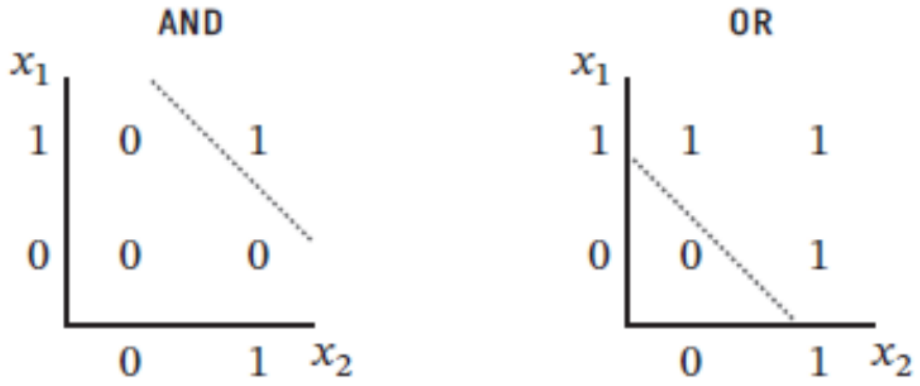
인공지능보안_중간과제발표

4조
20220896 정후리
20220967 이가현
20220817 손민정
20220625 손재윤

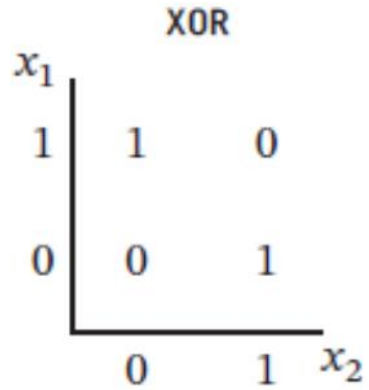
AND/OR 게이트

- AND, OR 연산을 수행하는 퍼셉트론



XOR 게이트

퍼셉트론의 작동 검증 - XOR 연산



XOR 연산에 대한 진리표

XOR		
x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

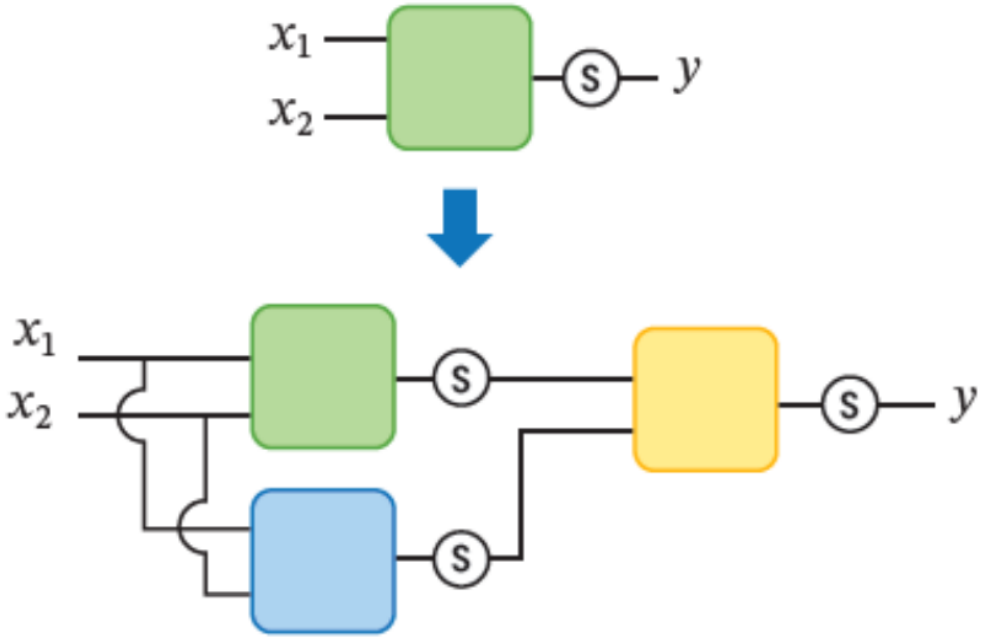
Q) 0그룹과 1그룹으로 완벽하게 분할하는 선을 긋는 방법

XOR 게이트 문제 해결

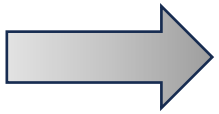
- 다층으로 구성되자 공간이 확장됐다

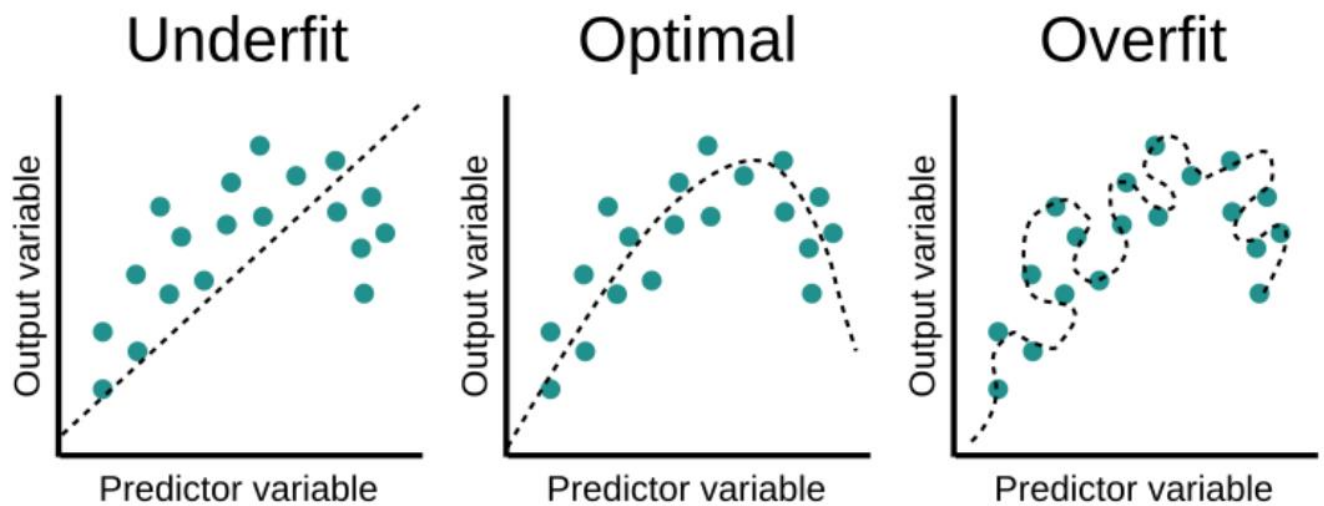
가설식이 존재하는 차원의 확장

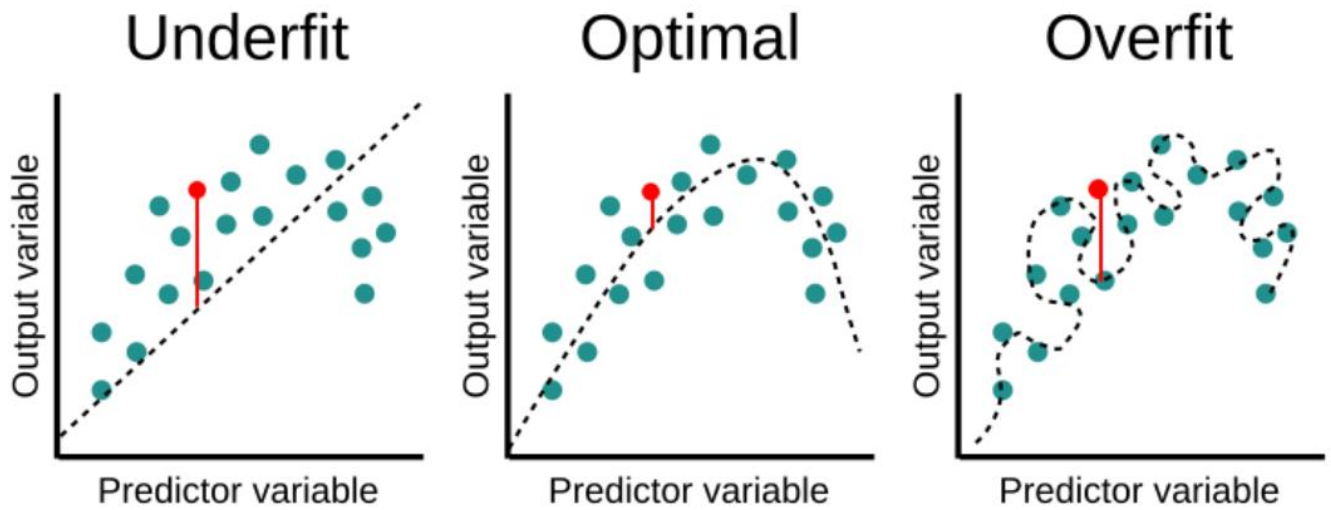
- 다층 퍼셉트론의 구현



XOR 게이트 훈련과정

XOR 클래스(신경망 정의)  **순전파 정의**  **모델링 관련 작업 수행**


Underfit
Optimal
Overfit
Output variable
Predictor variable
Output variable
Predictor variable
Output variable
Predictor variable



