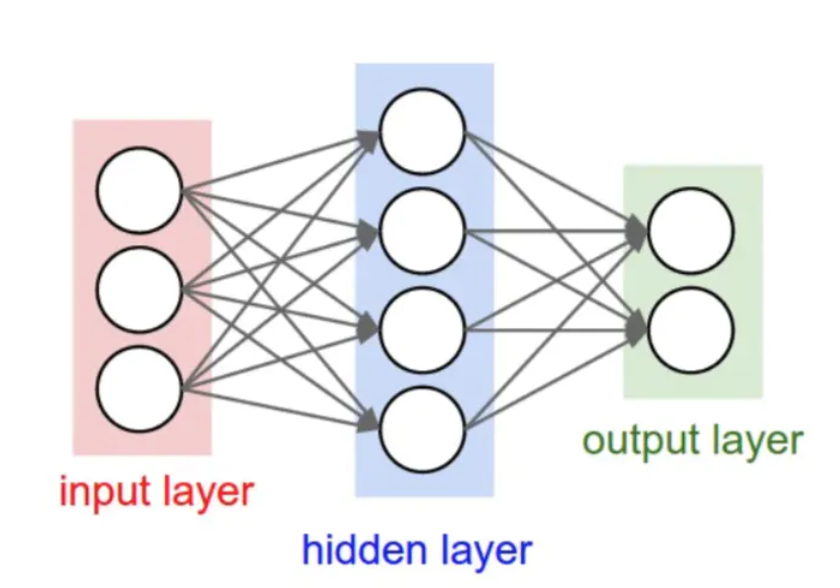
## 神经网络结构

## 分层组织

神经网络是对在非循环图中连接的神经元集合的建模。换句话说，一些神经元的输出可以成为其他神经元的输入。

最常见的层类型是完全连接的层，其中两个相邻层之间的神经元完全成对连接，但单个层内的神经元不共享连接。

以下是使用完全连接的图层的示例神经网络拓扑：



命名惯例：

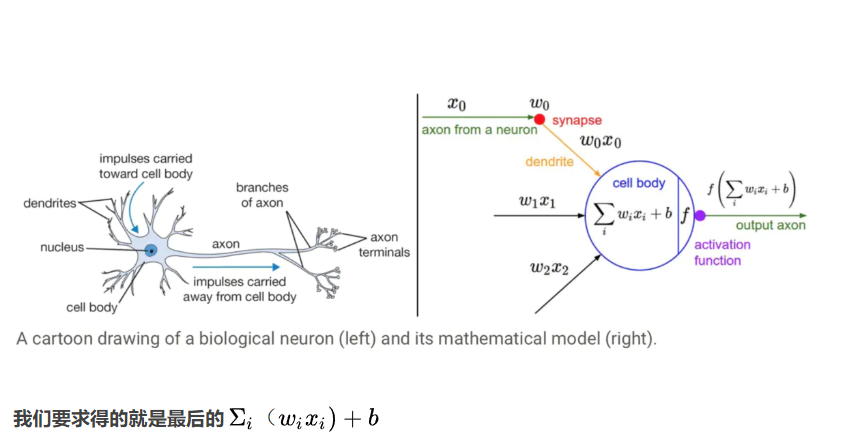
当说是N层神经网络时，我们不计算输入层。

同理，单层神经网络就描述了没有隐藏层的网络（输入直接映射到输出；只有输入层和输出层）。

测量神经网络大小：

通常用来测量神经网络大小的两个指标是神经元的数量，或者更常见的是参数的数量

首先来看一下生物神经元的卡通绘图（左）及其数学模型（右）。



这是一个重复矩阵乘法与激活函数交织的过程。

神经网络被组织成层的主要原因之一是这种结构使得使用矩阵向量运算来评估神经网络非常简单和有效。

输出层与神经网络中的其他层不同，输出层神经元通常不具有激活函数这是因为最后的输出层通常用于表示类别得分.

