Machine Learning

题目：**中山大学**

学习小组 05 机器学习作业

姓　　名	方桂安，刘玥，周敏
学　　号	20354027,20354229,20354187
院　　系	智能工程学院
专　　业	智能科学与技术
指导教师	彭卫文 (副教授)

2021 年 11 月 14 日

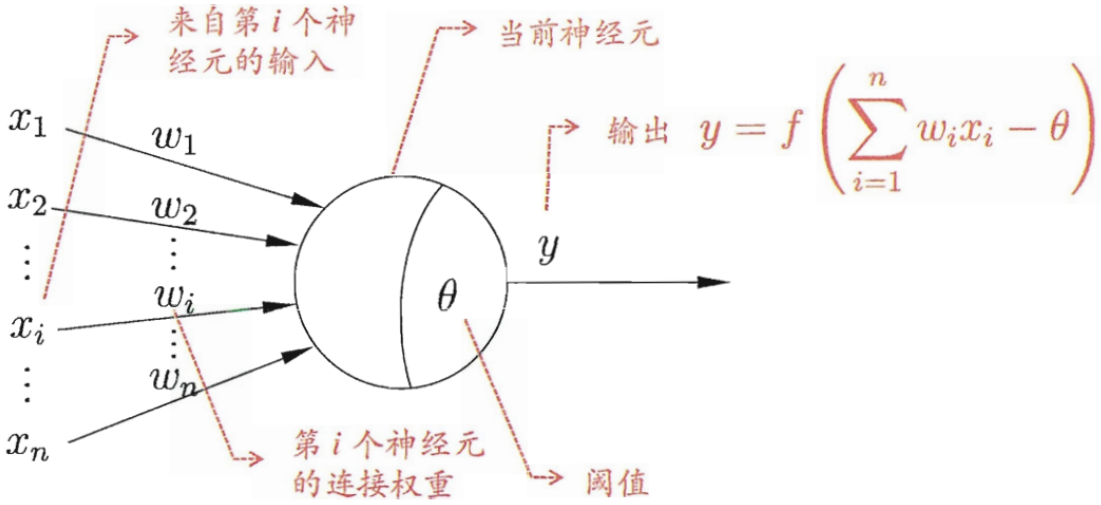
一、描述所使用的神经网络模型

1.1 神经元模型

1.1.1 神经元模型的定义

神经网络是由具有适应性的简单单元组成的广泛并行互联的网络，它的组织能够模拟生物神经系统对真实世界物体所作出的交互反应。神经网络中最基本的成分是神经元模型，即上述的“简单单元”。