Bias(편향)

Variance(분산)

학습률(lr)

가중치 및 편향 업데이트의 결정하므로 0.001~0.1 사이의 수를 대입하여 최적의 값 찾음

에폭(epoch)

데이터셋 전체를 한 번 학습하는 단위

HiddenLayer(은닉층) 2개

```
[ ] # hidden layers 2개
    class XOR(nn.Module):
        def __init__(self):
            super().__init__()
            self.model = nn.Sequential(
                nn.Linear(2,2),
                nn.Sigmoid(),
                nn.Linear(2,2),
                nn.Sigmoid(),
                nn.Linear(2,1),
                nn.Sigmoid()
            )

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

```
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: 0.00550277065485715 9
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]]))
```

```
[ ] 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))
```

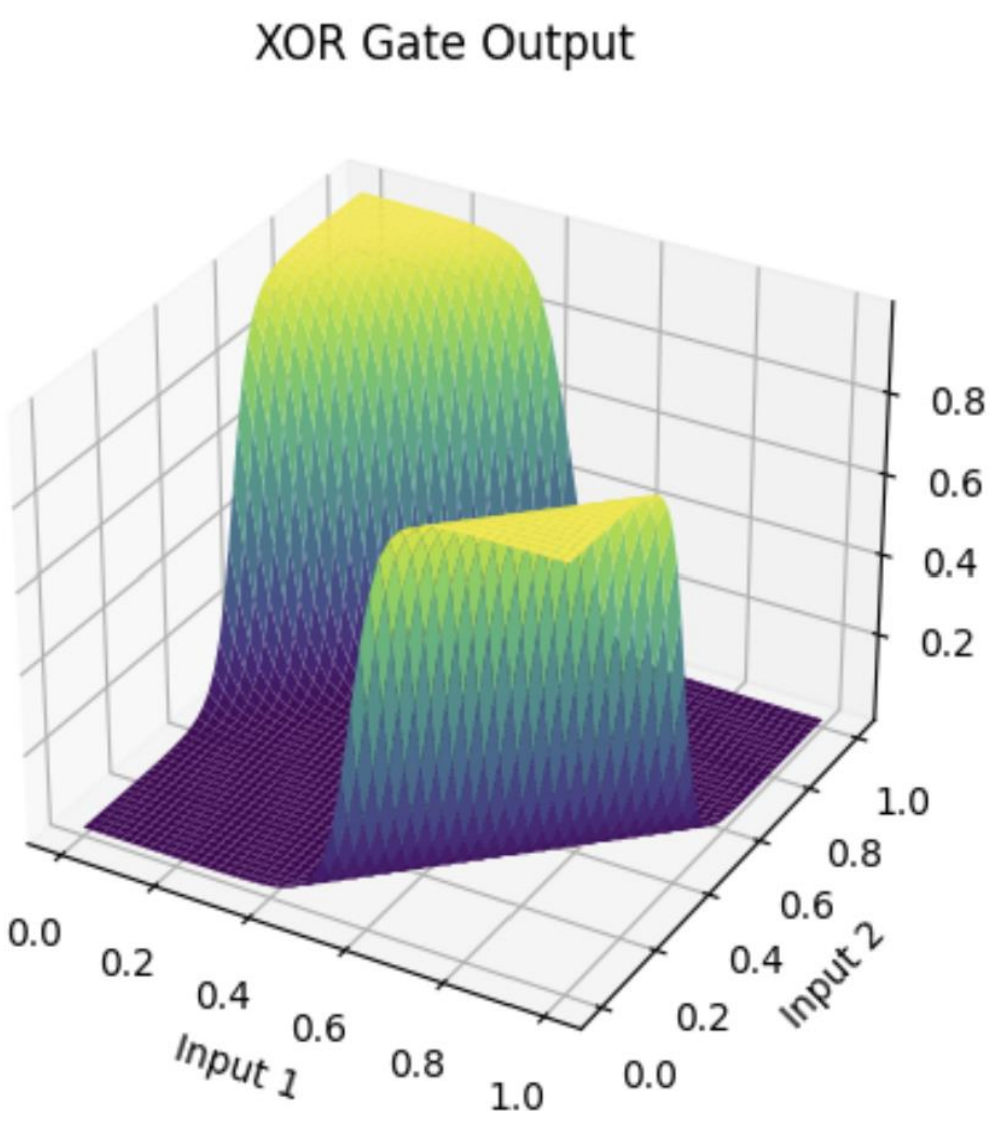
```python
import torch

# 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]])
```

```
[ ]
```

```python
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()
```

```python
# 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])

# 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()
```


XOR Gate Output
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0
Input 1
0.0
0.2
0.4
0.6
0.8
1.0
Input 2

HiddenLayer(은닉층) 3개

```
# hidden layers 3개
class XOR(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(2, 2),
            nn.Sigmoid(),
            nn.Linear(2, 2),
            nn.Sigmoid(),
            nn.Linear(2, 2),
            nn.Sigmoid(),
            nn.Linear(2, 1),
            nn.Sigmoid()
        )

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

```python
print("최적 시나리오:")
train(X, y, epochs=10000, lr=0.01, optimizer='Adam', model=model, every_print=1000)
predictions = model(X)
predicted_classes = torch.where(predictions > 0.5, 1, 0)
print("Predictions:", predictions)
print("Predicted Classes:", predicted_classes)

for name, param in model.named_parameters():
    print(name, param.data)
```

