数值方法在 BP 神经网络中的应用

朱睿 | 数值方法 | 2017/4/17

一． 概述：

BP 神经网络（backpropagation）是一种按误差逆传播算法训练的多层前馈网络，是目前应用最广泛的神经网络模型之一。BP 网络能学习和存贮大量的输入-输出模式映射关系，而无需事前揭示描述这种映射关系的数学方程。它的学习规则是使用梯度下降法，通过反向传播来不断调整网络的权值和阈值，使网络的误差平方和最小。BP 神经网络模型拓扑结构包括输入层（input）、隐层(hidden layer)和输出层(output layer)。

二． BP 神经网络的结构：

1．神经元：

简单说来，神经网络是由多个基本的神经元组成的网络系统，每一个神经元有对应的输入和相应的输出。利用下面的这幅图来说明单一的神经元的工作过程。

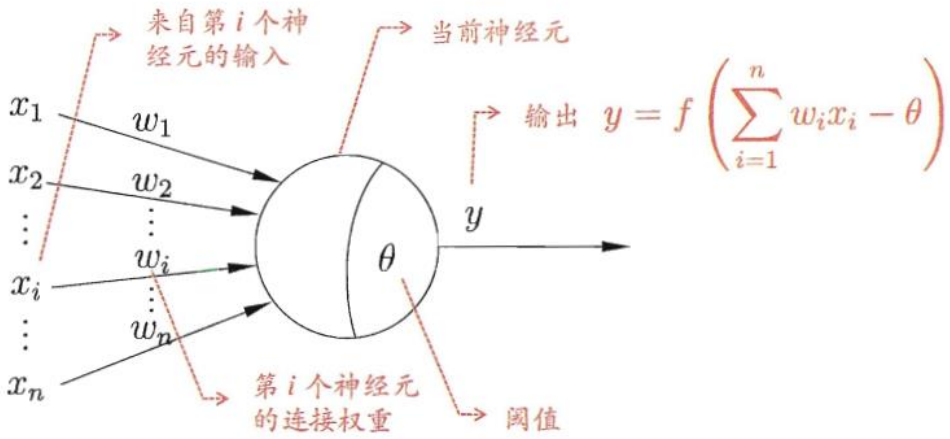


图 1 神经元结构

x1,x2…xn	某一个输入样例的 n 个特征，
ω1,ω2…ωn	每个特征所对应的权重
θ	该神经元的阈值
y	输出的值
f	激活函数

2．激活函数：

Sign 函数：使用该函数，处理输出的线性加权和，映射得到的输出 y 是 0 或者 1 的离散值，适用于二元分类。

Sigmoid 函数：使用该函数，处理输出的线性加权和，映射得到的输出 y 是连续值，适用于计算一个 0 到 1 的连续值。

