# 11行python实现神经网络（第一部分） 11行python实现神经网络（第一部分） 原文链接： 一种描述了反向传播内部工作的简单神经网络实现。 iamtrask 在2015年7月12日发布。 总结：我从我可以写来玩的代码玩意可以学到很多。这个教程通过一个非常简单的例子讲解了反向传播，仅通过简短的python代码实现。 编辑：一些朋友问我要跟进的文章，于是我打算写一个。当它在@iamtrask完成时我会发推文。 如果你有兴趣阅读它，请随时关注并感谢所有反馈！ 只是给了我这样的代码： X = np.array([ [0,0,1],[0,1,1],[1,0,1],[1,1,1] ])y = np.array([[0,1,1,0]]).Tsyn0 = 2\*np.random.random((3,4)) - 1syn1 = 2\*np.random.random((4,1)) - 1for j in xrange(60000): l1 = 1/(1+np.exp(-(np.dot(X,syn0)))) l2 = 1/(1+np.exp(-(np.dot(l1,syn1)))) l2\_delta = (y - l2)\*(l2\*(1-l2)) l1\_delta = l2\_delta.dot(syn1.T) \* (l1 \* (1-l1)) syn1 += l1.T.dot(l2\_delta) syn0 += X.T.dot(l1\_delta)

原文链接：[A Neural Network in 11 lines of Python (Part 1)](http://iamtrask.github.io/2015/07/12/basic-python-network/?from=hackcv&hmsr=hackcv.com&utm_medium=hackcv.com&utm_source=hackcv.com)

## 一种描述了反向传播内部工作的简单神经网络实现。

iamtrask 在2015年7月12日发布。

**总结**：我从我可以写来玩的代码玩意可以学到很多。这个教程通过一个非常简单的例子讲解了反向传播，仅通过简短的python代码实现。

**编辑**：一些朋友问我要跟进的文章，于是我打算写一个。当它在@iamtrask完成时我会发推文。 如果你有兴趣阅读它，请随时关注并感谢所有反馈！

## 只是给了我这样的代码：

X = np.array([ [0,0,1],[0,1,1],[1,0,1],[1,1,1] ])  
y = np.array([[0,1,1,0]]).T  
syn0 = 2\*np.random.random((3,4)) - 1  
syn1 = 2\*np.random.random((4,1)) - 1  
for j in xrange(60000):  
 l1 = 1/(1+np.exp(-(np.dot(X,syn0))))  
 l2 = 1/(1+np.exp(-(np.dot(l1,syn1))))  
 l2\_delta = (y - l2)\*(l2\*(1-l2))  
 l1\_delta = l2\_delta.dot(syn1.T) \* (l1 \* (1-l1))  
 syn1 += l1.T.dot(l2\_delta)  
 syn0 += X.T.dot(l1\_delta)

别的语言： ,   
别的语言： ,  
然而，这代码有些过于简练了些…….让我们把它分解成几个简单的部分。  
第一部分：一个简单的玩具神经网络  
一个用反向传播训练的神经网络是试着使用输入取预测输出。  
输入输出0010111110110110  
试着考虑从输出的结果中取预测给出的三列。我们可以通过简单的输入和输出的测量统计去解决这个问题。如果我们这样做了，我们将会看到最左边的输入列与输出完全相关。 反向传播，以其最简单的形式，测量这样的统计数据来制作模型。 让我们直接进入并使用它来做到这一点。

然而，这代码有些过于简练了些…….让我们把它分解成几个简单的部分。

### 第一部分：一个简单的玩具神经网络

一个用反向传播训练的神经网络是试着使用输入取预测输出。

输入输出0010111110110110

试着考虑从输出的结果中取预测给出的三列。我们可以通过简单的输入和输出的**测量统计**去解决这个问题。如果我们这样做了，我们将会看到最左边的输入列与输出完全相关。 反向传播，以其最简单的形式，测量这样的统计数据来制作模型。 让我们直接进入并使用它来做到这一点。

### 两层神经网络： 两层神经网络：

import numpy as np  
  
# sigmoid function  
def nonlin(x,deriv=False):  
 if(deriv==True):  
 return x\*(1-x)  
 return 1/(1+np.exp(-x))  
   
# input dataset  
X = np.array([ [0,0,1],  
 [0,1,1],  
 [1,0,1],  
 [1,1,1] ])  
   
# output dataset   
y = np.array([[0,0,1,1]]).T  
  
# seed random numbers to make calculation  
# deterministic (just a good practice)  
np.random.seed(1)  
  
# initialize weights randomly with mean 0  
syn0 = 2\*np.random.random((3,1)) - 1  
  
for iter in xrange(10000):  
  
 # forward propagation  
 l0 = X  
 l1 = nonlin(np.dot(l0,syn0))  
  
 # how much did we miss?  
 l1\_error = y - l1  
  
 # multiply how much we missed by the   
 # slope of the sigmoid at the values in l1  
 l1\_delta = l1\_error \* nonlin(l1,True)  
  
