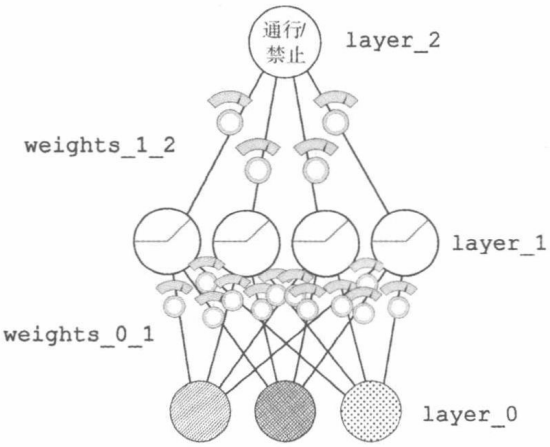
```
import numpy as np

np.random.seed(1)

def relu(x):
    return (x > 0) * x

alpha = 0.2
hidden_size = 4

streetlights = np.array( [[ 1, 0, 1 ],
                          [ 0, 1, 1 ],
                          [ 0, 0, 1 ],
                          [ 1, 1, 1 ] ] )

walk_vs_stop = np.array([[ 1, 1, 0, 0]]).T

weights_0_1 = 2*np.random.random((3,hidden_size)) - 1
weights_1_2 = 2*np.random.random((hidden_size,1)) - 1

layer_0 = streetlights[0]
layer_1 = relu( np.dot(layer_0,weights_0_1))
layer_2 = np.dot(layer_1,weights_1_2)
```

这个函数将所有负数设为0

两组权重经过随机初始化后，将三层网络连接在一起

第一层网络(layer_1)的输出通过relu函数的处理，将其中的负值转换为0，并作为输出传递给下一层神经网络，也就是第二层(layer_2)

请根据右图理解每一段代码的含义。输入数据进入第 0 层网络(layer_0)，通过 dot 函数，信号经过权重从 layer_0 传递给 layer_1(也就是说对于 layer_1 中的四个节点分别求加权和)。从 layer_1 算出的这些加权然后传递给 relu 函数，后者将其中的负值转换为 0。然后，最后一次加权和算出结果传递给最后的节点 layer_2。



6.22 反向传播的代码

你可以据此了解每项权重对最终误差的贡献。

在上一章的末尾，我提出了一个观点：把两层神经网络的代码记住是一件重要的事情，这样当我们讲到更高级的概念时，你可以便捷地回忆起它。这种时候记忆是有用的。

下面展示了新的学习代码，认识与理解前几章的内容是至关重要的。如果你