[**从零开始：用Python搭建神经网络**](http://blog.csdn.net/x_r_su/article/details/53043484)

标签： [python](http://www.csdn.net/tag/python)

2016-11-05 09:20 630人阅读 [评论](http://blog.csdn.net/x_r_su/article/details/53043484#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/x_r_su/article/details/53043484#report)

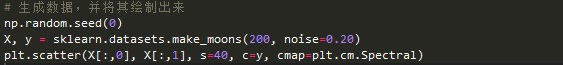
http://static.blog.csdn.net/images/category_icon.jpg 分类：

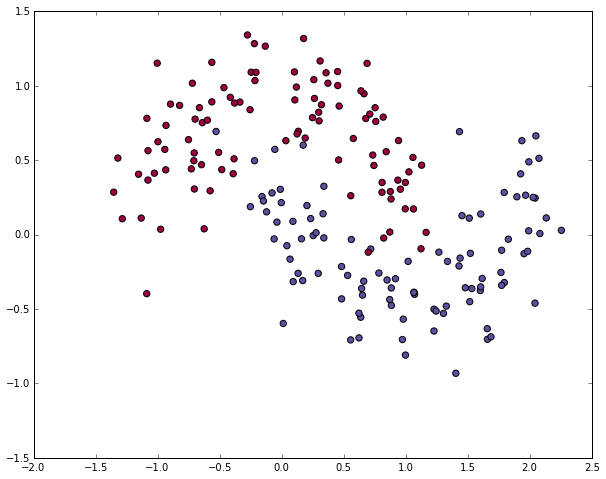
Python（34） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

在这篇博客里，我们将从零开始搭建一个三层的神经网络。我们不会对用到的数学原理一一赘述，但我保证你可以直观地了解到我们在做什么。另外，你也可以通过文章内的链接来获取更详细的信息。  
这儿我就假定你已经熟悉基础的微积分和[**机器学习**](http://lib.csdn.net/base/machinelearning)的一些概念，比如分类和规范化，最好还能懂得一些优化神经网络的技术，比如梯度下降法。当然，即便你对以上提到的这些都不是很了解，我相信你还是能从这篇文章找到乐子的;-)  
那从头来做到底有什么意义？我想，即便是以后使用神经网络库来做开发，比如[PyBrain](http://pybrain.org/" \t "_blank)，一次（甚至多次）从头开始搭建网络的经历仍然可以作为极其宝贵的练习经验。它能够帮助你更好地了解神经网络的运作机制。而这也是设计高效的模型必经之路。  
本文所展示的实例代码追求简单易懂，因此在效率上会打折扣。在我的下一篇推文中，我将展示如何通过[Theano](http://deeplearning.net/software/theano/)来实现一个高效率的神经网络。[链接](http://www.wildml.com/2015/09/speeding-up-your-neural-network-with-theano-and-the-gpu/" \t "_blank)

**生成数据集**

让我们从生成所需要的数据集开始吧。幸运的是，[scikit-learn](http://scikit-learn.org/)提供了一些很有用的数据集生成器，让我们不必为之再造轮子，我们先试试*make\_moons*。