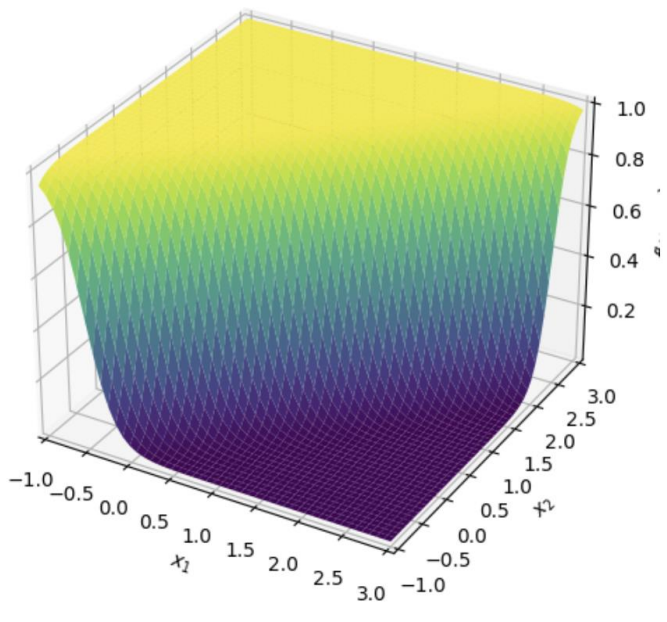
```
최적 시나리오:
Epoch: 1000, loss: 0.00987507775425911
Epoch: 2000, loss: 0.002397105097770691
Epoch: 3000, loss: 0.0010224109282717109
Epoch: 4000, loss: 0.0005195625126361847
Epoch: 5000, loss: 0.0002860369277186692
Epoch: 6000, loss: 0.0001641740382183432
Epoch: 7000, loss: 9.645272803027183e-05
Epoch: 8000, loss: 5.7423399994149804e-05
Epoch: 9000, loss: 3.44630861945916e-05
Epoch: 10000, loss: 2.077104181807954e-05
Predictions: tensor([[2.1226e-05],
        [9.9998e-01],
        [9.9998e-01],
        [2.1185e-05]], grad_fn=<SigmoidBackward0>)
Predicted Classes: tensor([[0],
        [1],
        [1],
        [0]])
model.0.weight tensor([[-6.1799,  6.3918],
        [-7.2902,  7.3291]])
model.0.bias tensor([ 3.1900, -4.3039])
model.2.weight tensor([[-5.7749,  6.0148],
        [ 5.7013, -6.6082]])
model.2.bias tensor([ 2.8003, -2.6538])
model.4.weight tensor([[ 6.3596, -6.5419],
        [-6.6853,  7.0563]])
model.4.bias tensor([ 0.1071, -0.1949])
model.6.weight tensor([[ 11.2224, -10.4749]])
model.6.bias tensor([-0.3518])
```

```python
import torch

# 주어진 파라미터
w0 = torch.tensor([[-6.1799,  6.3918], [-7.2902,  7.3291]])
b0 = torch.tensor([ 3.1900, -4.3039])
w2 = torch.tensor([[-5.7749,  6.0148], [ 5.7013, -6.6082]])
b2 = torch.tensor([ 2.8003, -2.6538])
w4 = torch.tensor([[ 6.3596, -6.5419],[-6.6853,  7.0563]])
b4 = torch.tensor([ 0.1071, -0.1949])
w6 = torch.tensor([[ 11.2224, -10.4749]])
b6 =  torch.tensor([-0.3518])

# 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)

# 세 번째 은닉층의 출력 계산
layer3_output = sigmoid(layer2_output @ w4.T + b4)

# 최종 출력 계산
final_output = sigmoid(layer3_output @ w6.T + b6)
print(final_output)
```

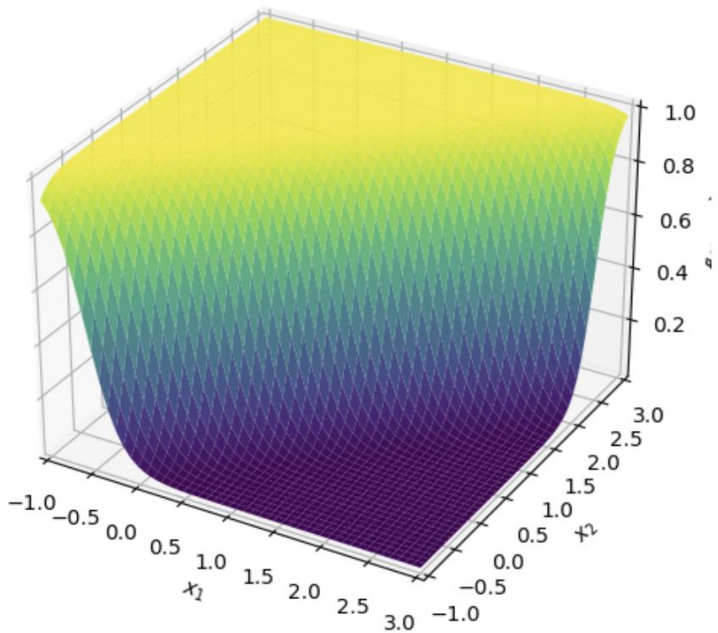
```
tensor([[2.1226e-05],
        [9.9998e-01],
        [9.9998e-01],
        [2.1185e-05]])
```

```
[ ]  import torch
     from sympy import symbols, exp
     from sympy.plotting import plot3d

     # 변수 정의
     x, x1, x2 = symbols('x x1 x2')
     w1, w2, w3, w4, w5, w6 = symbols('w1 w2 w3 w4 w5 w6')
     b1, b2, b3 = symbols('b1 b2 b3')
     sigmoid = 1 / (1+exp(-x))
```

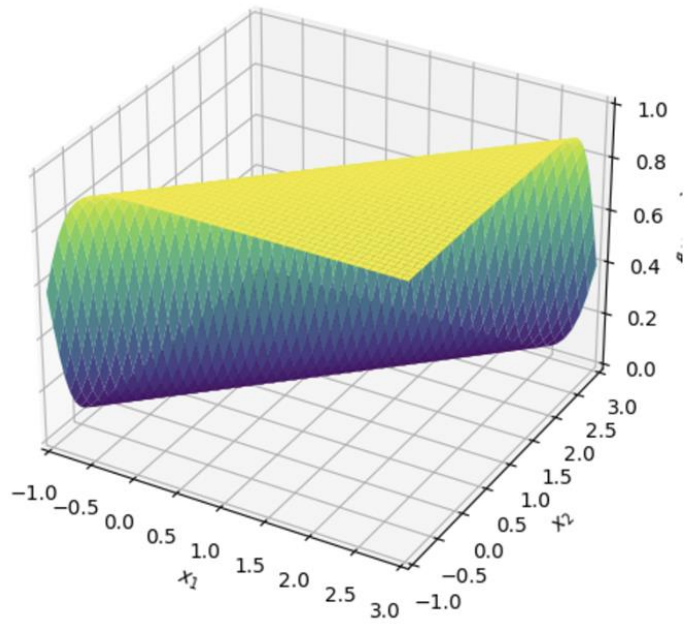
```
# 첫 번째 은닉층의 출력 계산식 정의
h1= sigmoid.subs({'x': w1*x1 + w2*x2 + b1})

#3d출력
plot3d(h1.subs({'w1': -6.1799, 'w2': 6.3918, 'b1': 3.1900}) , (x1, -1, 3), (x2, -1, 3))
```




```
# 두 번째 은닉층의 출력 계산식 정의
h2= sigmoid.subs({'x': w3*x1 + w4*x2 + b2})

#3d출력
plot3d(h2.subs({'w3': -5.7749, 'w4': 6.0148, 'b2': 2.8003}) , (x1, -1, 3), (x2, -1, 3))
```