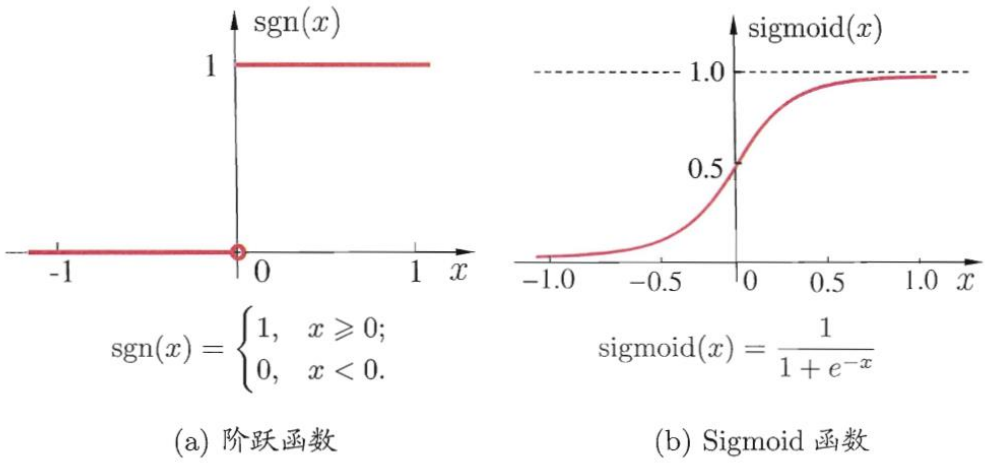


图 2 经典激活函数

3．BP 神经网络：

把上面许多神经元按一定的结构组成起来，就构成了神经网络：

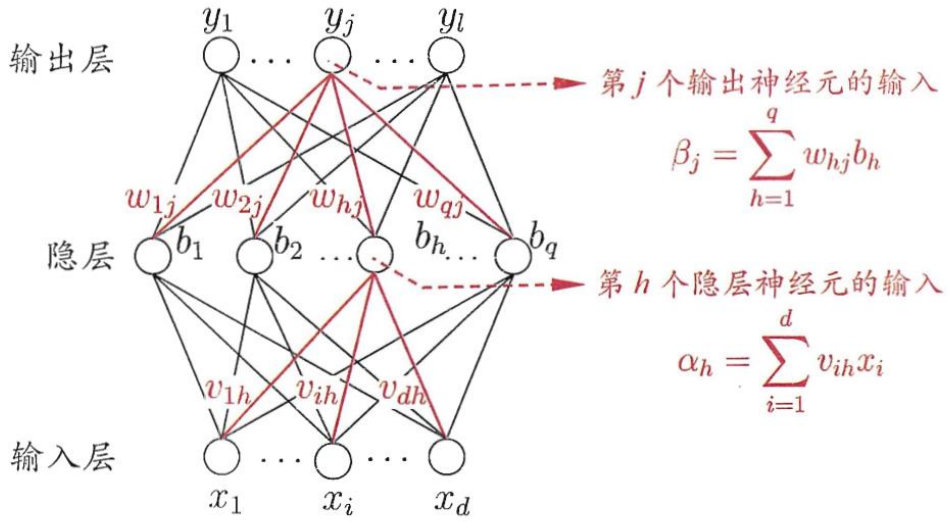


图 3 BP 神经网络结构

输入层：只负责输入样例 X 的 d 个特征，不参与计算

隐层：对输入的特征进行第一次映射，得到 b1,b2...bq 个特征

输出层：再一次映射，得到最终结果

4．误差逆传播算法：(图片来自周志华《机器学习》)

由于单个神经元的学习能力较弱，有些复杂问题并不能只靠神经元来解决，因此需要多层网络。采用目前最成功的误差逆传播算法（backpropagation）它的学习规则是使用最速梯度下降法，通过反向传播来不断调整网络的权值和阈值，使网络的误差平方和最小。接下来将具体分析这个算法，请参照图 3。

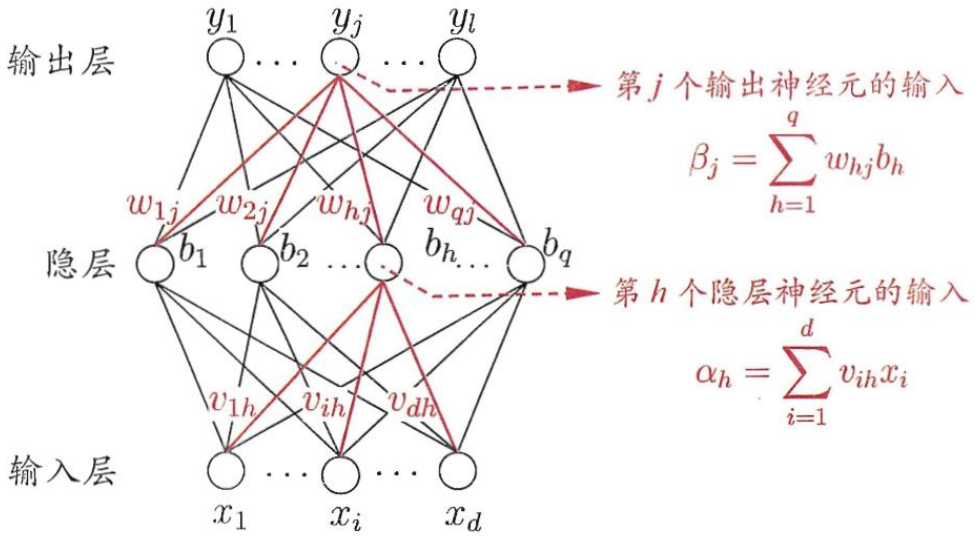


图 3 BP 神经网络结构

数据集：$D = \{(x_1, y_1), \ldots\ldots, (x_m, y_1)\}$,其中，输入样本是有 d 维的向量组成，输出样本是由 L 维的向量。一个维度对应一个特征。其中有 d 个输入，L 个输出，中间的隐层可以设置成 q 个。