（注释：类别得分的意思是不同的数代表不同的类别。比如CIFAR-10数据集就是有10个类别，那么数字1-10分别就代表了这个数据集中不同的类别）

**生成数据集**

**Network.py**

首先我们生成一个可以操作的数据不需要自己写代码来生成数据集.

**make\_moons**是函数用来生成数据集，在sklearn.datasets里，具体用法如下

n\_numbers:生成样本数量  
shuffle:是否打乱，类似于将数据集random一下  
noise:默认是false，数据集是否加入高斯噪声  
random\_state:生成随机种子，给定一个int型数据，能够保证每次生成数据相同。

**cmap=plt.cm.Spectral**

plt.cm.Spectral的作用在于在画图时为不同类别的样本分别分配不同的颜色。

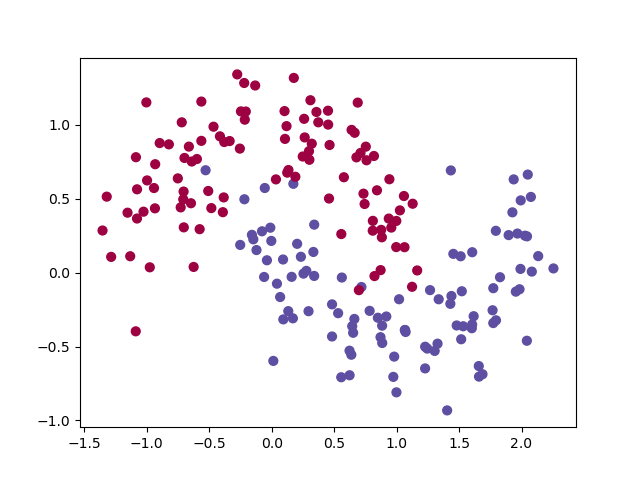
**np.random.seed(0)**

该参数指定了一个随机数生成的起始位置。每个参数对应一个位置。

类似的取随机数的还有这个：[numpy.random.RandomState](https://blog.csdn.net/weixin_41571493/article/details/80550076)

设置seed（）里的数字就相当于设置了一个盛有随机数的“聚宝盆”，一个数字代表一个“聚宝盆”，当我们在seed（）的括号里设置相同的seed，“聚宝盆”就是一样的，那当然每次拿出的随机数就会相同。如果不设置seed，则每次会生成不同的随机数。

代码输出如下：



生成的数据集中有两种类型的数据，分别用红点和蓝点标识了出来。

将蓝点视为男性患者，将红点视为女性患者，并且将x轴和y轴视为医疗方式。

我们的目标是训练一个机器学习分类器，让它在x、y坐标系下预测正确的类别（男性或者女性）。

这些数据不能被线性分割，我们无法画出一条直线将这两种类型的数据分开。

**Logistic回归**

**Network-1.py**

先训练一个Logistic回归分类器，实验一下。

这个分类器的输入是坐标x、y，它的输出是预测的数据类型（0或1）。

**np.meshgrid(\*xi, \*\*kwargs)**  
从坐标向量中返回坐标矩阵,生成网格点坐标矩阵。

Eg:直观的例子

二维坐标系中,X轴可以取三个值1,2,3, Y轴可以取三个值7,8, 请问可以获得多少个点的坐标?  
显而易见是6个:  
(1,7)(2,7)(3,7)  
(1,8)(2,8)(3,8)