1.1.2 M-P神经元模型



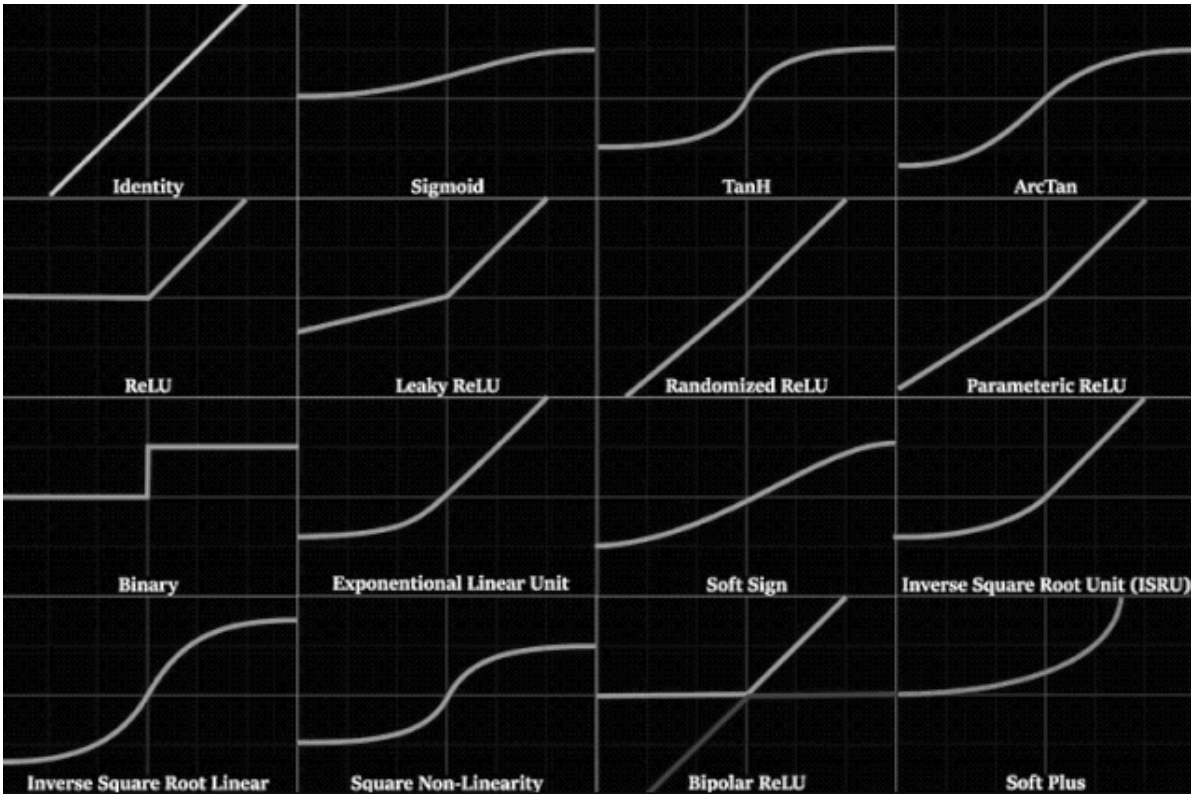
输入：来自其他n个神经元传递过来的输入信号

处理：输入信号通过带权重的连接进行传递，神经元接受到总输入值将其与神经元的阈值进行比较

输出：通过激活函数的处理以得到输出

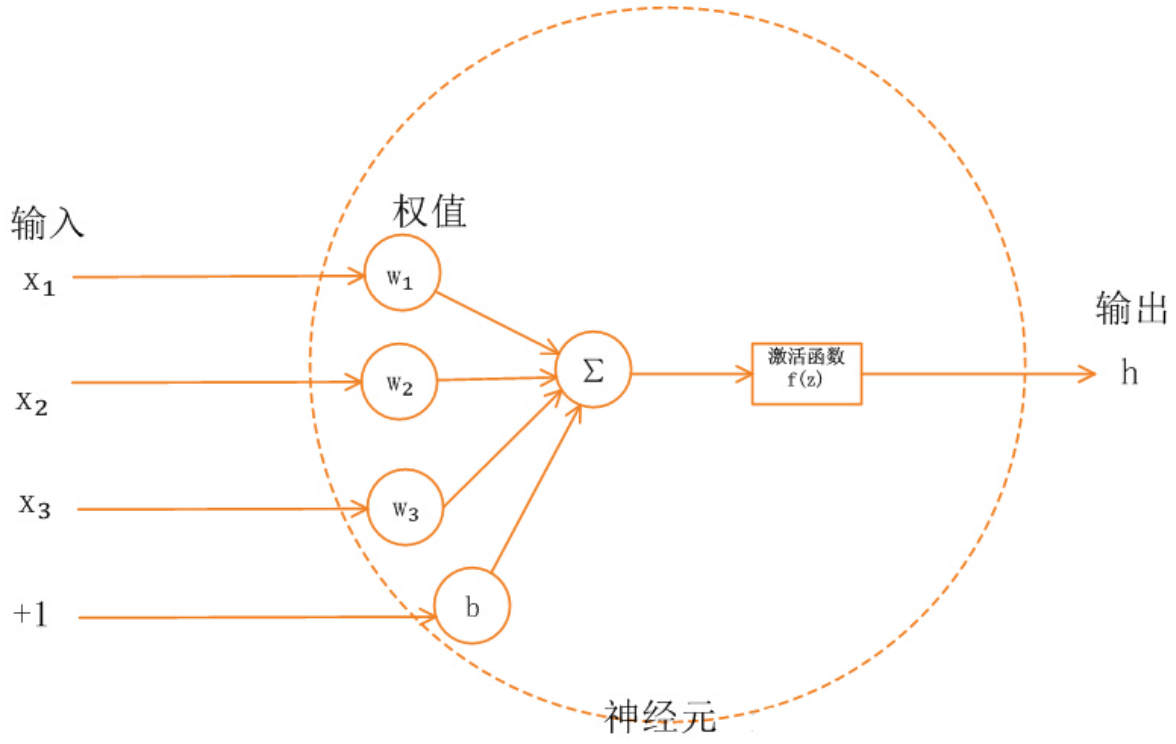
1.1.3 激活函数

激活函数	形式	导数形式
Sigmoid	$f(x)=\frac{1}{1+e^{-x}}$	$f'(x)=f(x)(1-f(x))$
Tanh	$f(x)=tanh(x)=\frac{e^x-e^{-x}}{e^x+e^{-x}}$	$f'(x)=1-(f(z))^2$
ReLU	$f(x)=max(0,x)=\begin{cases}0 & x\leq 0\\ x & x>0\end{cases}$	$f'(x)=\begin{cases}0 & x\leq 0\\ 1 & x>0\end{cases}$
Leaky ReLU	$f(x)=max(0.001x,x)=\begin{cases}0.001x & x\leq 0\\ x & x>0\end{cases}a$	$f(x)=max(0.001x,x)=\begin{cases}0.001 & x\leq 0\\ 1 & x>0\end{cases}$
