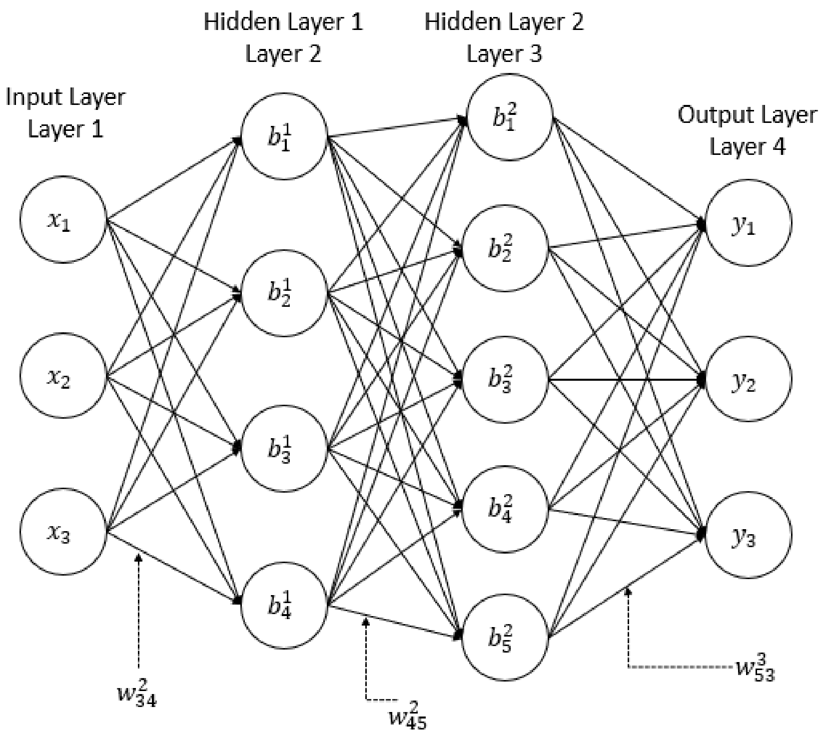
Chapter 1

Sample Chapter

1.1 Neural Network

变量表示： 在看本文之前你可能已经对Neural Network有了一定的了解，或者已经听说过。Neural Network可以抽象的表示为下图：



【图1】

Neural Network的最基本组成单位是Neuron，在图中用一个圆圈表示。该图所表示的是一个有两个Hidden Layer的Neural Network。该Neural Network的层数为5层。接下来我们定义一下变量：

相邻两个Layer的节点之间通过Synapse相连接，每个Synapse上有一个权重Weight。

w_{kj}^l表示第$(l-1)$层的第个k个neuron连接到第l层的第j个neuron的权重；

b_j^l 表示第l层的第j个neuron的bias；

z_j^l 表示第l层的第j个neuron的输入，即：$z_l^j = \sum_k w_{kj}^l a_k^{l-1} + b_j^l$

a_j^l 表示第l层的第j个neuron的输出，即：$a_l^j = \sigma(\sum_k w_{kj}^l a_k^{l-1} + b_j^l)$

其中，σ 表示Activation Function

有了变量的定义之后，我们还需要定义Loss Function。比较常见的Loss Function是Quadratic Loss Function：

$$C = \frac{1}{2n} \sum_x ||y(x) - a^L(x)||^2$$

其中，x表示输入的样本，y表示实际的分类情况，a^L表示预测的分类情况，L表示神经网络的最大层数。

Neural Network计算案例： 假设我们有这样一个数据：

$$X = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}, Y = \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}^\top$$

```
# Python Code:
# Generate Input and Output data:
import numpy as np
X = np.array([[0, 0, 1],[1, 1, 1],[1, 0, 1],[0, 1, 1]]) # 4 by
    3 matrix
Y = np.array([[0, 1, 1, 0]]).T # 4 by 1 matrix
```

如何将上面的Neural Network应用到这个数据上呢：

1. 首先我们先选择Activation Function为Sigmoid Function。即，$\sigma(x) = \frac{1}{1+e^{-x}}$。

该激活函数对x的导函数是:

$\sigma^{'}(x) = \frac{e^{-x}}{(1+e^{-x})^2}$

$= \frac{1+e^{-x}-1}{(1+e^{-x})^2}$

$= \frac{1}{1+e^{-x}} - \frac{1}{(1+e^{-x})^2}$

$= \sigma(x)(1-\sigma(x))$

```
# Python Code:
def sigmoid(x, derive=False):
    if derive == False:
        return 1 / (1 + np.exp(-x))
    else:
        return x * (1 - x)
```