**contour：**轮廓，等高线。

为等高线上注明等高线的含义：

plt.contourf 与 plt.contour 区别：

f：filled，也即对等高线间的填充区域进行填充（使用不同的颜色）；

contourf：将不会再绘制等高线（显然不同的颜色分界就表示等高线本身），

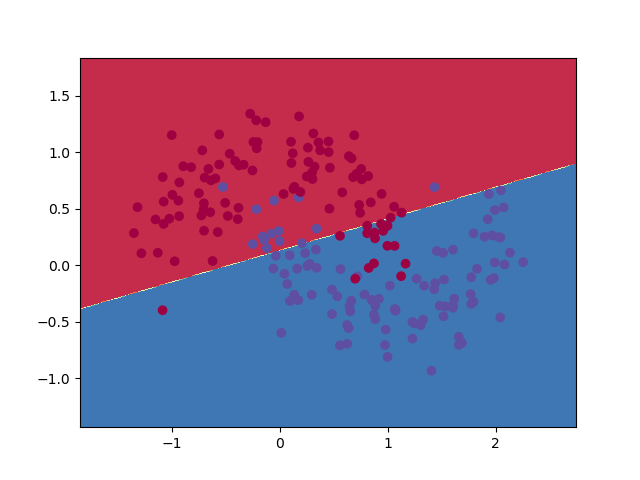
**增加 colorbar:**

默认 colorbar 是竖直放置，通过 orientation 关键字参数，可将其设置为水平放置；

cb = plt.colorbar(orientation='horizontal')

要画出等高线，核心函数是plt.contourf()，但在这个函数中输入的参数是x,y对应的网格数据以及此网格对应的高度值，因此我们调用np.meshgrid(x,y)把x,y值转换成网格数据.

