HW 05 – REPORT

소속 :

학번 :

이름 :

1. 서론

실습 목표 및 이론적 배경 기술 (1~2페이지)

실습 목표는 CNN(Convolution Neural Network)를 사용하여 CIFAR-10 이미지를 학습시키고, 정확도를 확인하여 CNN의 원리와 구현 방법을 알아보는 것이다.

퍼셉트론은 인공 신경망을 구성하는 기본 단위로, 입력 $x_1, \dots, x_n$ 각 항에 가중치 $w_1, \dots, w_n$을 곱하고 편향을 더한 다음, 비선형 함수(활성화 함수)를 사용하여 최종 출력을 만들어 낸다. 즉 출력 y는 다음과 같이 계산할 수 있다.

$$y = \tau(\mathrm{w^T x + b})$$

하나의 퍼셉트론은 일종의 선형 분류기이기 때문에 선형분리가 불가능한 데이터에는 한계가 있다. 따라서 이러한 퍼셉트론을 여러 층으로 구성하여 선형 분리가 불가능한 데이터도 분류할 수 있도록 할 수 있다. 층이 2개인 다층 퍼셉트론의 출력은 다음과 같이 계산될 수 있다.

$$y = \tau_2(\mathrm{w}_2^T(\tau_1(\mathrm{w}_1^T \mathrm{x} + \mathrm{b}_1) + \mathrm{b}_2))$$

활성화 함수에는 sigmoid, tanh, ReLU 등이 있다.

$$sigmoid(x) = \frac{1}{1+e^{-x}}, \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, ReLU(x) = \max(0, x)$$

손실 함수는 딥러닝이 추정한 값이 정답과 비교할 때 얼마나 정확한지 판단할 때 사용된다. 이러한 손실함수에는 평균제곱오차, 교차 엔트로피 등이 있다.

평균제곱오차는 출력값이 $y_1, y_2, \dots, y_n$이고 정답이 $t_1, t_2, \dots, t_n$일 때 다음과 같다.

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - t_i)^2$$

교차 엔트로피는 두 가지 확률 분포가 얼마나 비슷한지를 숫자로 나타낸 것이다. 정보 엔트로피는 발생 확률에 따라 정보량의 가중평균을 구한 것으로, 정보량의 기댓값이다. 교차 엔트로피는 정보 엔트로피에서, 정보량에 쓰이는 확률 분포와 가중 평균에 쓰이는 확률 분포를 다르게 설정한 것이다. 딥러닝 모델의 추정 확률 분포 P, 정답 확률 분포 Q가 있을 때 교차 엔트로피 H(P,Q)는 다음과 같이 계산할 수 있다

$$H(P, Q) = -\sum p_i log\ q_i$$

경사하강법은 함수의 기울기를 반복 계산하면서 이 기울기에 따라 함숫값이 낮아지는 방향으로 이동(퍼셉트론에서 가중치와 편향을 조정)하는 알고리즘이다. 순전파는 입력 데이터와 현재 파라미터값을 이용해 손실 함수를 계산하는 것을 의미한다. 역전파는 손실 함수값에 영향을 미친 성분에 대해 손실 기울기를 계산하고, 손실 기울기를 이용하여 실제로 파라미터를 조정한다. 손실 함수를 L이라고 할 때 한 파라미터 w는 다음과 같이 조정된다.

$$w_1 = w_1 - \alpha \frac{\partial L}{\partial w_1}$$

여기서 α는 학습률로, 손실 기울기에 얼마나 비례해서 조정할지를 나타낸다. 보통 0.01, 0.001 등과 같은 매우 작은 값으로 설정한다. SGD 등에서는 학습률이 고정이지만, AdaGrad, RMSprop, Adam은 파라미터 상태에 따라 학습률을 조정한다. $\frac{\partial L}{\partial w}$ 는 미분의 연쇄 법칙을 사용하여 다음과 같이 계산된다.

