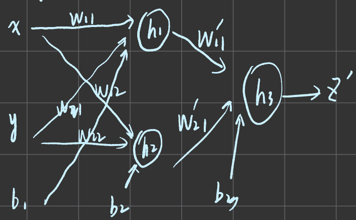
2. 解: (a) 因为如果没有这些非线性的激活函数，那么网络深度、宽度无论怎样增加整个网络都只是线性变换，则这些变换的复杂度将失去意义，所有模型都将只是简单的线性回归，而无法获取输入内容的更深层次的特征

(b) XOR真值表: 设 $z' = x \oplus y$

z' (y \ x)	0	1
0	0	1
1	1	0

结构:



设 $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix}^T \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$, $\begin{bmatrix} x_1' \\ y_1' \end{bmatrix} = \mathrm{ReLu}\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$

$z = \begin{bmatrix} W_{11}' \\ W_{21}' \end{bmatrix}^T \begin{bmatrix} x_1' \\ y_1' \end{bmatrix} + b_3$, $z' = \mathrm{ReLu}(z)$

利用pytorch获得参数，损失函数用MSE，优化器用SGD，学习率设置为0.2，初始化权重采用:

```
net.weight.data.normal_(0,1)

net.bias.fill_(0)
```

$y=h(w'x'+b')$

学习1000代，由于relu的采用，训练loss经常无法下降，需要反复尝试.

补充：反向传播函数，设 $h=\mathrm{relu}$，$x=\begin{bmatrix}x_1\\x_2\end{bmatrix}$，$x'=\begin{bmatrix}x_1'\\x_2'\end{bmatrix}$，$b=\begin{bmatrix}b_1\\b_2\end{bmatrix}$，$b'=b_3$

最终得到参数：（设输入$[x_1, x_2]^T$，输出y，中间relu层为$\begin{bmatrix}x_1'\\x_2'\end{bmatrix}$）

$$
\begin{cases}
\begin{bmatrix}x_1'\\x_2'\end{bmatrix}=\mathrm{Relu}\left(\begin{bmatrix}1.5145 & 1.7886\\0.9780 & 1.1392\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}+\begin{bmatrix}-1.6859\\0.0513\end{bmatrix}\right)\\
y=\mathrm{Relu}\left\{\begin{bmatrix}-2.3643 & 1.5052\end{bmatrix}\begin{bmatrix}x_1'\\x_2'\end{bmatrix}+(-0.5493)\right\}
\end{cases}
$$

$$
\begin{cases}
\frac{\partial y}{\partial w}=\frac{\partial y}{\partial x'}\cdot\frac{\partial x'}{\partial w}=h'(w'x'+b')w'^T\cdot h'(wx+b)\cdot x^T.\\
\frac{\partial y}{\partial b}=\frac{\partial y}{\partial x'}\cdot\frac{\partial x'}{\partial b}=h'(w'x'+b')w'^T\cdot h'(wx+b)\\
\frac{\partial y}{\partial w'}=h'(w'x'+b')\cdot x'^T\\
\frac{\partial y}{\partial b'}=h'(w'x'+b')
\end{cases}
$$

计算机在尝试$\begin{bmatrix}1\\0\end{bmatrix}$，$\begin{bmatrix}0\\1\end{bmatrix}$，$\begin{bmatrix}1\\1\end{bmatrix}$，$\begin{bmatrix}0\\0\end{bmatrix}$后输出分别为1,1,0,0，符合异或要求

部分函数代码与运行结果如下图所示：

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import numpy as np

data = np.array([[1, 0, 1], [0, 1, 1],
                 [1, 1, 0], [0, 0, 0]], dtype='float32')
x = data[:, :2]
y = data[:, 2]


def weight_init_normal(m):
    classname = m.__class__.__name__
    if classname.find('Linear') != -1:
        m.weight.data.normal_(0, 1)
        m.bias.data.fill_(0)
```

```python
class XOR(nn.Module):
    def __init__(self):
        super(XOR, self).__init__()
        self.fc1 = nn.Linear(2, 2)
        self.fc2 = nn.Linear(2, 1)

    def forward(self, x):
        h1 = F.relu(self.fc1(x))
        h2 = F.relu(self.fc2(h1))
        return h2
```

```python
criterion = nn.MSELoss()
optimizer = optim.SGD(net.parameters(), lr=0.2, momentum=0.9)


for epoch in range(1000):
    optimizer.zero_grad()
    out = net(x)
    loss = criterion(out, y)
    if epoch%100 == 0:
        print(loss)
    loss.backward()
    optimizer.step()
```

训练1000代，每100代输出一次loss.

