XOR 모델 설계 및 시각화

2개의 은닉층

(train해서 파라미터 얻기 전까진 3개 은닉층과 코드가 동일합니다!!!
3개 은닉층의 <공통> 부분 참고!!!!!)

모델의 구조
- 입력층 : 2개의 뉴런
- 은닉층 : 총 2개, 각 2개의 뉴런
- 출력층 : 1개의 뉴런
-> 활성화 함수 : 시그모이드 함수 사용

<은닉층 2개에서의 최적 train>

```
# 파라미터 히든레이어 2
# optimizer: SGD, learning rate
train(X, y, epochs=20000, lr=0.2, model=model, every_print=1000)
model(X), torch.where(model(X)>0.5, 1, 0)
```

```
Epoch: 1000, loss: 0.6930480003356934
Epoch: 2000, loss: 0.6929649114608765
Epoch: 3000, loss: 0.6927715539932251
Epoch: 4000, loss: 0.692120373249054
Epoch: 5000, loss: 0.6872801780700684
Epoch: 6000, loss: 0.5661376714706421
Epoch: 7000, loss: 0.032184965908527374
Epoch: 8000, loss: 0.009721485897898674
Epoch: 9000, loss: 0.005502770654857159
Epoch: 10000, loss: 0.003791974624618888
Epoch: 11000, loss: 0.0028768128249794245
Epoch: 12000, loss: 0.002310419688001275
Epoch: 13000, loss: 0.0019266318995505571
Epoch: 14000, loss: 0.0016500984784215689
Epoch: 15000, loss: 0.0014416251797229052
Epoch: 16000, loss: 0.0012790284818038344
Epoch: 17000, loss: 0.0011488031595945358
Epoch: 18000, loss: 0.0010421954793855548
Epoch: 19000, loss: 0.000953370297793299
Epoch: 20000, loss: 0.0008782774675637484
(tensor([[9.3357e-04],
         [9.9896e-01],
         [9.9926e-01],
         [8.0012e-04]], grad_fn=<SigmoidBackward0>),
 tensor([[0],
         [1],
         [1],
         [0]]))
```

Hyperparameter 결정 과정
- 학습률(learning rate): 0.2
- 최적화 기법(optimizer): SGD
- 손실 함수(loss function): 이진 교차 엔트로피 손실(Binary Cross-Entropy Loss)
- 에포크 수(epochs): 20,000

-> 위 조건으로 train 결과 손실 값이 점점 감소하고 최종 손실은 0.0008782774675637484로 낮은 값을 나타고 있음.
예측된 클래스는 예측된 확률에 기반하여 0.5보다 크면 1로, 그렇지 않으면 0으로 표시되도록 설정되어 있는데, 이 예측된 클래스가 [0, 1, 1, 0]으로
실제클래스 [0, 1, 1, 0] 와 정확히 일치하므로 학습이 잘 되었다고 볼 수 있음.

```
for x in model.named_parameters():
    print(x)
```

```
('model.0.weight', Parameter containing:
tensor([[ 5.0629, -4.8229],
        [ 5.5597, -5.6119]], requires_grad=True))
('model.0.bias', Parameter containing:
tensor([ 2.3824, -3.1163], requires_grad=True))
('model.2.weight', Parameter containing:
tensor([[ 5.7263, -6.0702],
        [-4.4636,  4.5147]], requires_grad=True))
('model.2.bias', Parameter containing:
tensor([-2.5706,  2.0293], requires_grad=True))
('model.4.weight', Parameter containing:
tensor([[-10.4020,   7.6603]], requires_grad=True))
('model.4.bias', Parameter containing:
tensor([1.5453], requires_grad=True))
```

학습 결과
- 첫 번째 은닉층(model.0) 가중치 [[5.0629, -4.8229], [5.5597, -5.6119]]
- 첫 번째 은닉층(model.0) 편향 [2.3824, -3.1163]
- 두 번째 은닉층(model.2) 가중치 [[5.7263, -6.0702], [-4.4636, 4.5147]]
- 두 번째 은닉층(model.2) 편향 [-2.5706, 2.0293]
- 출력층(model.4) 가중치 [-10.4020, 7.6603]
- 출력층(model.4) 편향 [1.5453]