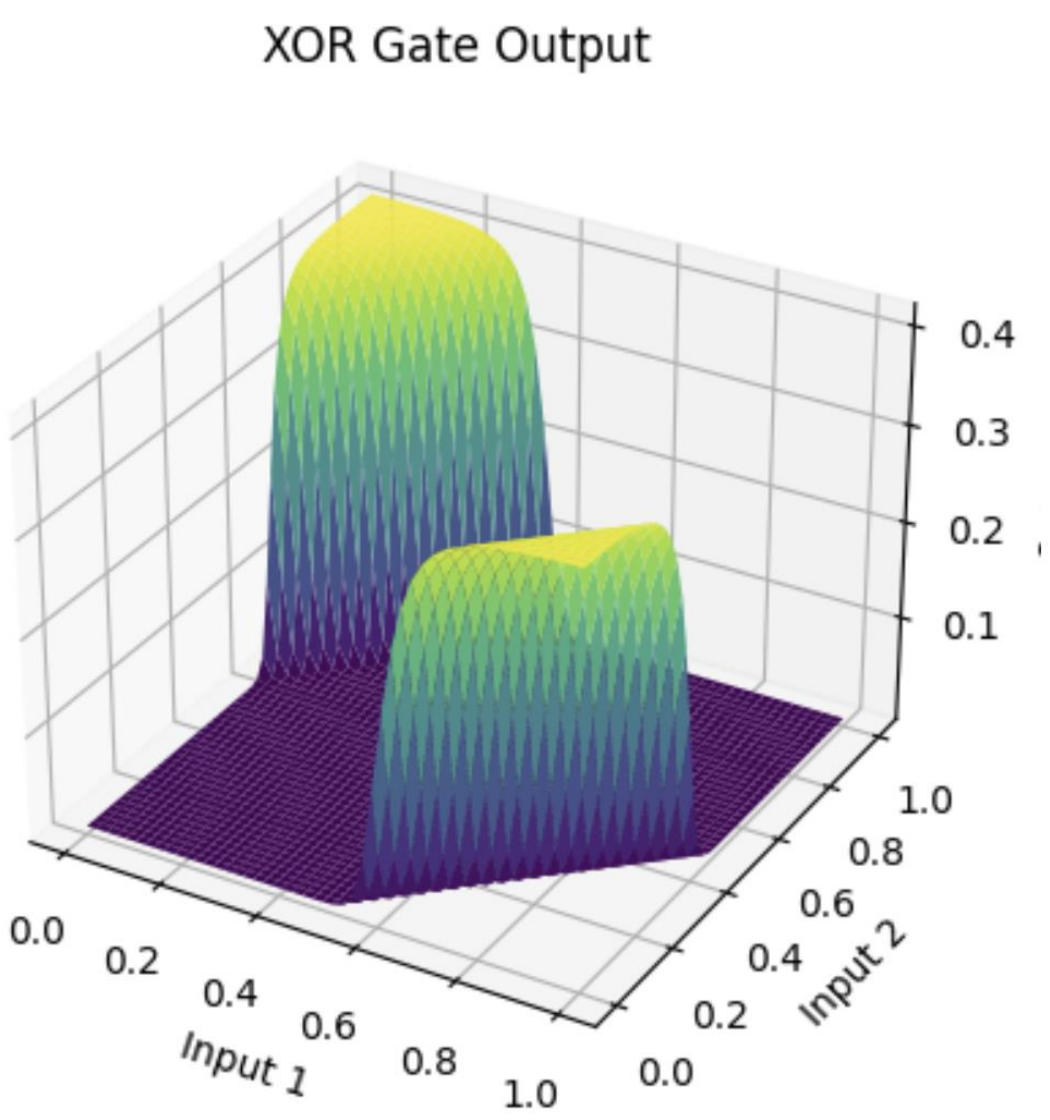



```
#세 번째 은닉층의 출력 계산식 정의
h3 = sigmoid.subs({'x': w5*x1 + w6*x2 + b3})

#3d
plot3d(h3.subs({'w5': 6.3596, 'w6': -6.5419, 'b3': 0.1071}) , (x1, -1, 3), (x2, -1, 3))
```



```
[ ] 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, 2)
            self.sigmoid3 = torch.nn.Sigmoid()
            self.layer4 = torch.nn.Linear(2, 1)
            self.sigmoid4 = 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)
            out = self.layer4(out)
            out = self.sigmoid4(out)
            return out

    xor_model = XOR()
```

```
[ ] # 학습된 파라미터 값 대입
    xor_model.layer1.weight.data = torch.tensor([[-6.1799, 6.3918],
                                                 [-7.2902, 7.3291]])
    xor_model.layer1.bias.data = torch.tensor([3.1900, -4.3039])
    xor_model.layer2.weight.data = torch.tensor([[-5.7749, 6.0148],
                                                 [5.7013, -6.6082]])
    xor_model.layer2.bias.data = torch.tensor([2.8003, -2.6538])
    xor_model.layer3.weight.data = torch.tensor([[-6.3596,   -6.5419],
                                                 [-6.6853, 7.0563]])
    xor_model.layer3.bias.data = torch.tensor([0.1071, -0.1949])
    xor_model.layer4.weight.data = torch.tensor([[11.2224,   -10.4749]])
    xor_model.layer4.bias.data = torch.tensor([-0.3518])

    # 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()
```


XOR Gate Output
0.4
0.3
0.2
0.1
0.0
0.2
0.4
0.6
0.8
1.0
Input 1
1.0
0.8
0.6
0.4
0.2
0.0
Input 2

HiddenLayer(은닉층) 4개

```
[ ]  # hidden layers 4개
     class XOR(nn.Module):
         def __init__(self):
             super().__init__()
             self.weight = torch.tensor([0.5, 0.5], dtype=torch.float)
             self.bias = torch.tensor([-0.25], dtype=torch.float)
             self.model = nn.Sequential(
                 nn.Linear(2, 2),  # 첫 번째 히든 레이어
                 nn.Sigmoid(),
                 nn.Linear(2, 2),  # 두 번째 히든 레이어
                 nn.Sigmoid(),
                 nn.Linear(2, 2),  # 세 번째 히든 레이어
                 nn.Sigmoid(),
                 nn.Linear(2, 2),  # 네 번째 히든 레이어
                 nn.Sigmoid(),
                 nn.Linear(2, 1),  # 출력 레이어
                 nn.Sigmoid()
             )

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

```
train(X, y, epochs=30000, lr=0.01, optimizer='Adam', model=model, every_print=1000, reset='xavier')
model(X) ,torch.where(model(X) > 0.5, 1, 0)
```

```
Epoch: 1000, loss: 0.40577399730682373
Epoch: 2000, loss: 0.004747707396745682
Epoch: 3000, loss: 0.0017148074693977833
Epoch: 4000, loss: 0.0008250309037975967
Epoch: 5000, loss: 0.00044312424142844975
Epoch: 6000, loss: 0.00025109335547313094
Epoch: 7000, loss: 0.00014644660404883325
Epoch: 8000, loss: 8.683284249855205e-05
Epoch: 9000, loss: 5.1992927183164284e-05
Epoch: 10000, loss: 3.128772004856728e-05
Epoch: 11000, loss: 1.8882270524045452e-05
Epoch: 12000, loss: 1.1437708963057958e-05
Epoch: 13000, loss: 6.949971066205762e-06
Epoch: 14000, loss: 4.1917319322237745e-06
Epoch: 15000, loss: 2.552897058194503e-06
Epoch: 16000, loss: 1.5286917687262758e-06
Epoch: 17000, loss: 9.340104725197307e-07
Epoch: 18000, loss: 5.752655169999343e-07
Epoch: 19000, loss: 3.464963924670883e-07
Epoch: 20000, loss: 2.210717440220833e-07
Epoch: 21000, loss: 1.1927453158477874e-07
Epoch: 22000, loss: 9.975076409318717e-08
Epoch: 23000, loss: 2.1067691591269977e-08
Epoch: 24000, loss: 1.1189852600068662e-08
Epoch: 25000, loss: 6.3672840333595104e-09
Epoch: 26000, loss: 3.7683127729337684e-09
Epoch: 27000, loss: 2.297172230214528e-09
Epoch: 28000, loss: 1.4371319689843176e-09
Epoch: 29000, loss: 9.272541068305884e-10
Epoch: 30000, loss: 6.187308443372785e-10
(tensor([[1.2353e-09],
         [1.0000e+00],
         [1.0000e+00],
         [1.2384e-09]], grad_fn=<SigmoidBackward0>),
 tensor([[0],
         [1],
         [1],
         [0]]))
```

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

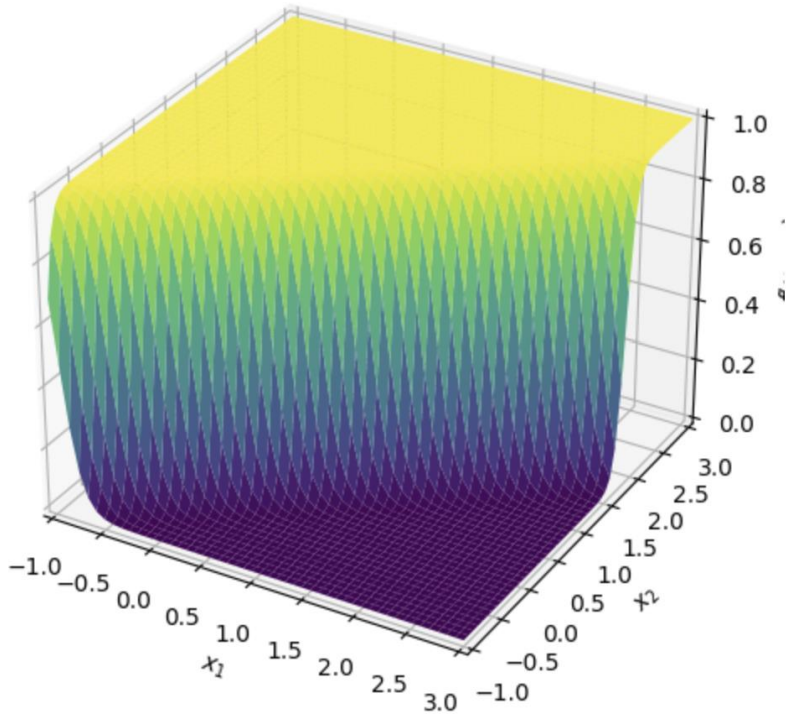
```
('model.0.weight', Parameter containing:
tensor([[-10.3267,  14.3530],
        [ -3.8846,   6.2506]], requires_grad=True))
('model.0.bias', Parameter containing:
tensor([ 4.8114, -3.5770], requires_grad=True))
('model.2.weight', Parameter containing:
tensor([[-4.5832,  5.1053],
        [ 6.1589, -5.0255]], requires_grad=True))
('model.2.bias', Parameter containing:
tensor([ 1.5159, -3.0480], requires_grad=True))
('model.4.weight', Parameter containing:
tensor([[ 5.6698, -6.0909],
        [-6.0369,  7.5744]], requires_grad=True))
('model.4.bias', Parameter containing:
tensor([ 0.4247, -1.0466], requires_grad=True))
('model.6.weight', Parameter containing:
tensor([[ 7.1937, -7.3540],
        [-7.5309,  7.5448]], requires_grad=True))
('model.6.bias', Parameter containing:
tensor([0.0227, 0.0733], requires_grad=True))
('model.8.weight', Parameter containing:
tensor([[ 17.7575, -19.4533]], requires_grad=True))
('model.8.bias', Parameter containing:
tensor([-1.0813], requires_grad=True))
```

