HW 06 – REPORT

소속 : 정보컴퓨터공학부

학번 : 201824523

이름 : 안혜준

1. 서론

Pytorch로 Convolutional Neural Network(CNN)를 구축하여 CIFAR-10 이미지 셋을 분류하고 성능을 평가해본다.

현대의 이미지 처리는 인공지능 기술이 많이 쓰이는 추세이다. 그 중 이미지 분류(라벨링)는 인공 신경망을 사용한 딥러닝으로 해결 가능하다. CIFAR-10 dataset은 32x32 픽셀의 60,000개 컬러 이미지가 포함되어 있으며, 10개의 클래스로 분류되어 있다. 데이터를 train / val / test 3부분으로 나누어 신경망 모델에 학습을 진행한다. 학습이 진행되며 정확도가 향상되는 것을 확인한다.

실습은 5개의 파트로 나누어져 진행된다.

Part I. Preparation

Part II. Barebones PyTorch

Part III. PyTorch Module API

Part IV. PyTorch Sequential API

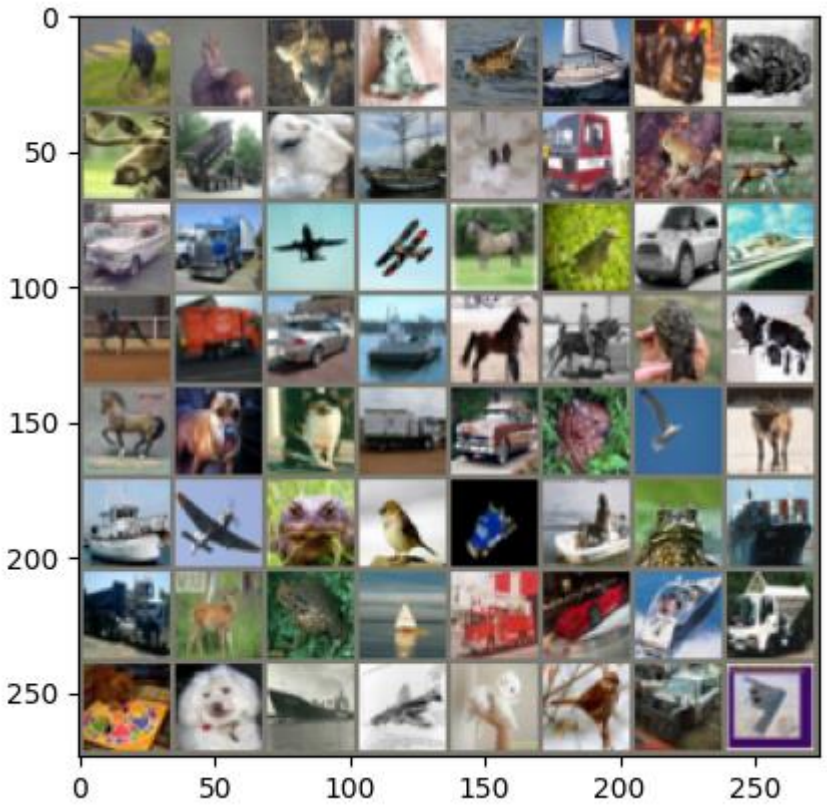
Part V. CIFAR-10 open-ended challenge

Part 2에서 4로 가며 추상화 수준이 높아지게 된다. Low level로 PyTorch를 다뤄보고, nn.Module과 nn.Sequential을 사용해보며 PyTorch를 익힌다. 최종적으로 Part 5에서 직접 모델을 구축하고 hyperparameter를 조정해 높은 정확도를 가지는 모델을 만들어본다.

2. 본론

1. Part I. Preparation

Torchvision 패키지로부터 CIFAR-10 datasets을 받아와 저장한다. 받아온 이미지들의 일부를 확인할 수 있다.



2. Part II. Barebones PyTorch

예제로 주어지는 two layers의 fully connected neural network와, 직접 만들어 하는 three layers의 convolutional neural network가 존재한다.

먼저 fully connected neural network는 입력에 첫번째 weight를 곱하고, relu 함수(비선형 activation 함수)를 적용하여 두번째 weight을 곱함으로써 간단하게 구현이 가능하다. Back propagation의 경우 PyTorch가 자동으로 적용해주므로 신경쓰지 않아도 된다. Input layer와 output layer 사이에 hidden layer가 하나 들어 있는 간단한 구조이다.

[Question 1]

구현해야하는 three_layer_convnet의 구조는 다음과 같다.

1. A convolutional layer (with bias)
2. ReLU nonlinearity
3. A convolutional layer (with bias)
4. ReLU nonlinearity
5. Fully-connected layer with bias

따라서 scores의 계산은 다음과 같이 이루어진다.

```
import torch.nn.functional as F

scores = F.relu(F.conv2d(x, conv_w1, conv_b1, padding=2))
scores = F.relu(F.conv2d(scores, conv_w2, conv_b2, padding=1))
scores = flatten(scores).mm(fc_w) + fc_b
```

이때 padding의 값에 따라 첫번째 커널을 5x5, 두번째 커널이 3x3으로 해야 이미지의 크기가 보존이 될 것이다. 두번째 레이어를 지났을 때는 필터와 이미지(2차원)의 형태로 남아 있기 때문에 fully Connected layer를 위해 flatten으로 1차원으로 바꾼 뒤 계산한다.