 # update weights  
 syn0 += np.dot(l0.T,l1\_delta)  
  
print "Output After Training:"  
print l1  
将numpy导入为np#sigmoid functiondef nonlin（x，deriv=False）：if（deriv==True）：返回x\*（1-x）返回1/（1+np.exp（-x））#输入数据集x=np.array（[[0,0,1]，[0,1,1]，[1,0,1]，[1,1,1]]）#输出数据集y=np.array（[[0,0,1,1]]）.T#种子随机数进行计算#确定性（只是一个好的实践）np.random.seed（1）#随机初始化权重，平均值为0syn0=2\*np.random.random（（3,1））-1，用于xrange（10000）：#正向传播l0=X 1 1=nonlin（np.dot（l0，syn0））#我们错过了多少？l1误差=y-l1#乘以我们漏掉了多少，再乘以l1δ=l1误差\*nonlin（l1，True）#update weights syn0+=np.dot（l0.T，l1δ）print“训练后输出：”print l1

Output After Training:  
[[ 0.00966449]  
 [ 0.00786506]  
 [ 0.99358898]  
 [ 0.99211957]]  
Output After Training:[[ 0.00966449] [ 0.00786506] [ 0.99358898] [ 0.99211957]]  
正如你在“Output After Training”中看到的，它生效了！！在我解释这个过程钱，我推荐你先运行一下这个代码对它的运行有一个直观的感受。你应该将它运行在一个（或者别的当脚本运行，但是我强烈推荐这个notebook）。以下这些位置需要在代码中好好看看：  
比较在第一次迭代后的l1和最后一个迭代后的l1。  
观察“nonlin”函数，这是使我们能得到一个输出概率的原因。  
观察在我们迭代过程中l1\_error是如何进行改变的。

|  |  |
| --- | --- |
| 变量 | 定义 |
| X | 输入数据集矩阵，其中每一行都是一个训练例子 |
| y | 输出数据集矩阵，其中每一行都是一个训练例子 |
| l0 | 神经网络的第一层，由输入数据指定 |
| l1 | 神经网络的第二层，也被称为隐藏层 |
| syn0 | 第一次的权重，突触0,将l0和l1连接 |
| \* | 元素乘法，因此两个相等大小的矢量将相应的值乘以1对1以生成相同大小的最终矢量。 |
| - | 元素减法，因此两个相等大小的矢量减去相应的值1到1，以生成相同大小的最终矢量 |
| x.dot(y) | 如果x和y是向量，则这是点积。 如果两者都是矩阵，那么它就是矩阵 - 矩阵乘法。 如果只有一个是矩阵，则它是向量矩阵乘法。 |

正如你在“Output After Training”中看到的，它生效了！！在我解释这个过程钱，我推荐你先运行一下这个代码对它的运行有一个直观的感受。你应该将它运行在一个[ipython notebook](http://ipython.org/notebook.html)（或者别的当脚本运行，但是我强烈推荐这个notebook）。以下这些位置需要在代码中好好看看：

* 比较在第一次迭代后的l1和最后一个迭代后的l1。
* 观察“nonlin”函数，这是使我们能得到一个输出概率的原因。
* 观察在我们迭代过程中l1\_error是如何进行改变的。
* 重点注意第36行，大部分的机密就在这里。  
  重点注意第36行，大部分的机密就在这里。  
  注意第39行。神经网络中的所有准备就是为了这个操作。  
  让我们一行一行的介绍这个代码  
  推荐:在两个屏幕中打开这个博客，这样你就可以在你阅读的同时海恩那个看到代码。当我写的时候我就是这么做的。:)  
  第1行:引入numpy库，这是一个线性代数的库。这是我们唯一的依赖。  
  第4行：这是我们的“非线性”函数。这里可以有好几种类型的方法，这里的非线性函数是用了一个叫“sigmoid”。一个可以将任意值的数转化为一个0到1之间的数。我们可以用它来将数转化为概率。它还有一个其他理想的属性使其适用于训练神经网络。
* 注意第39行。神经网络中的所有准备就是为了这个操作。

让我们一行一行的介绍这个代码

**推荐**:在两个屏幕中打开这个博客，这样你就可以在你阅读的同时海恩那个看到代码。当我写的时候我就是这么做的。:)

**第1行**:引入numpy库，这是一个线性代数的库。这是我们唯一的依赖。

**第4行**：这是我们的“非线性”函数。这里可以有好几种类型的方法，这里的非线性函数是用了一个叫“sigmoid”。一个[sigmoid function](https://en.wikipedia.org/wiki/Sigmoid_function)可以将任意值的数转化为一个0到1之间的数。我们可以用它来将数转化为概率。它还有一个其他理想的属性使其适用于训练神经网络。

第5行：注意这个函数还可以生成sigmoid的导数。（当deriv==True).sigomid函数的一个理想特性是它的输出可用于创建它的衍生物。如果sigmoid的输出是一个变量“out”,那么衍生物就很简单的是out\*(1-out)。这是非常有效率的。  
第5行：注意这个函数还可以生成sigmoid的导数。（当deriv==True).sigomid函数的一个理想特性是它的输出可用于创建它的衍生物。如果sigmoid的输出是一个变量“out”,那么衍生物就很简单的是out\*(1-out)。这是非常有效率的。  
如果你对于衍生物不是那么熟悉，就只需要把它看成是给定点处的sigmoid函数的斜率（正如你再上面看到的，不同的点有不同的斜率）想了解更多关于衍生物的知识，可以看Khan Academy的这个。  
第10行:这是初始化你的输入集为一个numpy矩阵。每一行都是一个单独的“训练集”。每个列对应于我们的输入节点之一。因此，我们有3个输入节点的网络和4个训练实例。