如图所示，第 h 个隐层神经元的输入α_h为 d 个输入神经元与各自对应的权值 v 的乘积之和，第 j 个输出神经元的输入为全部的 q 个隐层神经元的输出与各自对应的权值乘积之和。隐层第 h 个神经元的阈值用γ_h表示，输出层第 j 个神经元的阈值用θ_h表示。

现在假设神经元的输出表示为 $\hat{y}_k = (\hat{y}_1^k, \hat{y}_2^k, \ldots, \hat{y}_l^k)$，即：

$$\hat{y}_j^k = f(\beta_j - \theta_j) ,$$

采用均方误差最小化来最优化系数：

$$E_k = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k - y_j^k)^2 .$$

图 3 的网络中有(d+l+1)q+l 个参数需要确定：输入层到隐层的 d×q 个权值、隐层到输出层 q×l 个权值、q 个隐层神经元的阈值、l 个输出层神经元的阈值。BP 是一个迭代学习算法，在迭代的每一轮采用广义的感知机学习规则对参数进行更新估计，即：

$$w_i \leftarrow w_i + \Delta w_i ,$$

$$\Delta w_i = \eta(y - \hat{y})x_i ,$$

其中η∈（0,1）称为学习率。从式子中可以看出，若队训练样例预测正确，则w不发生变化，否则将根据错误的程度进行权重调整。类似地，任意参数υ的更新估计式为：

$$v \leftarrow v + \Delta v .$$

下面我们以图 3 中隐藏层到输出层的连接权w_{hj}为例来进行推导：
基于梯度下降策略，以目标的负梯度方向对参数调整，对于误差E_k，给定学习率η，有：

$$\Delta w_{hj} = -\eta\frac{\partial E_k}{\partial w_{hj}} .$$

由于$w_{h,j}$先影响到输出层第 j 个神经元的输入，再影响到其输出，最后影响到E_k，所以采用链式求导法则可得：

$$\frac{\partial E_k}{\partial w_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}} .$$

根据β_j的定义，有：

$$\frac{\partial \beta_j}{\partial w_{hj}} = b_h .$$

而 Sigmoid 函数有一个比较好的性质：

$$f'(x) = f(x)(1 - f(x)) ,$$

所以综合之后，可得：

$$\begin{aligned} g_j &= -\frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \\ &= -(\hat{y}_j^k - y_j^k) f'(\beta_j - \theta_j) \\ &= \hat{y}_j^k (1 - \hat{y}_j^k)(y_j^k - \hat{y}_j^k) . \end{aligned}$$

带入链式求导公式，可得：

$$\Delta w_{hj} = \eta g_j b_h \ .$$

类似可以得到：

$$\Delta \theta_j = -\eta g_j \ ,$$

$$\Delta v_{ih} = \eta e_h x_i \ ,$$

$$\Delta \gamma_h = -\eta e_h \ ,$$

其中：

$$\begin{aligned} e_h &= -\frac{\partial E_k}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h} \\ &= -\sum_{j=1}^{l} \frac{\partial E_k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} f'(\alpha_h - \gamma_h) \\ &= \sum_{j=1}^{l} w_{hj} g_j f'(\alpha_h - \gamma_h) \\ &= b_h(1-b_h) \sum_{j=1}^{l} w_{hj} g_j \ . \end{aligned}$$

对于每个训练样本，BP 算法先将输入样例提供给输入神经元，然后逐层的将信号向前传播，直到产生输出层的结果，然后计算输出层的误差，再将误差逆向传播到隐层神经元，然后根据神经元的误差来对连接权值和与之进行调整优化，知道训练达到很小的误差值或者迭代到一定的次数。具体过程如下图所示：

输入： 训练集 $D = \{(\boldsymbol{x}_k, \boldsymbol{y}_k)\}_{k=1}^{m}$;
学习率 η.