2. 接着，我们定义神经网络的结构（见【图2】），**自变量X的每一行就是一个training example。每一列就是一个Input Neuron。所以这里我们有4个training examples和3个Input Neuron。这里，由于我们的Neural Network一共有3层，所以需要两个随机权重矩阵，分别用W^1和W^2表示。W^1是一个3×4的矩阵。由于Y只有一列，所以W^2是一个4×1的矩阵** 第$(l-1)$层的第k个neuron与第l层的第j个neuron的权重用w^l_{kj}表示。

$$W^1=\begin{bmatrix}-0.01908238 & -0.54517074 & -0.49128704 & -0.88394168\\ -0.13116675 & -0.37640824 & 0.39268698 & -0.24449632\\ -0.64079264 & -0.95064254 & -0.86550074 & 0.35878555\end{bmatrix}, W^2=\begin{bmatrix}-0.09260631\\ 0.07315842\\ 0.79334259\\ 0.98067789\end{bmatrix}$$

```
# Python Code:
random.seed(0)
# W1 is Weight Matrix between Input Layer and Hidden Layer:
W1 = 2 * np.random.random((3,4)) - 1 # 3 by 4 matrix
# W2 is Weight Matrix between Hidden Layer and Output Layer:
W2 = 2 * np.random.random((4,1)) - 1 # 4 by 1 matrix
```

我们看上面的Code Block，W1是（权重矩阵一般随机初始化，比如从0均值的均匀分布或高斯分布中采样得到。如果我们将权重矩阵初始化成$[100.41, 100.1, 100.0]$，你会发现训练后的Neural Network的权重没有变化，说明这种权重矩阵初始化的方法不适合。）

3. 有了这些后，我们就可以建立一个for loop来训练Neural Network了。

```
# Python Code:
for i in range(10000):
    a1 = X
    # The 1st Layer is the Input Layer, X.
    # a1 --> 4 by 3 matrix

    z2 = np.dot(a1, W2)
    # z2 --> 3 by 3 matrix.
    # W2 --> 3 by 4 matrix
```

```
    a2 = sigmoid(z2, derive=False)
    # a2 --> 4 by 4 matrix
    # Here, we finished calculation before Hidden Layer

    z3 = np.dot(a2, W3)
    # W3 --> 4 by 1 matrix
    # z3 --> 4 by 1 matrix

    a3 = sigmoid(z3, derive=False)
    # a3 --> 4 by 1 matrix

    # Here, we finished calculation of forward feed.


    # Then, we'll calculate Back Propogation:
    e3 = Y - a3 # The Error from Output Layer.
    # e3 --> 4 by 1 matrix

    delta3 = e3 * sigmoid(a3, derive=True)
    # delta3 --> 4 by 1 matrix

    e2 = delta3.dot(W2.T) # The Error from Hidden Layer.
    # e2 --> 4 by 1 matrix

    delta2 = e2 * sigmoid(a2, derive=True)
    #

    # Gradient Descent (Update Weight Matrices):
    W3 = W3 + a2.T.dot(delta3)
    W2 = W2 + a1.T.dot(delta2)
print('Output after training: ', a3)
```

```
>>> Output after training:
[[0.00517135]
[0.99506876]
[0.99585861]
[0.00392782]]
```

4. 如果你已经读懂了上面的部分，那么可以跳过这一段。如果没懂的话，我们还是用上面的案例，用数学的方式计算一下这个Neural Network：在计算之前，如果你已经运行了上面的代码，能发现第一次for循环的结果为（你的结果可能与我的不同，但不要紧。我们只关心最后的结果）：

```
>>>Output after training:
[[0.43888284]
 [0.47587257]
 [0.46470967]
 [0.45109288]]
```

首先我们有自变量X和因变量Y:

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}, Y = \begin{bmatrix} y_{11} \\ y_{12} \\ y_{13} \\ y_{14} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