如果你对于衍生物不是那么熟悉，就只需要把它看成是给定点处的sigmoid函数的斜率（正如你再上面看到的，不同的点有不同的斜率）想了解更多关于衍生物的知识，可以看Khan Academy的这个[derivatives tutorial](https://www.khanacademy.org/math/differential-calculus/taking-derivatives/derivative_intro/v/calculus-derivatives-1)。

**第10行**:这是初始化你的输入集为一个numpy矩阵。每一行都是一个单独的“训练集”。每个列对应于我们的输入节点之一。因此，我们有3个输入节点的网络和4个训练实例。

第16行：这是初始化我们的输出集。在这种情况下，我生成水平空间的数据集（具有一行和4列）。t是转置函数。转置后，这个y矩阵有4行一列。就像我们的输入一样，每一行都是一个训练示例，每个列（只有一个）是一个输出节点。因此，我们的网络有3个输入和1个输出。  
第16行：这是初始化我们的输出集。在这种情况下，我生成水平空间的数据集（具有一行和4列）。t是转置函数。转置后，这个y矩阵有4行一列。就像我们的输入一样，每一行都是一个训练示例，每个列（只有一个）是一个输出节点。因此，我们的网络有3个输入和1个输出。  
第20行：将随机数字当作你的输入这是个好习惯。你的数字是随机分布的，但每次训练时，它们都会以同样的方式随机分布。这使你更容易看到你的更改如何影响网络。  
第23行：这是我们这个神经网络的权重矩阵。它被叫做“syn0”是“synapse zero”的简称。因为我们只有两层（输入和输出），所以我们只需要一个权重矩阵去连接它们。它的维度是（3，1）因为我们有3个输入和1个输出。另一种观察方法是L0的大小为3，L1的大小为1。因此，我们希望将L0中的每个节点连接到L1中的每个节点，这需要维度矩阵（3，1）。：）

**第20行**：将随机数字当作你的输入这是个好习惯。你的数字是随机分布的，但每次训练时，它们都会以同样的方式随机分布。这使你更容易看到你的更改如何影响网络。

**第23行**：这是我们这个神经网络的权重矩阵。它被叫做“syn0”是“synapse zero”的简称。因为我们只有两层（输入和输出），所以我们只需要一个权重矩阵去连接它们。它的维度是（3，1）因为我们有3个输入和1个输出。另一种观察方法是L0的大小为3，L1的大小为1。因此，我们希望将L0中的每个节点连接到L1中的每个节点，这需要维度矩阵（3，1）。：）

同时需要注意的是它是随机初始化的，平均值为零。在权重初始化方面有相当多的理论。现在，把它作为一个最佳实践，在权重初始化中有一个零均值是个好主意。  
同时需要注意的是它是随机初始化的，平均值为零。在权重初始化方面有相当多的理论。现在，把它作为一个最佳实践，在权重初始化中有一个零均值是个好主意。  
还有一点需要注意的是“神经网络”的真实就是这个矩阵，我们有l0层和l1层但是它们是基于数据集的瞬时值。我们不保存它们。所有的学习都存储在Sy0矩阵中。  
第25行：这开始了我们的实际工作神经网络训练的代码。这是通过多次迭代循环，以优化我们的网络应用于数据集。  
第28行：因为我们的第一层l0很简单的就是我们的数据。我们在这点上明确地描述了它。记住X包含4个训练示例（行）。在这个实现中，我们将同时处理它们。这就是所谓的“全批量”培训。因此，我们有4个不同的L0行，但是如果你想，你可以把它看作一个单独的训练例子。在这一点上没有什么区别。（如果我们想在不改变任何代码的情况下，我们可以加载1000或10000）。

还有一点需要注意的是“神经网络”的真实就是这个矩阵，我们有l0层和l1层但是它们是基于数据集的瞬时值。我们不保存它们。所有的学习都存储在Sy0矩阵中。

**第25行**：这开始了我们的实际工作神经网络训练的代码。这是通过多次迭代循环，以优化我们的网络应用于数据集。

**第28行**：因为我们的第一层l0很简单的就是我们的数据。我们在这点上明确地描述了它。记住X包含4个训练示例（行）。在这个实现中，我们将同时处理它们。这就是所谓的“全批量”培训。因此，我们有4个不同的L0行，但是如果你想，你可以把它看作一个单独的训练例子。在这一点上没有什么区别。（如果我们想在不改变任何代码的情况下，我们可以加载1000或10000）。

