20211584 장준영 자율주행 I 보고서

0. Implementation

```
# step1. Pass through the first fully connected layer (fc1) to perform a linear transformation.
x = self.fc1(x)

# step2. Apply the Rectified Linear Unit (ReLU) activation function to introduce non-linearity.
x = F.relu(x)

# step3. Pass through the second fully connected layer (fc2) to perform another linear transformation.
x = self.fc2(x)

# step4. Apply ReLU activation once again.
x = F.relu(x)

# step5. Pass through the third fully connected layer (fc3) to perform a linear transformation.
x = self.fc3(x)
```

forward method 내에서 주석에 작성된 지시대로 구현하였다.

```
Test set: Average loss: 0.3805, Accuracy: 8941/10000 (89%)

Total training and evaluation time: 51.8696 sec
```

실행 결과는 위와 같다.

1. Activation Functions

```
#################
x = self.fc1(x)
x = torch.tanh(x)
x = self.fc2(x)
x = torch.tanh(x)
x = self.fc3(x)
#################
```

```
#################
x = self.fc1(x)
x = torch.sigmoid(x)
x = self.fc2(x)
x = torch.sigmoid(x)
x = self.fc3(x)
#################
```

```
#################
x = self.fc1(x)
x = F.leaky_relu(x, negative_slope = 0.01)
x = self.fc2(x)
x = F.leaky_relu(x, negative_slope = 0.01)
x = self.fc3(x)
#################
```

차례대로 activation function을 tanh, sigmoid, leaky_relu(c = 0.01)로 변경한 코드이다. 그에 따른 결과는 다음과 같다.

```
Test set: Average loss: 0.5161, Accuracy: 8882/10000 (89%)

Total training and evaluation time: 52.8104 sec
```

```
Test set: Average loss: 2.2678, Accuracy: 1830/10000 (18%)

Total training and evaluation time: 54.8319 sec
```

```
Test set: Average loss: 0.3804, Accuracy: 8941/10000 (89%)

Total training and evaluation time: 53.4498 sec
```

결과를 다음과 같이 요약할 수 있다.

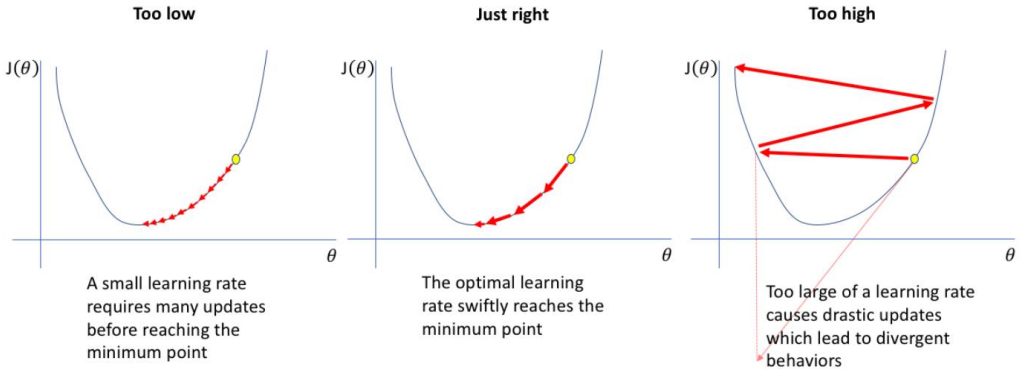
Average loss	leaky relu ≤ relu < tanh ≪ sigmoid
Accuracy	leaky relu = relu ≥ tanh ≫ sigmoid

1. ReLU는 음수에서 0이 된다. 이는 훈련을 빠르게 만들고 그래디언트 소실 문제를 줄여주지만, 'Dead ReLU' 문제를 유발할 수 있다. 이러한 장점 덕분에 ReLU는 깊은 네트워크에서 정확하고 빠르게 작동하고, loss를 빠르게 줄일 수 있다.
2. Leaky ReLU는 ReLU의 변형으로, 음수 입력에 대해 아주 작은 기울기를 적용하여 감소하도록 한다. 이는 'Dead ReLU' 문제를 완화하고, 모델의 성능을 더욱 향상시킬 수 있다. 이러한 이유로, Leaky ReLU의 성능이 ReLU와 동일하거나 약간 더 좋게 나올 수 있다.
3. Tanh 함수는 출력을 -1과 1 사이로 제한한다. 이는 모델의 출력을 더욱 일관성 있게 만들어 줄 순 있지만, 여전히 그래디언트 소실 문제가 있어 ReLU 계열의 activation function보다 성능이 약간 떨어질 수 있다.
4. Sigmoid 함수는 출력을 0과 1 사이 범위로 제한하는데, 이는 이진 분류 문제에서는 유용할 수 있으나 그 외의 경우에선 그래디언트 소실 문제를 심각하게 일으키는 경향이 있다. 다시 말해, 네트워크가 깊어질수록 그래디언트가 점점 소실되어 작아지고 weight update가 거의 이루어지지 않게 된다. 이로 인해 학습이 매우 느려지거나, 심지어는 전혀 이루어지지 않을 수도 있다. 따라서 Sigmoid의 성능이 매우 안좋게 나온 것으로 보인다.