以及最初随机生成的两个权重矩阵W^1和W^2:

$$W^1 = \begin{bmatrix} w^1_{11} & w^1_{12} & w^1_{13} & w^1_{14} \\ w^1_{21} & w^1_{22} & w^1_{23} & w^1_{24} \\ w^1_{31} & w^1_{32} & w^1_{33} & w^1_{34} \end{bmatrix} = \begin{bmatrix} -0.01908238 & -0.54517074 & -0.49128704 & -0.88394168 \\ -0.13116675 & -0.37640824 & 0.39268698 & -0.24449632 \\ -0.64079264 & -0.95064254 & -0.86550074 & 0.35878555 \end{bmatrix}$$

$$W^2 = \begin{bmatrix} w^2_{11} \\ w^2_{21} \\ w^2_{31} \\ w^2_{41} \end{bmatrix} = \begin{bmatrix} -0.09260631 \\ 0.07315842 \\ 0.79334259 \\ 0.98067789 \end{bmatrix}$$

另外要注意的一点是，我们通常将误差矩阵B初始化为全0矩阵，即：

$$B^1 = \begin{bmatrix} b^1_{11} & b^1_{12} & b^1_{13} & b^1_{14} \\ b^1_{21} & b^1_{22} & b^1_{23} & b^1_{24} \\ b^1_{31} & b^1_{32} & b^1_{33} & b^1_{34} \\ b^1_{41} & b^1_{42} & b^1_{43} & b^1_{44} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

那么对于神经网络来说，第一层(i.e. Input Layer)就是X；最后一层(i.e. Output Layer)就是Y；Hidden Layer只有一

4.1. 接下来计算从Input Layer到Hidden Layer的传播过程：我们用z^l_{kj}表示从第$(l-1)$层第k个neuron到第l层第j个neuron的输入值。特别地，$X = z^1$。另外，我们用h^l_{kj}表示从第$(l-1)$层第k个neuron到第l层第j个neuron的输出值。

第$(l-1)$层的输出值h^{l-1}_{kj}经过Activation Function之后，就得到了第l层的输入值$z^l kj$ 对于从Input Layer上的第k个neuron传向Hidden Layer上的第j个neuron的输出值h^2_{kj}有：

$$h^2_{kj} = h^1 \cdot W^1 + B^1 = X \cdot W^1 + B^1 = \begin{bmatrix} x^1_{11} & x^1_{12} & x^1_{13} \\ x^1_{21} & x^1_{22} & x^1_{23} \\ x^1_{31} & x^1_{32} & x^1_{33} \\ x^1_{41} & x^1_{42} & x^1_{43} \end{bmatrix} \cdot \begin{bmatrix} w^1_{11} & w^1_{12} & w^1_{13} & w^1_{14} \\ w^1_{21} & w^1_{22} & w^1_{23} & w^1_{24} \\ w^1_{31} & w^1_{32} & w^1_{33} & w^1_{34} \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} -0.01908238 & -0.54517074 & -0.49128704 & -0.88394168 \\ -0.13116675 & -0.37640824 & 0.39268698 & -0.24449632 \\ -0.64079264 & -0.95064254 & -0.86550074 & 0.35878555 \end{bmatrix}$$

$$= \begin{bmatrix} -0.64079264 & -0.95064254 & -0.86550074 & 0.35878555 \\ -0.79104177 & -1.87222152 & -0.9641008 & -0.76965245 \\ -0.65987502 & -1.49581328 & -1.35678778 & -0.52515613 \\ -0.77195939 & -1.32705078 & -0.47281376 & 0.11428923 \end{bmatrix}$$

4.2. 接着，我们用Sigmoid Function将h_{kj}^2激活后得到z_{kj}^2:

$$z_{kj}^2 = \sigma(h_{kj}^2) = \begin{bmatrix} \sigma(-0.64079264) & \sigma(-0.95064254) & \sigma(-0.86550074) & \sigma(0.35878555) \\ \sigma(-0.79104177) & \sigma(-1.87222152) & \sigma(-0.9641008) & \sigma(-0.76965245) \\ \sigma(-0.65987502) & \sigma(-1.49581328) & \sigma(-1.35678778) & \sigma(-0.52515613) \\ \sigma(-0.77195939) & \sigma(-1.32705078) & \sigma(-0.47281376) & \sigma(0.11428923) \end{bmatrix}$$

$$= \begin{bmatrix} z_{11}^2 & z_{12}^2 & z_{13}^2 & z_{14}^2 \\ z_{21}^2 & z_{22}^2 & z_{23}^2 & z_{24}^2 \\ z_{31}^2 & z_{32}^2 & z_{33}^2 & z_{34}^2 \\ z_{41}^2 & z_{42}^2 & z_{43}^2 & z_{44}^2 \end{bmatrix} = \begin{bmatrix} 0.34506738 & 0.27875562 & 0.29619137 & 0.58874642 \\ 0.31194502 & 0.13328488 & 0.2760579 & 0.31655429 \\ 0.34076769 & 0.18305079 & 0.20476287 & 0.37164735 \\ 0.3160554 & 0.20964762 & 0.38395048 & 0.52854125 \end{bmatrix}$$