第29行：这是我们的预测步骤。基本上，我们首先让网络“尝试”来预测给定输入的输出。然后，我们将研究它是如何执行的，这样我们就可以对它进行调整，以便在每次迭代中都做得更好。  
第29行：这是我们的预测步骤。基本上，我们首先让网络“尝试”来预测给定输入的输出。然后，我们将研究它是如何执行的，这样我们就可以对它进行调整，以便在每次迭代中都做得更好。  
这一行包含两步。第一矩阵乘以L0的Sy0。第二个通过S形函数传递我们的输出。考虑每个维度：  
(4 x 3) dot (3 x 1) = (4 x 1)  
矩阵乘法是有序的，因此方程中间的维数必须是相同的。由此生成的最终矩阵是第一个矩阵的行数和第二个矩阵的列数。  
因为我们一次性训练4个例子，我们最后会得到4个猜想结果，是一个（4×1）的矩阵。每个输出对应于网络对给定输入的猜测。当我们“加载”一个任意数量的训练例子，它就会变得十分直观。矩阵乘法仍然有效。：）

这一行包含两步。第一矩阵乘以L0的Sy0。第二个通过S形函数传递我们的输出。考虑每个维度：

(4 x 3) dot (3 x 1) = (4 x 1)

矩阵乘法是有序的，因此方程中间的维数必须是相同的。由此生成的最终矩阵是第一个矩阵的行数和第二个矩阵的列数。

因为我们一次性训练4个例子，我们最后会得到4个猜想结果，是一个（4×1）的矩阵。每个输出对应于网络对给定输入的猜测。当我们“加载”一个任意数量的训练例子，它就会变得十分直观。矩阵乘法仍然有效。：）

第32行：所以，正如上文那么对应每一个输入在l1都有一个猜想。我们现在可以通过比较它的猜想（l1）和正确答案（y）之间的差距去了解它的工作情况。l1\_error就是一个表现神经网络工作的好与不好的一个评价值。  
第32行：所以，正如上文那么对应每一个输入在l1都有一个猜想。我们现在可以通过比较它的猜想（l1）和正确答案（y）之间的差距去了解它的工作情况。l1\_error就是一个表现神经网络工作的好与不好的一个评价值。  
第36行：现在我们得到了一个有用的东西，这就是秘密所在。在这一行有很多事情，那么我们把它分成两个部分。  
第一部分：导数  
nonlin(l1,True)  
如果L1表示这三个点，则上面的代码生成下面的行的斜率。请注意，非常高的值，如x=2.0（绿点）和非常低的值，如x=-1.0（紫点），具有相当浅的斜率。你能得到的最高斜率是在x＝0（蓝点）。这起着重要的作用。还注意到所有的导数都在0到1之间。

**第36行**：现在我们得到了一个有用的东西，这就是秘密所在。在这一行有很多事情，那么我们把它分成两个部分。

### 第一部分：导数

nonlin(l1,True)

如果L1表示这三个点，则上面的代码生成下面的行的斜率。请注意，非常高的值，如x=2.0（绿点）和非常低的值，如x=-1.0（紫点），具有相当浅的斜率。你能得到的最高斜率是在x＝0（蓝点）。这起着重要的作用。还注意到所有的导数都在0到1之间。

### 完整的表达：误差加权导数 完整的表达：误差加权导数 l1\_delta = l1\_error \* nonlin(l1,True) 有比“误差加权导数”更“数学精确”的方法，但我认为这抓住了直觉。L1-误差是（4，1）矩阵。NoLin（L1，True）返回一个（4,1）矩阵。我们正在做的是将它们“元素化”。这返回具有乘法值的（411）矩阵l1\_delta。 当我们通过误差乘以“斜率”时，我们降低了高信噪比预测的误差。再看sigoid函数的图像！如果斜率非常浅（接近0），那么网络要么具有非常高的值，要么是非常低的值。这意味着网络是非常自信的。然而，如果网络猜到了接近（x＝0，y＝0.5），那么它就不是很有信心。我们更新这些“一厢情愿”的预测最多，而且我们倾向于将自信的预测乘以接近0的数字，从而将那些预测置之不理。

l1\_delta = l1\_error \* nonlin(l1,True)

有比“误差加权导数”更“数学精确”的方法，但我认为这抓住了直觉。L1-误差是（4，1）矩阵。NoLin（L1，True）返回一个（4,1）矩阵。我们正在做的是将它们“元素化”。这返回具有乘法值的（411）矩阵l1\_delta。

当我们通过误差乘以“斜率”时，我们降低了高信噪比预测的误差。再看sigoid函数的图像！如果斜率非常浅（接近0），那么网络要么具有非常高的值，要么是非常低的值。这意味着网络是非常自信的。然而，如果网络猜到了接近（x＝0，y＝0.5），那么它就不是很有信心。我们更新这些“一厢情愿”的预测最多，而且我们倾向于将自信的预测乘以接近0的数字，从而将那些预测置之不理。

第39行:我们现在准备去更新我们的网络！让我们看一个单独的例子。  
第39行:我们现在准备去更新我们的网络！让我们看一个单独的例子。  
在这个训练例子中，我们都设置更新我们的权重值，让我们更新最左边的权重值（9.5）。  
weight\_update = input\_value \* l1\_delta  
对于最左边的权重来说，这将乘以1×l1\_delta。据推测，这会增长9.5。为什么只有一个小数据呢？这个预测已经很有信心了，而且预测在很大程度上是正确的。小的误差和小的斜率意味着非常小的更新。考虑所有的权重。它会稍微增加三个。  
然而，因为我们使用的是“一批”的数据训练，我们在以上四个训练示例上执行上述步骤。所以，它看起来更像上面的图像。那么，第39行是怎么做的呢？它为每个训练示例计算每个权重的权重更新，对它们求和，并更新权重，所有这些都以简单的行为单位。玩矩阵乘法，你会看到它做到这一点！