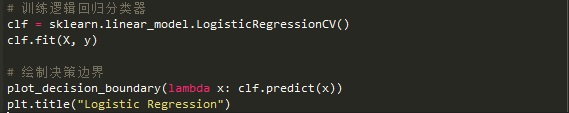


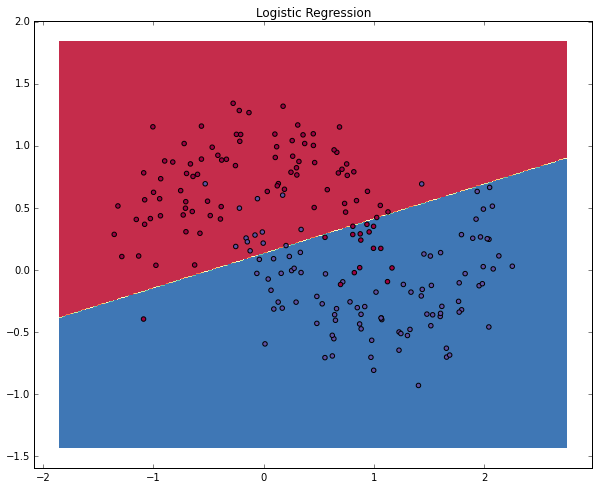


生成了两类数据集，分别用红点和蓝点表示。你可以把蓝点想象成男性病人，红点想象成女性病人，把x轴和y轴想象成药物治疗剂量。  
我们希望通过训练使得机器学习分类器能够在给定的x轴y轴坐标上预测正确的分类情况。我们无法用直线就把数据划分，可见这些数据样本呈非线性。那么，除非你手动构造非线性功能（例如多项式），否则，诸如逻辑回归（*Logistic Regression*）这类线性分类器将无法适用于这个案例。  
事实上，这也正是神经网络的一大主要优势。神经网络的隐藏层会为你去学习特征，所以你不需要为构造特征这件事去操心。

**逻辑回归**

为了证明（学习特征）这点，让我们来训练一个逻辑回归分类器吧。以x轴，y轴的值为输入，它将输出预测的类（0或1）。为了简单起见，这儿我们将直接使用*scikit-learn*里面的逻辑回归分类器。

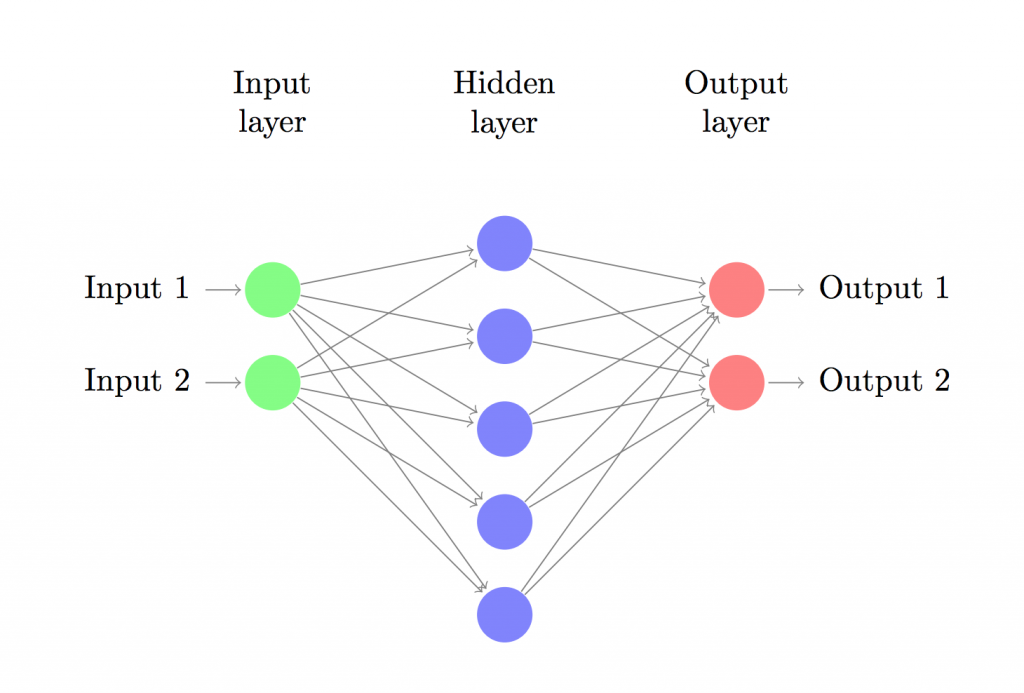




图表向我们展示了逻辑回归分类器经过学习最终得到的决策边界。尽管它尽可能地将数据区分为两类，却不能捕获到数据呈“月亮形状”的特性。

**训练一个神经网络**

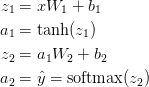
现在，我们搭建由一个输入层，一个隐藏层，一个输出层组成的三层神经网络。输入层中的节点数由数据的维度来决定，也就是2个。相应的，输出层的节点数则是由类的数量来决定，也是2个。（因为我们只有一个预测0和1的输出节点，所以我们只有两类输出，实际中，两个输出节点将更易于在后期进行扩展从而获得更多类别的输出）。以x，y坐标作为输入，输出的则是两种概率，一种是0（代表女），另一种是1（代表男）。结果如下：



