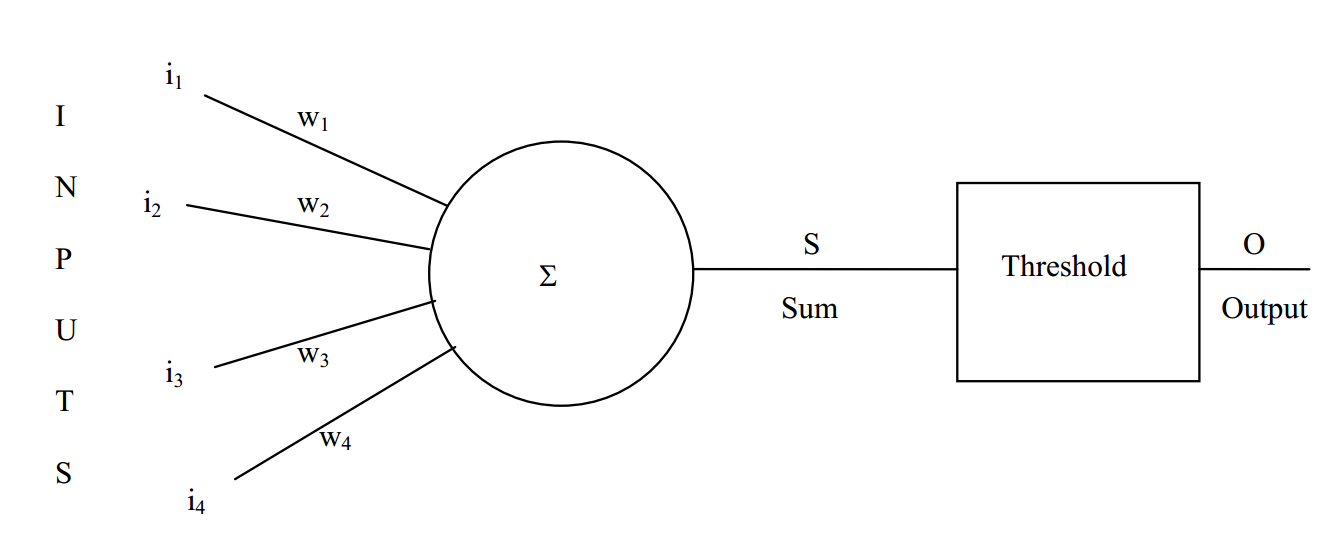
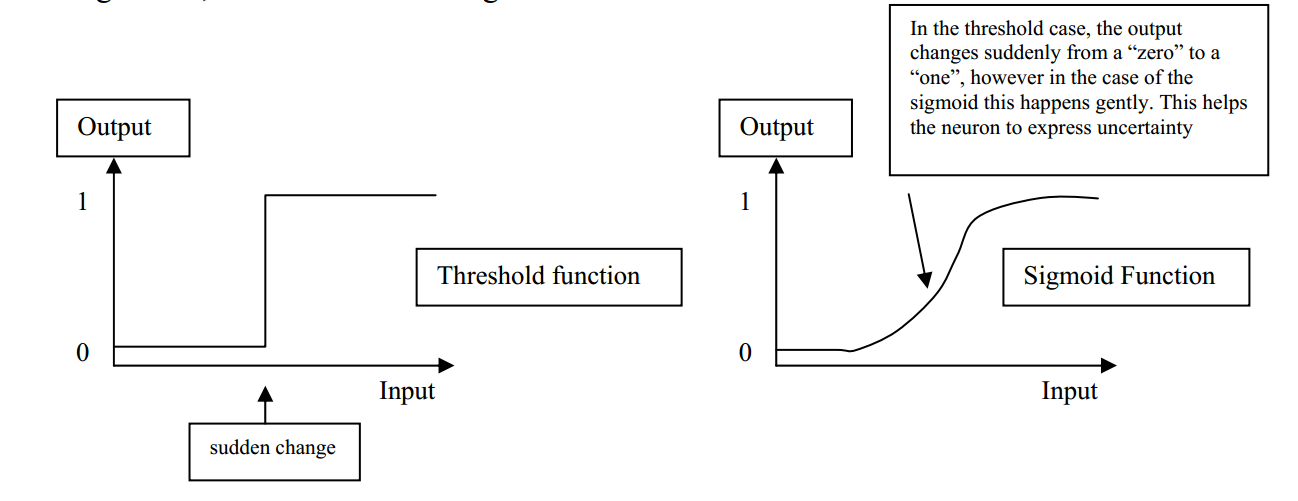
In 1949 Donald Hebb2 postulated one way for the network to learn. If a synapse is  
used more, it gets strengthened – releases more Neurotransmitter. This causes that  
particular path through the network to get stronger, while others, not used, get weaker.  
You might say that each connection has a weight associated with it – larger weights  
produce more stimulation and smaller weights produce less. These were the first steps  
to understanding the learning mechanism of the network. There are still gaps in our  
understanding of learning in the Biological Network, but this need not delay us from  
turning to Artificial Networks and how they are trained

