**神经网络中的权重初始化：**

**Why and How**

神经网络中的权重（weight）初始化是个常常被忽略的问题。

**权重初始化：Why**

在创建了神经网络后，通常需要对权重和偏置进行初始化，大部分的实现都是采取Gaussian distribution来生成随机初始值。假设现在输入层有1000个神经元，隐藏层有1个神经元，输入数据x为一个全为1的1000维向量，采取高斯分布来初始化权重矩阵w，偏置b取0。下面的代码计算隐藏层的输入z：

# -\*- coding:utf-8 -\*-

import numpy as np

import matplotlib.pyplot as plt

def run():

x = np.ones(1000)

w = np.random.randn(1000)

b = 0

# z为加权和

z = np.sum(x \* w) + b

print z

然而通过上述初始化后，**因为w服从均值为0、方差为1的正太分布，x全为1，b全为0，输入层一共1000个神经元，所以z服从的是一个均值为0、方差为1000的正太分布**。修改代码如下，生成20000万个z并查看其均值、方差以及分布图像：

def run():

# z的个数

t = 20000

z\_lst = np.empty(t)

x = np.ones(1000)

b = 0

for i in xrange(t):

w = np.random.randn(1000)

z = np.sum(x \* w) + b

z\_lst[i] = z

print 'z 均值：', np.mean(z\_lst)

print 'z 方差：', np.var(z\_lst)

plt.hist(z\_lst, bins=100)

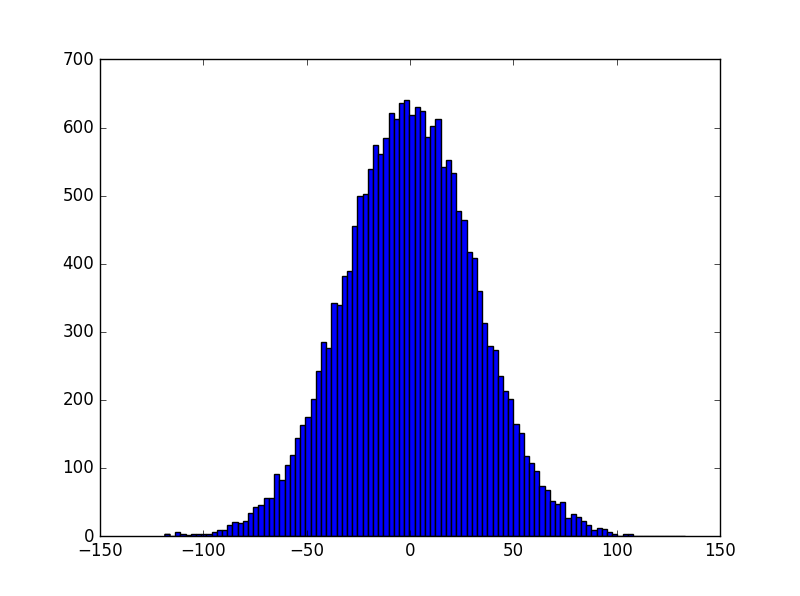
plt.show()

输出结果如下：

z 均值： -0.0402106463845

z 方差： 997.082082524

输出图像如下：



*z分布（1）*

在此情况下，**z有可能是一个远小于-1或者远大于1的数，通过激活函数（比如sigmoid）后所得到的输出会非常接近0或者1，也就是隐藏层神经元处于饱和的状态**。所以当出现这样的情况时，在权重中进行微小的调整仅仅会给隐藏层神经元的激活值带来极其微弱的改变。而这种微弱的改变也会影响网络中剩下的神经元，然后会带来相应的代价函数的改变。结果就是，这些权重在我们进行梯度下降算法时会学习得非常缓慢。

因此，我们可以通过改变权重w的分布，使|z|尽量接近于0。这就是我们为什么需要进行权重初始化的原因了。

**权重初始化：How**

一种简单的做法是修改w的分布，使得z服从均值为0、方差为1的标准正态分布。根据正太分布期望与方差的特性，将w除以sqrt(1000)即可。修改后代码如下：

def run():

# z的个数

t = 20000

z\_lst = np.empty(t)

# 输入神经元个数

m = 1000

x = np.ones(m)

b = 0

for i in xrange(t):

w = np.random.randn(m) / np.sqrt(m)

z = np.sum(x \* w) + b

z\_lst[i] = z

print 'z 均值：', np.mean(z\_lst)

print 'z 方差：', np.var(z\_lst)

# 保持与z分布（1）图像横坐标刻度不变，使得结果更加直观

plt.xlim([-150, 150])

plt.hist(z\_lst, bins=10)

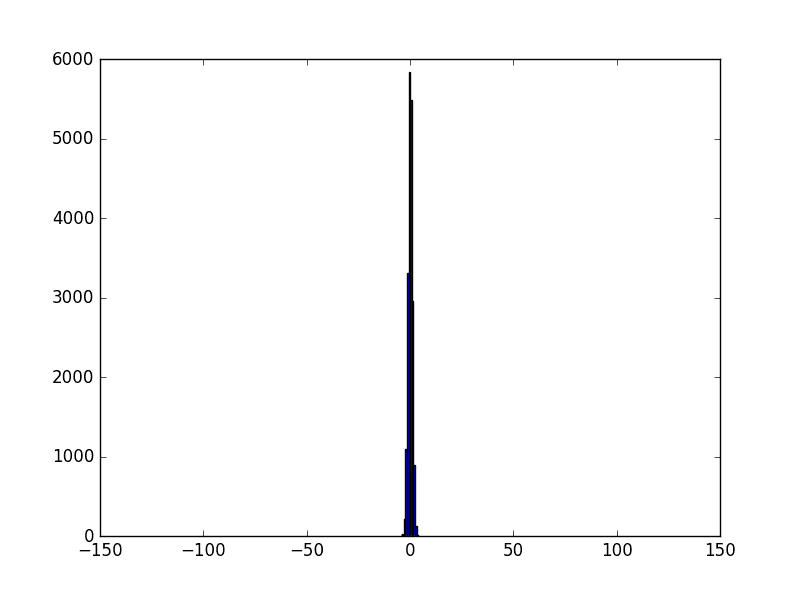
plt.show()

输出结果如下：

z 均值： 0.013468729222

z 方差： 1.00195898464

输出图像如下：



*z分布（2）*

这样的话z的分布就是一个比较接近于0的数，使得神经元处于不太饱和的状态，让BP过程能够正常进行下去。

除了这种方式之外（除以前一层神经元的个数n\_in的开方），还有许多针对不同激活函数的不同权重初始化方法，比如兼顾了BP过程的除以( (n\_in + n\_out)/2 )。