**model selection**

**模型选择 欠拟合和过拟合(梁明)**

1. 训练误差：模型在训练数据集上表现出的误差。
2. 泛化误差：模型在任意一个测试数据样本上表现出的误差的期望。
3. 模型选择
4. 验证数据集:用于模型选择的在训练数据集和测试数据集以外的数据。
5. k折交叉验证:用于改善大量验证数据集剩余的情况。

4. 欠拟合和过拟合:

1. 欠拟合:模型无法得到较低的训练误差。

2) 过拟合:模型的训练误差远小于测试数据集上的误差。

5. 多项式函数拟合实验

1. 生成数据集：

采用三阶多项式函数来生成标签 y=1.2x−3.4x2+5.6x3+5+ϵ

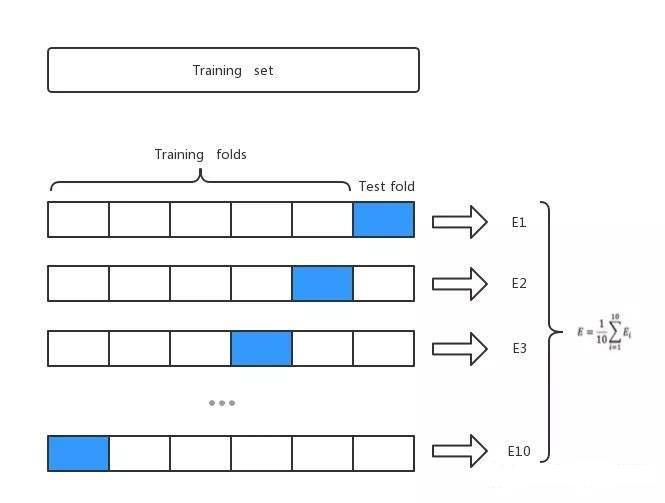
1. 多项式函数拟合（用平方损失函数）
2. 得到拟合结果

欠拟合和过拟合主要由两种因素影响：(邓楠)

1. 模型复杂度：模型复杂度高，参数选择空间大，更容易得到更低的训练误差。但是过高，泛化误差会上升导致过拟合。
2. 训练数据集：训练集过小，类比模型复杂度相对数据集过大，容易过拟合。模型复杂度过高过低使泛化误差过大，而训练集则是取更大的会更好。

**交叉验证（胡涛）**

**为了实现对测试误差的有效估计，最常用的方法就是做交叉验证(cross validation)。交叉验证的思想很简单，将已有数据划分成K份，次将其中的一份作为验证集(validation set)，而用剩下的K-1份数据作为训练集训练得到模型，这样就可以重复训练K次，可以得到K个验证误差(validation error)，将其平均作为对测试误差的估计。K的取值通常为5或10，特别的，当K=N时叫做留一验证法(leave-one-out)。值得一提的是，K当越大，验证误差与测试误差之间的偏差就越小，但是方差越大，反之亦然。**