神经网络的灵感来自真正的神经元的观察，上面这段话说明，不同的突触synapse是有不同的比重的，用的多比重就大，用的少比重就小，所以人们根据这种比重的关系，搞出了第一个神经元模型。



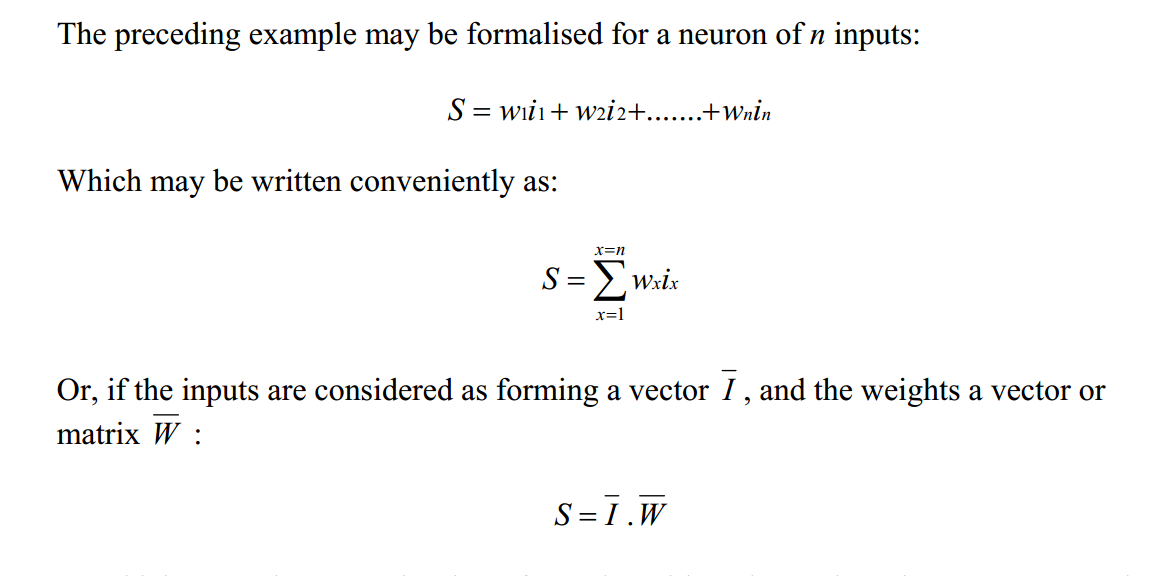
i像是一个个的突触，输入到神经处理单元，但是不同的突触起的作用不一样，所以给了一个w，w就代表每个突触的比重。

这样S求和之后的结果，经过一个阈值处理，输出的0,1这种间断的信号。人们觉着还是连续的好，所以s后面开始跟类似sigmod函数这样的。



阈值是一个突然跳跃，太吓人了。sigmod是一个缓慢上升而且连续的，确实好些？

下面是把计算转换成矩阵运算

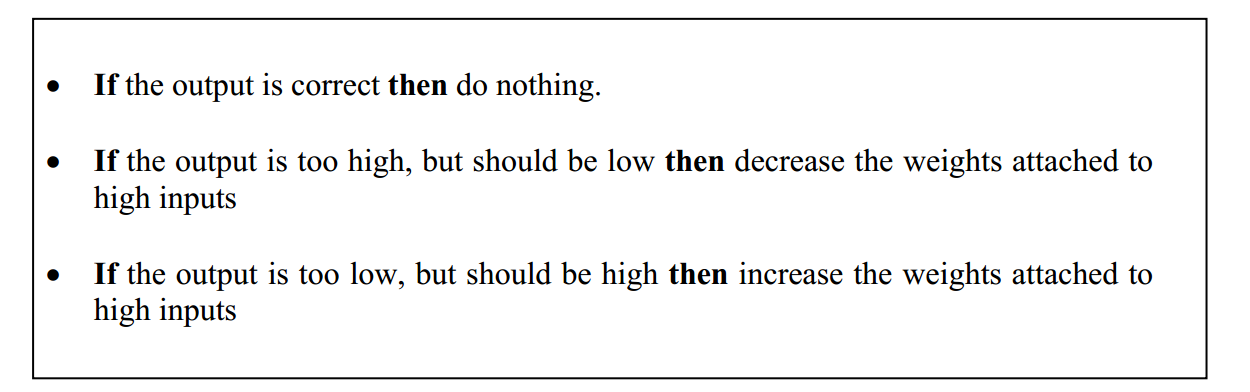


矩阵运算的优点呢？

Even if there were some “interference” to the basic picture (some of the white pixels  
weren’t quite 0 and some of the shaded ones weren’t quite 1) the neuron would still  
recognise it, as long as the total sum was greater than 1.5. So the neuron is “Noise  
Tolerant” or able to “Generalise” which means it will still successfully recognise the  
picture, even if it isn’t perfect - this is one of the most important attributes of Neural  
Networks