2. Learning Rate

```
>python 1/mnist.py --lr 0.003
```

위와 같이 --lr <learning_rate>를 추가해 learning rate를 조절할 수 있다. 강의 자료에서 확인할 수 있듯, learning rate가 너무 높으면 한 점에 수렴하지 못하여 학습을 진행할 수 없고, 너무 낮으면 수많은 연산 이후에 optimal point를 찾을 수 있다는 단점이 있다.



따라서, optimal learning rate는 주어진 문제에 따라 경험적으로 찾아낼 수 있다. 0.0005, 0.001, 0.003, 0.005, 0.01, 0.03, 0.05의 learning rate에 대한 학습 결과는 다음과 같다.

```
Test set: Average loss: 1.8454, Accuracy: 5794/10000 (58%)

Total training and evaluation time: 56.4328 sec
```

```
Test set: Average loss: 0.9573, Accuracy: 7932/10000 (79%)

Total training and evaluation time: 55.5749 sec
```

```
Test set: Average loss: 0.3805, Accuracy: 8941/10000 (89%)

Total training and evaluation time: 55.9640 sec
```

```
Test set: Average loss: 0.3101, Accuracy: 9117/10000 (91%)

Total training and evaluation time: 57.0126 sec
```

```
Test set: Average loss: 0.2453, Accuracy: 9275/10000 (93%)

Total training and evaluation time: 56.2088 sec
```

```
Test set: Average loss: 0.1509, Accuracy: 9543/10000 (95%)

Total training and evaluation time: 55.2493 sec
```

```
Test set: Average loss: 0.1160, Accuracy: 9628/10000 (96%)

Total training and evaluation time: 60.1588 sec
```

Performance(Average loss / Accuracy)	순서대로 증가함.
Total time	서로 큰 차이를 보이지 않음.
대입한 Learning rate 중 최적의 값은 0.05였다. 최적의 값을 기준으로 양 옆의 값이 서로 대칭적이고, 최적값에서 최고의 성능을 보일 것이라고 생각했는데, 대입한 값 중 제일 큰 값이 최적값이 되어서 확인할 수 없었다.	

따라서, 추가로 0.1, 0.5, 1, 5를 대입해보았다. 결과는 순서대로 다음과 같다.

```
Test set: Average loss: 0.0939, Accuracy: 9704/10000 (97%)

Total training and evaluation time: 59.4286 sec
```

```
Test set: Average loss: 0.1147, Accuracy: 9665/10000 (97%)

Total training and evaluation time: 55.5911 sec
```

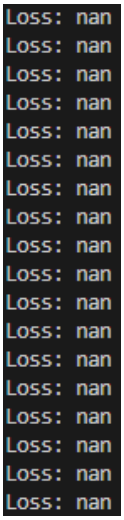
```
Test set: Average loss: 1.8444, Accuracy: 1889/10000 (19%)

Total training and evaluation time: 55.5905 sec
```

```
Test set: Average loss: nan, Accuracy: 980/10000 (10%)

Total training and evaluation time: 56.9580 sec
```

0.1까지 성능이 증가하다가, 0.5에서 아주 약간 감소하였다. 이후 1에서 급격하게 성능이 떨어지는 것을, 5에서는 제대로 된 학습이 진행되지 않는 것을 확인할 수 있다. optimal learning rate는 0.1 언저리의 값일 것이고, 1을 초과하는 learning rate에선 loss가 발산함을 알 수 있다.



실제로 learning rate를 5로 설정하고 진행했을 땐 loss 값이 nan이 되는 것을 확인할 수 있다.

3. Hidden Layer Depth

Hidden layer를 추가해 깊이를 조절하는 과정은 다음과 같다. 먼저, __init__ method에서 새로운 fully connected layer를 정의해야 한다.

```
def __init__(self):
    super(Net, self).__init__()

    self.fc1 = nn.Linear(28*28, 50)
    self.fc2 = nn.Linear(50, 50)
    self.fc3 = nn.Linear(50, 50)
    self.fc4 = nn.Linear(50, 50)
    self.fc5 = nn.Linear(50, 10)
```

위 사진에선 50차 벡터를 입력으로 받고 50차 벡터를 출력으로 하는 두 개의 fully connected layer를 추가하였다. 다음으론 forward method에서 앞 방향으로 진행되는 flow를 지정해주어야 한다.

```
################
x = self.fc1(x)
x = F.relu(x)
x = self.fc2(x)
x = F.relu(x)
x = self.fc3(x)
x = F.relu(x)
x = self.fc4(x)
x = F.relu(x)
x = self.fc5(x)
################
```