代码输出如下：



上图显示了Logistic回归分类器学习的决策边界，它用直线尽可能地把数据分开了，但是正确率确很低。

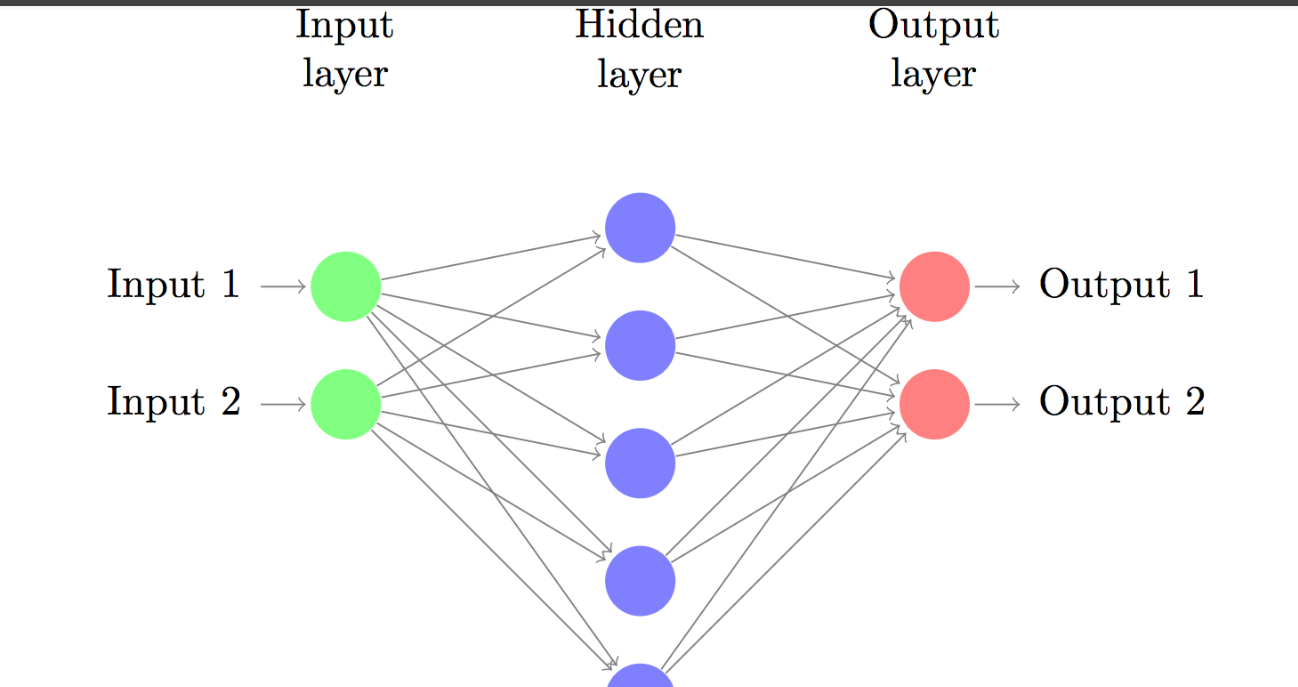
**训练一个神经网络**

现在搭建一个3层的神经网络，其中包含1个输入层，1个隐藏层以及1个输出层。

输出层的节点数取决于我们的数据的维度，也就是2

输出层的节点数取决于我们有多少类数据，这里也是2。

神经网络的输入是x、y坐标，而输出是两个概率，一个是类型为0（女性患者）的概率，另一个是类型为1（男性患者）的概率。这个神经网络看起来就像下面这样：



我们可以自己选择隐藏层的维度（节点数量），隐藏层节点的数量越多，我们能够适应的功能就越复杂。

如果节点数量很多，首先，为了预测结果以及学习神经网络参数，就需要进行大量的计算；其次，更多的参数数量意味着更容易造成对数据的过度拟合。

我们还需要为隐藏层选择一个激活函数，激活函数用来将该层的输入转换成输出。非线性激活函数能够使我们拟合非线性数据。

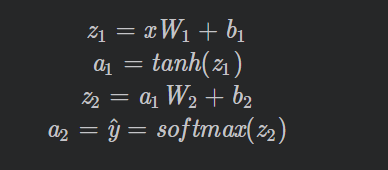
常用的激活函数有[tanh](https://reference.wolfram.com/language/ref/Tanh.html)、[sigmoid](https://en.wikipedia.org/wiki/Sigmoid_function)以及[RELUs](https://en.wikipedia.org/wiki/Rectifier_(neural_networks))，在这里我们使用tanh。那就是它的导数都能用它本身表示。

因为我们想要让神经网络输出概率，所以输出层的激活函数我们使用[softmax](https://en.wikipedia.org/wiki/Softmax_function)，它可以简单地把原始数值转换成概率。

**神经网络如何做出预测**

我们的神经网络利用正向传播来做出预测。

假设x是对神经网络的二维输入，那么我们按照如下步骤计算我们的预测结果



zi是第i层的输入，ai是第i层使用激活函数计算后的输出。

W1,b1,W2,b2是我们这个神经网络的参数，是需要通过训练数据来让神经网络学习的。

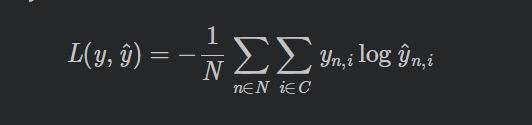
**学习参数**

神经网络学习参数意味着在训练数据上寻找最佳参数(W1,b1,W2,b2)，以此达到使错误最小化的目的。

但是我们应该如何定义“错误”呢？

我们将衡量错误的函数称为损失函数。对于softmax输出来说，一个常用的选择是分类交叉熵损失，又称为负对数似然 。

如果我们有N个训练样本，以及C个输出类别，那么我们的预测结果相对于真值标签相对于真值标签相对于真值标签 y的损失是这样定义的：



总结训练样本，如果预测错了类型，就增加损失。通过寻找使损失最小化的参数，我们可以最大限度地提高训练数据的似然。

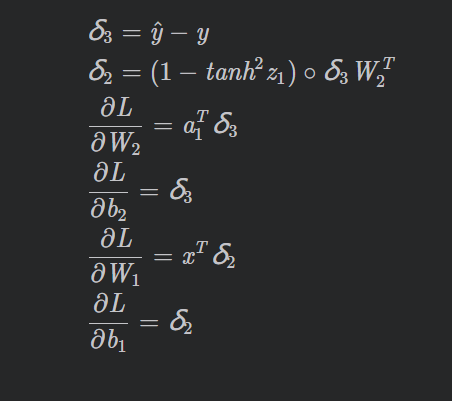
我们可以用[梯度下降法](http://cs231n.github.io/optimization-1/)来寻找损失函数的最小值。用固定的学习率实现一个最普通版本的梯度下降法，也称为批量梯度下降法。

