Week1 - ANN, DNN 보고서

분석 25기 김지수

1. OR gate 학습 및 시각화 (ANN)

- OR GATE를 은닉층 1개를 이용해 분리하기
- 코드 분석 및 시각화

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

# 1. input/output 정의
X = torch.Tensor([[0, 0], [0, 1], [1, 0], [1, 1]])
y = torch.Tensor([[0], [1], [1], [1]])

# 2. model 정의
class ANN(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(2, 2), # input 2D → hidden layer
            nn.Sigmoid(),    # 이진 분류용 활성화 함수
            nn.Linear(2, 1), # output layer
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.model(x)


model = ANN()
```

```
criterion = nn.BCELoss() # loss function
optimizer = optim.SGD(model.parameters(), lr=0.1) # SGD에 학습률은 0.1로 설정

# 3. Training
def train(epochs, X, y):
    for epoch in range(epochs):
        output = model(X)           # tensor with 4 probabilities(0-1)
        loss = criterion(output, y)  # compute the loss w/ BCE

        optimizer.zero_grad()       # initialize all the gradients in the previous epoc
        loss.backward()             # backpropagation(역전파) → compute the gradie
        optimizer.step()            # update the parameters(기울기 w, 편향 b) based o

        if epoch % 1000 == 0:
            print(f"Epoch {epoch}, loss: {loss.item():.4f}")

epochs = 10000
train(epochs, X, y)
```

```
Epoch 0, loss: 0.9114
Epoch 1000, loss: 0.2052
Epoch 2000, loss: 0.0486
Epoch 3000, loss: 0.0236
Epoch 4000, loss: 0.0150
Epoch 5000, loss: 0.0109
Epoch 6000, loss: 0.0084
Epoch 7000, loss: 0.0069
Epoch 8000, loss: 0.0058
Epoch 9000, loss: 0.0050
```

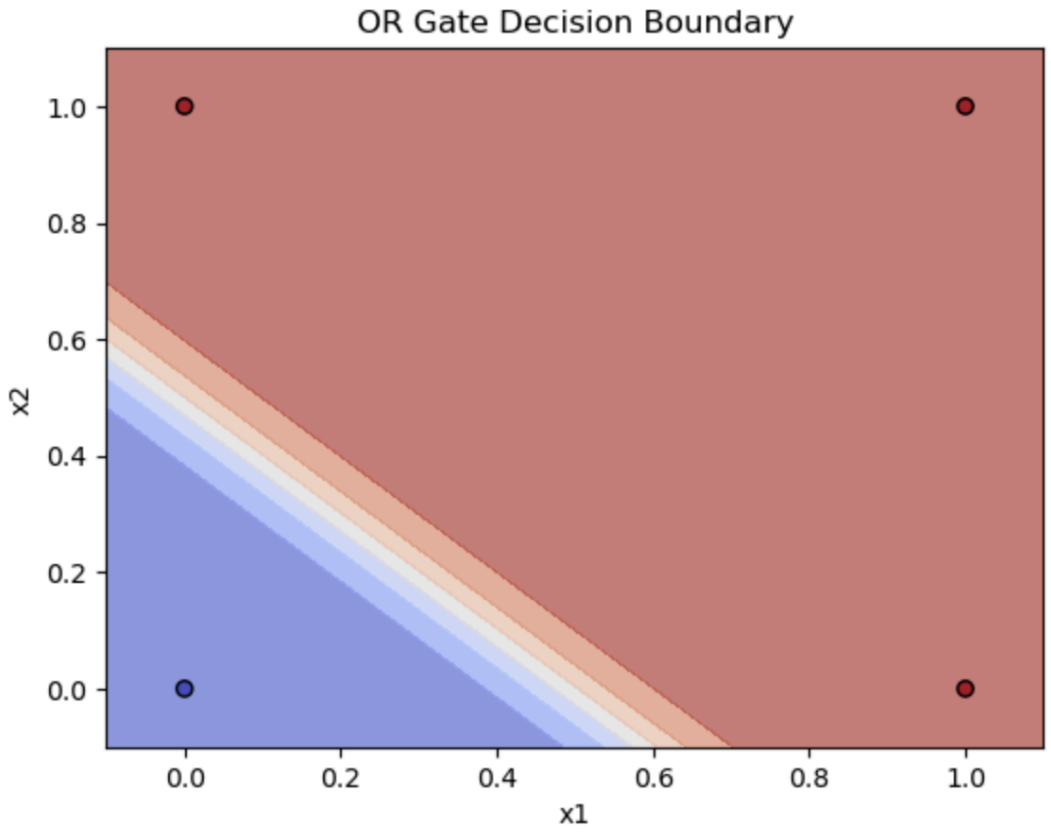
```
# Visualize the decision boundary

