Week1 과제: ANN, DNN 보고서

<목차>

1️⃣ OR 게이트 학습 및 시각화(ANN)

→ OR 게이트를 은닉층 1개 이용해 분리하세요.

→ 코드 분석과 함께 시각화를 해주세요.

[코드 전체 + 코드 분석(주석)]

```
'''
OR 게이트를 은닉층 1개 이용하여 분리하기
'''
import torch
import torch.nn as nn

torch.manual_seed(777)  # CPU 사용

# 1. 입력과 출력 정의 (OR 게이트)
X = torch.FloatTensor([[0,0], [0,1], [1,0],[1,1]])
Y = torch.FloatTensor([[0],[1],[1],[1]])


# 2. 은닉층 1개를 가지는 단층 퍼셉트론 모델 정의
# - 입력 2개, 출력 1개인 선형 레이어
layer = nn.Linear(2, 1, bias = True)
# - 활성화 함수 - 시그모이드 함수 사용(이진 분류에 적합)
sigmoid = nn.Sigmoid()
```

```
model = nn.Sequential(layer,sigmoid) # linear → sigmoid 순으로 연결


# 3. 비용함수, 옵티마이저 정의
# - Binary Cross Entropy Loss 사용(이진 분류에 적합)
criterion = nn.BCELoss()
# - 경사 하강법 기반 모델 파라미터 갱신
optimizer = torch.optim.SGD(model.parameters(), lr = 1)


# 4. 학습 - 오차 역전파 실행
for step in range(10001):  # Epoch = 100001
  # - 그래디언트 초기화
    optimizer.zero_grad()
  # - Foward
  hypothesis = model(X)

  # - Error 계산
  cost = criterion(hypothesis, Y)
  # - Backward
  cost.backward()
  # - 가중치 갱신
  optimizer.step()

  if step % 500 == 0:  # 가독성 상 500 step 마다 Loss 출력하도록 설정
    print(f"{step} {cost.item():.6f}")  # Loss 소수점 여섯째 자리까지 출력


# 5. 학습된 w, b를 바탕으로 예측값 평가 및 정확도 계산
with torch.no_grad():
  # - model에 X를 통과시켜 예측값 구하기
  hypothesis = model(X)
  # - 0.5 이상이면 1, 아니면 0
  predicted = (hypothesis > 0.5).float()
  accuracy = (predicted == Y).float().mean()
  print('\n Hypothesis: ', hypothesis.numpy(),
        '\n Correct: ', predicted.numpy(), # threshold 적용한 예측값
        '\n Accuracy: ', accuracy.item())  # 정확도
```

▼ 4 학습 Loss 결과

0 0.912537
500 0.018859
1000 0.009305
1500 0.006158
2000 0.004598
2500 0.003667
3000 0.003049
3500 0.002609
4000 0.002280
4500 0.002024
5000 0.001820
5500 0.001654
6000 0.001515
6500 0.001397
7000 0.001297
7500 0.001210
8000 0.001134
8500 0.001067
9000 0.001007
9500 0.000954
10000 0.000906

⇒ Loss가 0에 가까워지면서 예측이 정답에 근접함을 확인

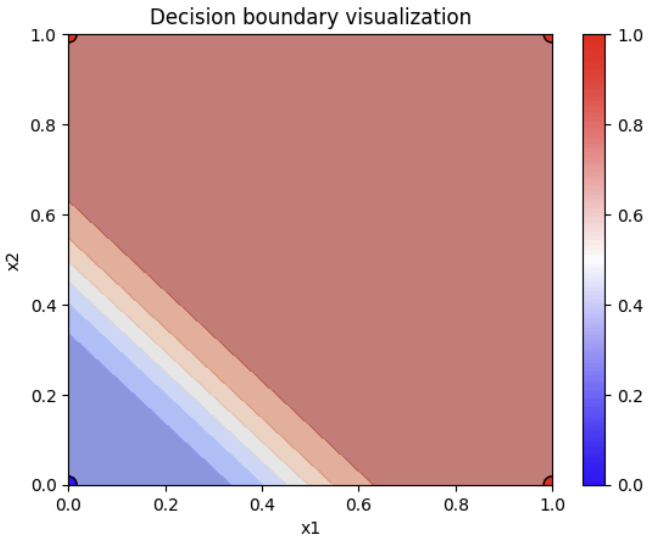
▼ 5 예측 값, 분리 성공 결과

Hypothesis: [[0.00201166]
[0.9991954]
[0.9991954]
[1.   ]]
Correct: [[0.]
[1.]
[1.]
[1.]]
Accuracy: 1.0