<최종출력>

```
import torch

# 주어진 파라미터 값
w0 = torch.tensor([[ 5.0629, -4.8229], [5.5597, -5.6119]])
b0 = torch.tensor([ 2.3824, -3.1163])
w2 = torch.tensor([[ 5.7263, -6.0702],[-4.4636,  4.5147]])
b2 = torch.tensor([-2.5706,  2.0293])
w4 = torch.tensor([[-10.4020,  7.6603]])
b4 = torch.tensor([1.5453])

# 입력 데이터
X = torch.tensor([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=torch.float32)

sigmoid = nn.Sigmoid()

# 첫 번째 은닉층의 출력 계산
layer1_output = sigmoid(X @ w0.T + b0)

# 두 번째 은닉층의 출력 계산
layer2_output = sigmoid(layer1_output @ w2.T + b2)

# 최종 출력 계산
sigmoid(layer2_output @ w4.T + b4)
```

```
tensor([[9.3354e-04],
        [9.9897e-01],
        [9.9926e-01],
        [8.0015e-04]])
```

위에 출력된 파라미터를 대입하여 각 은닉층의 출력과 최종출력을 계산

<3d 출력>

```
import numpy as np
import torch
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

x_xor = torch.tensor([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=torch.float)

class XOR(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = torch.nn.Linear(2, 2)
        self.sigmoid = torch.nn.Sigmoid()
        self.layer2 = torch.nn.Linear(2, 2)
        self.sigmoid2 = torch.nn.Sigmoid()
        self.layer3 = torch.nn.Linear(2, 1)
        self.sigmoid3 = torch.nn.Sigmoid()

    def forward(self, x):
        out = self.layer1(x)
        out = self.sigmoid(out)
        out = self.layer2(out)
        out = self.sigmoid2(out)
        out = self.layer3(out)
        out = self.sigmoid3(out)
        return out

xor_model = XOR()
```

(여기도 은닉층 3개랑 동일)
__init__ 메서드 : 다층 퍼셉트론 정의
-> 3개의 은닉층, 1개의 출력층
각 층은 선형 변환과 활성화 함수인 시그모이드 함수로 구성

forward 메서드 : 입력 데이터를 각 층에 순전파 시킴

```
# 학습된 파라미터 값 대입
xor_model.layer1.weight.data = torch.tensor([[ 5.0629, -4.8229], [5.5597, -5.6119]])
xor_model.layer1.bias.data = torch.tensor([ 2.3824, -3.1163])
xor_model.layer2.weight.data = torch.tensor([[ 5.7263, -6.0702],[-4.4636,  4.5147]])
xor_model.layer2.bias.data = torch.tensor([-2.5706,  2.0293])
xor_model.layer3.weight.data = torch.tensor([[-10.4020,   7.6603]])
xor_model.layer3.bias.data = torch.tensor([1.5453])
```

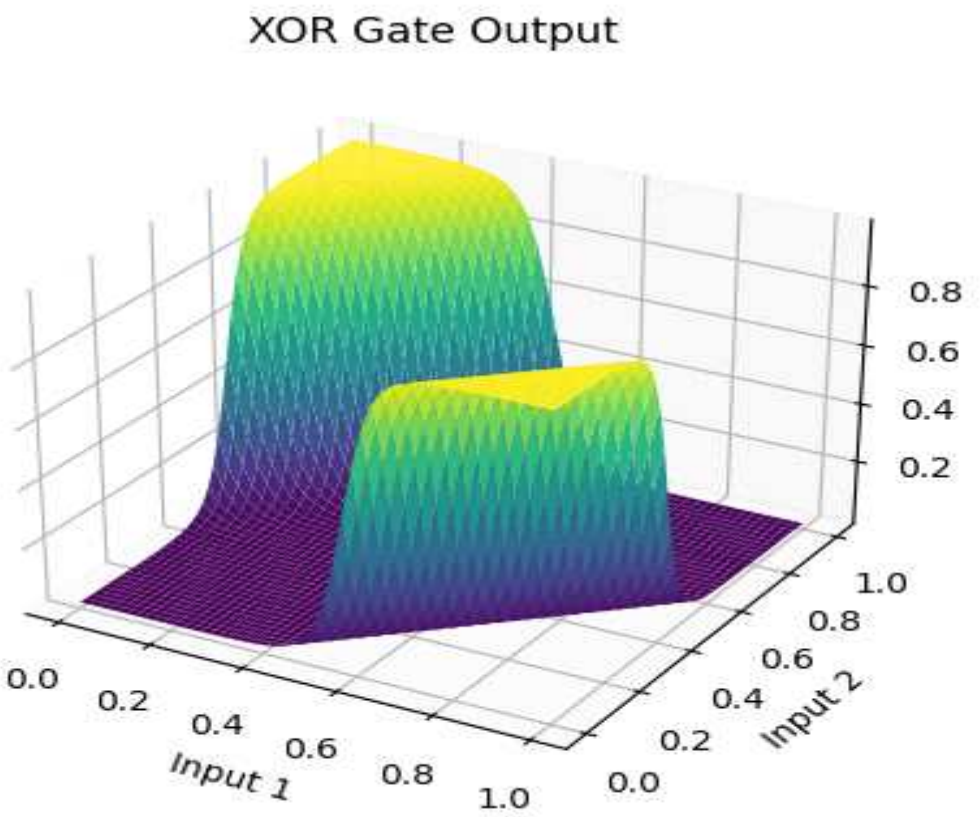
학습된 파라미터 대입

```
# 3차원 그래프 생성을 위한 입력 데이터 정의
x_values_xor = np.linspace(0, 1, 50)
y_values_xor = np.linspace(0, 1, 50)
xx_xor, yy_xor = np.meshgrid(x_values_xor, y_values_xor)
xy_xor = np.column_stack([xx_xor.ravel(), yy_xor.ravel()])
xy_tensor_xor = torch.tensor(xy_xor, dtype=torch.float)

# 각 점에서의 예측값 계산
with torch.no_grad():
    predictions_xor = xor_model(xy_tensor_xor).numpy().reshape(xx_xor.shape)

# 면 그래프로 출력 시각화
fig = plt.figure()
ax_xor = fig.add_subplot(111, projection='3d')
ax_xor.plot_surface(xx_xor, yy_xor, predictions_xor, cmap='viridis')

ax_xor.set_xlabel('Input 1')
ax_xor.set_ylabel('Input 2')
ax_xor.set_zlabel('Output')
ax_xor.set_title('XOR Gate Output')
plt.show()
```