在这个训练例子中，我们都设置更新我们的权重值，让我们更新最左边的权重值（9.5）。

**weight\_update = input\_value \* l1\_delta**

对于最左边的权重来说，这将乘以1×l1\_delta。据推测，这会增长9.5。为什么只有一个小数据呢？这个预测已经很有信心了，而且预测在很大程度上是正确的。小的误差和小的斜率意味着非常小的更新。考虑所有的权重。它会稍微增加三个。

然而，因为我们使用的是“一批”的数据训练，我们在以上四个训练示例上执行上述步骤。所以，它看起来更像上面的图像。那么，第39行是怎么做的呢？它为每个训练示例计算每个权重的权重更新，对它们求和，并更新权重，所有这些都以简单的行为单位。玩矩阵乘法，你会看到它做到这一点！

### 快速回顾一下： 快速回顾一下： 所以，现在我们一件看到了神经网络是如何进行更新的，让我们回顾一下我们的训练数据和反映。当我们的输入和输出都是1时，我们增大了它们之间的权重值。当我们的输入是1而输出是0时，我们减小了它们之间的权重。 因此，在下面的四个训练示例中，权重从输入开始直到输出，它将一直递增或保持不变，而另外两个权重将发现自己在训练示例中既增加又减少（抵消进度）。这种现象是导致我们的网络学习的基础上的输入和输出之间的相关性

所以，现在我们一件看到了神经网络是如何进行更新的，让我们回顾一下我们的训练数据和反映。当我们的输入和输出都是1时，我们增大了它们之间的权重值。当我们的输入是1而输出是0时，我们减小了它们之间的权重。

因此，在下面的四个训练示例中，权重从输入开始直到输出，它将一直递增或保持不变，而另外两个权重将发现自己在训练示例中既增加又减少（抵消进度）。这种现象是导致我们的网络学习的基础上的输入和输出之间的相关性

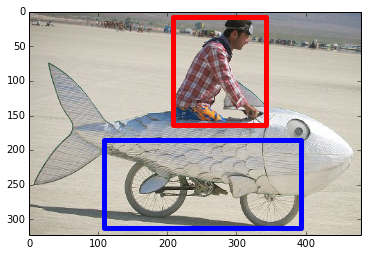
### 第二部分：一个更难一些的问题 第二部分：一个更难一些的问题 InputsOutput0010011110111110 尝试预测给定两个输入列的输出列。一个关键的方法应该是，两列都没有与输出相关。每一列有50%的机会预测1和50%的机会预测0。 那么，模式是什么？它似乎与列三完全无关，它总是1。然而，列1和2给出了更清晰。如果任一列1或2是1（但不是两者都是！）然后输出为1。这是我们的模式。 这被叫做“非线性”模式，因为它在输入和输出之间不是一种直接一对一的关系。在这里它时输入组合的一对一的关系，即第一列和第二列。

InputsOutput0010011110111110

尝试预测给定两个输入列的输出列。一个关键的方法应该是，两列都没有与输出相关。每一列有50%的机会预测1和50%的机会预测0。

那么，模式是什么？它似乎与列三完全无关，它总是1。然而，列1和2给出了更清晰。如果任一列1或2是1（但不是两者都是！）然后输出为1。这是我们的模式。

这被叫做“非线性”模式，因为它在输入和输出之间不是一种直接一对一的关系。在这里它时输入组合的一对一的关系，即第一列和第二列。



img  
img  
信不信由你，图像识别是一个简单的问题，如果一个人有100个相同大小的管道和自行车的图像，那么没有单独的像素位置会直接与自行车或管道的存在相关。从纯统计的角度来看，像素也可能是随机的。然而，像素的某些组合不是随机的，即形成自行车或人的图像的组合。  
我们的策略  
为了首先将像素组合成与输出具有一对一关系的东西，我们需要添加另一个层。我们的第一层将组合输入，然后第二层将使用第一层的输出作为输入将它们映射到输出。在我们跳进一个实现之前，先看看这个表。

信不信由你，图像识别是一个简单的问题，如果一个人有100个相同大小的管道和自行车的图像，那么没有单独的像素位置会直接与自行车或管道的存在相关。从纯统计的角度来看，像素也可能是随机的。然而，像素的某些组合不是随机的，即形成自行车或人的图像的组合。