**应旻昊总结**

booststrapping

**这个方法的核心就是”自助采样”，如果有m个样本。则我们随机从这样本集中有放回的抽取m个数。这样必然有数据被多次采到，当然也有数据从未被采到。通过极限计算可以得到从未被采到的数据大约有37%。**

**我们可以将采集到的m个样本作为训练集，将从未采集到的样本作为测试集。当然多次的采集会改变样本本身的分布规律。这种方法在小样本集上是非常好用的。**

稳定训练

**（李思梦总结）**

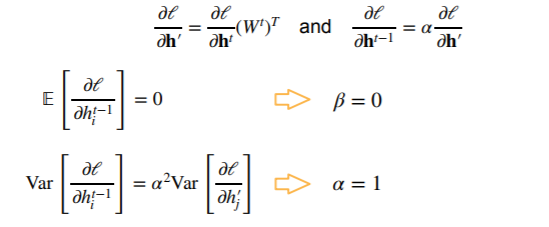
**目的：确保梯度值在适当范围内，例如在10-6---103之间；**

**批量标准化，梯度裁剪；**

**正确的重量初始化和激活功能；**

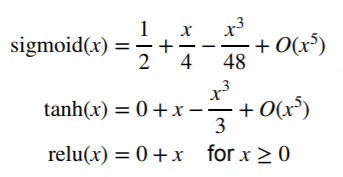
**激活：就是一个简单的线性函数和激活函数；**

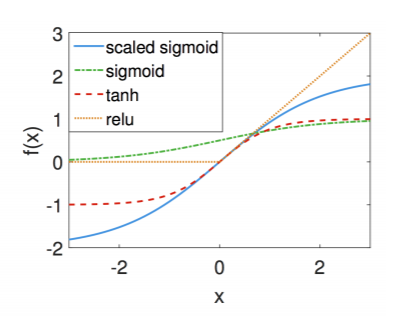
**假设σ(x) = αx + β，**



**修改激活功能**

**泰勒展开式：**



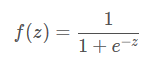


**每次sigmoid（x）=4\*sigmoid（x）-2；**

**（叶钰莹总结）**

**一．常见激活函数：**

1. **Sigmoid 函数：**

****

1. **tanh函数**

****

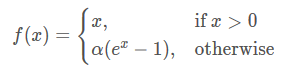
1. **ReLU函数**

****

1. **Leaky ReLU 函数（PReLU）**

****

1. **ELU函数**

****

**二．深层神经网络的两个问题**

**1.Gradient Exploding problem(梯度爆炸)**

**用ReLU作为激活函数**

1. **会得到一个超出范围的无穷大值（严重使用16位浮点数）**

**②对学习速度（LR）敏感（不够小的LR->大的权重->更大的梯度；太小的LR ->没有进展；可能需要在培训期间大幅度更改LR）**

**2.Gradient Vanishing problem(梯度消失)**

**用sigmoid作为激活函数**

1. **值为0的梯度（严重的16位浮点数）**
2. **训练没有进展（无论如何选择学习速度）**
3. **底部图层严重（只有顶层得到了训练）**

**三．Hardware for Deep Learning(深度学习硬件)**

**1.Intel i7-6700k(因特尔）**

**提高CPU的利用率1：**

**①在计算前先准备好数据**

**②改善时间和空间记忆的局限性**

**时态：重用数据，以便将它们保存在缓存中**

**空间：按顺序读取数据，以便预取数据**

**提高CPU的利用率2：**

**①服务器CPU可能有几十个核心**

**②并行化以使用所有核心（超线程可能没有帮助，因为共享寄存器）**

**Nvidia Titan X(Pascal) 英伟达泰坦X(帕斯卡)**

**提高CPU的利用率：**

**①平行化，利用数千个线程**

**②内存位置，简单的缓存架构和较小的缓存大小**

1. **简单的控制流，（非常有限的支持，大型同步）**

**（刘一鸣总结）**

**深度神经网络的两个问题**

**1、 梯度爆炸：（1.5^100≈4E17）**

**使用情况**

**1、 使用范围在（6E-5到6E4之间的16位浮点数）**

**2、 对于学习速度很敏感**

**速度太快：梯度太大**

**速度太慢，可能没结果**

**需要在学习期间大幅更改学习速度**

**2、梯度消失：（0.8^100≈2E-10）**

**使用情况**

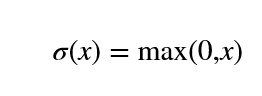