我们可以选择隐藏层的维度。放进去的节点越多，实现的功能就可以越复杂。但是维度过高也是会有代价的。首先，更多的预测以及学习网络参数意味着更高的计算强度，更多的参数也会带来过拟合的风险。  
那么该如何判断隐藏层的规模呢？尽管总会有许多通用性很好的引导和推荐，但问题的差异性也不该被忽视。在我看来，选择规模这件事绝不仅仅是门科学，它更像是一门艺术。通过待会儿的演示，我们可以看到隐藏层里的节点数是怎么影响我们的输出的。  
另外，我们还需要为隐藏层选择激活函数（*activation function*）。激活函数会将输入转化成输出。非线性的激活函数可以帮助我们处理非线性的假设。通常选用的激活函数有[tanh](https://reference.wolfram.com/language/ref/Tanh.html" \t "_blank), [the sigmoid function](https://en.wikipedia.org/wiki/Sigmoid_function), [ReLUs](https://en.wikipedia.org/wiki/Rectifier_%28neural_networks" \t "_blank)。在这里我们将使用tanh这样一个适用性很好地函数。这些函数有一个优点，就是通过原始的函数值便可以计算出它们的导数。例如tanh的导数就是1-tanh2x。这让我们可以在推算出tanh⁡x一次后就重复利用这个得到导数值。  
鉴于我们希望我们的网络输出的值为概率，所以我们将使用[softmax](https://en.wikipedia.org/wiki/Softmax_function)作为输出层的激活函数，这个函数可以将原始的数值转化为概率。如果你很熟悉逻辑回归函数，你可以把它当做是逻辑回归的一般形式。

**我们的网络是如何做出预测的呢？**

神经网络通过前向传播做出预测。前向传播仅仅是做了一堆矩阵乘法并使用了我们之前定义的激活函数。如果该网络的输入x是二维的，那么我们可以通过以下方法来计算其预测值 预测值.png：



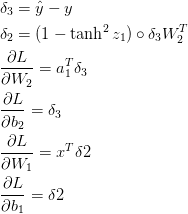
zi是第i层的输入，ai是该层应用激活函数后的输出i，Wi,bi是需要我们通过训练数据来获取的神经网络参数，你可以把它们当作在网络的层与层之间用于转化数据的矩阵。这些矩阵的维度可以通过上面的矩阵乘法看出来。如果我们在隐藏层上使用500个节点，那么就有 latex_012.png, latex_017.png , latex_016.png , latex_019.png 。可以看出，隐藏层的规模与可以用在隐藏层的节点数是呈正相关的。

**研究参数**

研究参数是为了找到能够使我们的训练数据集错误率最小化的参数（latex_007.png ）。但该如何定义错误呢？我们在这里会用损失函数（*loss function*）来检测错误。通常对softmax的输出，我们会选择明确的交叉熵损失（*cross-entropy loss*）（或者叫负对数似然）。如果我们有 latex_004.png个训练示例，latex_020.png 个类别，那么预测latex_013.png 相对于真实的有标签数据的损失则可以通过如下方法来计算获得:

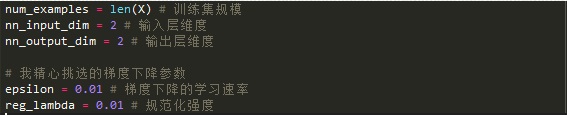
latex_018.png

这个公式看起来很复杂，它的功能就是对我们的训练示例进行求和，并加上预测值错误造成的损失。所以，标签值latex_003.png 与预测值latex_013.png 相差越大，损失就越大。通过寻找降低错误率的参数，我们可以实现最大似然。  
我们可以使用[梯度下降法](http://cs231n.github.io/optimization-1/)来找到这些参数，这里，我会使用一种最普遍的方法：批量梯度下降法。这种方法的学习速率是固定的。它的衍化版例如SGD（随机梯度下降*stochastic gradient descent*）或者最小批量梯度下降（*minibatch gradient descent*）通常在实际使用中会有更好的效果。所以，如果您真的想要学习或者使用，那么请选择这些方法，最好还要能够实现[逐渐衰减学习速率](http://cs231n.github.io/neural-networks-3/#anneal)。  
作为输入，梯度下降法需要各参数对应损失函数的梯度（导数的参数）latex_023.png , latex_008.png , latex_015.png , latex_005.png 。为了计算这些梯度，我们使用著名的后向传播[**算法**](http://lib.csdn.net/base/datastructure)。这种方法在通过输出计算梯度方面效率很高。您可以通过这些链接（[链接1](http://colah.github.io/posts/2015-08-Backprop/)，[链接2](http://cs231n.github.io/optimization-2/)）来了解它，在这里我就不详细介绍它的原理了。  
通过后向传播公式，我们找到了下面这些数据：

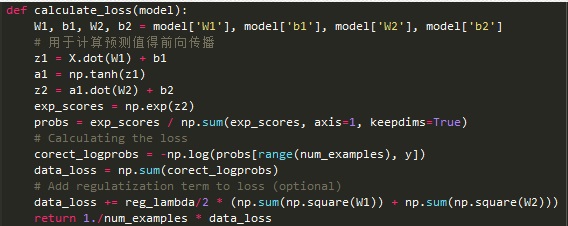


**实现**

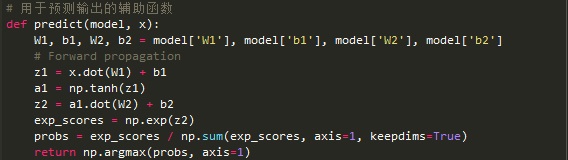
接下来我们就要实现这个三层的神经网络了。首先，我们需要定义一些对用于梯度下降法的变量和参数。



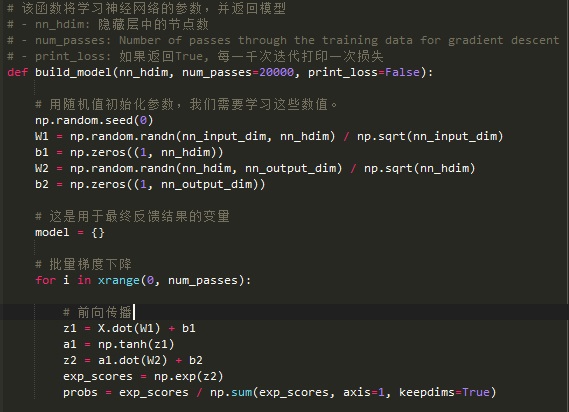
首先，我们先实现之前定义的损失函数，这将用来评估我们的模型。

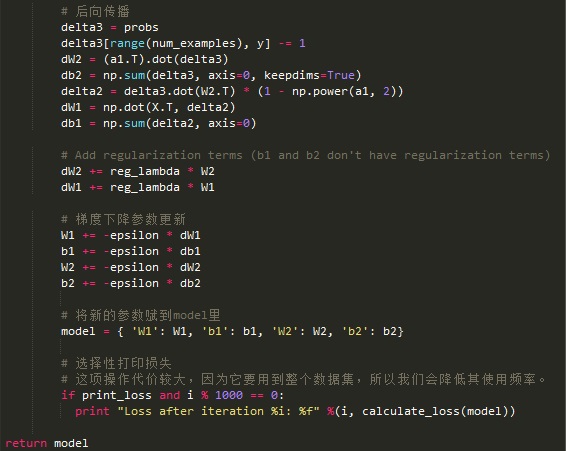


我们还要实现一个用于计算输出的辅助函数。它会通过定义好的前向传播方法来返回拥有最大概率的类别。



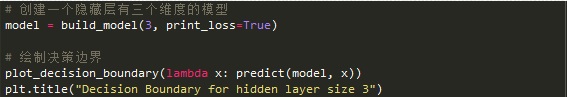
最后是训练神经网络的函数。它会使用我们之前找到的后向传播导数来进行批量梯度下降运算。