过程：

在$(0,1)$范围内随机初始化网络中所有连接权和阈值
repeat
  for all $(\boldsymbol{x}_k, \boldsymbol{y}_k) \in D$ **do**
    根据当前参数和式(5.3) 计算当前样本的输出 $\hat{\boldsymbol{y}}_k$;
    根据式(5.10) 计算输出层神经元的梯度项 g_j;
    根据式(5.15) 计算隐层神经元的梯度项 e_h;
    根据式(5.11)-(5.14) 更新连接权 w_{hj}, v_{ih} 与阈值 θ_j, γ_h
  end for
until 达到停止条件

输出： 连接权与阈值确定的多层前馈神经网络

图 4 逆误差传播算法：

注意，BP 算法的目标是最小化训练集 D 上的累计误差：

$$E = \frac{1}{m}\sum_{k=1}^{m} E_k \ ,$$

三． MNIST 数据集：

MNIST 数据集是一个手写体数据集，通过阅读官网可以知道，这个数据集由四部分组成，分别是：

```
train-images-idx3-ubyte.gz:  training set images (9912422 bytes)
train-labels-idx1-ubyte.gz:  training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz:   test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz:   test set labels (4542 bytes)
```

一个训练图片集，一个训练标签集，一个测试图片集，一个测试标签集。可以看出这个其实并不是普通的文本文件或是图片文件，而是一个压缩文件，下载并解压出来，我们看到的是二进制文件，其中训练图片集的内容部分如此：

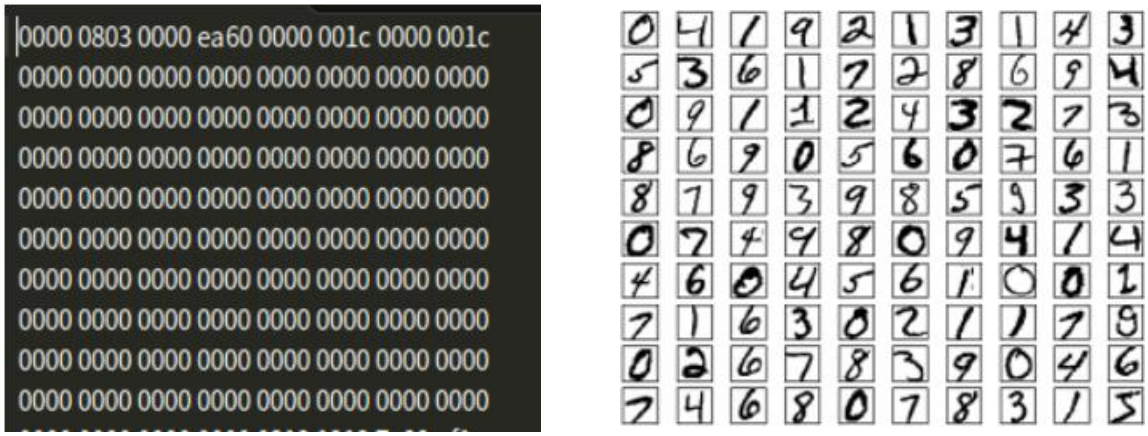


图 5 MNIST 数据集（二进制文件；实际信息）

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	60000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
....			
xxxx	unsigned byte	??	label

The labels values are 0 to 9.

图 6 训练集性质

训练集有 60000 个用例，即该文件里面包含了 60000 个标签内容，每一个标签的值为 0 到 9 之间的一个数；回到训练标签集上，先解析每一个属性的含义，offset 代表了字节偏移量，也就是这个属性的二进制值的偏移是多少；type 代表了这个属性的值的类型；value 代表了这个属性的值是多少；description 是对这个的说明。使用网上的方法将二进制文件解析成.mat 文件（没学过数据库，所以用网上的方法），方便使用 Python 读取。

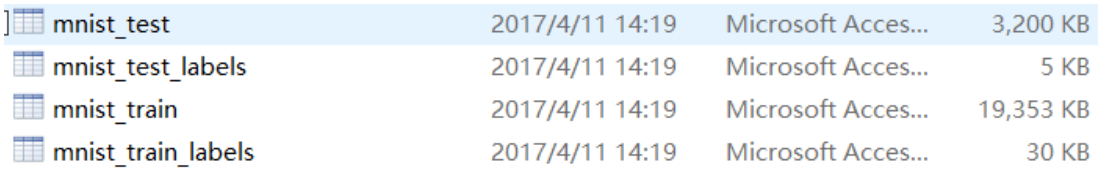


图 7 解析得到的 MNIST 的 mat 文件

四． 小组所做假设：

神经网络的一个大问题就是容易陷入局部最优，达不到所要求的正确率，我们在跑代码的时候也发现了这个问题：有时候很快就跑完了，有时候要等较长时间才能跑完。所以我们提出如下假设：