```
from torch import nn

w0 = torch.tensor([[-10.3267,  14.3530], [ -3.8846,   6.2506]])
b0 = torch.tensor([4.8114, -3.5770])
w2 = torch.tensor([[-4.5832,  5.1053], [6.1589, -5.0255]])
b2 = torch.tensor([1.5159, -3.0480])
w4 = torch.tensor([[5.6698, -6.0909],[-6.0369,  7.5744]])
b4 = torch.tensor([0.4247, -1.0466])
w6 = torch.tensor([[7.1937, -7.3540],[-7.5309,  7.5448]])
b6 =  torch.tensor([0.0227, 0.0733])
w8 = torch.tensor([[ 17.7575, -19.4533]])
b8 =  torch.tensor([-1.0813])

# 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)

# 세 번째 은닉층의 출력 계산
layer3_output = sigmoid(layer2_output @ w4.T + b4)

# 네 번째 은닉층의 출력 계산
layer4_output = sigmoid(layer3_output @ w6.T + b6)

#최종 출력 계산
final_output = sigmoid(layer4_output @ w8.T + b8)
print(final_output)
```

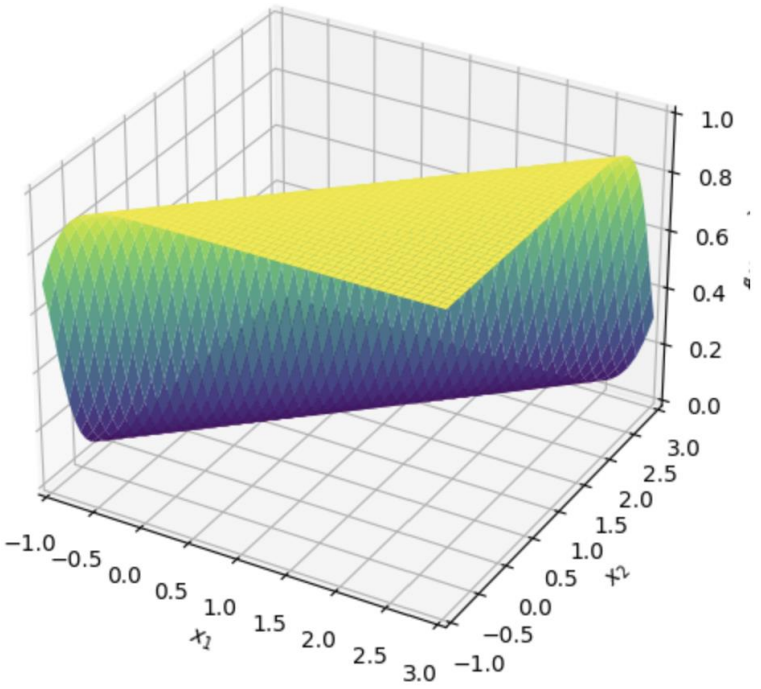
```
tensor([[1.2353e-09],
        [1.0000e+00],
        [1.0000e+00],
        [1.2384e-09]])
```

```
[ ]  import torch
     from sympy import symbols, exp
     from sympy.plotting import plot3d

     # 변수 정의
     x, x1, x2 = symbols('x x1 x2')
     w1, w2, w3, w4, w5, w6, w7, w8 = symbols('w1 w2 w3 w4 w5 w6 w7 w8')
     b1, b2, b3 = symbols('b1 b2 b3')
     sigmoid = 1 / (1+exp(-x))
```

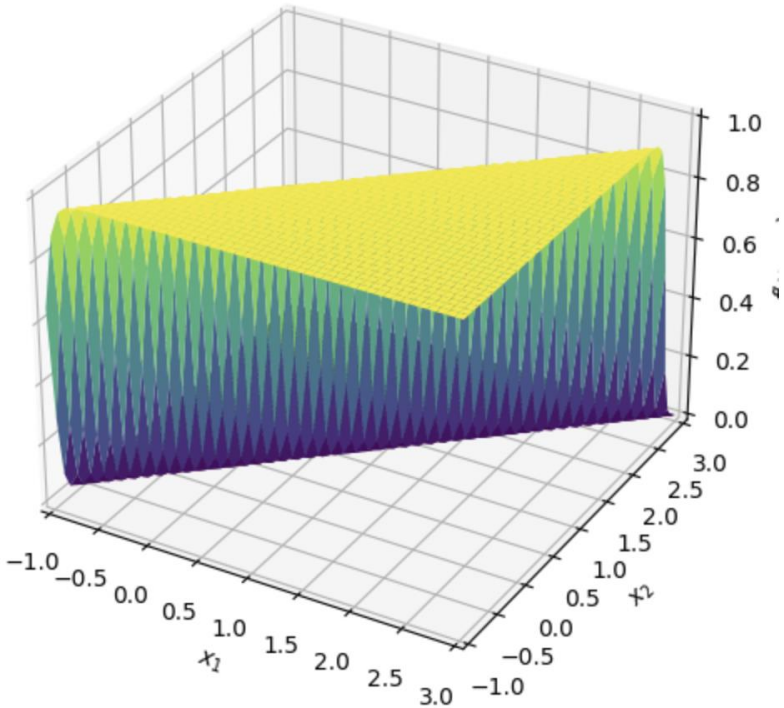
```
# 첫 번째 은닉층의 출력 계산식 정의
h1= sigmoid.subs({'x': w1*x1 + w2*x2 + b1})
plot3d(h1.subs({'w1': -10.3267, 'w2': 14.3530, 'b1': 4.8114}), (x1, -1, 3), (x2, -1, 3))
```



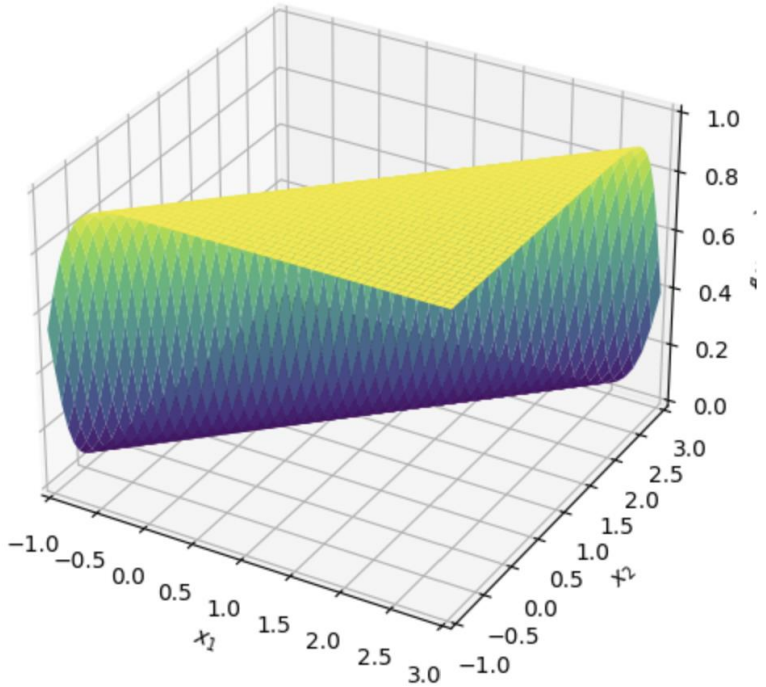