형식 자체가 수정 전과 동일하므로 쉽게 이해할 수 있다. 이렇게 하면, hidden layer를 두 개 더 추가해 depth가 2 늘어난다. 동일한 과정을 거쳐 hidden layer가 0개, 1개, 5개, 10개인 경우를 테스트해보았다. 결과는 순서대로 다음과 같다.

```
Test set: Average loss: 0.3396, Accuracy: 9045/10000 (90%)

Total training and evaluation time: 52.0864 sec
```

```
Test set: Average loss: 0.3805, Accuracy: 8941/10000 (89%)

Total training and evaluation time: 53.5670 sec
```

```
Test set: Average loss: 2.2995, Accuracy: 1135/10000 (11%)

Total training and evaluation time: 57.3189 sec
```

```
Test set: Average loss: 2.3014, Accuracy: 1135/10000 (11%)

Total training and evaluation time: 59.0605 sec
```

Average loss	$(0) < (1) \ll (5) < (10)$
Accuracy	$(0) > (1) \gg (5) > (10)$
Total time	$(0) < (1) < (5) < (10)$

내가 생각한 것과 정반대의 결과가 나왔다. 나는 hidden layer가 늘어날수록 복잡한 패턴을 학습할 수 있게 되어 정확도가 증가할 것이라고 생각했다. 그런데 오히려 hidden layer가 늘어날수록 average loss가 증가하고 accuracy가 감소했다. 나는 원인으로 다음을 꼽았다.

1. 모델의 복잡성이 늘어날수록 모델은 학습 데이터의 노이즈(일반화 불가한 데이터 특수 경향성, 학습 데이터의 잘못된 부분)까지 학습하려는 경향이 있다. 이에 따라 훈련 데이터에 대한 성능은 향상되지만, 이 외의 테스트 데이터에 대한 성능은 저하되는 과적합(overfitting)을 초래할 수 있다.
2. 모델의 복잡성이 데이터의 복잡성을 넘어서면 오히려 성능이 떨어질 수도 있다. MNIST 데이터셋은 비교적 단순한 데이터셋이므로, 너무 많은 hidden layer가 오히려 성능을 저하시킬 수 있다.
3. 이 외에도 learning rate나 initial value 등이 특정 상황에 대해 안좋은 결과를 내도록 설정되어 있을 수도 있다. 이는 개념적으로 확인이 불가능하기 때문에 추측으로 남겨두었다.

내가 테스트 한 방법이 옳은 방법인지는 모르겠지만, 내 예상을 빗나가는 결과가 나와 의외였다. 그 외에, Total time은 예상한대로 오름차순이 되었다. 통과해야 하는 fully connected layer가 늘어나 weight 연산량도 덩달아 늘어나기 때문이다.

4. Hidden Layer Width

원본처럼 하나의 hidden layer를 유지하되, 그 크기(너비)를 바꾸는 테스트를 할 것이다. Depth와 마찬가지로, __init__ method에서 너비 요소를 바꿔주기만 하면 된다. 다음은 hidden layer의 width를 100으로 바꾼 코드이다.

```
def __init__(self):
    super(Net, self).__init__()

    self.fc1 = nn.Linear(28*28, 100)
    self.fc2 = nn.Linear(100, 100)
    self.fc3 = nn.Linear(100, 10)
```

Hidden layer width를 10, 50, 200, 750로 바꾸어 테스트한 결과는 순서대로 다음과 같다.

```
Test set: Average loss: 0.7646, Accuracy: 7662/10000 (77%)

Total training and evaluation time: 54.0514 sec
```

```
Test set: Average loss: 0.3805, Accuracy: 8941/10000 (89%)

Total training and evaluation time: 51.8696 sec
```

```
Test set: Average loss: 0.3502, Accuracy: 8997/10000 (90%)

Total training and evaluation time: 56.0005 sec
```

```
Test set: Average loss: 0.3309, Accuracy: 9068/10000 (91%)

Total training and evaluation time: 80.6758 sec
```

Perfomance(Average loss / Accuracy)	(750) > (200) > (50) > (10)
Total time	(50) < (10) < (200) < (750)
Hidden layer의 width를 늘리면 성능이 향상하는 것은 쉽게 알 수 있다. hidden layer의 width가 늘어나는 것은 뉴런의 수를 늘리는 것과 동일한데, 이에 따라 동시에 처리할 수 있는 정보의 양이 늘어난다. 이는 더 많은 특징과 더 복잡한 패턴을 수월하게 학습할 수 있도록 한다. 신기한 것은 Total time이 (50)보다 (10)이 더 큰 것이었는데, 이는 컴퓨터 시스템 상의 문제가 있었던 것으로 추측하고 있다. 보통은, width가 늘어남에 따라 weight 연산량이 증가하기 때문에 더 오랜 시간이 걸린다.	

극단적인 케이스로 2500, 5000 까지도 테스트해보았다. 결과는 순서대로 다음과 같다.

```
Test set: Average loss: 0.2984, Accuracy: 9153/10000 (92%)

Total training and evaluation time: 235.7294 sec
```

```
Test set: Average loss: 0.2756, Accuracy: 9227/10000 (92%)

Total training and evaluation time: 678.2316 sec
```

성능이 확실히 좋아지지만, width 상승 폭에 정비례하진 않고 성능 상승 폭이 줄어드는 것을 확인할 수 있다. 따라서, width 만을 늘려서 얻을 수 있는 성능적 이득은 한계가 존재한다.