**1、 梯度值为0（严格使用16位浮点数）**

**2、 不管怎么选择学习速度，训练都没进展**

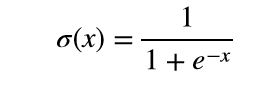
**3、 使用底层较为苛刻，只有顶层可以得到很好的训练，做底层意义不大**

**激活函数**

**1：Pelu:**

****

**2:sigmoid:**

****

**3:tanh:**

**σ（x）=(*ex*−*e*−*x)/(ex*+*e*−*x)***

**稳定训练：**

**1、 目标：确保渐变值在一个合适的范围**

**2、 正常化**

**3、 恒定每一层的方差**

**4、 前向方差、反向均值和方差，Xevier初始化**

**硬件**

**RAM⬄CPU⬄GPU**

**提高CPU利用率方法一：**

**RAM->L3->L2->L1->寄存器**

**改善时间空间内存分布**

**提高CPU利用率方法二：**

**多核心多线程**

**提高GPU利用率方法：**

**1、 并行化，使用数千核心**

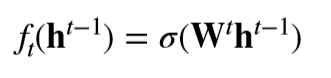
**2、 简单的缓存架构和较小的缓存大小**

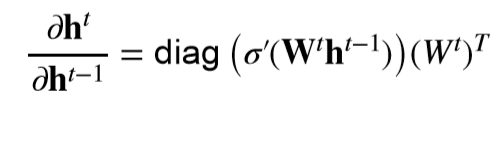
**3、 简单的控制流**

**其它硬件：DSP FPGA AI ASIC Systolic Array**

**CPU、GPU、DSP、FPGA、AI ASIC等灵活性、通用性以此降低，能效比依次提高。**

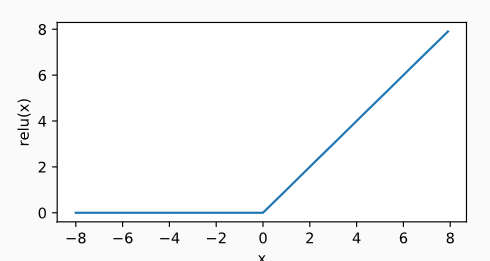
**（徐明雪总结）**

1. **Numerical Stability**
2. **在MLP中，假设没有偏差。**

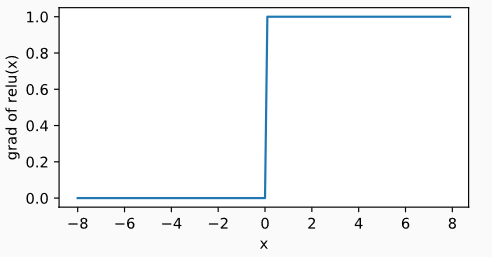
****

1. **激活函数**
2. **Relu函数**

**定义：Relu(x)=max(x,0) (负数元素清零)**

**图像：**

**导数图像：**

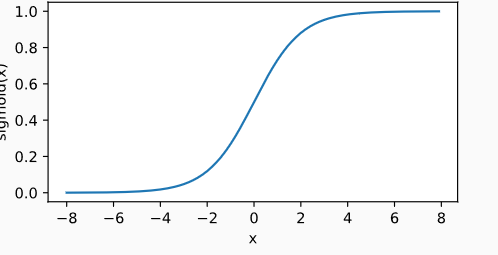
****

1. **sigmoid函数**

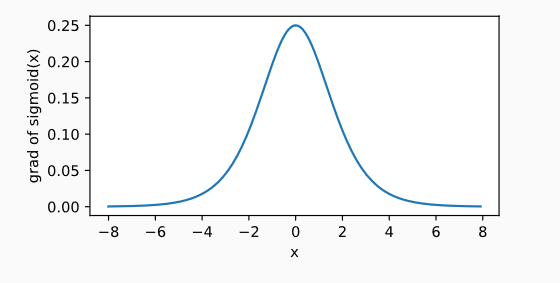
**(可将值变换到0—1之间)**

**定义：sigmoid(x)=1/(1+exp(-x))**

**导数：sigmoid’(x)=sigmoid(x)(1-sigmoid(x))**

**图像：**

**注：在输入接近0时，函数接近线性变换；**

**导数图像：**

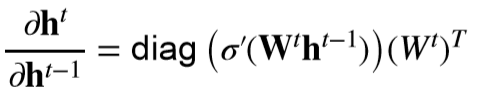
**3、梯度爆炸的一些问题**

1. **简而言之，即梯度的值超出范围**
2. **对学习率敏感**

**具体：不够小的LR->大的权重->更大的梯度**

**太小的LR->培训过程中可能不会显著改变LR;**

**3、梯度消失的一些问题**

1. **无论如何选择学习率，只有顶层经过良好的培训，无法到达深层次的培训；**

**关于梯度的总结：**

**以上，忽略第一项，权重w的更新会影响梯度的变化，如若，学习率较大，便会使得w的变化十分剧烈，从而无法得到最优化的 w值，此时，在其他条件一定时，学习率较大，便会造成梯度爆炸；但是，如若学习率较小，w的值便会更新的不明显，此时，其他条件不变时，梯度消失。**