⇒ 출력: `[[0.], [1.], [1.], [1.]]` **, 정확도 = 1.0 (100%) <OR 게이트 분리 성공>**

[시각화]

```
import matplotlib.pyplot as plt
import numpy as np
import torch

def plot_decision_boundary(model, X, Y):
  # 0~1 구간을 100*100으로 나눠 2차원 격자 생성
  x = np.linspace(0, 1, 100)
  y = np.linspace(0, 1, 100)
  xx, yy = np.meshgrid(x, y)
  # 각 (x1, x2) 좌표쌍을 (10000, 2) 형태로 정리하여 모델 입력으로 변환
  grid = torch.tensor(np.c_[xx.ravel(), yy.ravel()], dtype = torch.float32)

  # 모델 예측 (no_grad: 학습은 X)
  with torch.no_grad():
      zz = model(grid).reshape(xx.shape)

  # 배경 색상 설정(이진분류에 적합한 'coolwarm' 사용)
  plt.contourf(xx, yy, zz, cmap = 'coolwarm', alpha = 0.7)

  # 실제 데이터 점 (X, Y) 찍기 (빨강 = 1, 파랑 = 0)
  X_np = X.numpy()  # tensor > numpy 배열로 변환
  Y_np = Y.numpy().ravel()
  plt.scatter(X_np[:, 0], X_np[:, 1], c=Y_np, cmap='bwr', edgecolors='k', s=100)

  plt.colorbar()
  plt.title("Decision boundary visualization")
  plt.xlabel("x1")
  plt.ylabel("x2")
  plt.show()

plot_decision_boundary(model, X, Y)
```



- 배경 색 (contourf)
  - 색상 = 모델의 예측 확률
  - 파란색 (0) = 출력이 0일 확률이 높다
  - 빨간색 (1) = 출력이 1일 확률이 높다
  - 경계선 근처의 옅은 색 = 예측 확률이 0.5 정도이다.
- 빨간 점 : $y = 1$
- 파란 점 : $y = 0$
- (0,0) 은 파란색 영역(예측값 0)에 있고, (0,1), (1,0), (1,1) 은 빨간색 영역(예측값 1)에 있으므로

  단층 퍼셉트론으로 OR 게이트 구분 가능

2️⃣ XOR 게이트 학습 시도 및 한계 시각화 (ANN)

→ XOR 게이트를 은닉층 1개 이용해 분리하세요.

→ 분리가 되지 않는 이유를 적어주세요.

[코드]

1️⃣ 에서 이미 단층 퍼셉트론을 구현하였기 때문에, OR 게이트만 XOR 게이트로 **출력값(y)만 변경**하여 실행하였다. (나머지 코드는 1️⃣과 동일)

```
# 1. 입력과 출력 정의 (XOR 게이트)
X = torch.FloatTensor([[0,0], [0,1], [1,0], [1,1]])
XOR_Y = torch.FloatTensor([[0],[1],[1],[1]])  # 이 부분만 변경 (XOR 게이트)
```

▼ 4 학습 Loss 결과

0 0.727397
500 0.693147
1000 0.693147
1500 0.693147
2000 0.693147
2500 0.693147
3000 0.693147
3500 0.693147
4000 0.693147
4500 0.693147
5000 0.693147
5500 0.693147
6000 0.693147
6500 0.693147
7000 0.693147
7500 0.693147
8000 0.693147
8500 0.693147
9000 0.693147
9500 0.693147
10000 0.693147

⇒ Loss가 0.693147에 수렴하면서 모델이 학습하지 못하는 것을 확인

▼ 5 예측값, 분리 실패 결과

Hypothesis: [[0.5]
[0.5]
[0.5]
[0.5]]
Correct: [[0.]
[0.]