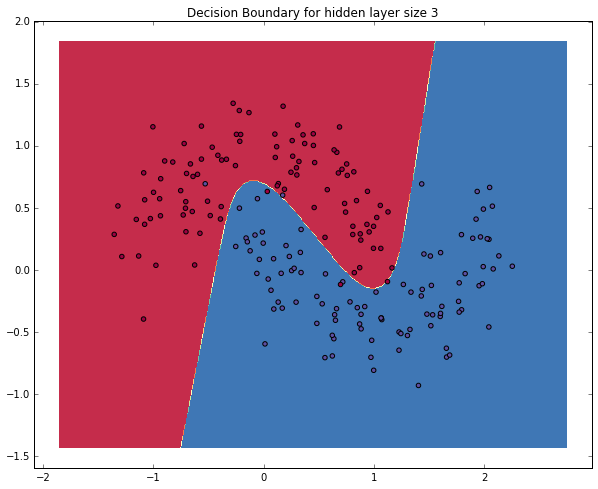




**一个隐藏层规模为3的网络**

让我们看看训练一个隐藏层规模为3的网络会发生什么。

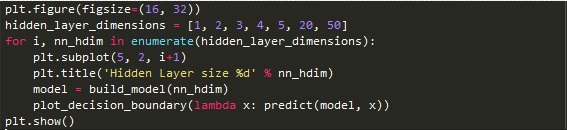


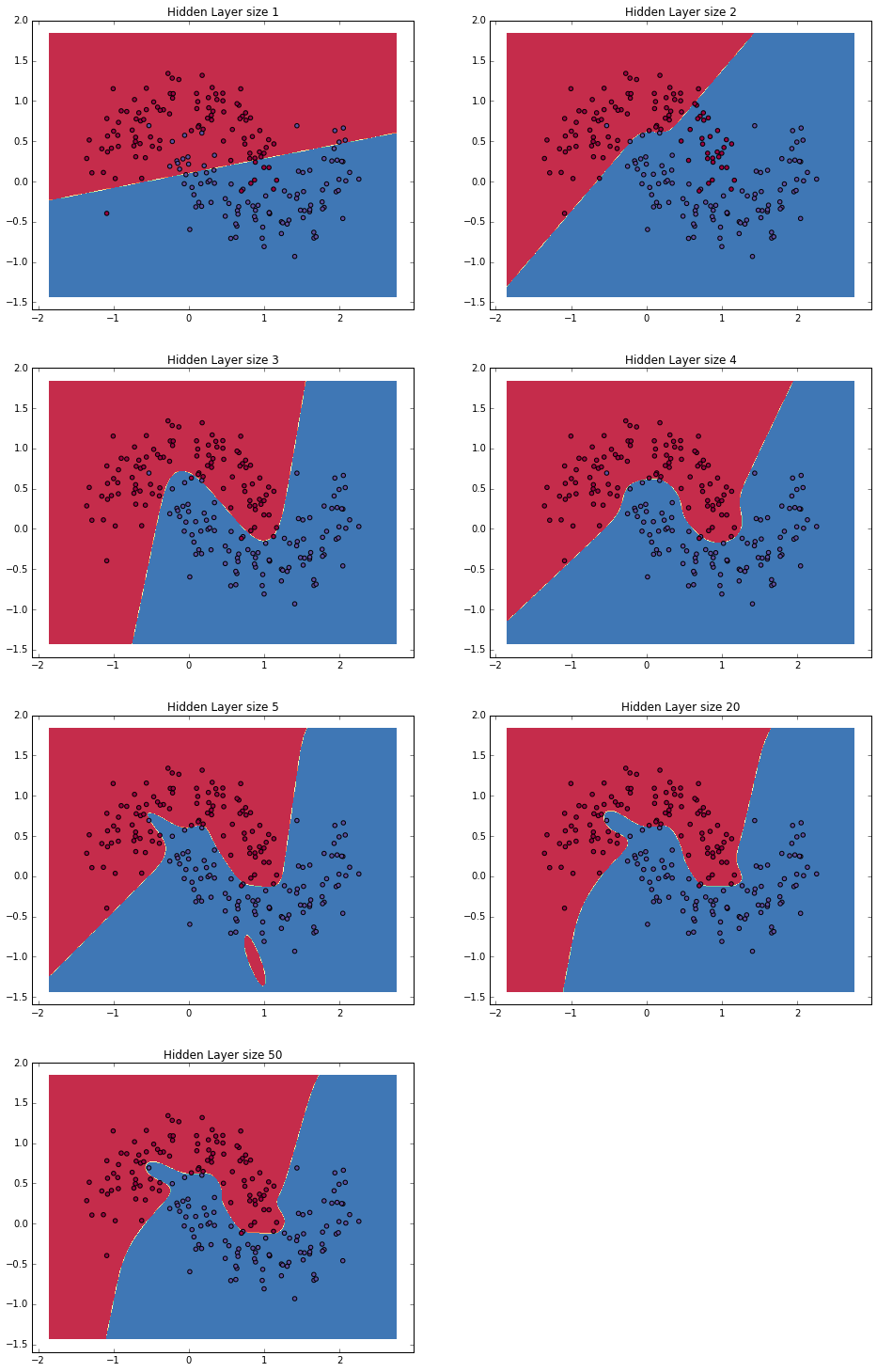


喔~这看起来相当不错。我们的神经网络能够成功地找到区分不同类别的决策边界了。

**变更隐藏层规模**

在刚刚的示例中，我们选择了一个隐藏层规模为3的网络，现在我们来看看不同规模的隐藏层会带来什么样的效果。





我们可以看到，低维度的隐藏层很好地抓住了数据的整体趋势。高维度的隐藏层则显现出过拟合的状态。相对于整体性适应，它们更倾向于精确记录各个点。如果我们要在一个分散的数据集上进行[**测试**](http://lib.csdn.net/base/softwaretest)（你也应该这么做），那么隐藏层规模较小的模型会因为更好的通用性从而获得更好的表现。虽然我们可以通过强化规范化来抵消过拟合，但选择正确的隐藏层规模相对来说会更“经济实惠”一点。

**小练习**

这儿有些可以帮助你更进一步理解代码的干货：  
1.    用[最小批量梯度下降法](http://cs231n.github.io/optimization-1/#gd)替换梯度下降法来训练网络，这会帮助你的模型在实战中获得更好的表现。  
2.    我们使用的是固定学习速率来进行梯度下降。试着去实现[学习速率逐步衰减的方法](http://cs231n.github.io/neural-networks-3/#anneal)。  
3.    本文中，我们用tanh作为激活函数。试试其他的激活函数（有些我们之前已经提到过）。记住，改变激活函数也就意味着改变了后向传播导数。  
4.    将网络从两个类别拓展到三个类别。您将需要为此准备一个合适的数据集。  
5.    将网络拓展到四层，对不同层的规模进行测试。增加新的隐藏层意味着你的前向传播和后向传播代码都需要调整。  
[所有的代码都可以在Github的iPython notebook上找到](https://github.com/dennybritz/nn-from-scratch)。欢迎您提出问题或者反馈:-)

**以上内容来源自**[**Python**](http://lib.csdn.net/base/python)**部落：http://python.freelycode.com/contribution/detail/300**

**英文原文：http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/**