def plot_GATE(model):
    x_min, x_max = -0.1, 1.1
    y_min, y_max = -0.1, 1.1
    h = 0.01
```

```
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max,
    grid = torch.FloatTensor(np.c_[xx.ravel(), yy.ravel()])
    with torch.no_grad():
        Z = model(grid).reshape(xx.shape)
    Z = Z.numpy()

    plt.contourf(xx, yy, Z, cmap="coolwarm", alpha=0.7)
    plt.scatter(X[:, 0], X[:, 1], c=y[:, 0], cmap="coolwarm", edgecolors='k')
    plt.title("OR Gate Decision Boundary")
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.show()

plot_GATE(model)
```



- 모델이 입력값을 받고 0일 확률/1일 확률이 각각 어느 정도인지 보여주는 출력값을 색으로 표현하였음
    - sigmoid 함수를 통해 0-1 사이 실수값을 뽑았기 때문에 중간 부분의 색깔이 확률처럼 연속적으로 변하는 모습

- 그래프 상 네 개의 점들은 모두 실제 학습에 사용된 tensor 데이터이다.
- 빨간 영역과 파란 영역은 각각 모델이 1, 0이라고 예측하는 부분이고, 중간의 그라데이션으로 시각화한 부분은 모델이 애매하게 판단하는 decision boundary 라고 볼 수 있으므로 해당 모델은 OR GATE를 hidden layer 1개인 단순한 모델로도 잘 분리해내었다고 판단할 수 있다.

---

2. XOR gate 학습 시도 및 한계 시각화 (ANN)

- XOR GATE를 은닉층 1개를 이용해 분리하기
- 분리가 되지 않는 이유?

- input/output 외에 modeling code들을 1번과 동일하여 함수 인자를 바꾸는 작업만 진행하였습니다.

```
# Define XOR GATE
X = torch.Tensor([[0, 0], [0, 1], [1, 0], [1, 1]])
y = torch.Tensor([[0], [1], [1], [0]])