4.3. 得到z_{kj}^2后，我么将z_{kj}^2作为Output Layer的输入h_{kj}^2，来计算Output Layer处的输出h_{kj}^3。同样地，先计算:

$$h_{kj}^3 = z_{kj}^2 \cdot W^2 = \sigma(h_{kj}^2) \cdot W^2 = \begin{bmatrix} z_{11}^2 & z_{12}^2 & z_{13}^2 & z_{14}^2 \\ z_{21}^2 & z_{22}^2 & z_{23}^2 & z_{24}^2 \\ z_{31}^2 & z_{32}^2 & z_{33}^2 & z_{34}^2 \\ z_{41}^2 & z_{42}^2 & z_{43}^2 & z_{44}^2 \end{bmatrix} \cdot \begin{bmatrix} w_{11}^2 \\ w_{21}^2 \\ w_{31}^2 \\ w_{41}^2 \end{bmatrix}$$

$$= \begin{bmatrix} 0.34506738 & 0.27875562 & 0.29619137 & 0.58874642 \\ 0.31194502 & 0.13328488 & 0.2760579 & 0.31655429 \\ 0.34076769 & 0.18305079 & 0.20476287 & 0.37164735 \\ 0.3160554 & 0.20964762 & 0.38395048 & 0.52854125 \end{bmatrix} \cdot \begin{bmatrix} -0.09260631 \\ 0.07315842 \\ 0.79334259 \\ 0.98067789 \end{bmatrix}$$

$$= \begin{bmatrix} 0.80078972 \\ 0.51030912 \\ 0.50874791 \\ 0.80900175 \end{bmatrix}$$

4.4. Forward Feed过程的最后一步就是将Activation Function放到最后一层的输出h_{kj}^3上：

$$z_{kj}^3 = \sigma(h_{kj}^3) = \begin{bmatrix} z_{11}^3 \\ z_{21}^3 \\ z_{31}^3 \\ z_{41}^3 \end{bmatrix} = \begin{bmatrix} \sigma(h_{11}^3) \\ \sigma(h_{21}^3) \\ \sigma(h_{31}^3) \\ \sigma(h_{41}^3) \end{bmatrix}$$

$$= \begin{bmatrix} \sigma(0.80078972) \\ \sigma(0.51030912) \\ \sigma(0.50874791) \\ \sigma(0.80900175) \end{bmatrix} = \begin{bmatrix} 0.69014339 \\ 0.62487894 \\ 0.62451291 \\ 0.69189674 \end{bmatrix}$$

到此，我们发现我们计算的输出和Python代码的第一次输出相同。因此完成了第一次的迭代过程。

4.5. 接下来进行Back Propagation：
Back Propogation过程中最重要的就是更新权重矩阵$W^l(old)$得到$W^l(new)$。
而$W^l(new) = W^l(old) + a^{(l-1)_{kj}T} \cdot \delta^l$。
其中，δ^l是第l层到第$(l-1)$层产生的错误。用$\delta^l = E^l \circ \sigma'(a^l_{kj})$求得。其中$E^l$是Back Propogation过程中，第$l$层产生的Error。
本文中，点积运算的两个矩阵之间用·连接；哈达玛积运算的两个矩阵用∘表示。
哈达玛积是两个矩阵的对应位置的元素之间进行乘法运算，可以用`np.multiply(array1, array2)`来在Python中实现。
Error通常是用由Loss Function（或Cost Function）求导得来，记作C。本例是分类问题，所以用$\frac{1}{2}\sum_{i=1}^{n}(Y_i - a_i^L)^2$计算作为Loss Function，其中$n$是样本个数。

$$\begin{aligned} a^L &= \sigma(W^L \cdot a^{L-1} + b^L) \\ a^{L-1} &= \sigma(W^{L-1} \cdot a^{L-2} + b^{L-1}) \\ &= \sigma(W^L \cdot \sigma(W^{L-1} \cdot a^{L-2} + b^{L-1}) + b^L) \\ &\vdots \\ a^1 &= \sigma(W^1 \cdot a^0 + b^1) \\ a^0 &= X \end{aligned} \tag{1.1}$$

在Back Propagation的过程中，如果是从最后一层L到倒数第二层$(L-1)$，那么$E^L = \frac{\partial C}{\partial Y} = Y - a^L$。同理，如果是计算从第$L$层到第$l$层的$Error^l$，则