1. 迭代步长的影响，有时候迭代步长较短，当学习算法跑到局部最优的时候，很难从局部最优跳出来，所以要设置合适的步长值。然而这方面有许多人做出了优化，比如使用变步长的方式，用一种反馈的算法改变每次要走的步长，从而确保能够跳出局部最优。因此我们不对步长做研究。
2. 权值初始值的影响：这方面研究的人比较少，我们查到的资料，论文相对较少，我们采用了两种方式，第一种完全随机，使用 Python 的内置随机方法，针对不同的步长，各做 1000 次实验，来对单次代码运算时间，迭代次数，测试集准确率取平均值，评价它的性能。第二种采用了正态分布随机初始权值的方式（自然界大多数数据符合正态分布），第三种就采用了均匀分布的方式，评价性能的方法同上。（实验数据会附在压缩包里面）

实验结果

实验 1：

(a) 两个权值取 0.1,0.3，按照不同的排列组合来做实验，在训练时，在训练集内达到 90%的准确率就跳出，然后进行测试集验证效果。为防止局部最优浪费太多时间，我们规定迭代 500 次还没有跳出训练集的学习为陷入局部最优，达到 500 次自动跳到测试集验证效果。

(b)每种方法对应不同的步长都做了 1000 次实验，得到迭代时间，测试准确率，迭代次数的 1000 次的平均值。

(c)均匀分布方法得到的效果非常差，其他方法跑两天就出了结果，而均匀分布方法需要跑两个星期，得到的结果也非常差。

(d)方法：步长 1=0.3，步长 2= 0.1 同样耗时过多，不参与最终讨论。

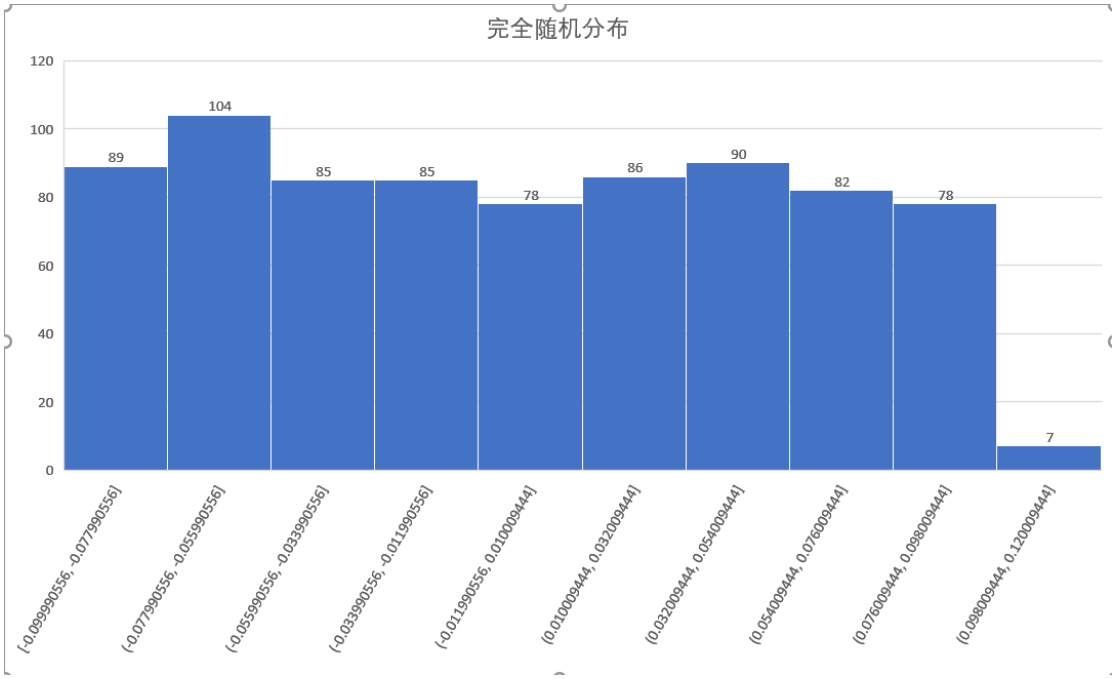
(e)在得到的随机方式初始权值上面，我们每个都乘以了 0.2 进行归一化处理，否则运算时间会大大加长