epochs = 10000
train(epochs, X, y) # 인자만 바꾸고 동일한 은닉층 1개 모델에서 실행
```

```
Epoch 0, loss: 1.8836
Epoch 1000, loss: 0.4935
Epoch 2000, loss: 0.4887
Epoch 3000, loss: 0.4870
Epoch 4000, loss: 0.4859
Epoch 5000, loss: 0.4850
Epoch 6000, loss: 0.4843
Epoch 7000, loss: 0.4837
Epoch 8000, loss: 0.4832
Epoch 9000, loss: 0.4828
```

- OR gate보다 확실히 loss가 epoch을 거듭해도 계속 큰 값을 띠는 것을 볼 수 있다. 즉, 분리가 되지 않음.

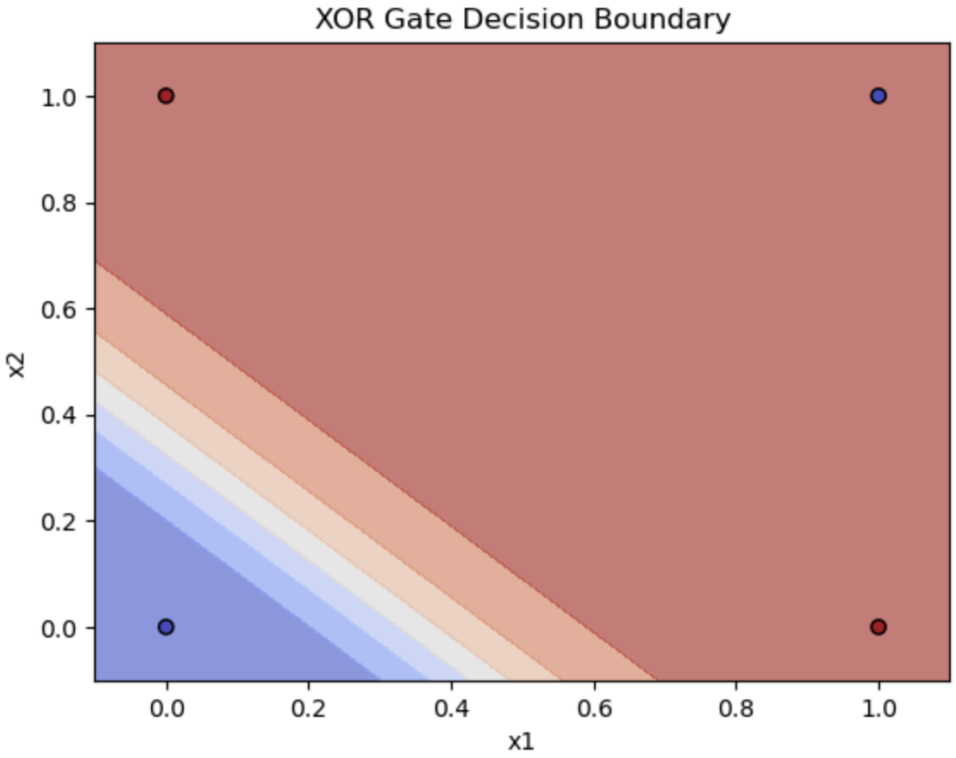
```
# Visualize the decision boundary

def plot_XOR_GATE(model):
    x_min, x_max = -0.1, 1.1
    y_min, y_max = -0.1, 1.1
    h = 0.01

    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max,
    grid = torch.FloatTensor(np.c_[xx.ravel(), yy.ravel()])
    with torch.no_grad():
        Z = model(grid).reshape(xx.shape)
    Z = Z.numpy()

    plt.contourf(xx, yy, Z, cmap="coolwarm", alpha=0.7)
    plt.scatter(X[:, 0], X[:, 1], c=y[:, 0], cmap="coolwarm", edgecolors='k')
    plt.title("XOR Gate Decision Boundary")
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.show()

plot_XOR_GATE(model)
```



- 앞선 OR gate보다 훨씬 배경색이 경계선을 따라 선명하게 구분되지 않고 복잡하게 섞여있는 것을 볼 수 있다. 이는 XOR의 정답 구조 자체가 한 직선으로 나눌 수 없는 구조이기 때문이며, 위 시각화 결과를 통해 이러한 비선형 문제는 하나의 decision boundary로 분리가 불가능함을 볼 수 있다.
- XOR 게이트는 이러한 단순한 ANN으로는 완벽히 분리가 불가능하므로더 깉은 DNN이 필요하며 충분한 hidden layer를 통해 비선형 관계를 학습할 수 있다.

---

3. XOR gate 학습 시각화 (DNN)

- XOR GATE를 DNN을 이용해 분리하기
- 분리가 되는 이유?
- 코드 분석과 시각화를 해주세요.