테스트를 위해 64개의 minibatch로 3개 채널(rgb)을 가진 이미지(32x32)를 넣었을 때, 결과의 크기가 64x10이 나오는 것을 확인할 수 있다.

```
torch.Size([64, 10])
```

64개의 이미지에 대한 각각의 라벨(10가지 종류) 정보를 가지고 있다는 걸을 알 수 있다.

[Question 2]

실행을 위해 각 weight, bias를 랜덤으로 초기화하여 실행한다.

```
conv_w1 = random_weight((channel_1, 3, 5, 5))
conv_b1 = zero_weight((channel_1,))
conv_w2 = random_weight((channel_2, channel_1, 3, 3))
conv_b2 = zero_weight((channel_2,))
fc_w = random_weight((32 * 32 * 16, 10))
fc_b = zero_weight((10, ))
```

결과는 다음과 같다.

```
Iteration 0, loss = 3.0858
Checking accuracy on the val set
Got 117 / 1000 correct (11.70%)

Iteration 100, loss = 1.7278
Checking accuracy on the val set
Got 339 / 1000 correct (33.90%)

Iteration 200, loss = 1.8941
Checking accuracy on the val set
Got 376 / 1000 correct (37.60%)

Iteration 300, loss = 1.9220
Checking accuracy on the val set
Got 405 / 1000 correct (40.50%)

Iteration 400, loss = 1.7406
Checking accuracy on the val set
Got 423 / 1000 correct (42.30%)

Iteration 500, loss = 1.4956
Checking accuracy on the val set
Got 446 / 1000 correct (44.60%)

Iteration 600, loss = 1.5524
...
Iteration 700, loss = 1.4275
Checking accuracy on the val set
Got 466 / 1000 correct (46.60%)
```

반복을 거치며 validation에 대한 정확도가 개선이 되어 가는 모습을 볼 수 있다.

Validation에 대한 최종 정확도는 46.60%이다.

3. Part III. PyTorch Module API

함수를 만드는 것이 아닌 PyTorch의 nn.Module을 상속받는 모델 클래스를 만든다.

[Question 3]

ThreeLayerConvNet 클래스의 생성자를 만든다.

```
def __init__(self, in_channel, channel_1, channel_2, num_classes):
    super().__init__()

    self.conv1 = nn.Conv2d(in_channel, channel_1, 5, padding=2)
    self.conv2 = nn.Conv2d(channel_1, channel_2, 3, padding=1)
    self.fc = nn.Linear(channel_2 * 32 * 32, num_classes)

    nn.init.kaiming_normal_(self.conv1.weight)
    nn.init.kaiming_normal_(self.conv2.weight)
    nn.init.kaiming_normal_(self.fc.weight)
```

각 layer를 정의하고 사용될 weight을 초기화한다.

[Question 4]

ThreeLayerConvNet 클래스의 forward 메서드를 정의한다.

```
def forward(self, x):
    scores = F.relu(self.conv1(x))
    scores = F.relu(self.conv2(scores))
    scores = self.fc(flatten(scores))

    return scores
```

Class에 저장된 layer를 지난다. Activation function은 relu를 사용한다.

Question 1과 마찬가지로 테스트를 했을 때 크기는 64x10으로 나옴을 알 수 있다.

```
torch.Size([64, 10])
```

[Question 5]

객체를 만들고 모델 학습을 시킨다. 모델을 학습시키고 평가한다. optimizer로는 stochastic gradient descent(SGD)를 사용한다.

```
model = ThreeLayerConvNet(3, channel_1, channel_2, 10)
optimizer = optim.SGD(model.parameters(), lr=learning_rate)
```

결과는 다음과 같다.

```
Iteration 0, loss = 4.2430
Checking accuracy on validation set
Got 111 / 1000 correct (11.10)

Iteration 100, loss = 1.9718
Checking accuracy on validation set
Got 328 / 1000 correct (32.80)

Iteration 200, loss = 1.7157
Checking accuracy on validation set
Got 386 / 1000 correct (38.60)

Iteration 300, loss = 1.8186
Checking accuracy on validation set
Got 419 / 1000 correct (41.90)

Iteration 400, loss = 1.5958
Checking accuracy on validation set
Got 428 / 1000 correct (42.80)

Iteration 500, loss = 1.6835
Checking accuracy on validation set
Got 456 / 1000 correct (45.60)

Iteration 600, loss = 1.5961
...
Iteration 700, loss = 1.4772
Checking accuracy on validation set
Got 472 / 1000 correct (47.20)
```

Validation에 대한 최종 정확도는 47.20%이다.

4. Part IV. PyTorch Sequential API

더 간단히 모델을 만들 수 있는 nn.Sequential을 사용한다. Part III와 다르게, 각 layer에 bias를 추가하고 optimizer로 Nesterov momentum 0.9인 SGD를 사용해 구현한다.