PReLU	$f(x)=max(\alpha x,x)=\begin{cases}\alpha x & x\leq 0\\ x & x>0\end{cases}$	$f(x)=max(\alpha x,x)=\begin{cases}\alpha & x\leq 0\\ 1 & x>0\end{cases}$
RReLU	PReLU中的 α 随机取值	
ELU	$f(x)=\begin{cases}x & x\geq 0\\ \alpha(e^x-1) & x<0\end{cases}$	$f(x)=\begin{cases}1 & x\geq 0\\ \alpha e^x & x<0\end{cases}$
Maxout	$f(x)=max(w_1^Tx+b_1,w_2^Tx+b_2)$	$f(x)=max(w_1,w_2)$



1.1.3.1 激活函数的介绍

如下图所示，神经网络中的每个神经元节点接受上一层神经元的输出值作为本神经元的输入值，并将输入值传递给下一层，输入层神经元节点会将输入属性值直接传递给下一层（隐层或输出层）。在多层神经网络中，上层节点的输出和下层节点的输入之间具有一个函数关系，这个函数称为激活函数（又称激励函数）。



1.1.3.2 激活函数的用途

如果不用激励函数（相当于激励函数是f(x) = x），每一层节点的输入都是上层输出的线性函数，这样无论神经网络有多少层，输出都是输入的线性组合，这种情况就是最原始的感知机，网络的逼近能力就相当有限。当我们引入非线性函数作为激励函数，深层神经网络表达能力就更加强大，不再是输入的线性组合，几乎可以逼近任意函数。