要用到Chain Rule，有$E^l = \frac{\partial C}{\partial W^l} = \frac{\partial C}{\partial Y} \cdot \frac{\partial Y}{\partial W^l}$。我们总结出如下规律：

$$
\begin{aligned}
E^L &= \frac{\partial C}{\partial a^L} = a^L - Y \\
&\vdots \\
E^l &= \frac{\partial C}{\partial W^l} = \frac{\partial C}{\partial a^l} \cdot \frac{\partial a^l}{\partial W^l} = (a^L - Y) \cdot \frac{\partial a^l}{\partial W^l} \\
&= (a^L - Y) \cdot \sigma^{'}(W^l \cdot a^{l-1} + b^l) \cdot (a^{l-1})^T \\
E^{l-1} &= \frac{\partial C}{\partial W^{l-1}} = \frac{\partial C}{\partial a^{l-1}} \cdot \frac{\partial a^{l-1}}{\partial W^{l-1}} = \frac{\partial C}{\partial a^l} \cdot \frac{\partial a^l}{\partial a^{l-1}} \cdot \frac{\partial a^{l-1}}{\partial W^{l-1}} \\
&= \frac{\partial C}{\partial a^l} \cdot \frac{\partial a^l}{\partial a^{l-1}} \cdot \sigma^{'}(W^{l-1} a^{l-2} + b^{l-1})(a^{l-2})^T \\
&\vdots
\end{aligned} \tag{1.2}
$$

由于我们的神经网络有三层（Input Layer，Hidden Layer，和Output Layer）
所以在Back Propagation的过程中，从Output Layer到Hidden Layer产生的Error，$E^3 = a^3 - Y$（见公式1.2）。从Hidden Layer到Input Layer的Error：

$$
E^2 = \frac{\partial C}{\partial a^3} \cdot \frac{\partial a^3}{\partial a^2} \cdot \frac{\partial a^2}{W^2} = (Y - a^3) \circ \sigma'(a^3) \cdot (W^3)^T \tag{1.3}
$$

1. 从后往前计算每层的Error。首先计算Neural Network最后一层的Error，也就是代码中的 `e3 = Y - a3`，记作E^3。所以：

$$
E^3 = Y - z_{kj}^3 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 0.69014339 \\ 0.62487894 \\ 0.62451291 \\ 0.69189674 \end{bmatrix} = \begin{bmatrix} -0.69014339 \\ 0.37512106 \\ 0.37548709 \\ -0.69189674 \end{bmatrix}
$$

2. 与此同时从后往前计算每层的错误δ^l等于当前层的Error E^l与激活函数的导数$\sigma'_{z_{kj}^l}$的哈达玛积(Hadamard Product)。哈达玛积是两个矩阵的对应位置的元素之间进行乘法运算，可以用`np.multiply(array1, array2)`来在Python中实现。
<u>**本文中，点积运算的两个矩阵之间用·连接；哈达玛积运算的两个矩阵用∘表示。**</u>

那么最后一层的错误δ^3：

$$\delta^3 = E^3 \circ \sigma^{'}(z_{kj}^3) = \begin{bmatrix} -0.69014339 \\ 0.37512106 \\ 0.37548709 \\ -0.69189674 \end{bmatrix} \circ \begin{bmatrix} \sigma^{'}(0.69014339) \\ \sigma^{'}(0.62487894) \\ \sigma^{'}(0.62451291) \\ \sigma^{'}(0.69189674) \end{bmatrix} = \begin{bmatrix} -0.69014339 \\ 0.37512106 \\ 0.37548709 \\ -0.69189674 \end{bmatrix} \circ \begin{bmatrix} 0.21384549 \\ 0.23440525 \\ 0.23449654 \\ 0.21317564 \end{bmatrix}$$

$$= \begin{bmatrix} -0.69014339 \times 0.21384549 \\ 0.37512106 \times 0.23440525 \\ 0.37548709 \times 0.23449654 \\ -0.69189674 \times 0.21317564 \end{bmatrix} = \begin{bmatrix} -0.14758405 \\ 0.08793035 \\ 0.08805042 \\ -0.14749553 \end{bmatrix}$$