이건 추가로... 이클래스에 인공신경망 설계한 구조 설명하라는 내용이 있어서 챗지피티에게 이 코드 보여주고 인공신경망 설계한 구조 설명하라 했더니 이렇게 나와서 혹시나 참고할 수 있을까 싶어 보냅니다.. 은닉층 3개 보고서에서 설명한 내용이랑 비슷하지만 더 자세함!! 그저 참고용

데이터 준비:
입력 데이터(X)는 XOR 게이트의 입력값을 나타내며, 두 개의 특성을 가진 4개의 샘플로 구성됩니다. 각 샘플은 [0, 0], [0, 1], [1, 0], [1, 1]과 같이 두 개의 이진 특성을 가지고 있습니다.
출력 데이터(y)는 XOR 게이트의 결과값을 나타내며, 각 입력값에 대한 XOR 게이트의 출력값을 나타내는 4개의 이진 값으로 구성됩니다.

신경망 정의:
XOR 클래스는 PyTorch의 nn.Module을 상속하여 정의됩니다. 이 클래스는 다층 퍼셉트론(Multi-Layer Perceptron, MLP)을 구현합니다.
nn.Linear를 사용하여 선형 레이어를 정의합니다. 이 코드에서는 입력 차원이 2이고 출력 차원이 2인 레이어를 두 번, 입력 차원이 2이고 출력 차원이 1인 레이어를 한 번 사용합니다.
활성화 함수로는 nn.Sigmoid를 사용하여 출력값을 [0, 1] 범위로 변환합니다.

순방향 전파 정의:
forward 메서드는 입력 데이터를 받아 신경망을 통해 전달하여 출력값을 반환합니다.
nn.Sequential을 사용하여 각 레이어를 차례대로 적용합니다. 입력 데이터는 각 레이어를 통과하면서 변환되어 최종적으로 출력값이 생성됩니다.
모델 초기화:
XOR 클래스의 인스턴스를 생성하여 모델을 초기화합니다. 이것은 이후에 훈련을 진행할 모델을 나타냅니다.

모델 훈련 및 평가:
train 함수를 사용하여 모델을 훈련합니다. 이 함수는 주어진 데이터와 모델에 대해 주어진 epoch 수만큼 반복하여 모델을 최적화합니다.
훈련된 모델은 입력 데이터에 대한 예측을 생성하고, 이를 기반으로 예측된 클래스를 계산합니다.

모델 파라미터 확인:
named_parameters 메서드를 사용하여 훈련된 모델의 파라미터를 확인합니다. 이를 통해 각 레이어의 가중치(weight)와 편향(bias) 값을 확인할 수 있습니다.
이 코드는 주어진 입력에 대한 XOR 게이트의 출력을 예측하기 위해 다층 퍼셉트론을 구축하고 훈련하는 과정을 담고 있습니다. 해당 신경망은 PyTorch를 사용하여 구현되었으며, 역전파(backpropagation)를 통해 최적화되는 과정을 포함하고 있습니다.