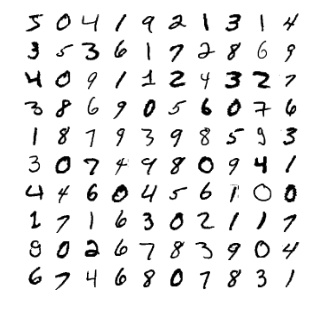
用30行Python识别MNIST

为了更好地理解神经网络，我想看到一个用最少量的代码实现的。我很快找到了11行Python中的神经网络。 学习异或函数并显示反向传播的一小段代码。然而，我觉得我需要一个稍微复杂一些的例子 - 一个可以识别MNIST手写数字数据集的神经网络：

迈克尔·尼尔森（Michael Nielsen）关于神经网络的伟大论著就是建立在这个分类问题之上的，但是最简单的实现是74行代码（没有评论）。我把这个代码重构成了25行（源代码）：

**import** numpy **as** np #导入numpy

**import** mnist #导入mnist数据

**def** **feed\_forward**(X, weights): #前馈函数

a = [X]

**for** w **in** weights:

a.append(np.maximum(a[-1].dot(w),0))

**return** a

**def** **grads**(X, Y, weights):梯度函数

grads = np.empty\_like(weights)

a = feed\_forward(X, weights)

delta = a[-1] - Y

grads[-1] = a[-2].T.dot(delta)

**for** i **in** range(len(a)-2, 0, -1):

delta = (a[i] > 0) \* delta.dot(weights[i].T)

grads[i-1] = a[i-1].T.dot(delta)

**return** grads / len(X)

trainingX, trainingY, testX, testY = mnist.load\_data()

weights = [np.random.randn(\*w) \* 0.1 **for** w **in** [(784, 100), (100, 10)]]

num\_epochs, batch\_size, learn\_rate = 30, 20, 0.1

**for** i **in** xrange(num\_epochs):

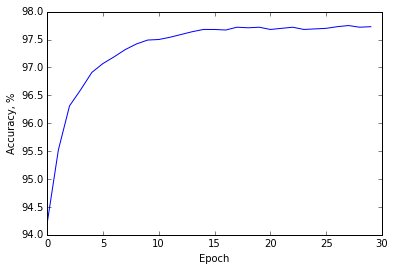
**for** j **in** xrange(0, len(trX), batch\_size):

X, Y = trainingX[j:j+batch\_size], trainingY[j:j+batch\_size]

weights -= learn\_rate \* grads(X, Y, weights)

prediction = np.argmax(feed\_forward(testX, weights)[-1], axis=1)

**print** i, np.mean(prediction == np.argmax(testY, axis=1))

如果你运行这个代码，你将在测试集上达到约97.8％的准确度。

算法实际上与原始实现相同。该网络有一个隐藏层，有100个神经元，并使用小批量梯度下降来学习权重。它有两个依赖关系：numpy，处理向量和矩阵操作; mnist，它是一个简单的脚本，用于下载MNIST数据并将其加载到内存中。

主要区别在于，我使用矩阵操作而不是循环，这意味着您可以将整个小批量处理为矩阵，而不是遍历每个训练示例。 因此，训练比原来的代码快了5倍，而你仍然可以看到底下发生了什么。我将激活函数从sigmoid改为ReLU，这就是f(x)= max(0，x)，它的导数就像x> 0(应用元素方式)一样简单。我也放弃了偏见条款，因为在这个问题上，他们并没有显着提高准确性。