3. 同理，用公式（1.2）（1.3），计算Hidden Layer 到Input Layer的Error E^2和错误δ^2：

$$E^2 = \delta^3 \cdot (W^3)^T = \begin{bmatrix} -0.14758405 \\ 0.08793035 \\ 0.08805042 \\ -0.14749553 \end{bmatrix} \cdot \begin{bmatrix} -0.09260631 \\ 0.07315842 \\ 0.79334259 \\ 0.98067789 \end{bmatrix}^T$$

$$= \begin{bmatrix} 0.01366721 & -0.01079702 & -0.11708471 & -0.14473241 \\ -0.00814291 & 0.00643285 & 0.06975889 & 0.08623135 \\ -0.00815402 & 0.00644163 & 0.06985415 & 0.0863491 \\ 0.01365902 & -0.01079054 & -0.11701449 & -0.14464561 \end{bmatrix}$$

$$\delta^2 = E^2 \circ \sigma^{'}(a^2)$$

$$= E^2 \circ \begin{bmatrix} \sigma^{'}(0.34506738) & \sigma^{'}(0.27875562) & \sigma^{'}(0.29619137) & \sigma^{'}(0.58874642) \\ \sigma^{'}(0.31194502) & \sigma^{'}(0.13328488) & \sigma^{'}(0.2760579) & \sigma^{'}(0.31655429) \\ \sigma^{'}(0.34076769) & \sigma^{'}(0.18305079) & \sigma^{'}(0.20476287) & \sigma^{'}(0.37164735) \\ \sigma^{'}(0.3160554) & \sigma^{'}(0.20964762) & \sigma^{'}(0.38395048) & \sigma^{'}(0.52854125) \end{bmatrix}$$

$$= \begin{bmatrix} 0.00308873 & -0.00217075 & -0.02440772 & -0.0350432 \\ -0.00174776 & 0.00074312 & 0.01394131 & 0.01865595 \\ -0.00183176 & 0.0009633 & 0.0113747 & 0.02016473 \\ 0.00295259 & -0.00178794 & -0.02767773 & -0.03604357 \end{bmatrix}$$

4. 有了E^l和δ^l，就能用梯度下降更新权重矩阵W^{l-1}了。因为$l=3$，所以我们要用梯度下降前的权重矩阵$W^2(old)$求更新后的权重矩阵$W^2(new)$：

$$W^2(new) = W^2(old) + (z^2_{kj})^T \cdot \delta^3$$

$$= \begin{bmatrix} -0.09260631 \\ 0.07315842 \\ 0.79334259 \\ 0.98067789 \end{bmatrix} + \begin{bmatrix} 0.34506738 & 0.27875562 & 0.29619137 & 0.58874642 \\ 0.31194502 & 0.13328488 & 0.2760579 & 0.31655429 \\ 0.34076769 & 0.18305079 & 0.20476287 & 0.37164735 \\ 0.3160554 & 0.20964762 & 0.38395048 & 0.52854125 \end{bmatrix}^T \cdot \begin{bmatrix} -0.14758405 \\ 0.08793035 \\ 0.08805042 \\ -0.14749553 \end{bmatrix}$$

$$= \begin{bmatrix} -0.13271534 \\ 0.02893393 \\ 0.73530181 \\ 0.87638927 \end{bmatrix}$$

5. 同样，计算从第二层（Hidden Layer）到第一层（Input Layer）之间的权重矩阵$W^1(new)$：

$$W^1(new) = W^1(old) + (a^1_{kj})^T \cdot \delta^2 \tag{1.4}$$

$$\text{, where } W^1(old) = \begin{bmatrix} -0.01908238 & -0.54517074 & -0.49128704 & -0.88394168 \\ -0.13116675 & -0.37640824 & 0.39268698 & -0.24449632 \\ -0.64079264 & -0.95064254 & -0.86550074 & 0.35878555 \end{bmatrix}$$

$$(a^1_{kj})^T = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}^T$$

$$\delta^2 = \begin{bmatrix} 0.00308873 & -0.00217075 & -0.02440772 & -0.0350432 \\ -0.00174776 & 0.00074312 & 0.01394131 & 0.01865595 \\ -0.00183176 & 0.0009633 & 0.0113747 & 0.02016473 \\ 0.00295259 & -0.00178794 & -0.02767773 & -0.03604357 \end{bmatrix}$$

所以（1.4）的结果为

$$W^1(new) = \begin{bmatrix} -0.0226619 & -0.54346432 & -0.46597103 & -0.845121 \\ -0.12996192 & -0.37745306 & 0.37895056 & -0.26188394 \\ -0.63833084 & -0.95289481 & -0.89227018 & 0.32651946 \end{bmatrix}$$