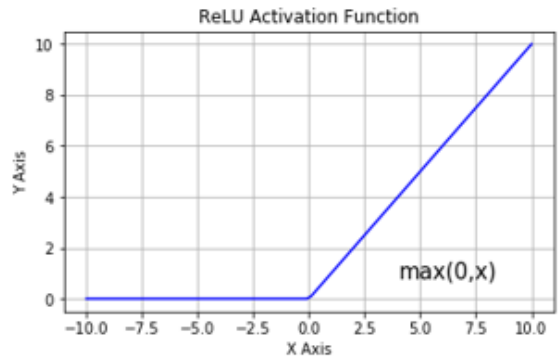
1.1.3.3 一些常见的激活函数及其性质

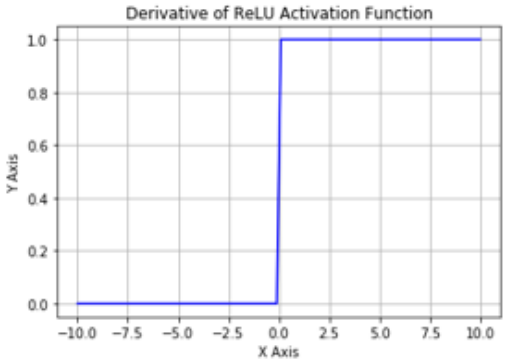
1.1.3.3.1 Relu激活函数

Relu函数的解析式：

$$f_{Relu}(x) = max(0, x)$$

Relu函数及其导数的图像如下图所示：





$$f_{ReLU}(x) = \max(0, x)$$

$$f'_{ReLU}(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$$

优点：

①计算效率较高

②兼具线性和非线性特性

缺点：

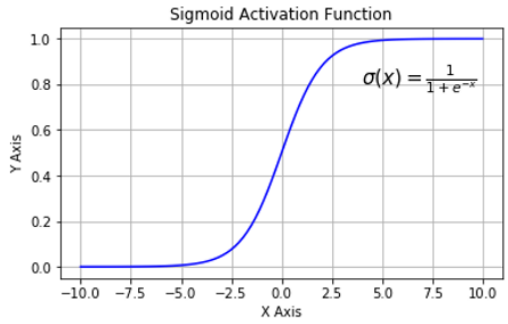
梯度消失问题：在x<0时，神经元保持非激活状态，且在反向传导（backward pass）中梯度为零

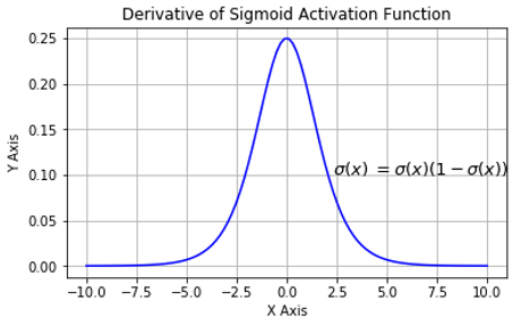
1.1.3.3.2 Sigmoid激活函数

Sigmoid 函数的解析式：

$$f_{Sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid函数及其导数的图像如下图所示：





$$f_{Sigmoid}(x) = \frac{1}{1+e^{-x}} \qquad f'_{Sigmoid}(x) = f_{Sigmoid}(x)\left(1 - f_{Sigmoid}(x)\right)$$

优点：

①梯度的“平滑性”

②输出在“0-1区间”

缺点：

①梯度消失问题：神经网络使用 Sigmoid 激活函数进行反向传播时，输出接近0或1的神经元其梯度趋近于0

②计算成本问题：涉及指数计算

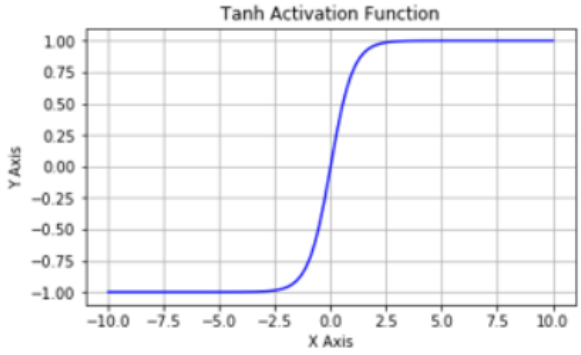
③不以零为中心：Sigmoid 输出不以零为中心

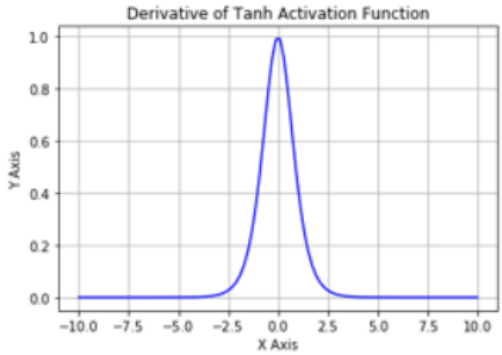
1.1.3.3.3 tanh 激活函数

tanh函数的解析式：

$$f_{tanh}(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

tanh函数及其导数的图像如下图所示：





$$f_{tanh}(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \qquad f'_{tanh}(x) = 1 - f_{tanh}(x)^2$$