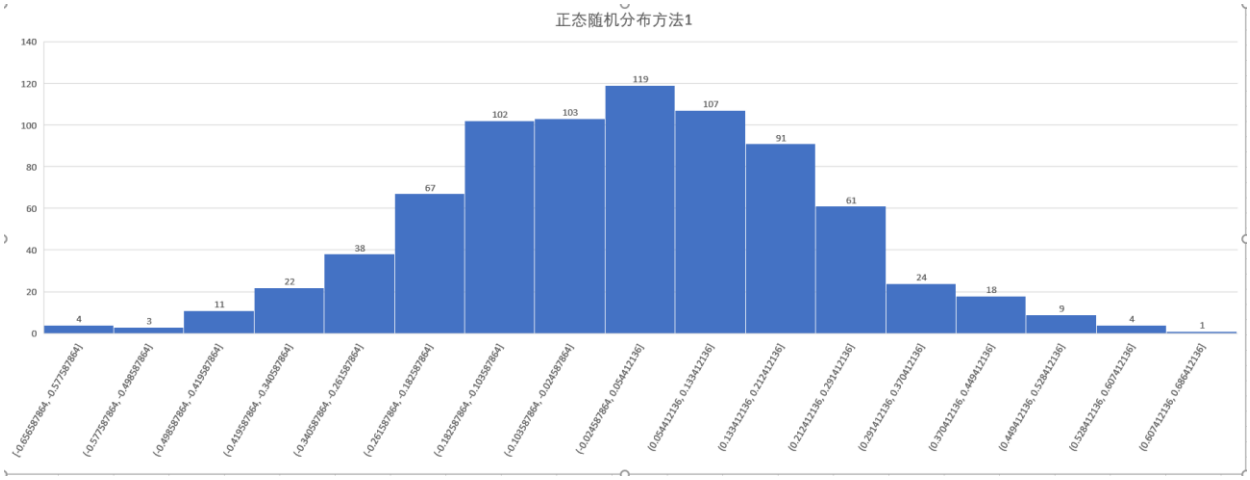
(f)下面这张表格是实验的结果（并没有将超出 500 次的实验数据剔除）：

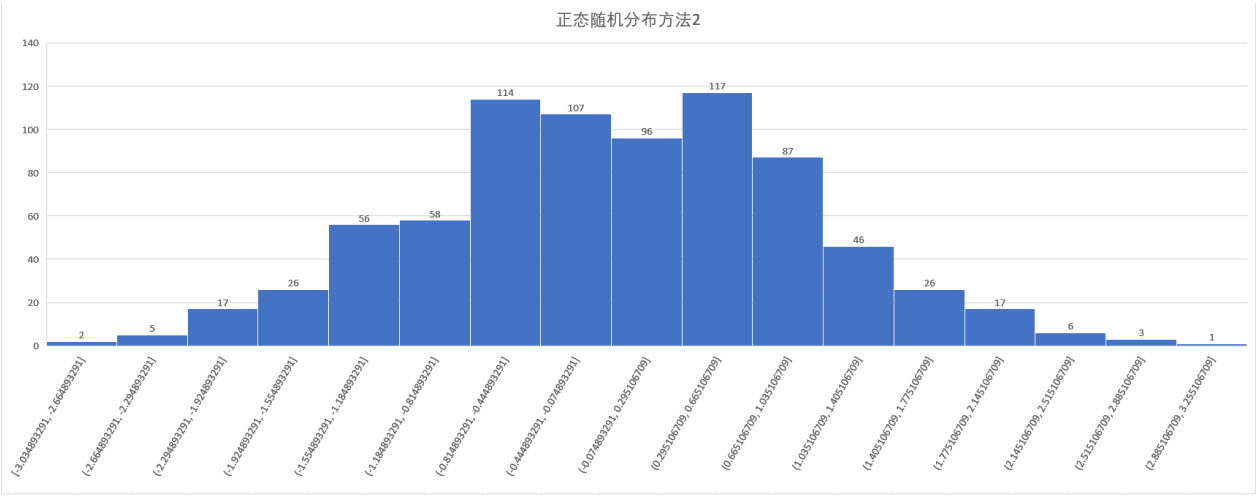
可以看到，完全随机方法不论在迭代时间还是迭代次数上面都优于正太随机分布。（1000 次实验，3 种不同方式都如此）