```
# 두 번째 은닉층의 출력 계산식 정의
h2= sigmoid.subs({'x': w3*x1 + w4*x2 + b2})
plot3d(h2.subs({'w3': 5.6698, 'w4': -6.0909, 'b2':  0.4247}), (x1, -1, 3), (x2, -1, 3))
```



```
# 세 번째 은닉층의 출력 계산식 정의
h3 = sigmoid.subs({'x': w5*x1 + w6*x2 + b3})
plot3d(h3.subs({'w5': 7.1937, 'w6': -7.3540, 'b3': 0.0227}), (x1, -1, 3), (x2, -1, 3))
```

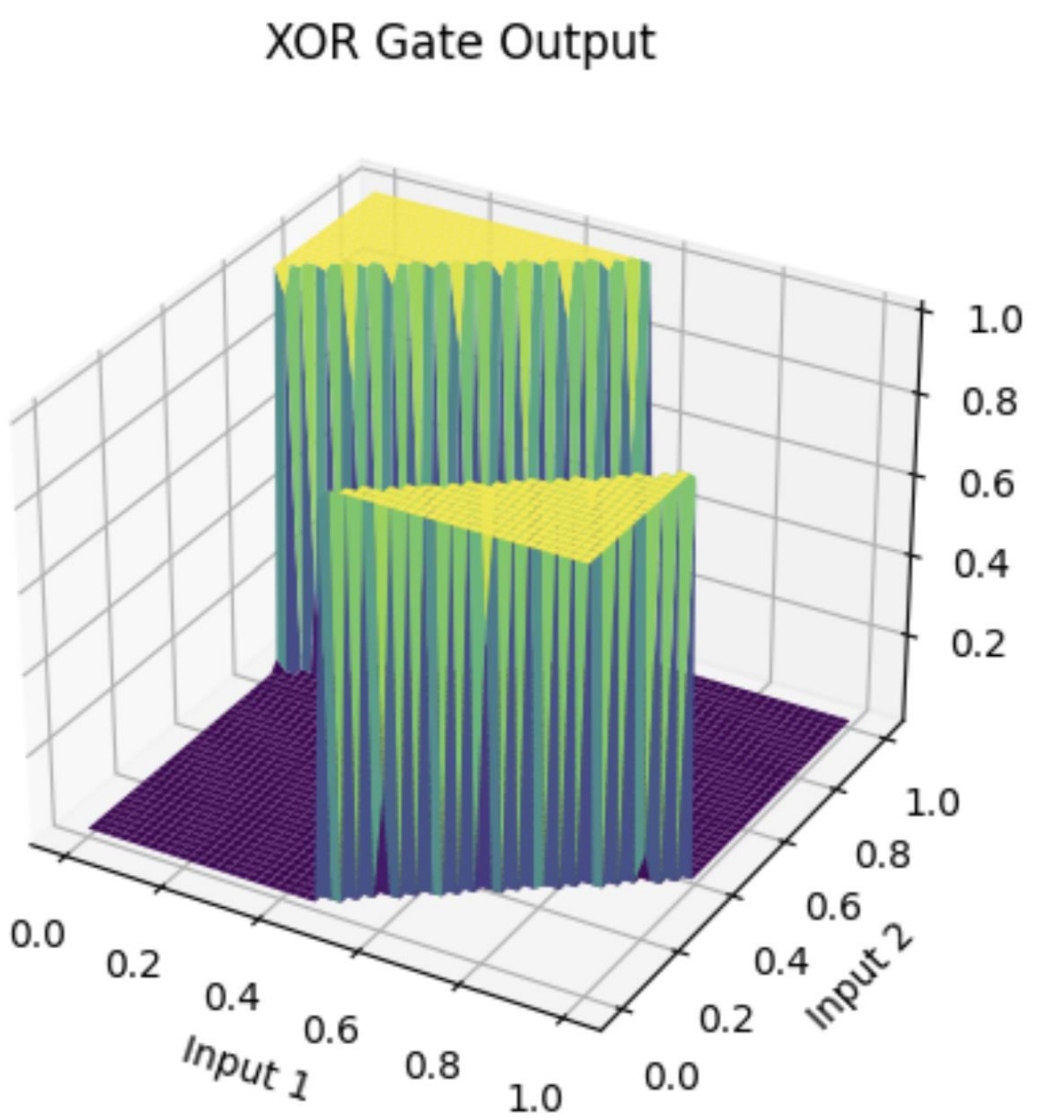



```
# 네 번째 은닉층의 출력 계산식 정의
h4 = sigmoid.subs({'x': 17.7575*x1 - 19.4553*x2 - 1.0813})
plot3d(h4.subs({'w7': 17.7575, 'w8': -19.4553, 'b4': -1.0813}), (x1, -1, 3), (x2, -1, 3))
```



```python
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, 2)
        self.sigmoid3 = torch.nn.Sigmoid()
        self.layer4 = torch.nn.Linear(2, 2)
        self.sigmoid4 = torch.nn.Sigmoid()
        self.layer5 = torch.nn.Linear(2, 1)
        self.sigmoid5 = 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)
        out = self.layer4(out)
        out = self.sigmoid4(out)
        out = self.layer5(out)
        out = self.sigmoid5(out)
        return out

xor_model = XOR()
```

```python
# 학습된 파라미터 값 대입
xor_model.layer1.weight.data = torch.tensor([[-10.3267,  14.3530],
                                             [ -3.8846,   6.2506]])
xor_model.layer1.bias.data = torch.tensor([ 4.8114, -3.5770])
xor_model.layer2.weight.data = torch.tensor([[-4.5832,  5.1053],
                                             [ 6.1589, -5.0255]])
xor_model.layer2.bias.data = torch.tensor([ 1.5159, -3.0480])
xor_model.layer3.weight.data = torch.tensor([[ 5.6698, -6.0909],
                                             [-6.0369,  7.5744]])
xor_model.layer3.bias.data = torch.tensor([ 0.4247, -1.0466])
xor_model.layer4.weight.data = torch.tensor([[ 7.1937, -7.3540],
                                             [-7.5309,  7.5448]])
xor_model.layer4.bias.data = torch.tensor([0.0227, 0.0733])
xor_model.layer5.weight.data = torch.tensor([[ 17.7575, -19.4533]])
xor_model.layer5.bias.data = torch.tensor([-1.0813])

# 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)

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()
```


XOR Gate Output
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0
Input 1
0.0
0.2
0.4
0.6
0.8
1.0
Input 2

HiddenLayer(은닉층) 5개

```
[ ] # hidden layers 5개
    class XOR(nn.Module):
        def __init__(self):
            super().__init__()
            self.model = nn.Sequential(
                nn.Linear(2, 2),
                nn.Sigmoid(),
                nn.Linear(2, 2),
                nn.Sigmoid(),
                nn.Linear(2, 2),
                nn.Sigmoid(),
                nn.Linear(2, 2),
                nn.Sigmoid(),
                nn.Linear(2, 2),
                nn.Sigmoid(),
                nn.Linear(2, 1),
                nn.Sigmoid()
            )