作为输入，梯度下降法需要计算损失函数对于参数的梯度（导数向量）：

捕获3

利用著名的反向传播算法来计算这些梯度，从输出开始用反向传播计算梯度会很高效。

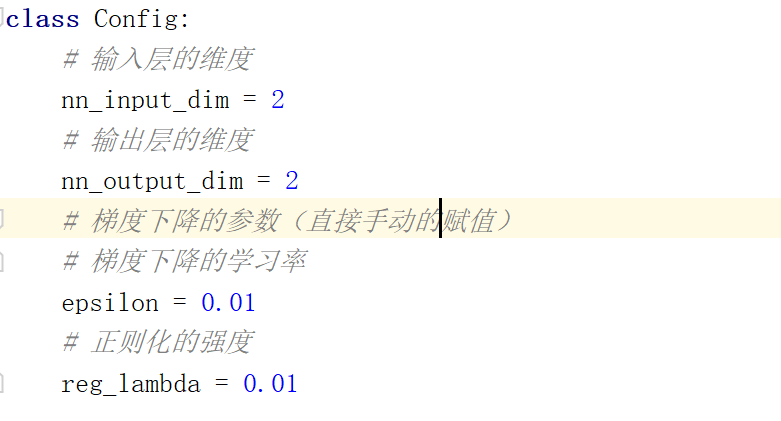
下面的公式为反向传播：



**实现**

**Network-2.py**

现在我们已经准备好实现一个神经网络了，我们为梯度下降定义一些变量和参数



**numpy.tanh(array[, out] )**

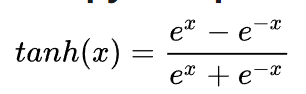
此函数用于计算作为参数传递的数组的所有元素的双曲正切。

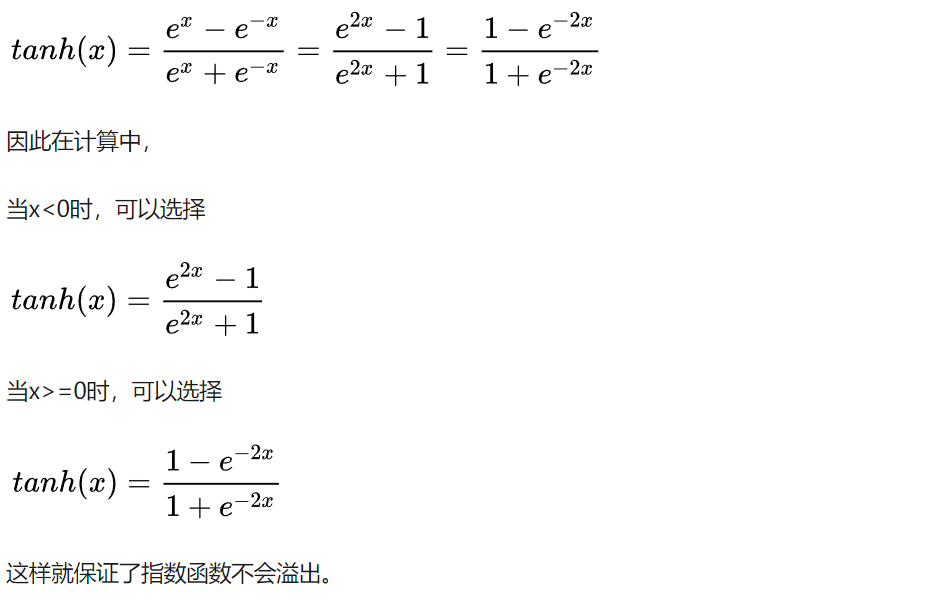
array：要计算切线值(以弧度为单位)的数组元素。out：输出数组的形状。

返回具有三角正切正弦的数组。

# numpy中的np.tanh函数是如何做到数值稳定并且不溢出的？

函数的结果范围为(-1,1)。





import numpy as np

np.dot(a, b, out=None)

该函数的作用是获取两个元素a,b的乘积.