1. **权重初始化**
2. **随机初始化：**

**例：随机初始化，在线性回归中，net.initialize(init.Normal(sigmal=0.001))权重参数，正态分布的随机初始化方式。**

1. **Xavier的随机初始化**

**输入a个，输出b个。**

**则随机均匀分布（-（b/(a+b)(1/2)）, （b/(a+b)(1/2)））;**

1. **重改激活函数**

**泰勒展开式**

1. **Hardware for deep learning**
2. **Intel i7-6700K，四核处理器；**
3. **提高CPU利用率**

**注：在计算前，需要调用数据，顺序，主存储器->高速缓冲存储器->寄存器。**

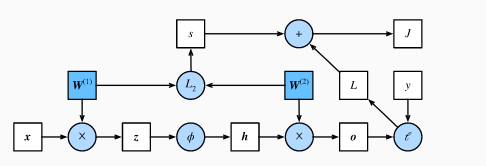
1. **时间：重用数据，以便使他们保存在缓存中；**
2. **空间：按顺序读取数据，以便我们可以预取数据；**
3. **多核心**
4. **多线程运行**
5. **提高GPU利用率**
6. **并行化，使用多个线程；**
7. **简单的缓存架构和较小的缓存位置**
8. **简单的控制流**
9. **CPU和GPU的比较**

**（王树仁总结）**

1. **正向、反向传播算法**

**（1）正向传播是指对神经网络沿着从输入层到输出层的顺序，依次计算并存储模型的中间变量（包括输出）。为简单起见，假设输入是一个特征为x∈R的样本，且不考虑偏差项**

**绘制例模型正向传播的计算图，其中左下角是输入，右上角是输出。可以看到，图中箭头方向大多是向右和向上，其中方框代表变量，圆圈代表运算符，箭头表示从输入到输出之间的依赖关系。**

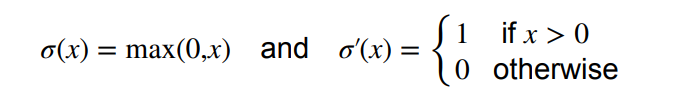
****

**隐藏层的权重参数W(1)x，中间变量z=W(1)x，激活函数ϕ向量长度为h的隐藏层变量h=ϕ(z)，向量长度为q的输出层变量o=W(2)h，单个数据样本的损失项L=ℓ(o,y)，在给定的数据样本上带正则化的损失J=L+s**

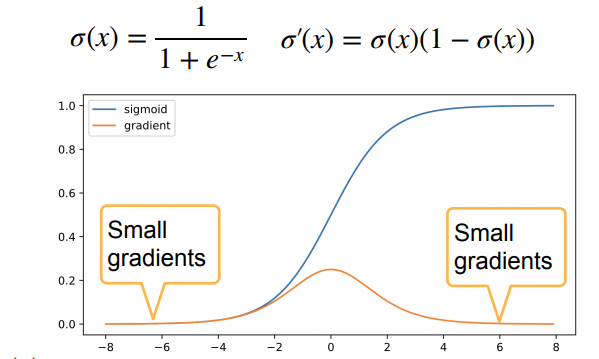
**（2）反向传播指的是计算神经网络参数梯度的方法。总的来说，反向传播依据微积分中的链式法则，沿着从输出层到输入层的顺序，依次计算并存储目标函数有关神经网络各层的中间变量以及参数的梯度。**

* **正向传播沿着从输入层到输出层的顺序，依次计算并存储神经网络的中间变量。**
* **反向传播沿着从输出层到输入层的顺序，依次计算并存储神经网络中间变量和参数的梯度。**
* **在训练深度学习模型时，正向传播和反向传播相互依赖。**

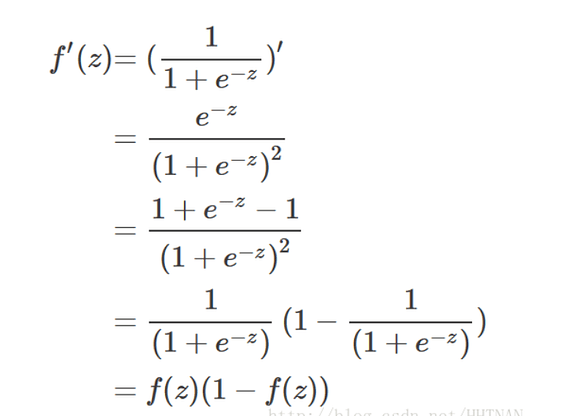
**（4）梯度爆炸用ReLU函数作为激活函数**

****

**梯度衰减用sigmoid函数作为激活函数**

****

**sigmoid函数推导过程：**

****

1. **数值稳定性**
2. **深度模型有关数值稳定性的典型问题是衰减（vanishing）和爆炸（explosion）当神经网络的层数较多时，模型的数值稳定性容易变差。**

