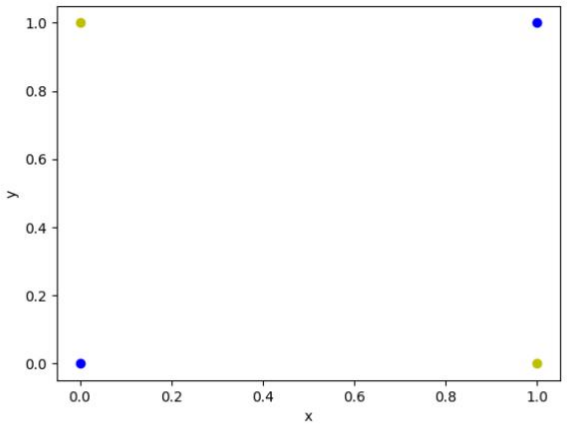
# 多层神经网络案例实践

## 目录

# 异或问题

# 异或问题

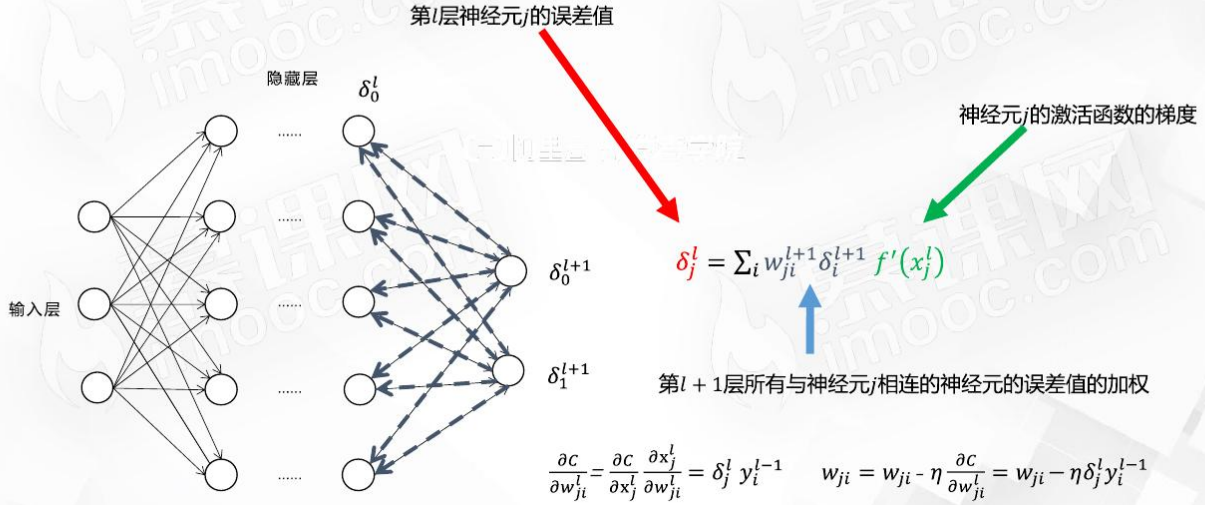
◆ 多层感知器可以解决单层感知器无法解决的异或问题




```
# 输入数据X，各维度表示偏置，x坐标，y坐标
X = np.array([[1,0,0],
              [1,0,1],
              [1,1,0],
              [1,1,1]])
# 标签Y
Y = np.array([0,1,1,0])
# 2层感知器
V = (np.random.random((3,4)) -0.5)*2 # 第一个网络层参数矩阵，初始化输入层权值,取值范围-1到1
W = (np.random.random((4,1))-0.5)* 2 # 第二个网络层参数矩阵，初始化输出层权值,取值范围-1到1
```

# 误差反向传播算法原理

◆ 基于反向传播算法的参数更新




# 多层感知器求解

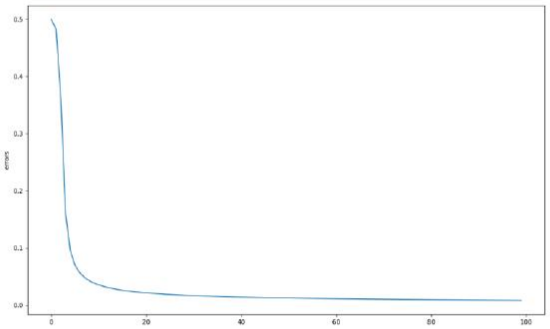
# 基于Python求解多层感知器算法

◆ 代码实现与结果示意

```
# 更新权值（2个权值矩阵，V和W）
def update():
    global X,Y,W,V,lr
    L1 = sigmoid(np.dot(X,V))# 隐藏层输出(4*3)×(3*4)=(4,4)
    L2 = sigmoid(np.dot(L1,W))# 输出层输出(4,4)×(4*1)=(4,1)
    L2_delta = (Y.T - L2)*dsigmoid(L2)# 输出层的Delta
    L1_delta = L2_delta.dot(W.T)*dsigmoid(L1)# 输出层的前一层的Delta

    W_C = lr*L1.T.dot(L2_delta)
    V_C = lr*X.T.dot(L1_delta)
    W = W + W_C
    V = V + V_C

errors = [] # 记录误差
for i in range(100000):
    update()# 更新权值
    if i % 1000 == 0:# 输出误差
        L1 = sigmoid(np.dot(X,V))
        L2 = sigmoid(np.dot(L1,W))
        errors.append(np.mean(np.abs(Y.T - L2)))
        print('Error:',np.mean(np.abs(Y.T - L2)))
```




```
[[0.01102541]
 [0.98857222]
 [0.9937592 ]
 [0.00890389]]
0
1
1
0
```

## 下次预告：时序神经网络