### 我们的策略

为了首先将像素组合成与输出具有一对一关系的东西，我们需要添加另一个层。我们的第一层将组合输入，然后第二层将使用第一层的输出作为输入将它们映射到输出。在我们跳进一个实现之前，先看看这个表。

nputs (l0)Hidden Weights (l1)Output (l2)0010.10.20.50.200110.20.60.70.111010.30.20.30.911110.20.10.30.80  
nputs (l0)Hidden Weights (l1)Output (l2)0010.10.20.50.200110.20.60.70.111010.30.20.30.911110.20.10.30.80  
如果我们随机初始化权重，我们将得到第1层的隐藏状态值。注意什么？第二列（第二隐藏节点）与输出已经略有关联！它并不完美，但就在那里。信不信由你，这是神经网络训练的一个重要部分。（可以说，这是神经网络训练的唯一方式）下面的训练就是放大这种相关性。它既要更新Sy1，也要将它映射到输出，并且更新SeN0以更好地从输入产生它！  
注意：添加更多层来建模更多关系组合的领域称为“深度学习”，因为建模的层越来越深。

如果我们随机初始化权重，我们将得到第1层的隐藏状态值。注意什么？第二列（第二隐藏节点）与输出已经略有关联！它并不完美，但就在那里。信不信由你，这是神经网络训练的一个重要部分。（可以说，这是神经网络训练的唯一方式）下面的训练就是放大这种相关性。它既要更新Sy1，也要将它映射到输出，并且更新SeN0以更好地从输入产生它！

注意：添加更多层来建模更多关系组合的领域称为“深度学习”，因为建模的层越来越深。

### 三层神经网络 三层神经网络

import numpy as np  
  
def nonlin(x,deriv=False):  
 if(deriv==True):  
 return x\*(1-x)  
  
 return 1/(1+np.exp(-x))  
   
X = np.array([[0,0,1],  
 [0,1,1],  
 [1,0,1],  
 [1,1,1]])  
   
y = np.array([[0],  
 [1],  
 [1],  
 [0]])  
  
np.random.seed(1)  
  
# randomly initialize our weights with mean 0  
syn0 = 2\*np.random.random((3,4)) - 1  
syn1 = 2\*np.random.random((4,1)) - 1  
  
for j in xrange(60000):  
  
 # Feed forward through layers 0, 1, and 2  
 l0 = X  
 l1 = nonlin(np.dot(l0,syn0))  
 l2 = nonlin(np.dot(l1,syn1))  
  
 # how much did we miss the target value?  
 l2\_error = y - l2  
   
 if (j% 10000) == 0:  
 print "Error:" + str(np.mean(np.abs(l2\_error)))  
   
 # in what direction is the target value?  
 # were we really sure? if so, don't change too much.  
 l2\_delta = l2\_error\*nonlin(l2,deriv=True)  
  
 # how much did each l1 value contribute to the l2 error (according to the weights)?  
 l1\_error = l2\_delta.dot(syn1.T)  
   
 # in what direction is the target l1?  
 # were we really sure? if so, don't change too much.  
 l1\_delta = l1\_error \* nonlin(l1,deriv=True)  
  
 syn1 += l1.T.dot(l2\_delta)  
 syn0 += l0.T.dot(l1\_delta)  
Error:0.496410031903  
Error:0.00858452565325  
Error:0.00578945986251  
Error:0.00462917677677  
Error:0.00395876528027  
Error:0.00351012256786  
导入numpy作为npdef nonlin（x，deriv=False）：if（deriv==True）：返回x\*（1-x）返回1/（1+np.exp（-x））x=np.array（[[0,0,1]，[0,1,1]，[1,0,1]，[1,1]，[1,1,1]]）y=np.array（[[0]，[1]，[0]]）np.random.seed（1）#使用平均值0syn0随机初始化权重=2\*np.random.random（（3,4））-1syn1=2\*np.random.random（（4,1））-1对于xrange（60000）中的j：#通过第0、1和2层前馈l0=X 1 1=nonlin（np.dot（l0，syn0））l2=nonlin（np.dot（l1，syn1））#我们错过了多少目标值？l2\_error=y-l2 if（j%10000）==0:print“error:”+str（np.mean（np.abs（l2\_error））#目标值在哪个方向？#我们真的确定吗？如果是，不要改变太多。l2δ=l2误差\*nonlin（l2，deriv=True）#每个l1值对l2误差的贡献是多少（根据权重）？l1\_error=l2\_delta.dot（syn1.T）#目标l1在哪个方向？#我们真的确定吗？如果是，不要改变太多。l1增量=l1增量错误\*nonlin（l1，deriv=True）syn1+=l1.T.dot（l2增量）syn0+=l0.T.dot（l1增量）错误：0.496410031903错误：0.00858452565325错误：0.00578945986251错误：0.00462917677677错误：0.00395876528027错误：0.00351012256786

|  |  |
| --- | --- |
| 变量 | 定义 |
| X | 输入数据矩阵其中每一行代表训练集 |
| y | 输出数据矩阵其中每一行代表训练集 |
| l0 | 神经网络的第一层，由输入数据指定 |
| l1 | 神经网络的第二层，也被称为隐藏层 |
| l2 | 神经网络的第三层，这是我们的假设，当我们训练时，它应该近似正确答案。 |
| syn0 | 第一层的权重值，连接第一和第二层。 |
| syn1 | 第二层的权重值，连接第二和第三层。 |
| l2\_error | 神经网络犯错的概率。 |
| l2\_delta | 这是对网络误差的置信度放大。它几乎与错误完全相同，只是非常自信的错误被消除了 |
| l1\_error | 利用Syth1中的权重对L2\_detlta进行加权，可以计算中/隐层中的误差。 |
| l1\_delta | 这是由置信度缩放的网络的L1误差。同样，它与L1\_error几乎相同，除了自信错误是静默的 |