**exp：**[高等数学](https://baike.baidu.com/item/%E9%AB%98%E7%AD%89%E6%95%B0%E5%AD%A6)里以自然常数e为底的[指数函数](https://baike.baidu.com/item/%E6%8C%87%E6%95%B0%E5%87%BD%E6%95%B0)

numpy.exp()：返回e的幂次方，e是一个常数为2.71828

np.exp(1) 为自身  
np.exp(2) 为平方

**numpy.square（）:**

返回输入的元素平方

**numpy.argmax(array, axis)**

用于返回一个numpy数组中最大值的索引值。当一组中同时出现几个最大值时，返回第一个最大值的索引值。

在运算时，相当于剥掉一层中括号，返回一个数组，分为一维和多维。

一维数组剥掉一层中括号之后就成了一个索引值，是一个数，而n维数组剥掉一层中括号后，会返回一个 n-1 维数组，而剥掉哪一层中括号，取决于axis的取值。

n维的数组的 axis 可以取值从 0 到 n-1，其对应的括号层数为从最外层向内递进.

**对一个一维向量**

import numpy as np

a = np.array([3, 1, 2, 4, 6, 1])

b=np.argmax(a)#取出a中元素最大值所对应的索引，此时最大值位6，其对应的位置索引值为4，（索引值默认从0开始）

print(b)#4

**对2维向量（通常意义下的矩阵）**

import numpy as np

a = np.array([[1, 5, 5, 2],[9, 6, 2, 8],3, 7, 9, 1]])

b=np.argmax(a, axis=0)

#对二维矩阵来讲a[0][1]会有两个索引方向，第一个方向为a[0]，默认按列方向搜索最大值

#a的第一列为1，9，3,最大值为9，所在位置为1，

#a的第一列为5，6，7,最大值为7，所在位置为2，

#此此类推，因为a有4列，所以得到的b为1行4列，

print(b)#[1 2 2 1]

c=np.argmax(a, axis=1)#现在按照a[0][1]中的a[1]方向，即行方向搜索最大值，

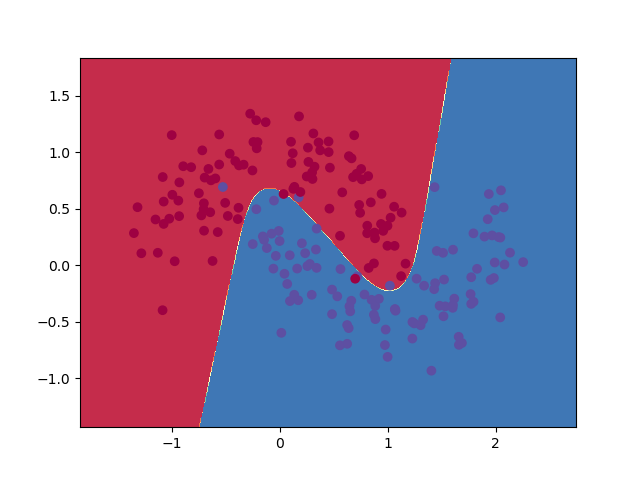
#a的第一行为1，5，5，2,最大值为5（虽然有2个5，但取第一个5所在的位置），索引值为1，