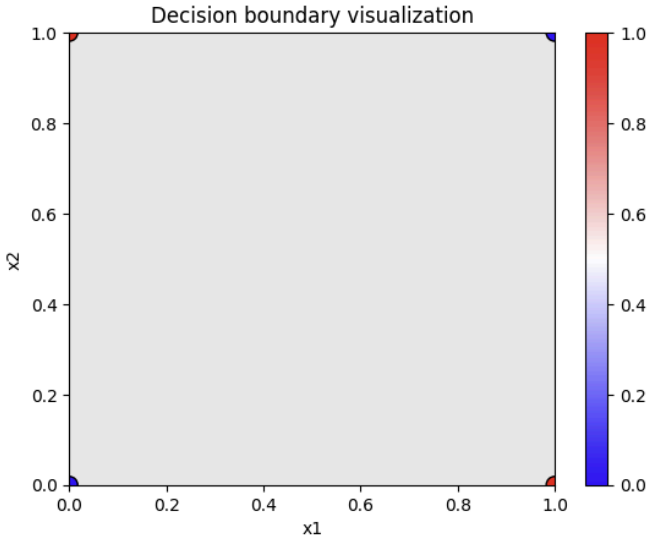
[0.]
[0.]]
Accuracy: 0.5

⇒ 출력: `[[0.], [0.], [0.], [0.]]` , 정확도 = 0.5 (50%) <XOR 게이트 분리 실패>

[시각화]

1️⃣ 에서 생성한 시각화 코드 함수 사용

```
plot_decision_boundary(model, X, XOR_Y)
```



[분리가 되지 않는 이유]

- (1) 시각적 근거
  - 시각화를 통해 결정 경계가 생기지 않는 것을 확인 (=하나의 직선으로는 분리할 수 없어 학습에 실패)
- (2) 수학적 구조
  - 단층 퍼셉트론은 $w_1 * x_1 + w_2 * x_2 + b = 0$ (선형 방정식)으로 결정 경계를 만드는데,
  - XOR 게이트의 경우,
    - $f(0,0) = \sigma(b) \approx 0 \quad \Rightarrow \quad b \ll 0$

(시그모이드 함수 성질에 따라 b가 매우 작은 음수여야 $\sigma(b)$가 0에 가까워짐)

- $f(0,1) = \sigma(w_2 + b) \approx 1 \quad \Rightarrow \quad w_2 + b \gg 0$
- $f(1,0) = \sigma(w_1 + b) \approx 1 \quad \Rightarrow \quad w_1 + b \gg 0$
- $f(1,1) = \sigma(w_1 + w_2 + b) \approx 0 \quad \Rightarrow \quad w_1 + x_2 + b \ll 0$

위의 네 조건을 동시에 만족하는 w_1, w_2, b 값은 없다.

- 따라서 수학적으로도 모순이 발생하기 때문에, **은닉층 1개로는 XOR 분리에 한계를 가진다.**

3️⃣ XOR 게이트 학습 및 시각화 (DNN)

→ XOR 게이트를 DNN을 이용해 분리하세요.

→ 분리가 되는 이유를 적어주세요.

→ 코드 분석과 시각화를 해주세요.

[코드 전체 + 코드 분석(주석)]

2️⃣ 에서 단층 퍼셉트론 → 다층 퍼셉트론 정의 부분만 변경. (1️⃣과 동일한 코드 부분은 코드 분석 생략)

```
'''
XOR 게이트를 DNN을 이용하여 분리하기
'''
import torch
import torch.nn as nn

torch.manual_seed(777)

# 1. XOR 게이트 입력, 출력 정의 유지
X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]])
XOR_Y = torch.FloatTensor([[0], [1], [1], [0]])


# 2. 다층 퍼셉트론 모델 정의 (은닉층 3개 사용)
```

```
'''
- 깊고(은닉층 늘리기(3개)), 넓게(features 수 늘리기(2>10)) 신경망 구조 확장
- 활성화 함수: 일반적으로 은닉층에는 ReLU(기울기 소실 문제 해결), 출력층에는 Sigmo
'''

model = nn.Sequential(
  nn.Linear(2, 10, bias=True),   # 은닉층1 (입력 2 > 출력 10)
  nn.ReLU(),
  nn.Linear(10, 10, bias=True),  # 은닉층2 (입력 10 > 출력 10)
  nn.ReLU(),
  nn.Linear(10, 10, bias=True),  # 은닉층3 (입력 10 > 출력 10)
  nn.ReLU(),
  nn.Linear(10, 1, bias=True),   # 출력층 (입력 10 > 출력 1)
  nn.Sigmoid()
)



# 3. 비용 함수 및 옵티마이저 설정 (동일)
criterion = torch.nn.BCELoss()  # 이진 분류이므로 BCE 유지
optimizer = torch.optim.SGD(model.parameters(), lr=1)

# 4. 모델 학습 (동일)
for epoch in range(10001):
  optimizer.zero_grad()
  hypothesis = model(X)
  cost = criterion(hypothesis, XOR_Y)
  cost.backward()
  optimizer.step()

  if epoch % 500 == 0:
    print(f"{epoch} {cost.item():.6f}")

# 5. 평가 (동일)
with torch.no_grad():
  hypothesis = model(X)
  predicted = (hypothesis > 0.5).float()
  accuracy = (predicted == XOR_Y).float().mean()
```

```
print('\n Hypothesis: ', hypothesis.numpy(),
      '\n Correct: ', predicted.numpy(),
      '\n Accuracy: ', accuracy.item())
```