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_1} = \frac{\partial L}{\partial y}\frac{\partial \mathrm{w^T x}}{\partial w_1}$$

이러한 다층 퍼셉트론은 한 퍼셉트론이 인접한 양쪽 계층의 모든 퍼셉트론과 연결되어 있다. 그러나 이는 이미지를 학습시킬 때는 매우 큰 가중치 파라미터를 갖게 되고, 메모리 부담, 학습 속도 감소, 많은 데이터 필요 등의 문제가 발생할 수 있다. 따라서 이미지 학습에는 합성곱 신경망(Convolution Neural Network)를 사용한다.

합성곱 계층은 작은 사각 모양의 가중치 파라미터(합성곱 커널 가중치)를 이용하여 입력 픽셀값들로부터 출력 픽셀값을 계산한다. 출력 픽셀은 합성곱 커널 가중치의 영역에 들어가는 입력 픽셀값과 커널 가중치들을 짝지어 곱한 후 커널의 편향값과 합산해 구한다. 이때 입력 픽셀 행렬에 커널이 이동하며 반복 적용되면서 출력 픽셀을 계산한다. 합성곱 계층에서 출력 픽셀값을 계산하려고 할 때, 입력 픽셀값의 범위를 넘어서거나, 출력이 입력보다 줄어들 수 있다. 이를 해결하기 위해 입력 픽셀 가장자리에 0을 추가할 수 있고, 이를 패딩이라고 한다. 출력 크기는 입력 크기가 N, 커널 크기를 F, 보폭(stride)를 s라 할 때 다음과 같이 구할 수 있다.

$$\frac{N-F}{s} + 1$$

폴링은 일정 영역의 입력 픽셀값들로부터 그 최대치(MaxPooling)나 평균치(AvgPooling) 같은 대푯값을 구해 출력 픽셀로 생성하는 처리이다. 폴링 계층을 이용하면, 특히 합성곱 계층 사이에 배치되면 작은 해상도에 정보가 집약되게 만들게할 수 있다. 즉, 합성곱 계층이 미세한 변화에 휘둘리지 않고 필요한 패턴을 쉽게 포착할 수 있게 해준다.

2. 본론

실습 내용 및 결과 기술 (2페이지 이상)

(1) Q1 – Forward pass for three-layer ConvNet

```
x=F.conv2d(x,conv_w1,conv_b1,padding=2) # apply convolution(with w1 and b1) to x, padding=2
x=F.relu(x) # apply ReLU activation function
x=F.conv2d(x,conv_w2,conv_b2,padding=1) # apply convolution(with w2 and b2) to x, padding=1
x=F.relu(x) # apply ReLU activation function
x=flatten(x) # flatten x
x=x.mm(fc_w) # matrix multiplication(x*fc_w)
x=x+fc_b # matrix addition(x+fc_b)
scores = x
```

torch.nn.functional에 내장된 conv2d, relu를 사용하여 convolution 연산과 relu 연산을 하고, flatten을 적용한 pytorch tensor x와 가중치 fc_w를 곱하고 편향 fc_b를 더한다(fully connected network).

```
def three_layer_convnet_test():
    x = torch.zeros((64, 3, 32, 32), dtype=dtype)  # minibatch size 64, image size [3,

    conv_w1 = torch.zeros((6, 3, 5, 5), dtype=dtype)  # [out_channel, in_channel, kerne
    conv_b1 = torch.zeros((6,))  # out_channel
    conv_w2 = torch.zeros((9, 6, 3, 3), dtype=dtype)  # [out_channel, in_channel, kerne
    conv_b2 = torch.zeros((9,))  # out_channel

    # you must calculate the shape of the tensor after two conv layers, before the ful
    fc_w = torch.zeros((9 * 32 * 32, 10))
    fc_b = torch.zeros(10)

    scores = three_layer_convnet(x, [conv_w1, conv_b1, conv_w2, conv_b2, fc_w, fc_b])
    print(scores.size())  # you should see [64, 10]
three_layer_convnet_test()
```

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

테스트 결과 최종 출력의 크기가 (64, 10)이 나왔다(미니배치 64개, 10개의 분류 카테고리).