优点：

①梯度的“平滑性”

②输出以零为中心

缺点：

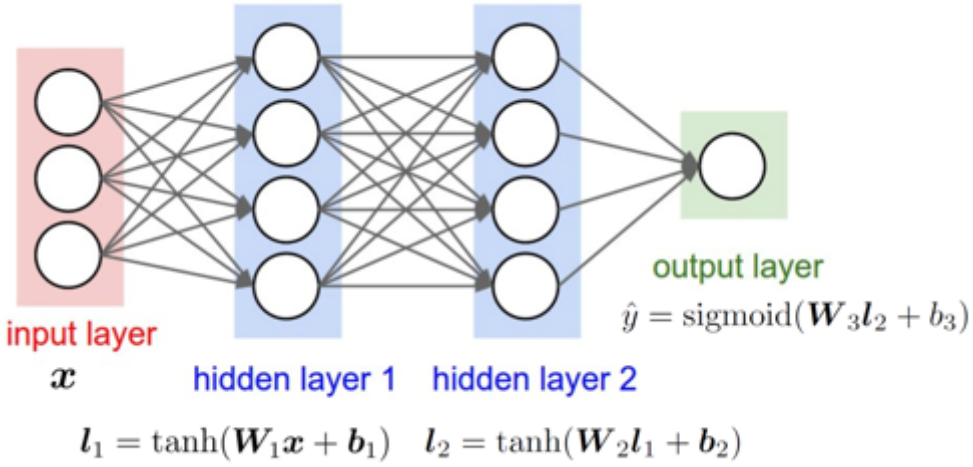
①梯度消失问题：神经网络使用 tanh 激活函数进行反向传播时，输出接近-1或1的神经元其梯度趋近于0

②计算成本问题：涉及指数计算

1.2 神经网络模型

1.2.1 神经网络模型的定义

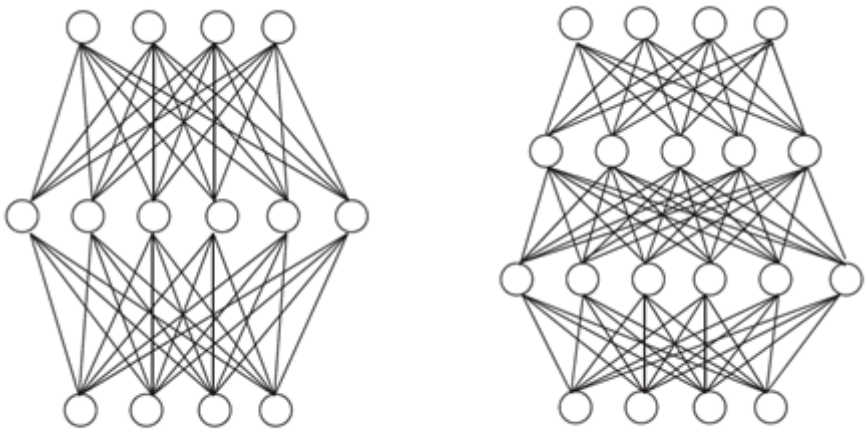
将若干神经元按一定的层次结构连接起来就得到了神经网络，可将神经网络视为包含了若干参数的数学模型，这个模型是由若干个函数相互（嵌套）代入而得。



1.2.2 多层前馈神经网络

1.2.2.1 定义

每层神经元与下一层神经元全互联，神经元之间不存在同层连接也不存在跨层连接。输入层接受外界输入，隐含层与输出层神经元对信号进行加工，最终结果由输出层神经元输出。根据训练数据来调整神经元之间的“连接权”以及每个功能神经元的“阈值”。



(a) 单隐层前馈网络　　(b) 双隐层前馈网络

1.2.2.2 模型训练

数据：

$$D = \{(x_1, y_1), (x_2, y_2), \ldots\ldots, (x_m, y_m)\}, x_i \in R^d, y_i \in R^l$$