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

```
train(X, y, epochs=10000, lr=0.01, optimizer='Adam', model=model, every_print=1000)
model(X), torch.where(model(X)>0.5, 1, 0)
```

```
Epoch: 1000, loss: 0.004637928679585457
Epoch: 2000, loss: 0.0012940652668476105
Epoch: 3000, loss: 0.0005670603131875396
Epoch: 4000, loss: 0.0002909837930928916
Epoch: 5000, loss: 0.0001608799066161736 8
Epoch: 6000, loss: 9.252912423107773e-05
Epoch: 7000, loss: 5.439734013634734e-05
Epoch: 8000, loss: 3.241356898797676e-05
Epoch: 9000, loss: 1.9457962480373 68e-05
Epoch: 10000, loss: 1.171560961665818 5e-05
(tensor([[1.1839e-05],
         [9.9999e-01],
         [9.9999e-01],
         [1.1646e-05]], grad_fn=<SigmoidBackward0>),
 tensor([[0],
         [1],
         [1],
         [0]]))
```

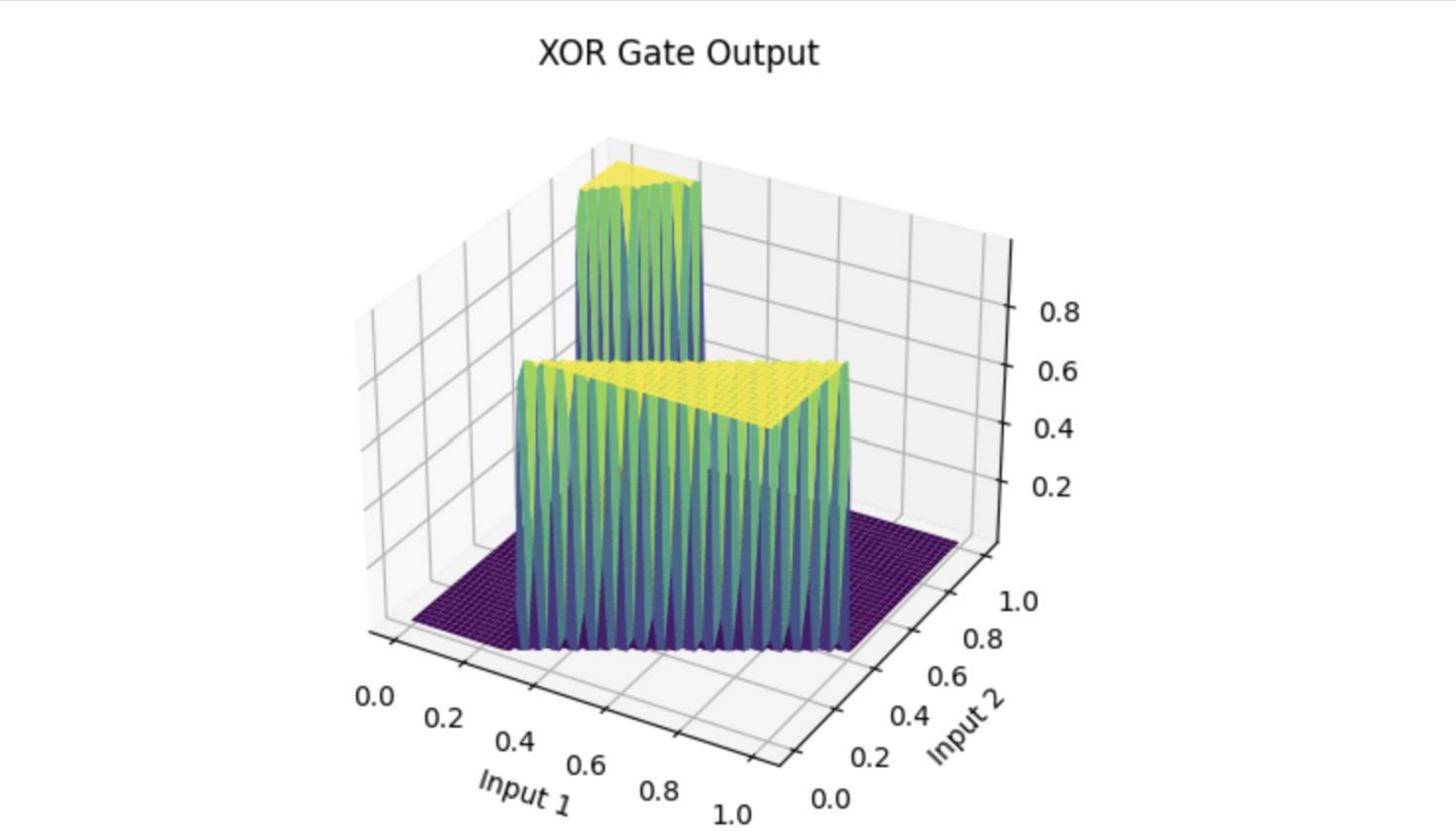
```python
for x in model.named_parameters():
    print(x)
```

```
('model.0.weight', Parameter containing:
tensor([[-6.1971,  7.7637],
        [ 2.8193, -4.9432]], requires_grad=True))
('model.0.bias', Parameter containing:
tensor([-5.9793,  0.4553], requires_grad=True))
('model.2.weight', Parameter containing:
tensor([[-5.2246, -3.3166],
        [-4.9951, -3.9336]], requires_grad=True))
('model.2.bias', Parameter containing:
tensor([2.1632, 2.6653], requires_grad=True))
('model.4.weight', Parameter containing:
tensor([[ 4.9992,  5.4489],
        [-5.2453, -5.6580]], requires_grad=True))
('model.4.bias', Parameter containing:
tensor([-4.0145,  4.1987], requires_grad=True))
('model.6.weight', Parameter containing:
tensor([[-5.7084,  5.1748],
        [-4.9997,  5.9229]], requires_grad=True))
('model.6.bias', Parameter containing:
tensor([ 0.1955, -0.5327], requires_grad=True))
('model.8.weight', Parameter containing:
tensor([[ 6.3776,  6.0508],
        [-6.3812, -5.9339]], requires_grad=True))
('model.8.bias', Parameter containing:
tensor([-5.8751,  5.6763], requires_grad=True))
('model.10.weight', Parameter containing:
tensor([[ 11.3377, -11.4957]], requires_grad=True))
('model.10.bias', Parameter containing:
tensor([0.0604], requires_grad=True))
```

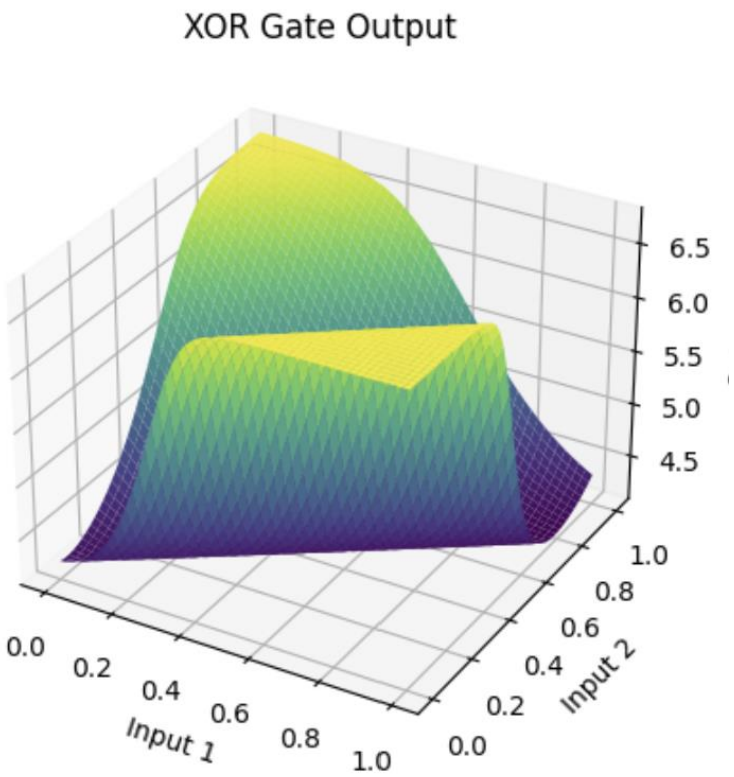
```python
import torch
import torch.nn as nn

# 주어진 파라미터
w0 = torch.tensor([[-6.1971,  7.7637], [ 2.8193, -4.9432]])
b0 = torch.tensor([-5.9793,  0.4553])
w2 = torch.tensor([[-5.2246, -3.3166], [-4.9951, -3.9336]])
b2 = torch.tensor([2.1632, 2.6653])
w4 = torch.tensor([[ 4.9992,  5.4489], [-5.2453, -5.6580]])
b4 = torch.tensor([-4.0145,  4.1987])
w6 = torch.tensor([[-5.7084,  5.1748], [-4.9997,  5.9229]])
b6 =  torch.tensor([ 0.1955, -0.5327])
w8 = torch.tensor([[ 6.3776,  6.0508], [-6.3812, -5.9339]])
b8 = torch.tensor([-5.8751,  5.6763])
w10 = torch.tensor([[ 11.3377, -11.4957]])
b10 = torch.tensor([0.0604])

# 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)

# 세 번째 은닉층의 출력 계산
layer3_output = sigmoid(layer2_output @ w4.T + b4)

# 네 번째 은닉층의 출력 계산
layer4_output = sigmoid(layer3_output @ w6.T + b6)

# 다섯 번째 은닉층의 출력 계산
layer5_output = sigmoid(layer4_output @ w8.T + b8)

# 최종 출력 계산
final_output = sigmoid(layer5_output @ w10.T + b10)
print(final_output)
```