方法	步长A	步长B	目标训练准确率		迭代时间	测试准确率		平均迭代次数
完全随机	0.1	0.3	0.9		204.5686	0.887005		13.902
正态分布随机1	0.1	0.3	0.9		227.934	0.886113		16.374
正态分布随机2	0.1	0.3	0.9		240.3645	0.8866741		16.183
完全随机	0.1	0.1	0.9		331.0821	0.884863		22.465
正态分布随机1	0.1	0.1	0.9		390.8022	0.8850523		26.404
正态分布随机2	0.1	0.1	0.9		436.3845	0.8845548		31.119
完全随机	0.3	0.3	0.9		831.798	0.885331		55.639
正态分布随机1	0.3	0.3	0.9		992.029	0.884774333		66.75
正态分布随机2	0.3	0.3	0.9		933.7868	0.8846976		62.4

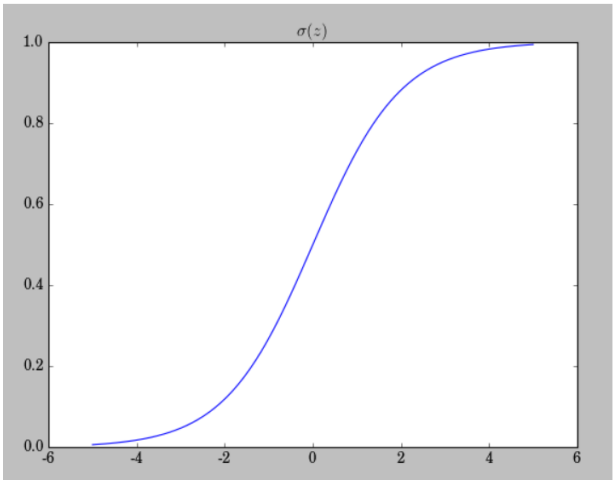
接下来，我们对实验步长 A=0.1，步长 B=0.3 的初始向量进行操作，画出直方图：