一切看起来都很熟悉！事实上，这仅仅是先前实现的2个。第一层（L1）的输出是对第二层的输入。这里发生的唯一新事情是在第43行  
一切看起来都很熟悉！事实上，这仅仅是先前实现的2个。第一层（L1）的输出是对第二层的输入。这里发生的唯一新事情是在第43行  
推荐:在两个屏幕中打开这个博客，这样你就可以在你阅读的同时海恩那个看到代码。当我写的时候我就是这么做的。:)  
第43行：使用来自L2的“置信加权误差”来为L1建立误差。为了做到这一点，它简单地将错误从权重从L2发送到L1。这给出了所谓的“贡献加权错误”，因为我们了解了l1中的每个节点值对l2中的错误“贡献”了多少。这个步骤被称为“反向传播”，是算法的命名空间。然后，我们使用与2层实现相同的步骤更新syn0。

**推荐**:在两个屏幕中打开这个博客，这样你就可以在你阅读的同时海恩那个看到代码。当我写的时候我就是这么做的。:)

**第43行**：使用来自L2的“置信加权误差”来为L1建立误差。为了做到这一点，它简单地将错误从权重从L2发送到L1。这给出了所谓的“贡献加权错误”，因为我们了解了l1中的每个节点值对l2中的错误“贡献”了多少。这个步骤被称为“反向传播”，是算法的命名空间。然后，我们使用与2层实现相同的步骤更新syn0。

### 第三部分：结论和进一步的工作 第三部分：结论和进一步的工作 我的推荐： 如果你真的对神经网络有兴趣，那么我有一个建议。你应该试着从内存中重建这个网络。我知道这听起来有一些疯狂，但它是十分有帮助的。如果你想根据新的学术论文创建任意的架构，或者阅读和理解这些不同架构的示例代码，我认为这是一个杀手练习。我认为这是有用的，即使你使用的框架，如,，或。在做这个练习之前，我用神经网络工作了几年，这是我在这个领域所做的最好的时间投资（而且没花多长时间）。

#### 我的推荐：

如果你真的对神经网络有兴趣，那么我有一个建议。你应该试着从内存中重建这个网络。我知道这听起来有一些疯狂，但它是十分有帮助的。如果你想根据新的学术论文创建任意的架构，或者阅读和理解这些不同架构的示例代码，我认为这是一个杀手练习。我认为这是有用的，即使你使用的框架，如[Torch](http://torch.ch/),[Caffe](http://caffe.berkeleyvision.org/)，或[Theano](http://deeplearning.net/software/theano/)。在做这个练习之前，我用神经网络工作了几年，这是我在这个领域所做的最好的时间投资（而且没花多长时间）。

#### 进一步的工作 进一步的工作 这个简单的例子仍然需要相当多的钟声和哨声才能真正接近最先进的建筑。如果你想进一步改进你的网络，这里有几件事你可以看一下。（也许我会在下面的帖子里。 • Alpha • • • Delta Trimming • • • • • • GPU Compatability • Other Awesomeness You Implement 想要在机器学习领域工作？ 学习机器学习最好的方法之一就是找一份专业练习机器学习的工作。我鼓励你在求职过程中核对数字推理的位置。如果你有任何关于数字推理职位或生活的问题，请随时在我的LinkedIn上给我发信息。我很高兴听到你想去哪里，并帮助你评估数字推理是否适合。

这个简单的例子仍然需要相当多的钟声和哨声才能真正接近最先进的建筑。如果你想进一步改进你的网络，这里有几件事你可以看一下。（也许我会在下面的帖子里。

• Alpha • [Bias Units](http://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks) • [Mini-Batches](https://class.coursera.org/ml-003/lecture/106) • Delta Trimming • [Parameterized Layer Sizes](https://www.youtube.com/watch?v=XqRUHEeiyCs) • [Regularization](https://class.coursera.org/ml-003/lecture/63) • [Dropout](http://videolectures.net/nips2012_hinton_networks/) • [Momentum](https://www.youtube.com/watch?v=XqRUHEeiyCs) • [Batch Normalization](http://arxiv.org/abs/1502.03167) • GPU Compatability • Other Awesomeness You Implement

### 想要在机器学习领域工作？

学习机器学习最好的方法之一就是找一份专业练习机器学习的工作。我鼓励你在求职过程中核对数字推理的位置。如果你有任何关于数字推理职位或生活的问题，请随时在我的LinkedIn上给我发信息。我很高兴听到你想去哪里，并帮助你评估数字推理是否适合。

