XOR ¶

In [1]:

```
from torch import nn
import torch
device = torch.device("cuda" if torch.cuda.is_avail
print(device)
```

```
cuda
```

In [2]:

```
net = nn.Sequential(
    nn.Linear(2,5),
    nn.ReLU(),
    nn.Linear(5,1),
    nn.Sigmoid()
).to(device)
optimizer = torch.optim.SGD(net.parameters(),lr=0.(
loss_func = nn.MSELoss()
```

In [3]:

```
x = [[0,0],[0,1],[1,0],[1,1]]
y = [[0],[1],[1],[0]]
x_tensor = torch.tensor(x).float().to(device)
y_tensor = torch.tensor(y).float().to(device)
for epoch in range(5000):
    out = net(x_tensor)
    loss = loss_func(out,y_tensor)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    if epoch % 1000 ==0:
        print(f'迭代次数:{epoch}')
        print(f'误差:{loss}')
```

```
迭代次数:0
误差:0.2519087791442871
迭代次数:1000
误差:0.13895517587661743
迭代次数:2000
误差:0.018672369420528412
迭代次数:3000
误差:0.006432113237679005
迭代次数:4000
误差:0.003489910392090678
```

In [4]:

```
out = net(x_tensor).cpu()
print(f'out:{out.data}')
torch.save(net, './XOR.pkl')
```

```
out:tensor([[0.0604],
        [0.9580],
        [0.9556],
        [0.0423]])
```

测试

In [5]:

```
x= [[0,0],[0,1],[1,0],[1,1]]
x_tensor = torch.tensor(x).float().to(device)
out_tensor = net(x_tensor).cpu()
for out in out_tensor:
    if out>0.5:
        print(1)
    else:
        print(0)
```

```
0
1
1
0
```

权重输出

In [6]:

```
for name, param in net.named_parameters():
    print(name)
    print(param.data)
    print("-----------------------------------")
```

```
0.weight
tensor([[-0.5128,  0.6508],
        [ 1.9218, -1.9219],
        [-2.2416,  2.2416],
        [ 0.0645,  0.3347],
        [ 0.4294, -0.4276]], device='cuda:0')
-----------------------------------
0.bias
tensor([ 1.2435e+00,  2.8582e-06, -4.6039e-05,
3.8934e-01, -6.4944e-01],
       device='cuda:0')
-----------------------------------
2.weight
tensor([[-1.3233,  2.6887,  3.0749, -0.4799, -
0.3514]], device='cuda:0')
-----------------------------------
2.bias
tensor([-0.9122], device='cuda:0')
-----------------------------------
```

结论

权重和偏置如上一个单元格输出所示

使用的激活函数为ReLU()和Sigmoid()

In [8]:

```
print(net)
```

```
Sequential(
  (0): Linear(in_features=2, out_features=5, bi
as=True)
  (1): ReLU()
  (2): Linear(in_features=5, out_features=1, bi
as=True)
  (3): Sigmoid()
)
```

网络结构如上所示