(2) train two-layer network

한 개의 은닉층(4000개의 퍼셉트론을 가짐)을 가진 다층 퍼셉트론을 이용하여 CIFAR-10을 학습한다.

```
hidden_layer_size = 4000
learning_rate = 1e-2

w1 = random_weight((3 * 32 * 32, hidden_layer_size))
w2 = random_weight((hidden_layer_size, 10))

train_part2(two_layer_fc, [w1, w2], learning_rate)
```

```
Iteration 700, loss = 1.6867
Checking accuracy on the val set
Got 437 / 1000 correct (43.70%)
```

loss는 1.6867, 정확도는 43.7%이다.

(3) Q2 – Training a ConvNet : Initialize the parameter with random_weight and zero_weight

```
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

conv_w1 = random_weight((channel_1,3,5,5)) # initialze Convolutional layer (with bias
conv_b1 = zero_weight((channel_1))
conv_w2 = random_weight((channel_2,channel_1,3,3)) # initialize Convolutional layer (
conv_b2 = zero_weight((channel_2))
fc_w = random_weight((channel_2 * 32 * 32,10)) # initialize Fully-connected layer for
fc_b = zero_weight((10))

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
################################################################################
#                                END OF YOUR CODE                              #
################################################################################

params = [conv_w1, conv_b1, conv_w2, conv_b2, fc_w, fc_b]
train_part2(three_layer_convnet, params, learning_rate)
```

```
Iteration 700, loss = 1.6182
Checking accuracy on the val set
Got 477 / 1000 correct (47.70%)
```

loss는 1.6182, 정확도는 47.7%이다.

(4) Q3 – making three-layer ConvNet with nn.Module and pytorch.nn

```
class ThreeLayerConvNet(nn.Module):
    def __init__(self, in_channel, channel_1, channel_2, num_classes):
        super().__init__()
        ### Question 3
        ################################################################################
        # TODO: Set up the layers you need for a three-layer ConvNet with the  #
        # architecture defined above.                                          #
        ################################################################################
        # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

        self.conv1 = nn.Conv2d(in_channel,channel_1,(5,5),padding=2,bias=True) #C
        self.conv2 = nn.Conv2d(channel_1,channel_2,(3,3),padding=1,bias=True) #Co
        self.fc1 = nn.Linear(channel_2 * 32 * 32,num_classes) # Fully-connected l
        nn.init.kaiming_normal_(self.conv1.weight) # initialize weight metrices w
        nn.init.kaiming_normal_(self.conv2.weight) # initialize weight metrices w
        nn.init.kaiming_normal_(self.fc1.weight) # initialize weight metrices wit
```

nn.Module을 상속받고, nn.Conv2d를 이용하여 convolution layer를, nn.Linear를 사용하여 fully connected layer를 만들고, nn.init.kaiming_normal을 이용하여 각 계층의 가중치를 kiming normal initialization을 사용하여 초기화한다.

(5) Q4 – forward function using defined in __init__(in Q3)

```
x = F.relu(self.conv1(x)) # apply conv1 and relu
x = F.relu(self.conv2(x)) # apply conv2 and relu
x=flatten(x) # flatten
scores = self.fc1(x) # apply fully connected network.
```

입력 x에 conv1을 적용시키고 relu 함수를 적용한 다음, conv2를 적용시키고, relu 함수를 적용시킨다. 그리고 flatten을 사용하여 fully connected network에 입력 가능하게 하고,

Q3에서 만든 fc1를 적용시킨다

```
def test_ThreeLayerConvNet():
    x = torch.zeros((64, 3, 32, 32), dtype=dtype)  # minibatch size 64, image size [3, 
    model = ThreeLayerConvNet(in_channel=3, channel_1=12, channel_2=8, num_classes=10)
    scores = model(x)
    print(scores.size())  # you should see [64, 10]
test_ThreeLayerConvNet()
```

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

테스트 결과 최종 출력의 크기가 (64,10) 이다.

(6) making and training two-layer fc model with subclass of nn.Module

```
hidden_layer_size = 4000
learning_rate = 1e-2
model = TwoLayerFC(3 * 32 * 32, hidden_layer_size, 10)
optimizer = optim.SGD(model.parameters(), lr=learning_rate)

train_part34(model, optimizer)
```

nn.Module을 상속받은 TwoLayerFC 클래스를 이용하여 32*32 RGB 이미지(CIFAR-10)를 학습시킨다. optimizer는 SGD로 설정한다.

```
Iteration 700, loss = 1.6824
Checking accuracy on validation set
Got 440 / 1000 correct (44.00)
```

loss는 1.6824, 정확도는 44%이다.