如果上面的位置没有一个感觉很好。继续搜索！机器学习专业知识是当今就业市场最有价值的技能之一，许多公司都在寻找专业人员。也许下面的这些服务会帮助你进行狩猎。  
如果上面的位置没有一个感觉很好。继续搜索！机器学习专业知识是当今就业市场最有价值的技能之一，许多公司都在寻找专业人员。也许下面的这些服务会帮助你进行狩猎。  
Machine Learning Jobs  
Lab126 - Sunnyvale, CA  
Intel - Hillsboro, OR  
FICO - San Jose, CA  
Autonodyne - Boston, MA  
Workday - Boulder, CO  
Integrity Applications… - Kīhei, HI  
MIT Lincoln Laboratory - Massachusetts  
MIT Lincoln Laboratory - Massachusetts  
MIT Lincoln Laboratory - Massachusetts  
Lab126 - Seattle, WA  
Uber - Boulder, CO  
hiretual.com - Mountain View, CA  
Microsoft - Malvern, PA

Machine Learning Jobs

[Software Development Engineer, Alexa…](http://www.indeed.com/viewjob?t=Software+Development+Engineer&c=&l=Sunnyvale%2C+CA&jk=394bd4217b10b942&indpubnum=9172611916208179&atk=&chnl=none) Lab126 - Sunnyvale, CA

[Software engineer in Artificial…](http://www.indeed.com/viewjob?t=Software+Engineer+Artificial+Intelligence&c=Intel&l=Hillsboro%2C+OR&jk=1329b56d82f1830f&indpubnum=9172611916208179&atk=&chnl=none) Intel - Hillsboro, OR

[Analytics Software – Artificial…](http://www.indeed.com/viewjob?t=Analytic+Software+Artificial+Intelligence+Engineer&c=FICO&l=San+Jose%2C+CA&jk=b240d252a351684c&indpubnum=9172611916208179&atk=&chnl=none) FICO - San Jose, CA

[Software Engineer, Artificial…](http://www.indeed.com/viewjob?t=Software+Engineer&c=Autonodyne&l=Boston%2C+MA&jk=3867ac099f2b58c9&indpubnum=9172611916208179&atk=&chnl=none) Autonodyne - Boston, MA

[Machine Learning Engineer (All levels) -…](http://www.indeed.com/viewjob?t=Machine+Learning+Engineer&c=Workday&l=Boulder%2C+CO&jk=08f69256386e417a&indpubnum=9172611916208179&atk=&chnl=none) Workday - Boulder, CO

[Machine Learning Engineer](http://www.indeed.com/viewjob?t=Machine+Learning+Engineer&c=Integrity+Applications&l=K%C4%ABhei%2C+HI&jk=d7aec4c807b5fb3d&indpubnum=9172611916208179&atk=&chnl=none) Integrity Applications… - Kīhei, HI

[Artificial Intelligence (AI) / Machine…](http://www.indeed.com/viewjob?t=Artificial+Intelligence&c=MIT+Lincoln+Laboratory&l=Massachusetts&jk=7813c1afa57b74b7&indpubnum=9172611916208179&atk=&chnl=none) MIT Lincoln Laboratory - Massachusetts

[Machine Learning / Artificial…](http://www.indeed.com/viewjob?t=Machine+Learning&c=MIT+Lincoln+Laboratory&l=Massachusetts&jk=ee006b3c3557ec3e&indpubnum=9172611916208179&atk=&chnl=none) MIT Lincoln Laboratory - Massachusetts

[Artificial Intelligence (AI) / Machine…](http://www.indeed.com/viewjob?t=Artificial+Intelligence&c=MIT+Lincoln+Laboratory&l=Massachusetts&jk=2b756fb21c2121f9&indpubnum=9172611916208179&atk=&chnl=none) MIT Lincoln Laboratory - Massachusetts

[Software Development Engineer - AWS…](http://www.indeed.com/viewjob?t=Software+Development+Engineer&c=&l=Seattle%2C+WA&jk=82c94c327256f70f&indpubnum=9172611916208179&atk=&chnl=none) Lab126 - Seattle, WA

[Deep Learning Engineer](http://www.indeed.com/viewjob?t=Deep+Learning+Engineer&c=Uber&l=Boulder%2C+CO&jk=fb1da92cd255416e&indpubnum=9172611916208179&atk=&chnl=none) Uber - Boulder, CO

[Machine Learning Engineer](http://www.indeed.com/viewjob?t=Machine+Learning+Engineer&c=hiretual.com&l=Mountain+View%2C+CA&jk=b27d334104e7a4f9&indpubnum=9172611916208179&atk=&chnl=none) hiretual.com - Mountain View, CA

[Data & AI Architect](http://www.indeed.com/viewjob?t=Data+Ai+Architect&c=Microsoft&l=Malvern%2C+PA&jk=0d477979f64a27fe&indpubnum=9172611916208179&atk=&chnl=none) Microsoft - Malvern, PA

[Staff Engineer - vRNI - Machine…](http://www.indeed.com/viewjob?t=Staff+Engineer&c=VMware&l=Palo+Alto%2C+CA&jk=105b2e0b2cb0659f&indpubnum=9172611916208179&atk=&chnl=none) VMware - Palo Alto, CA

[Senior AI Engineer - Machine Learning](http://www.indeed.com/viewjob?t=Senior+Ai+Engineer&c=Verizon&l=Irving%2C+TX&jk=ff3fbb942c275c15&indpubnum=9172611916208179&atk=&chnl=none) Verizon - Irving, TX

View More Job Search Results