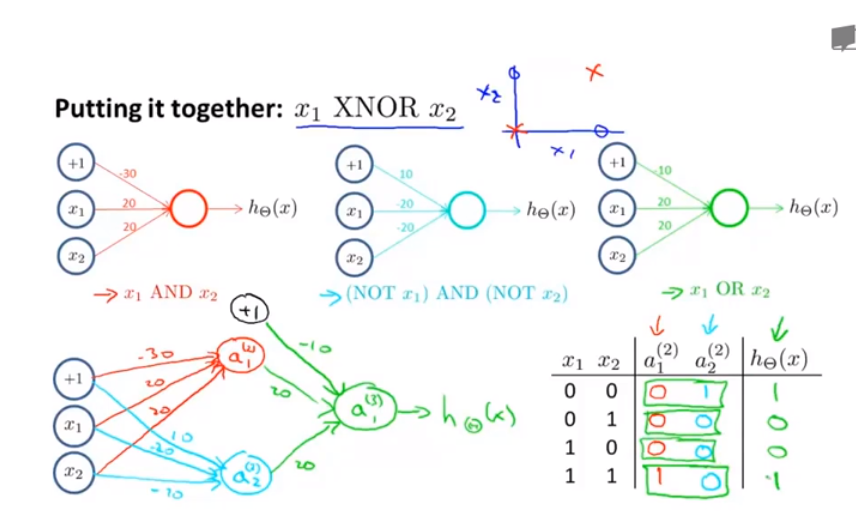
神经网络一个最大的优点是，抗干扰性。

算法学习的过程

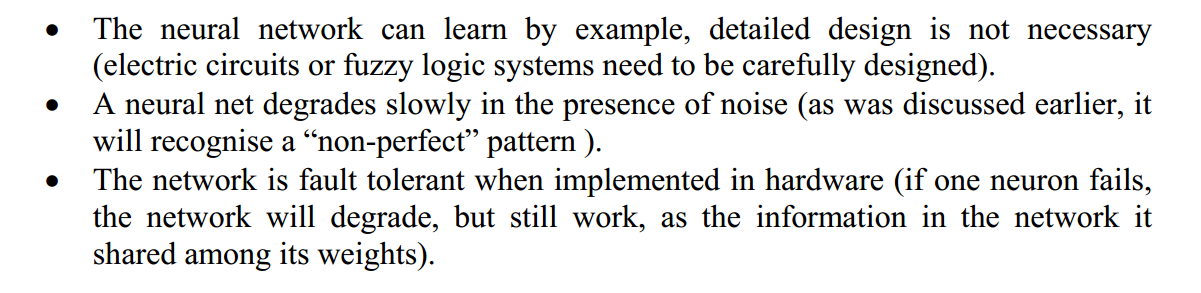


2.4单个神经元的局限性

对于一个神经元无法实现异或操作，在吴恩达的课有做过异或，需要两个神经元才行。所以要突破单个神经元的局限。



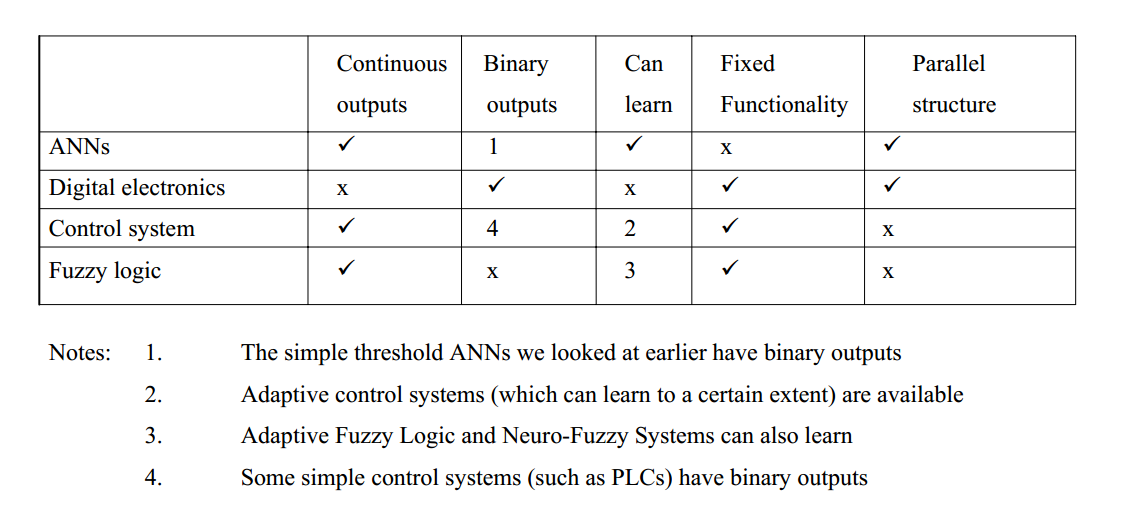
神经网络和数字电路的区别：



数字电路是基于硬件的，很明显有很大的局限性，不够灵活。

神经网络是可以训练和学习的，因此会比较灵活，容错性也比较好。

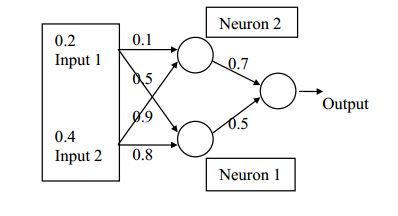
看一下更具体的对比：



在计算的时候尤其要注意，sigmod函数的定义，以及其导数的特殊性，每一层的输出都是一个sigmod函数啊。

### 正向传播

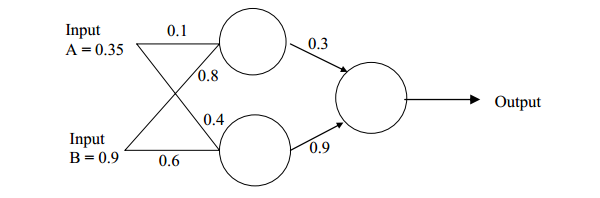
计算下面这个简单神经网络的正向传播：



import math  
  
  
def sigmoid(input):  
 return 1/(1+math.exp(-input))  
  
def calcute(input,weight):  
 size=len(input)  
 out=0  
 for i in range(0,size):  
 out += input[i]\*weight[i]  
 #print(round(out,3))  
 return out  
  
def forward(input, weight):  
 out = calcute(input, weight)  
 result = sigmoid(out)  
 return result  
  
  
  
def network(input,weight1,weight2,weight3):  
 result1 = forward(input, weight1)  
 print(round(result1, 3))  
  
 result2 = forward(input, weight2)  
 print(round(result2, 3))  
  
 result=[result1,result2]  
  
 result3= forward(result,weight3)  
 print(round(result3, 3))  
input=[0.2,0.4]  
weight1=[0.5,0.8]  
weight2=[0.1,0.9]  
weight3=[0.5,0.7]  
network(input,weight1,weight2,weight3)

### 反向传播

计算下面正向加反向传播



import math  
  
  
def sigmoid(input):  
 return 1/(1+math.exp(-input))  
  
def calcute(input,weight):  
 size=len(input)  
 out=0  
 for i in range(0,size):  