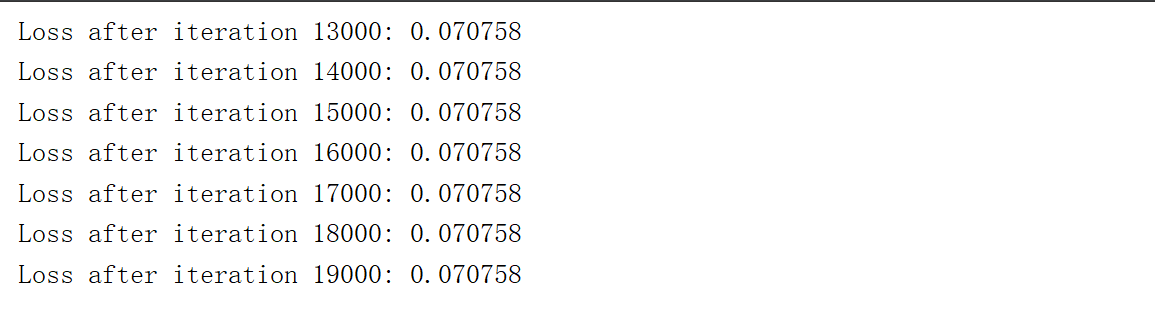
#a的第2行为9，6，2，8,最大值为9，索引值为0，

#因为a有3行，所以得到的c有3个值，即为1行3列

print(c)#[1 0 2]

