AND

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(2,1),
    nn.Sigmoid()
).to(device)
optimizer = torch.optim.SGD(net.parameters(),lr=0.0
loss_func = nn.MSELoss()
```

In [3]:

```
x = [[0,0],[0,1],[1,0],[1,1]]
y = [[0],[0],[0],[1]]
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:0.3514098525047302
迭代次数:1000
误差:0.09353005886077881
迭代次数:2000
误差:0.05892698094248772
迭代次数:3000
误差:0.04221417754885345
迭代次数:4000
误差:0.032368630170822144
```

In [4]:

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

```
out:tensor([[0.0115],
        [0.1728],
        [0.1729],
        [0.7903]], device='cuda:0')
```

测试

In [5]:

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

```
0
0
0
1
```

权重输出

In [6]:

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

```
0.weight
tensor([[2.8926, 2.8921]], device='cuda:0')
-----------------------------------
0.bias
tensor([-4.4578], device='cuda:0')
-----------------------------------
```

结论

权重如上一个单元格所示，为**[2.8926, 2.8921]**，偏置为**[-4.4578]**
使用的激活函数为Sigmoid()

In [8]:

```python
print(net)
```

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

网络结构如上所示