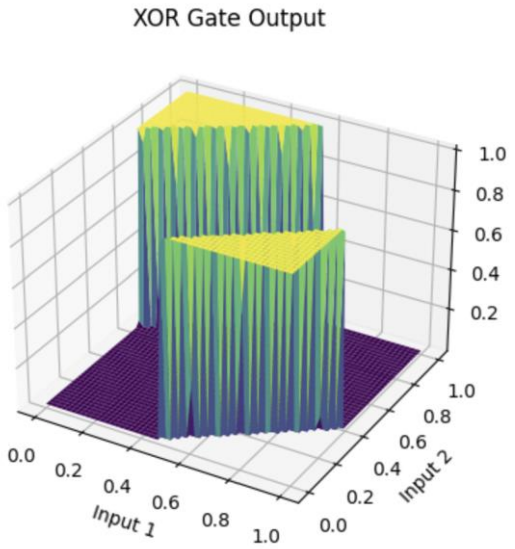
```
tensor([[1.1840e-05],
        [9.9999e-01],
        [9.9999e-01],
        [1.1647e-05]])
```

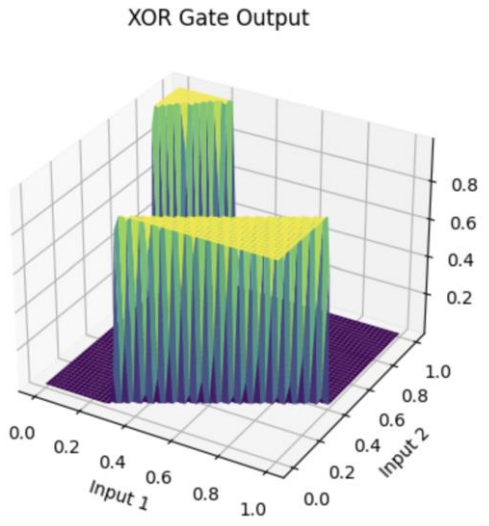
```python
import numpy as np
import torch
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# XOR 게이트를 위한 입력 데이터 정의
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, 2)
        self.sigmoid3 = torch.nn.Sigmoid()
        self.layer4 = torch.nn.Linear(2, 2)
        self.sigmoid4 = torch.nn.Sigmoid()
        self.layer5 = torch.nn.Linear(2, 2)
        self.sigmoid5 = torch.nn.Sigmoid()
        self.layer6 = torch.nn.Linear(2, 1)

    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)
        out = self.layer4(out)
        out = self.sigmoid4(out)
        out = self.layer5(out)
        out = self.sigmoid5(out)
        out = self.layer6(out)
        return torch.sigmoid(out)

xor_model = XOR()

# 학습된 파라미터 값 대입
xor_model.layer1.weight.data = torch.tensor([[-6.1971,  7.7637], [ 2.8193, -4.9432]])
xor_model.layer1.bias.data = torch.tensor([-5.9793,  0.4553])
xor_model.layer2.weight.data = torch.tensor([[-5.2246, -3.3166], [-4.9951, -3.9336]])
xor_model.layer2.bias.data = torch.tensor([2.1632, 2.6653])
xor_model.layer3.weight.data = torch.tensor([[ 4.9992,  5.4489], [-5.2453, -5.6580]])
xor_model.layer3.bias.data = torch.tensor([-4.0145,  4.1987])
xor_model.layer4.weight.data = torch.tensor([[-5.7084,  5.1748], [-4.9997,  5.9229]])
xor_model.layer4.bias.data = torch.tensor([0.1955, -0.5327])
xor_model.layer5.weight.data = torch.tensor([[ 6.3776,  6.0508], [-6.3812, -5.9339]])
xor_model.layer5.bias.data = torch.tensor([-5.8751,  5.6763])
xor_model.layer6.weight.data = torch.tensor([[11.3377, -11.4957]])
xor_model.layer6.bias.data = torch.tensor([0.0604])

# 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().squeeze()  # Fix: Add .squeeze()

# 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.reshape(xx_xor.shape), cmap='viridis')  # Fix: Reshape predictions_xor

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()
```

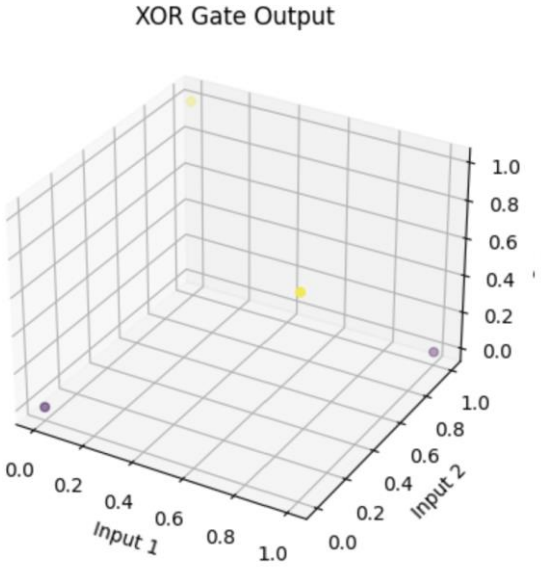

XOR Gate Output
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0
Input 1
0.0
0.2
0.4
0.6
0.8
1.0
Input 2

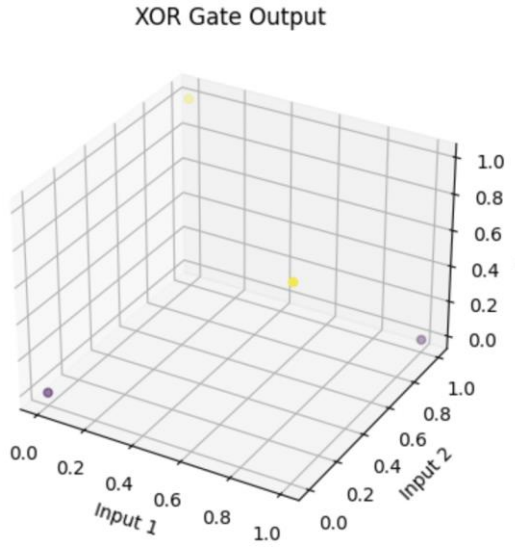
Insight


XOR Gate Output
6.5
6.0
5.5
5.0
4.5
0.0
0.2
0.4
0.6
0.8
1.0
Input 1
Input 2


XOR Gate Output
1.0
0.8
0.6
0.4
0.2
0.0
Input 1
Input 2


XOR Gate Output
0.8
0.6
0.4
0.2
0.0
Input 1
Input 2


XOR Gate Output
1.0
0.8
0.6
0.4
0.2
0.0
Input 1
Input 2


XOR Gate Output
1.0
0.8
0.6
0.4
0.2
0.0
Input 1
Input 2