(7) Q5 - making and training three-layer convnet model with subclass of nn.Module (defined in Q3)

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

TheeLayerConvNet 클래스에 input channel 개수(3), 첫번째 합성곱 층의 channel 개수(32개), 두번째 합성곱 층의 channel 개수(16개) 초기값을 넣어 새로운 모델을 만든다. optimizer는 SGD를 사용하였다.

```
Iteration 700, loss = 1.4595
Checking accuracy on validation set
Got 473 / 1000 correct (47.30)
```

학습결과 loss는 1.4595, 정확도는 47.3%이다.

(8) making and training two-layer fc with nn.Sequential

```
model = nn.Sequential(
    Flatten(),
    nn.Linear(3 * 32 * 32, hidden_layer_size),
    nn.ReLU(),
    nn.Linear(hidden_layer_size, 10),
)

# you can use Nesterov momentum in optim.SGD
optimizer = optim.SGD(model.parameters(), lr=learning_rate,
                      momentum=0.9, nesterov=True)

train_part34(model, optimizer)
```

nn.Sequental을 사용하여 모델을 만든다. Flatten()을 적용하여 이미지를 1차원으로 만들고, (입력층)-퍼셉트론(nn.Linear) – 비선형 활성화 함수(nn.ReLU) – 퍼셉트론(nn.Linear)으로 배치하여 은닉층이 하나있는 모델을 만든다. optimizer는 SGD에서, Nesterov momentum을 0.9로 설정하여 사용한다.

```
Iteration 700, loss = 1.7032
Checking accuracy on validation set
Got 434 / 1000 correct (43.40)
```

학습 결과 loss는 1.7032이고, 정확도는 43.4%이다.

(9) Q6 – making and training three-layer ConvNet with nn.Sequential

```
model = nn.Sequential(
    nn.Conv2d(3,channel_1,(5,5),padding=2,bias=True), #Convolutional layer (with
    nn.ReLU(), #ReLU
    nn.Conv2d(channel_1,channel_2,(3,3),padding=1,bias=True), #Convolutional lay
    nn.ReLU(), #ReLU
    Flatten(), #Flatten to 1d
    nn.Linear(channel_2 * 32 * 32,10), #Fully-connected layer (with bias) to com
)


optimizer = optim.SGD(model.parameters(), lr=learning_rate,
                      momentum=0.9, nesterov=True) #optimize the model using stoc
```

nn.Sequential을 사용하여 모델을 만든다. nn.Conv2d를 사용하여 convolution layer를 만들고 nn.ReLU를 사용하여 비선형 활성화 함수로 ReLU를 적용한다. 마지막에 Flatten을 한 후 Fully connected Network로 보내어 최종 출력을 생성한다.

```
Iteration 700, loss = 1.1333
Checking accuracy on validation set
Got 577 / 1000 correct (57.70)
```

학습 결과 loss는 1.1333, 정확도는 57.7%이다.