模型：若干神经元按一定的层次结构连接起来，每层神经元与下一层神经元全互联，神经元之间不存在同层连接也不存在跨层连接，所形成的神经网络模型。

策略：

①平方损失（回归问题）

$$L(y_i, f(x_i)) = (y_i - f(x_i)^2), ERM$$

②交叉熵损失（二分类问题）

$$L(y_i, f(x_i)) = -y_i log(p(y_i = 1|x_i)) - (1 - y_i) log(p(y_i = 0|x_i))$$

算法：误差逆传播算法（error Back Propagation）

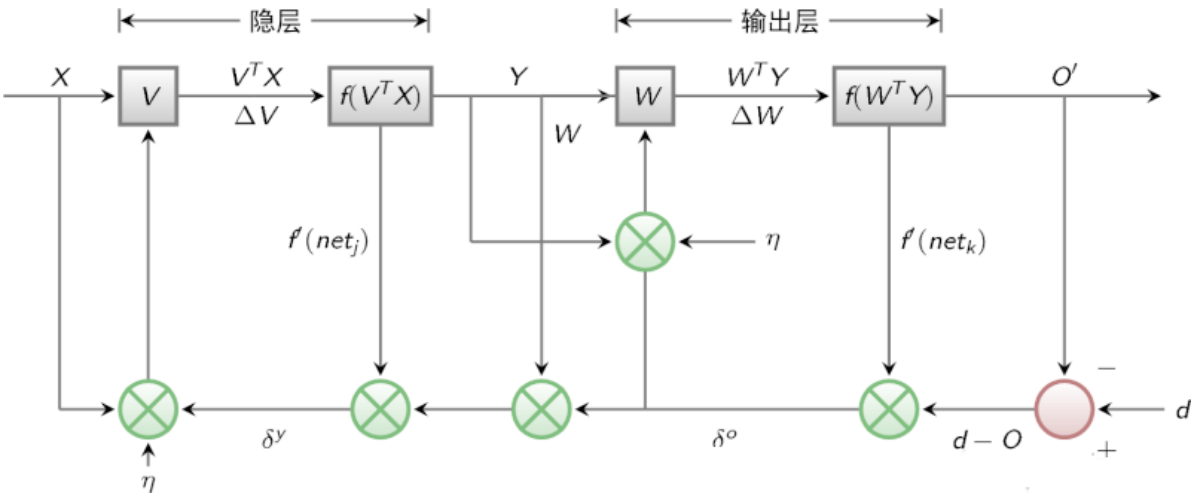
二、描述训练模型所使用的算法

2.1 误差逆传播算法

2.1.1 应用领域

反向传播算法应用较为广泛，从字面意思理解，与前向传播相互对应。在简单的神经网络中，反向传播算法，可以理解为最优化损失函数过程，求解每个参与运算的参数的梯度的方法。在前馈神经网中，反向传播从求解损失函数偏导过程中，步步向前求解每一层的参数梯度。在卷积神经网络中，反向传播可以求解全连接层的参数梯度。在循环神经网络中，反向传播算法可以求解每一个时刻t或者状态t的参数梯度（在RNN\LSTM\GRU中，反向传播更多是BPTT）。如今对于BP的理解，认为是在优化损失函数或者目标函数过程中，求解参与运算的参数的梯度方法，是一种比较普遍的说法。