```python
test = net(x)
print(test)
for name,parameters in net.named_parameters():
    print(name,':',parameters.size())
for parameters in net.parameters():
    print(parameters)
```

```
tensor(0.5000, grad_fn=<MseLossBackward0>)
tensor(2.0631e-05, grad_fn=<MseLossBackward0>)
tensor(3.0597e-10, grad_fn=<MseLossBackward0>)
tensor(1.4211e-13, grad_fn=<MseLossBackward0>)
tensor(1.4211e-14, grad_fn=<MseLossBackward0>)
tensor(0., grad_fn=<MseLossBackward0>)
tensor(0., grad_fn=<MseLossBackward0>)
tensor(0., grad_fn=<MseLossBackward0>)
tensor(0., grad_fn=<MseLossBackward0>)
tensor(0., grad_fn=<MseLossBackward0>)
tensor([[1.],
        [1.],
        [0.],
        [0.]], grad_fn=<ReluBackward0>)
<bound method Module.parameters of XOR(
  (fc1): Linear(in_features=2, out_features=2, bias=True)
  (fc2): Linear(in_features=2, out_features=1, bias=True)
```

4. 解: H_0: $\begin{bmatrix} 0.2 & 0.2 \\ 0.3 & 0.3 \end{bmatrix} \triangleq \begin{bmatrix} H_1 & H_2 \\ H_3 & H_4 \end{bmatrix}$

输入: $\begin{bmatrix} 1 & 1 & 2 \\ 1 & 1 & 2 \\ 1 & 1 & 2 \end{bmatrix} \triangleq x_{ij}\,(3\times 3)$

输出值: $\begin{bmatrix} 1 & 1.5 \\ 1 & 1.5 \end{bmatrix}$ 理论: $\begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}$

损失 $L = \frac{1}{2}(y-d)^2 = \frac{1}{2}(y_1-d_1)^2 + \frac{1}{2}(y_2-d_2)^2 + \frac{1}{2}(y_3-d_3)^2 + \frac{1}{2}(y_4-d_4)^2$

$$\begin{bmatrix} H_1^* & H_2^* \\ H_3^* & H_4^* \end{bmatrix} = \begin{bmatrix} H_1 & H_2 \\ H_3 & H_4 \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial L}{\partial H_1} & \frac{\partial L}{\partial H_2} \\ \frac{\partial L}{\partial H_3} & \frac{\partial L}{\partial H_4} \end{bmatrix}$$

$$\frac{\partial L}{\partial H_1} = \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial H_1} + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial H_1} + \frac{\partial L}{\partial y_3}\frac{\partial y_3}{\partial H_1} + \frac{\partial L}{\partial y_4}\cdot\frac{\partial y_4}{\partial H_1} = (y_1-d_1)x_{11} + (y_2-d_2)x_{12} + (y_3-d_3)x_{21} + (y_4-d_4)x_{22}$$

$$= 0.5\times 1 + (-1)\times 1 + 0.5\times 1 = 0$$

$$\frac{\partial L}{\partial H_2} = \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial H_2} + \frac{\partial L}{\partial y_2}\cdot\frac{\partial y_2}{\partial H_2} + \frac{\partial L}{\partial y_3}\frac{\partial y_3}{\partial H_2} + \frac{\partial L}{\partial y_4}\frac{\partial y_4}{\partial H_2} = (y_2-d_2)x_{13} + (y_3-d_3)x_{22} + (y_4-d_4)x_{32}$$

$$= 0.5\times 2 + (-1)\times 1 + 0.5\times 2 = 1$$

同理: $\frac{\partial L}{\partial H_3} = 0.5\times 1 + (-1)\times 1 + 0.5\times 1 = 0$ $\frac{\partial L}{\partial H_4} = 0.5\times 2 + (-1)\times 1 + 0.5\times 2 = 1$

$$\therefore \begin{bmatrix} H_1^* & H_2^* \\ H_3^* & H_4^* \end{bmatrix} = \begin{bmatrix} 0.2 & 0.2 \\ 0.3 & 0.3 \end{bmatrix} - 1 \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.2 & -0.8 \\ 0.3 & -0.7 \end{bmatrix}$$