(10) CIFAR-10 open-ended challenge

```
channel_1 = 64
channel_2 = 32
channel_3 = 32
channel_4 = 32
channel_5 = 16
channel_6 = 16

learning_rate = 3e-3

model = nn.Sequential(
    nn.Conv2d(3,channel_1,(5,5),padding="same",bias=True), # convolution layer(with bias) with 64 5x5 filter. same padding
    nn.BatchNorm2d(channel_1), # batch normalization
    nn.ReLU(), # ReLU
    nn.Conv2d(channel_1,channel_2,(3,3),padding="same",bias=True),  # convolution layer(with bias) with 32 3x3 filter. same padding
    nn.BatchNorm2d(channel_2),# batch normalization
    nn.ReLU(),# ReLU
    nn.Conv2d(channel_2,channel_3,(3,3),padding="same",bias=True), # convolution layer(with bias) with 32 3x3 filter. same padding
    nn.BatchNorm2d(channel_3), ,# batch normalization
    nn.ReLU(), # ReLU
    nn.MaxPool2d(2), # MaxPooling with 2x2
    nn.Conv2d(channel_3,channel_4,(3,3),padding="same",bias=True), # convolution layer(with bias) with 32 3x3 filter. same padding
    nn.BatchNorm2d(channel_4),# batch normalization
    nn.ReLU(),# ReLU
    nn.Conv2d(channel_4,channel_5,(3,3),padding="same",bias=True),# convolution layer(with bias) with 16 3x3 filter. same padding
    nn.BatchNorm2d(channel_5),# batch normalization
    nn.ReLU(),# ReLU
    nn.Conv2d(channel_5,channel_6,(3,3),padding="same",bias=True),# convolution layer(with bias) with 16 3x3 filter. same padding
    nn.BatchNorm2d(channel_6),# batch normalization
    nn.ReLU(),# ReLU
    nn.MaxPool2d(2),# MaxPooling with 2x2
    Flatten(), # Flatten
    nn.Linear(channel_6 * 8 * 8,10), #due to the 2 maxpooling(2), matrices are 8*8. apply fully connected network
    nn.BatchNorm1d(10) # batch normalization.
)
```

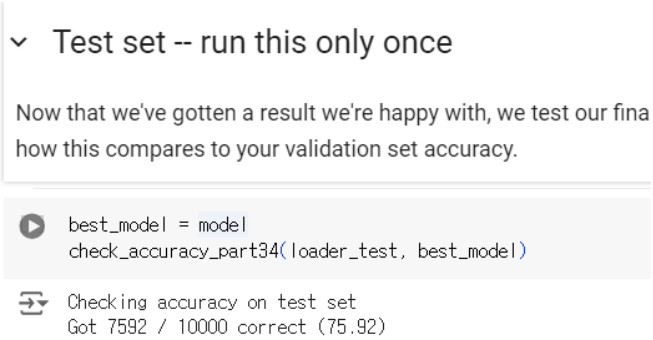
```
optimizer = optim.Adam(model.parameters(), lr=learning_rate,weight_decay=1e-05)
```

convolution layer를 6개로 늘리고, 각 convolution layer에 BatchNormalization을 적용하였다. 활성화함수는 ReLU를 그대로 적용하였다. 그리고 convolution layer 3개 마다 maxpooling(2)를 적용하여 32×32 이미지를 8×8 크기로 만들었다. Flatten 후 fully connected layer 후에도 BatchNormalization을 적용하였다. 첫번째 convolution layer만 커널 크기가 5x5이고, 나머지는 모두 3x3이다. 또한 첫번째 convolution layer의 채널 개수는 64(이해 channel_x), channel_2, channel_3, channel_4는 32, channel_5와 channel_6의 채널 개수는 16으로 설정하였다.

optimizer는 Adam을 사용하였으며 weight_decay(L2 regularization)는 1e-05로 설정하였다.

```
Iteration 700, loss = 0.6168
Checking accuracy on validation set
Got 756 / 1000 correct (75.60)
```

validation set의 정확도는 75.6%이다.



test set의 정확도는 75.92%이다.

3. 결론

토의 및 결론 (1페이지)

pytorch를 사용하여 convolution neural network를 다양한 방법으로 구성하고, CIFAR-10 이미지를 사용하여 학습하고, 정확도를 확인하였다. nn.Sequential을 사용하면 유연성은 낮지만, 상대적으로 쉽게 model을 구성할 수 있음을 알게 되었다. 또한 이미지 학습에 있어서 fully-connected network보다 convolution neural network가 더 효과적임을 알게 되었다. 또한 학습에 있어서 batch normalization, optimizer, 활성화 함수, 정규화 등 다양한 것을 적용하여 정확도를 향상시킬 수 있음을 알게 되었다.

참고 자료는 다음과 같다.

- Richard Szeliski. *Computer Vision: Algorithms and Applications*.2nd ed. Springer,2021
- 윤덕호.『파이썬 날코딩으로 알고 짜는 딥러닝』. 한빛미디어, 2019
- 오일석.『컴퓨터 비전과 딥러닝』. 한빛아카데미, 2023