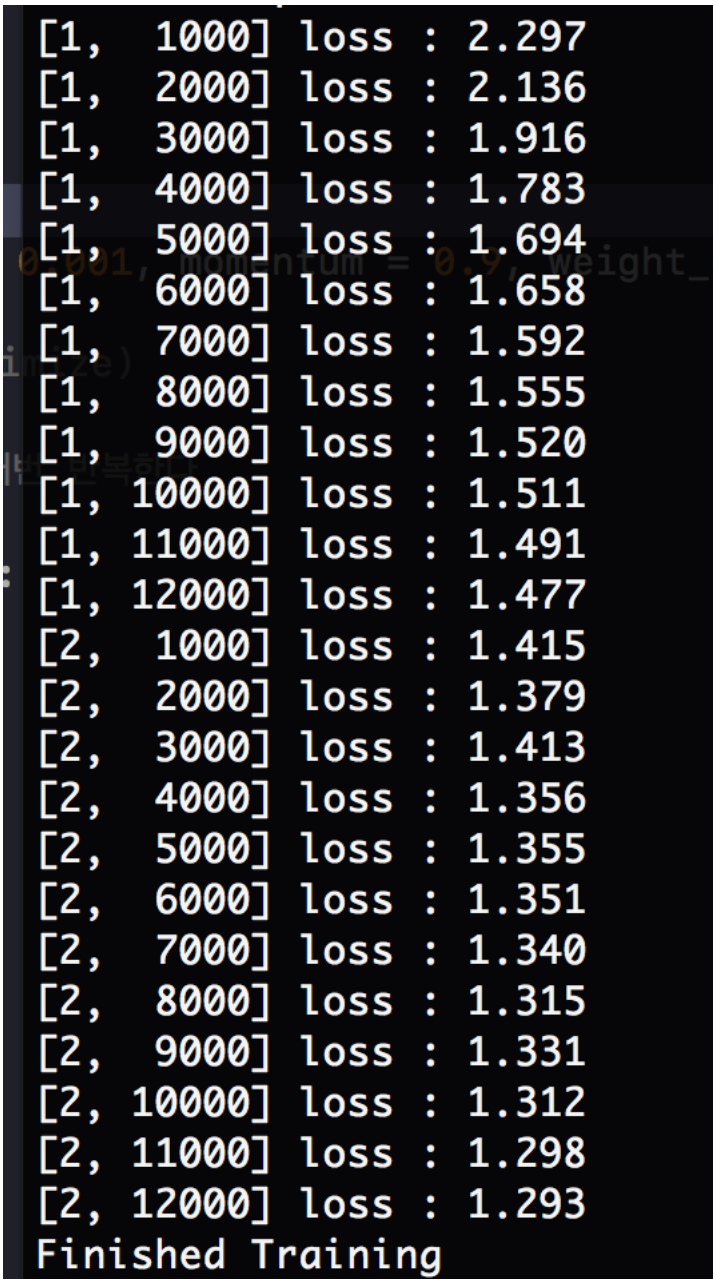
- **# 3-(6)** 2번 문제의 신경망에 L2 놈 적용

<코드>

```
optimizer = optim.SGD(net.parameters(), lr = 0.001, momentum = 0.9, weight_decay = 1e-6)
```

⇨ Weight_decay 를 1e-6으로 설정하여 가중치 감쇠 효과를 주었다.

<실행결과>



```
Accuracy of the network on the 10000 test images: 55 %
Accuracy of plane : 73 %
Accuracy of   car : 68 %
Accuracy of  bird : 37 %
Accuracy of   cat : 31 %
Accuracy of  deer : 39 %
Accuracy of   dog : 41 %
Accuracy of  frog : 75 %
Accuracy of horse : 69 %
Accuracy of  ship : 56 %
Accuracy of truck : 59 %
```

⇨ 적절히 학습되었다.

⇨ 2번과 유사한 일반화 성능을 갖는다.

4

```python
# -*- coding: utf -8 -*-
import numpy as np

def softmax(x):
    e_x = np.exp(x - np.max(x))
    return e_x/e_x.sum()

x = np.array([0.4, 2.0, 0.001, 0.32])

print("소프트맥스함수 적용 결과 : {}".format(softmax(x)))

```

```
소프트맥스함수 적용 결과 : [0.13250053 0.65627943 0.08890663 0.12231341]
```

5

```python
# y가 예측값 , t 가 ans
def MSE(y, t):
    temp = 0
    for i in range(len(t)):
        temp += pow(y[i]-t[i] ,2)
    return 0.5 * temp

def cross_entropy_error(y,t):
    delta = 1e-7
    temp = 0
    for i in range(len(t)):
        temp += t[i] * np.log(y[i] + delta)
    return -temp

def loglikelihood(y, t):
    for i in range(len(t)):
        if t[i] == 1:
            break
    return -np.log2(y[i])

prediction = [0.001, 0.9, 0.001, 0.098]
ans = [0, 0, 0, 1]

print("MSE")
print(MSE(prediction, ans))
print("교차엔트로피")
print(cross_entropy_error(prediction, ans))
print("로그우도")
print(loglikelihood(prediction,ans))
```

```
MSE
0.811803
교 차 엔 트 로 피
2.3227867799039226
로 그 우 도
3.3510744405468786
```