```
# DNN model (more than 2 hidden layers)
class DNN(nn.Module):
```

```
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(2, 4),   # 1st hidden layer
            nn.ReLU(),
            nn.Linear(4, 4),   # 2nd hidden layer
            nn.ReLU(),
            nn.Linear(4, 1),   # Output layer
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.model(x)
```

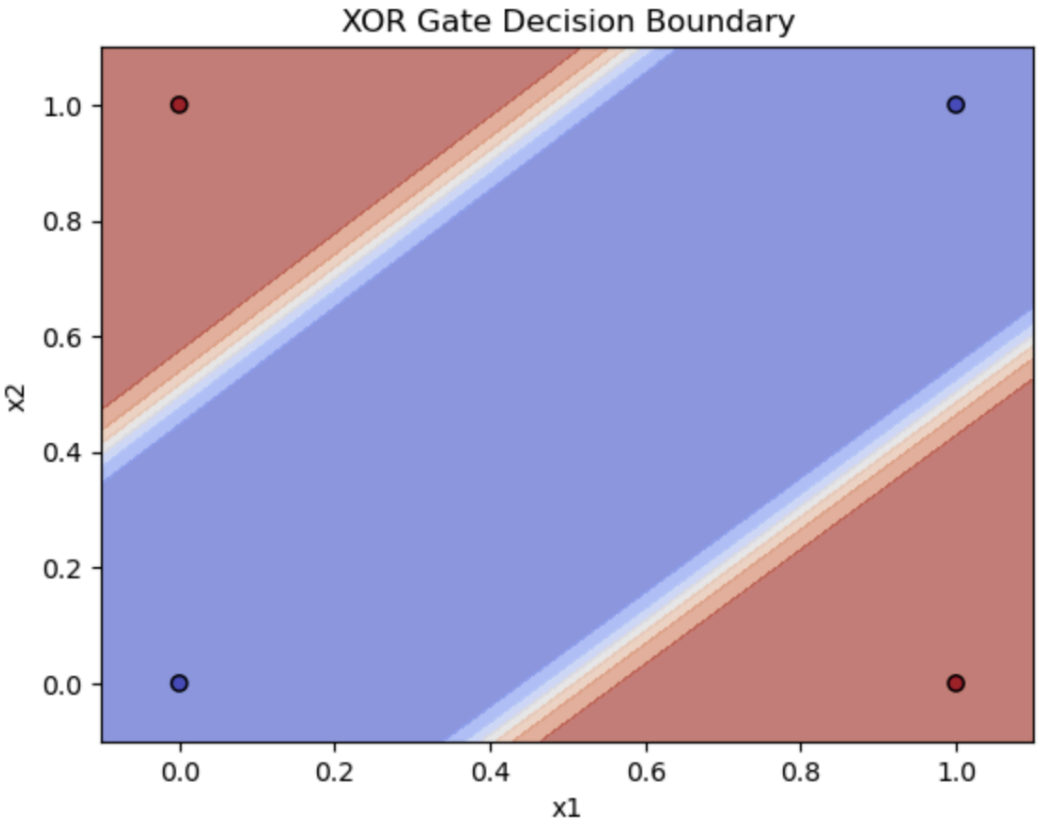
```
model = DNN()
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

epochs = 10000
train(epochs, X, y) # 앞서 생성해놓은 train function 재사용
```

```
Epoch 0, loss: 0.7157
Epoch 1000, loss: 0.0001
Epoch 2000, loss: 0.0000
Epoch 3000, loss: 0.0000
Epoch 4000, loss: 0.0000
Epoch 5000, loss: 0.0000
Epoch 6000, loss: 0.0000
Epoch 7000, loss: 0.0000
Epoch 8000, loss: 0.0000
Epoch 9000, loss: 0.0000
```

- ANN으로 XOR gate를 분리할 때보다 loss가 확연하게 줄어든 것을 볼 수 있다.

```
plot_XOR_GATE(model)
```



- 두 개의 1 output point (0,1), (1,0)은 붉은 영역에 위치하고, 0 output point (0,0), (1,1)은 파란 영역에 위치하는 것으로 보아 ANN과 달리 이 구조에서는 분리가 성공한 것을 볼 수 있다.

- 분리가 가능해진 이유
  - XOR은 비선형 문제이기 때문에 직선으로는 분리가 불가능하다.
  - 그러나 DNN은 층을 깊게 쌓는 과정을 통해 복잡한 곡선 경계를 학습할 수 있으며, 여기에 이번엔 ReLU와 같은 비선형 함수도 추가하였기 때문에 모델이 nonlinear decision boundary를 만들어내어 4개의 data point를 정확히 분류하는 경계가 만들어진 것을 볼 수 있다.