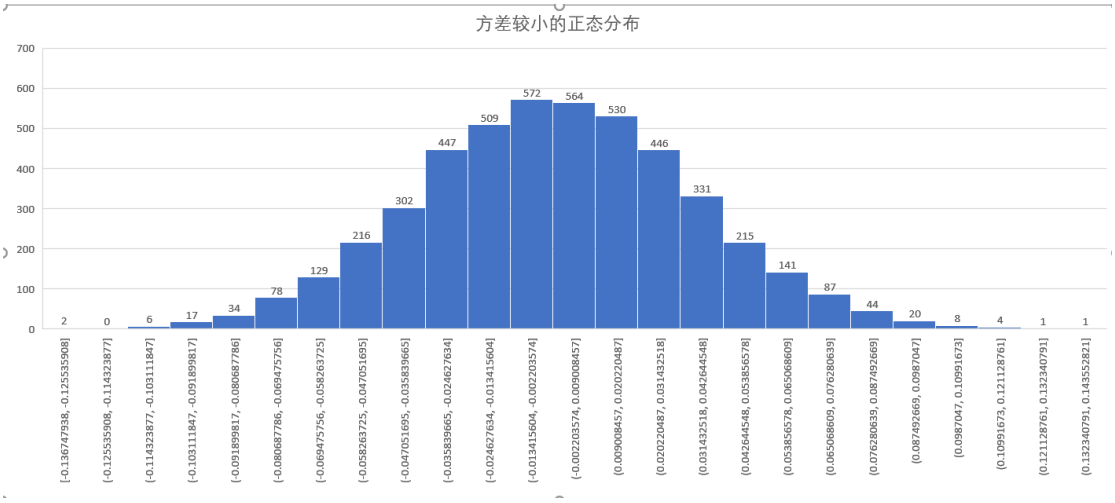


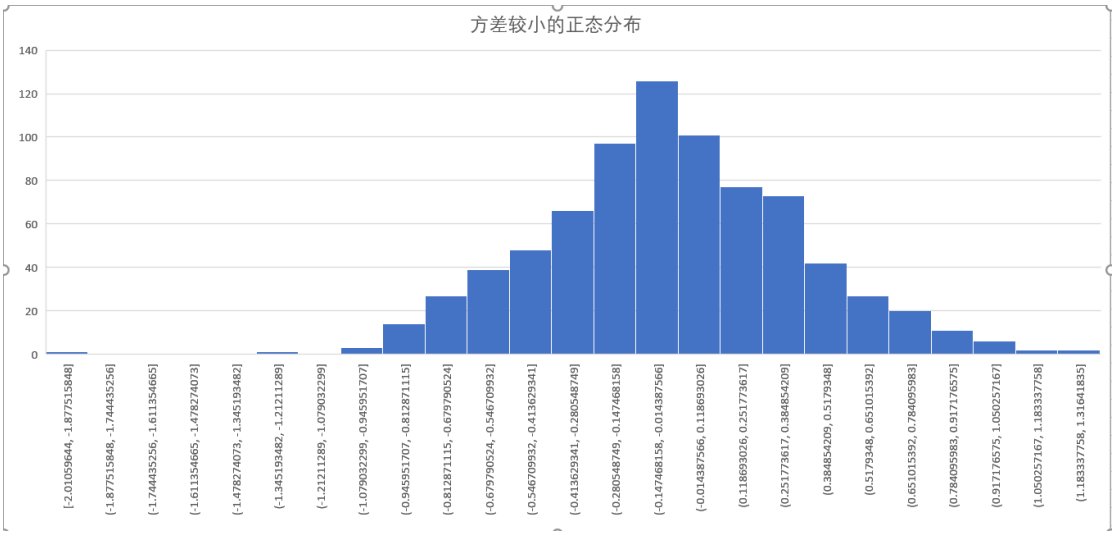


我们分析之所以性能相对较差是因为正态分布的方差较大，导致分布较扁平，所以想办法使得正态分布变得相对尖锐一些，大部分初始值集中在 0 附近，这样效果会比较好：因为权值的公式和激励函数的导数成正比，所以要将 w 的初始值集中在导数很大的地方，就是接近 0 的地方（看下面的图），提高学习速率。



所以我们降低初始两种方法的方差，使数据集中：





得到的实验结果如下：

改动正态分布1	0.1	0.3	0.9		**191.314**	**0.8872616**		**13.966**
改动正态分布2	0.1	0.3	0.9		**197.199**	**0.8873782**		**14.309**
完全随机	0.1	0.3	0.9		**204.569**	**0.887005**		**13.902**

可以与上面的表格对比，速度提升很大，1000 组实验提升 3.6 小时，而且迭代次数从 16 降到 14，很可观的结果。（上面的表格数据都是 1000 次实验取平均）

五．python 代码：

代码附在提交的压缩包里面。

附录

1．梯度下降法：

梯度下降法是一种一阶优化方法，解决无约束优化问题最简单，最经典的方法之一。

考虑无约束优化问题 $min_x f(x)$，其中 f（x）为连续可微函数，若能构造一个序列 $x^0, x^1, x^2, \ldots$，满足：

$$f(\boldsymbol{x}^{t+1}) < f(\boldsymbol{x}^t), \quad t = 0, 1, 2, \ldots$$

若不断执行该过程即可收敛到局部极小点，欲满足上式，根据泰勒展开式有：

$$f(\boldsymbol{x} + \Delta\boldsymbol{x}) \simeq f(\boldsymbol{x}) + \Delta\boldsymbol{x}^{\mathrm{T}} \nabla f(\boldsymbol{x}) \ ,$$

于是，欲满足 f（**x**+Δ**x**）<f（**x**），可以选择：

$$\Delta\boldsymbol{x} = -\gamma \nabla f(\boldsymbol{x}) \ ,$$

其中步长γ是一个小常数（步长），这就是梯度下降法。
若目标函数 f（**x**）满足一些条件，则通过选取合适的步长，就能确保通过梯度下降收敛到局部极小点。

引用：

1.人工神经元网络及其应用-袁曾任

2.Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights-Derrick Nguyen and Bernard Widrow

3.N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in Conference on Computer Vision and Pattern Recognition. IEEE, 2005, vol. I, pp. 886–893.

4.机器学习—周志华

5.Text Classification-Duda