▼ 4 학습 Loss 결과

0 0.692673
500 0.000152
1000 0.000058
1500 0.000034
2000 0.000024
2500 0.000018
3000 0.000014
3500 0.000012
4000 0.000010
4500 0.000009
5000 0.000008
5500 0.000007
6000 0.000006
6500 0.000006
7000 0.000005
7500 0.000005
8000 0.000004
8500 0.000004
9000 0.000004
9500 0.000004
10000 0.000003

⇒ Loss가 0에 빠르게 가까워지면서 예측이 정답에 매우 근접함을 확인

▼ 5 예측값, 분리 성공 결과

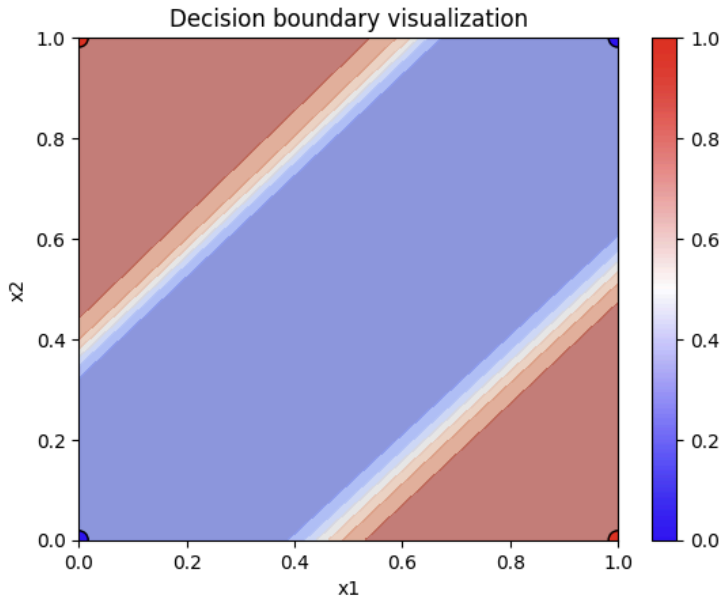
Hypothesis: [[4.6016412e-06]
[9.9999774e-01]
[9.9999714e-01]
[4.0127561e-06]]
Correct: [[0.]
[1.]
[1.]
[0.]]
Accuracy: 1.0

⇒ 출력: `[[0.], [1.], [1.], [0.]]` **, 정확도 = 1.0 (100%) <XOR 게이트 분리 성공>**

[시각화]

1️⃣ 에서 생성한 시각화 코드 함수 사용

```
plot_decision_boundary(model, X, XOR_Y)
```



- 배경 색, 점 의미는 1️⃣과 동일
- (0,0)과 (1,1)은 파란색 영역(예측값 0)에 있고, (0,1), (1,0)은 빨간색 영역(예측값 1)에 있으므로

  DNN으로 비선형적인 XOR 게이트 분류 가능

[분리가 되는 이유]

- 시각화를 통해 DNN으로 하나의 직선으로는 나눌 수 없는 XOR 게이트에 2개의 경계가 생기는 것을 확인
- 은닉층 1개 이상 + 비선형 함수를 조합할 경우,
  입력 공간을 선형이 아닌 방식으로 변환하여 복잡한 결정 경계를 만드는 DNN의 특성 때문에 분리가 가능해진다.

⬆️ 처음으로