**假设一个层数为LL的多层感知机的第ll层H(l)的权重参数为W(l)，输出层H(L)的权重参数为W(L)。为了便于讨论，不考虑偏差参数，且设所有隐藏层的激活函数为恒等映射（identity mapping）ϕ(x)=x。给定输入X，多层感知机的第l层的输出H(l)=XW(1)W(2)…W(l)H(l)=XW(1)W(2)…W(l)。此时，如果层数l较大，H(l)的计算可能会出现衰减或爆炸。**

**（2）在神经网络中，通常需要随机初始化模型参数。**

**如果将每个隐藏单元的参数都初始化为相等的值，那么在正向传播时每个隐藏单元将根据相同的输入计算出相同的值，并传递至输出层。在反向传播中，每个隐藏单元的参数梯度值相等。因此，这些参数在使用基于梯度的优化算法迭代后值依然相等。之后的迭代也是如此。在这种情况下，无论隐藏单元有多少，隐藏层本质上只有1个隐藏单元在发挥作用。通常将神经网络的模型参数，特别是权重参数，进行随机初始化**

**MXNet将使用默认的随机初始化方法：权重参数每个元素随机采样于-0.07到0.07之间的均匀分布，偏差参数全部清零**

**Xavier随机初始化。假设某全连接层的输入个数为a，输出个数为b，Xavier随机初始化将使该层中权重参数的每个元素都随机采样于均匀分布**

1. **硬件**

**RAM⬄CPU⬄GPU**

**（1）提高CPU利用率**

**时间：重用数据，以便使他们保存在缓存中；**

**空间：按顺序读取数据，以便我们可以预取数据；**

**多核心、多线程运行**

**（2）提高GPU利用率**

**并行化，使用多个线程；**

**简单的缓存架构和较小的缓存位置**

**简单的控制流**

**（钱鹏屿总结）**

**一、神经网络的反向传播算法**

**将神经网络的输出值和标准值进行比较，从而得到误差值。然后计算网络的每一层对这个误差值的“贡献”，并对每一层的权重和偏置进行调整的过程。**

**它从输出层开始，在反向传播过程中，需要计算输出 对所有权重 和偏置 的偏导数，直到输入层为止。然后，利用这些计算出来的偏导数，更新对应的权重和偏置，从而达到反向传播的目的。**

**二、深度神经网络的两个问题**

**1、梯度消失**

**在深度网络中，为了计算初始层的梯度，我们会累乘多个 项 ，最终计算结果将会呈指数级变小，这也就是梯度丢失产生的原因。**

**（1）激活函数：sigmoid**

**（2）超出值的范围：0**

**（3）训练没有进展**

**底部图层严重，只有顶层可以得到有效训练**

**2、梯度爆炸**

**当我们选取的权重值较大时， 将大于1。当累乘这些项的时候，计算结果将呈指数级增长。**

**（1）激活函数：ReLU**

**（2）超出值的范围：无穷大**

**（3）对LR（学习速度）敏感：**

**不够小的LR->大的权重->更大的梯度**

**太小的LR ->没有进展**

**可能需要在培训期间大幅度更改LR**

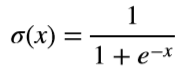
**三、稳定训练**

**四．激活函数**

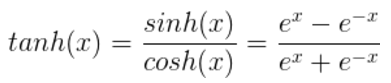