 out += input[i]\*weight[i]  
 #print(round(out,3))  
 return out  
  
def forward(input, weight):  
 out = calcute(input, weight)  
 result = sigmoid(out)  
 return result  
  
  
  
def network(input,weight1,weight2,weight3,out):  
 result1 = forward(input, weight1)  
 print(round(result1, 3))  
  
 result2 = forward(input, weight2)  
 print(round(result2, 3))  
 out.append(round(result1, 3))  
 out.append(round(result2, 3))  
 result=[result1,result2]  
 result3= forward(result,weight3)  
 return round(result3, 3)  
input=[0.35,0.9]  
weight1=[0.1,0.8]  
weight2=[0.4,0.6]  
weight3=[0.3,0.9]  
#输出层的结果  
out=[]  
target=0.5  
for i in range(0,10):  
 out = []  
 result = network(input,weight1,weight2,weight3,out)  
 print(result)  
 #输出错误  
 error=(target-result)\*(1-result)\*result  
 error= round(error,4)  
 print(error)  
  
 #更新最近的输出层权重  
 weight3[0]=round(weight3[0]+error\*out[0],6)  
 print(weight3[0])  
  
 weight3[1]=round(weight3[1]+error\*out[1],5)  
 print(weight3[1])  
  
  
 #更新隐藏层的权重  
 eh1=error\*weight3[0]\*(1-out[0])\*(out[0])  
 print(eh1)  
 eh2=error\*weight3[1]\*(1-out[1])\*(out[1])  
 print(eh2)  
  
 weight1[0]=weight1[0] + eh1\*input[0]  
 print(weight1[0])  
  
 weight1[1]=weight1[1] + eh2\*input[0]  
 print(weight1[1])  
  
 weight2[0]=weight2[0] + eh1\*input[1]  
 print(weight2[0])  
  
 weight2[1]=weight2[1] + eh2\*input[1]  
 print(weight2[1])