代码输出如下：

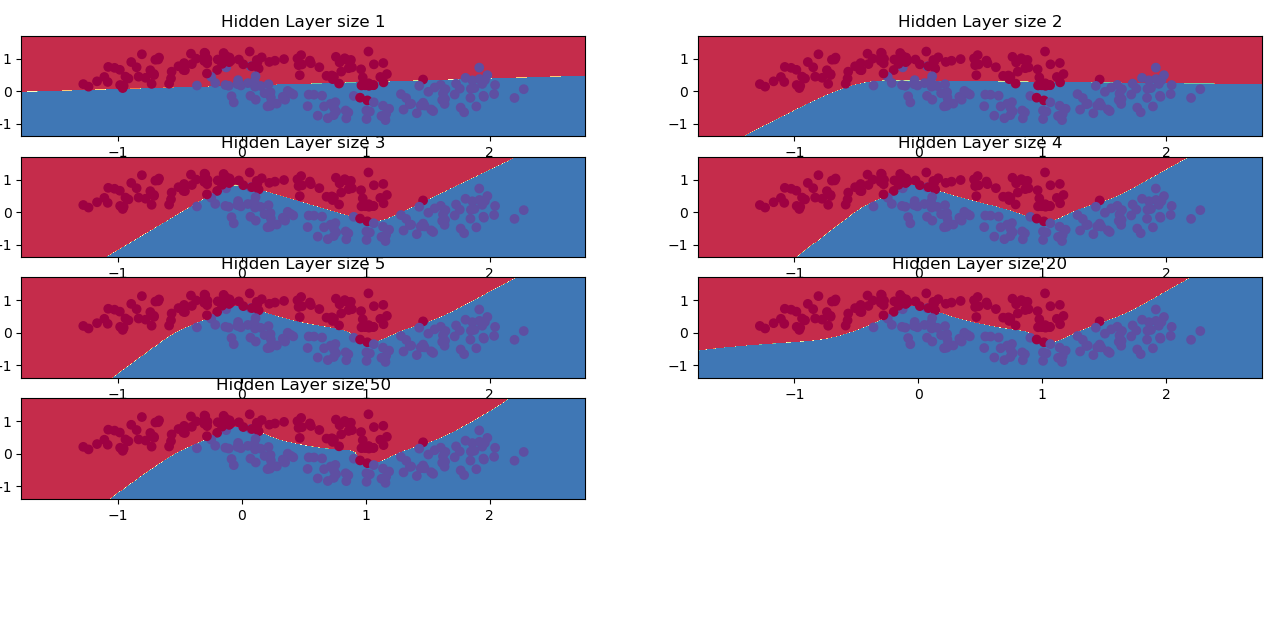




神经网络发现的决策边界能够成功地区分数据类别。

**改变隐藏层的大小**

**Network-4.py**

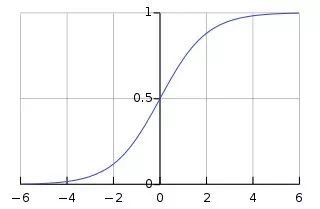
****

我们可以看到低维度的隐藏层可以很好地捕获数据的大致边界，而高维度的隐藏层则更易出现过度拟合。

**Network-3.py**

下面的激活函数，应用Sigmoid函数

绘制一个特征“S”型曲线，作为神经网络的激活函数。



此函数可以将任何值映射到0到1之间的值，它将有助于对输入的加权和归一化。

**下面是神经网络示例的训练过程：**

第一步：从训练数据集中提取输入，根据训练数据集的权重进行调整，并通过一种计算神经网络输出的方法对其进行筛选。

第二步：计算反向传播错误率。

在这种情况下，它是神经元的预测输出与训练数据集的期望输出之间的差异。

第三步：利用误差加权导数公式，根据所得到的误差范围，进行了一些较小的权值调整。

第四步：对这一过程进行迭代。在每次迭代中，整个训练集被同时处理。

接下来，解决网络的权重和偏置的问题。

神经网络在初始化时会随机分配权重与偏置，用高斯分布来取值。

**np.random.normal(loc=0.0, scale=1.0, size=None)**

作用：  
   生成高斯分布的概率密度随机数

loc：float

概率分布的均值（对应着整个分布的中心centre）

scale：float

概率分布的标准差（对应于分布的宽度，scale越大越矮胖，scale越小，越瘦高）

size：输出的shape，默认为None，只输出一个值

超参数： 学习率/迭代次数，需要自己设定，学习率与梯度的乘积为步长。

学习率太高会导致迈过最优点，学习率太低则走的太慢。

**numpy.apply\_along\_axis(func, axis, arr, \*args, \*\*kwargs)**

作用：  
将arr数组的每一个元素经过func函数变换形成的一个新数组

func是我们写的一个函数  
axis表示函数func对arr是作用于行还是列  
arr便是我们要进行操作的数组了

import numpy as npdef f(a):

return (a[0]+a[1])\*2

b=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])

np.apply\_along\_axis(f,0,b)

#结果:array([12, 16, 20, 24])

#(1+5)\*2=12 (2+6)\*2=16依次类推

代码输出如下：



### **BP神经网络结构模型**

 BP算法的基本思想是：学习过程由信号的正向传播和误差的反向传播俩个过程组成，输入从输入层输入，经隐层处理以后，传向输出层。

如果输出层的实际输出和期望输出不符合，就进入误差的反向传播阶段。

误差反向传播是将输出误差以某种形式通过隐层向输入层反向传播，并将误差分摊给各层的所有单元，从而获得各层单元的误差信号，这个误差信号就作为修正个单元权值的依据。

直到输出的误差满足一定条件或者迭代次数达到一定次数。

**BP神经网络的优点：**

非线性映射能力

泛化能力

容错能力：允许输入样本中带有较大误差甚至个别错误。

**BP神经网络的局限性：**

梯度下降法的缺陷：目标函数必须可微。

如果一片区域比较平坦会花费较多时间进行训练。

可能会陷入局部极小值，而没有到达全局最小值。

**BP神经网络的缺陷**

需要的参数过多，而且参数的选择没有有效的方法。

确定一个BP神经网络需要知道：网络的层数、每一层神经元的个数和权值。

权值可以通过学习得到，隐层神经元数量太多会引起过学习，如果隐层神经元个数太少会引起欠学习。此外学习率的选择也是需要考虑。

属于监督学习，对于样本有较大依赖性。

由于权值是随机给定的，所以BP神经网络具有不可重现性。