[Question 6]

각 layer에 bias=True 옵션을 추가하고, optim.SGD에 Nesterov=True, momentum=0.9 옵션을 추가한다.

```
model = nn.Sequential(
    nn.Conv2d(3, channel_1, 5, padding=2, bias=True),
    nn.ReLU(),
    nn.Conv2d(channel_1, channel_2, 3, padding=1, bias=True),
    nn.ReLU(),
    Flatten(),
    nn.Linear(channel_2 * 32 * 32, 10, bias=True)
)

optimizer = optim.SGD(model.parameters(), lr=learning_rate,
                      momentum=0.9, nesterov=True)
```

결과는 다음과 같다.

```
Iteration 0, loss = 2.3127
Checking accuracy on validation set
Got 143 / 1000 correct (14.30)

Iteration 100, loss = 1.4948
Checking accuracy on validation set
Got 453 / 1000 correct (45.30)

Iteration 200, loss = 1.3789
Checking accuracy on validation set
Got 505 / 1000 correct (50.50)

Iteration 300, loss = 1.2711
Checking accuracy on validation set
Got 491 / 1000 correct (49.10)

Iteration 400, loss = 1.5587
Checking accuracy on validation set
Got 527 / 1000 correct (52.70)
```

```
Iteration 500, loss = 1.2164
Checking accuracy on validation set
Got 550 / 1000 correct (55.00)

Iteration 600, loss = 1.4301
...
Iteration 700, loss = 1.1563
Checking accuracy on validation set
Got 564 / 1000 correct (56.40)
```

Validation에 대한 최종 정확도는 56.40%이다. Optimizer-SGD 옵션을 변경하고 bias를 추가하니 정확도가 증가하였다.

5. Part V. CIFAR-10 open-ended challenge

[Question 7]

이제 앞서 사용한 것들을 토대로 정확도가 70%를 넘도록 모델을 생성해본다.

```
channel_1 = 32
channel_2 = 64
learning_rate = 1e-2

model = nn.Sequential(
    nn.Conv2d(3, channel_1, 3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d((2, 2), (2, 2)),
    nn.Conv2d(channel_1, channel_2, 3, padding=1),
    nn.MaxPool2d((2, 2), (2, 2)),
    nn.ReLU(),
    Flatten(),
    nn.Linear(channel_2 * 8 * 8, 128),
    nn.ReLU(),
    nn.Linear(128, 10)
)

optimizer = optim.SGD(model.parameters(), lr=learning_rate,
                      momentum=0.9, nesterov=True)
```

32 채널로 3x3 커널의 Convolution을 한다. maxpooling으로 크기를 줄이고 다시 64채널로 3x3 커널의 Convolution을 수행한다. Fully connected linear layer를 2개 적용한다. 두 layer간의 hidden layer 사이즈는 128이다.

학습은 총 10 epochs 만큼 수행한다.

```
Iteration 0, loss = 2.2994
Checking accuracy on validation set
Got 113 / 1000 correct (11.30)

Iteration 100, loss = 1.6375
Checking accuracy on validation set
Got 418 / 1000 correct (41.80)

Iteration 200, loss = 1.4135
Checking accuracy on validation set
Got 492 / 1000 correct (49.20)

Iteration 300, loss = 1.2248
Checking accuracy on validation set
Got 534 / 1000 correct (53.40)

Iteration 400, loss = 1.1917
Checking accuracy on validation set
Got 574 / 1000 correct (57.40)

Iteration 500, loss = 1.0524
Checking accuracy on validation set
Got 598 / 1000 correct (59.80)

Iteration 600, loss = 1.0764
...
Iteration 700, loss = 0.2499
Checking accuracy on validation set
Got 735 / 1000 correct (73.50)
```

10 epochs 이후 validation에 대한 최종 정확도는 73.50%이다.

이제 test set에 대해 정확도를 평가해본다.

```
Checking accuracy on test set
Got 7129 / 10000 correct (71.29)
```

Test에 대한 최종 정확도는 71.29%이다.

3. 결론

PyTorch를 사용해 CNN 실습을 하였다. CNN은 Convolution 연산을 수행하는 신경망이다. 이미지는 2차원 데이터로 개별적인 픽셀의 정보뿐만 아니라, 주변 픽셀들과의 관계 또한 중요하다. 그렇기 때문에 2d convolution을 수행하여 중요한 정보들로 압축하여 학습을 시킴으로써 더 좋은 모델을 만들 수 있다.

최종 모델을 만들기 위해 convolution layer와 maxpooling layer, linear fully-connected layer를 사용하였다. 처음 three layer convNet을 만들어 1 epoch만 수행했을 때 정확도가 50%를 못 넘겼는데, 최종 모델을 10 epochs 수행하자 정확도가 70%가 넘게 되었다. 모델을 만들 때는 여러 layer를 사용하여 많은 epochs을 수행하면 높은 정확도를 얻을 수 있다. 높은 정확도의 모델을 얻기 위해 다양한 시도를 해보아야 함을 알 수 있었다.