**1、rule**

****

**2、sigmoid**

****

**3、tanh**

****

**五、提高CPU和GPU的利用率**

**CPU：（1）多级缓存**

**在计算之前，需要先准备好数据，将数据读入缓存中。**

**改善时间和空间记忆的局部性**

**时态：重用数据，以便将它们保存在缓存中**

**空间：按顺序读取数据，以便预取数据**

**（2）多核心+并行**

**GPU：（1）并行多线程**

**（2）简单的缓存架构和较小的缓存大小**

**（3）简单的控制流**

* **权重衰减（Weight Decay）**

**1. 平方规范化（Squared Norm Regularization）**

**1.1 目的：为了降低过拟合的程度**

**1.2 硬约束（Hard Constraint）**

**1.2.1 形式：在||w||^2<=θ的限制条件下，最小化损失函数（min ℓ(w, b)）**

**1.2.2 通常不对b进行正则化（进行或不进行不会对训练造成过多影响）**

**1.2.3 使用一个更小的θ，意味着对模型进行更大的正则化**

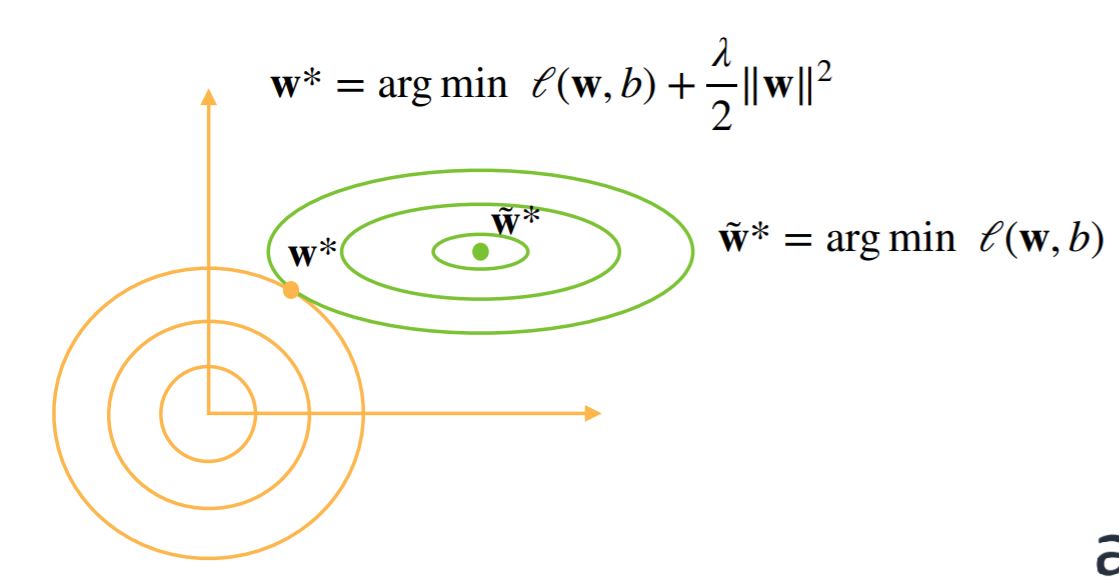
**1.3 软约束（Soft Constraint）**

**1.3.1 形式：最小化损失函数与正则化项之和（min (ℓ(w, b) +λ/2\*∥w∥^2)）**

**1.3.2 λ控制着正则化的程度**

**若λ=0，则表示无正则化；若λ→∞，则权重大小趋向于0**

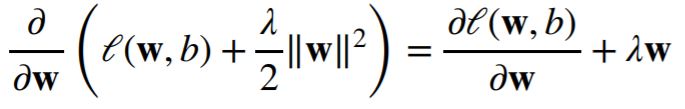
**1.4 图像表示**

****

**交点即为正则化后的最优解**

**2.更新规则（Update Rule）**

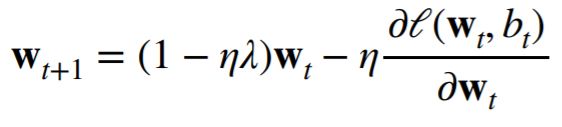
**2.1 计算含正则化项的损失函数的梯度**

****

**2.2 梯度下降中更新权重**

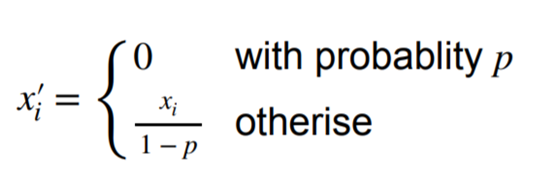
**W(t+1) = W(t) - η \* ∂loss/∂W**

**化简后即为**

****

* **丢弃法**

**丢弃法通过减少变量个数来简化模型，从而改善过拟合问题。丢弃法试图通过添加无偏差噪声来影响数据集，并希望经过噪声影响的数据集与原数据集的期望相同。因此它向数据集中每一个元素施加了丢弃概率：**

****

**元素被丢弃的概率为p，被保留的概率为1-p，元素的取值如图，计算得知E(xi’)=xi,即丢弃法不改变数据的期望值。然而，这种被丢弃的不确定性也使得结果具有不确定性，这不是我们所期望的，因此一般不使用丢弃法。**

* **例：**

**在神经网络里（多层感知机），我们可以随机（概率p）丢弃输入层或隐藏层中的节点，在现代神经网络中，我们所指的丢弃法，通常是对输入层或者隐藏层进行操作：**

**1. 随机选择一部分该层的输出作为丢弃元素**

**2. 把丢弃元素乘以0**

**3. 把非丢弃元素拉伸（按1-p拉伸）**