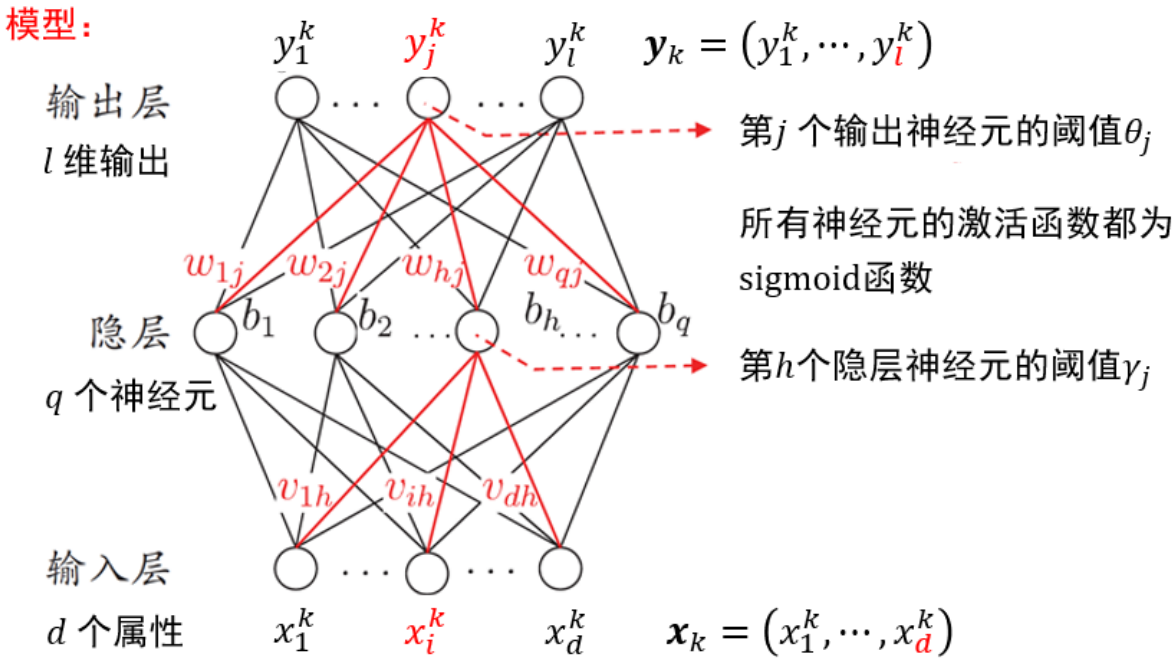
2.1.2 网络结构



1. 正向传播求损失，反向传播回传误差
2. 根据误差信号修正每层的权重
3. f是激活函数；f(netj)是隐层的输出； f(netk）是输出层的输出O; d是target。

2.1.2 基本参数结构

为了方便讨论，我们以一个隐层的神经网络结构进行推导。多隐层的神经网络推导思想与此类似，可推广。如下图为一个神经网络结构。



2.1.2.1参数简述

输入参数：$x_1^k,\ldots,x_i^k,\ldots,x_d^k$

输入层到第一隐层第h个神经元的权重：$v_{1h},\ldots,v_{ih},\ldots,v_{dh}$

第一层第h个神经元输入：$\alpha_h=\sum_{i=1}^{d}v_{ih}x_i^k$

第一隐层阈值：$\gamma_1,\ldots,\gamma_h,\ldots,\gamma_q$

第一隐层第h个输出：$b_h=f_{sigmoid}(\alpha_h-\gamma_h)$

第一隐层到第j个输出神经元的权重：$w_{1j},\ldots,w_{hj},\ldots,w_{qj}$

第j个输出神经元的输入：$\beta_j=\sum_{h=1}^{q}w_{hj}b_h$

输出层阈值：$\theta_1,\ldots,\theta_j,\ldots,\theta_l$

输出值：$y_j^k=f_{sigmoid}(\alpha_h-\gamma_h)$

所以前向传播计算误差为：

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

2.1.3 参数调整策略

BP算法的核心思想：使用梯度下降来搜索可能的权向量的假设空间，以找到最佳的拟合样例的权向量。具体而言，即利用损失函数，每次向损失函数负梯度方向移动，直到损失函数取得最小值。或者说，反向传播算法，是根据损失函数，求出损失函数关于每一层的权值及偏置项的偏导数，也称为梯度，用该值更新初始的权值和偏置项，一直更新到损失函数取得最小值或是设置的迭代次数完成为止。以此来计算神经网络中的最佳的参数。

$$\text{损失函数：}E_k=\frac{1}{2}\sum_{j=1}^{l}\left(y_j^k-\hat{y}_j^k\right)^2$$

2.1.3.1 计算准备

$$\frac{\partial E_k}{\partial \hat{y_j^k}} = (y_j^k - \hat{y_j^k})(-1)$$

$$f_{sigmoid}(x)^{(1)} = f_{sigmoid}(x)(1 - f_{sigmoid}(x))$$

$$\eta：学习率$$

下面，我们讲分别讨论每个参数的更新：

2.1.3.2 w更新

$$更新公式：w_{hj} = w_{hj} + \triangle w_{hj}$$

下面对$\triangle w_{hj}$进行讨论:

$$\begin{aligned}\triangle w_{hj} =& -\eta \frac{\partial E_k}{\partial w_{hj}} \\ =& -\eta(\frac{\partial E_k}{\partial \hat{y_j^k}} \cdot \frac{\partial \hat{y_j^k}}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}}) \\ =& -\eta(y_j^k - \hat{y_j^k})(-1) \cdot \hat{y_j^k}(1 - \hat{y_j^k}) \cdot b_h \\ 令g_j =& (y_j^k - \hat{y_j^k}) \cdot \hat{y_j^k}(1 - \hat{y_j^k})\end{aligned}$$

$$最终可得 \triangle w_{hj} = \eta g_j b_h$$

2.1.3.3 θ更新

$$更新公式：\theta_j = \theta_j + \triangle \theta_j$$

下面对$\triangle \theta_j$进行讨论：

$$\begin{aligned}\triangle \theta_j =& -\eta \frac{\partial E_k}{\partial \theta_j} \\ =& -\eta(\frac{\partial E_k}{\partial \hat{y_j^k}} \cdot \frac{\partial \hat{y_j^k}}{\partial \theta_j}) \\ =& -\eta(y_j^k - \hat{y_j^k})(-1) \cdot \hat{y_j^k}(1 - \hat{y_j^k}) \cdot (-1) \\ =& -\eta g_j\end{aligned}$$

2.1.3.4 v更新

$$更新公式：v_{ih} = v_{ih} + \triangle v_{ih}$$

下面对$\triangle v_{ih}$进行讨论：

$$
\begin{aligned}
\triangle v_{ih} &= -\eta \frac{\partial E_k}{\partial v_{ih}} \\
&= -\eta \frac{\partial E_k}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h} \cdot \frac{\partial \alpha_h}{\partial v_{ih}} \\
&= -\eta (\sum_{j=1}^{l} \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h}) \cdot b_h(1-b_h) \cdot x_i^k \\
&= \eta \sum_{j=1}^{l} g_j w_{hj} b_h (1-b_h) x_i^k
\end{aligned}
$$

2.1.3.5 γ更新

更新公式：$\gamma_h = \gamma_h + \triangle \gamma_h$

下面对$\triangle \gamma_h$进行讨论：

$$
\begin{aligned}
\triangle \gamma_h &= -\eta \frac{\partial E_k}{\partial \gamma_h} \\
&= -\eta \frac{\partial E_k}{\partial b_h} \cdot \frac{\partial b_h}{\partial \gamma_h} \\
&= -\eta (\sum_{j=1}^{l} \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h}) \cdot b_h(1-b_h) \cdot (-1) \\
&= \eta \sum_{j=1}^{l} g_j w_{hj} b_h (1-b_h)(-1)
\end{aligned}
$$

2.1.3 算法伪代码

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

三、描述模型超参数确定的过程，分析模型训练结果

3.1 超参数的概念

大部分机器学习算法都需要花费大量时间去训练，而在训练之前需要提前配置一些变量。这些变量对训练结果影响很大，但没有对任何数据集都适用的一组变量，需要根据具体应用具体配置，这些需要配置的变量称之为超参数（hyperparameters）。区分超参数和模型参数最大的一点就是是否通过数据来进行调整，模型参数通常是有数据来驱动调整，超参数则不需要数据来驱动，而是在训练前或者训练中人为的进行调整的参数。例如卷积核的具体核参数就是指模型参数，这是由数据驱动的。而学习率则是人为来进行调整的超参数。这里需要注意的是，通常情况下卷积核数量、卷积核尺寸这些也是超参数，注意与卷积核的核参数区分。

3.2 神经网络包含的超参数

3.2.1 超参数种类

通常可以将超参数分为三类：网络参数、优化参数、正则化参数。

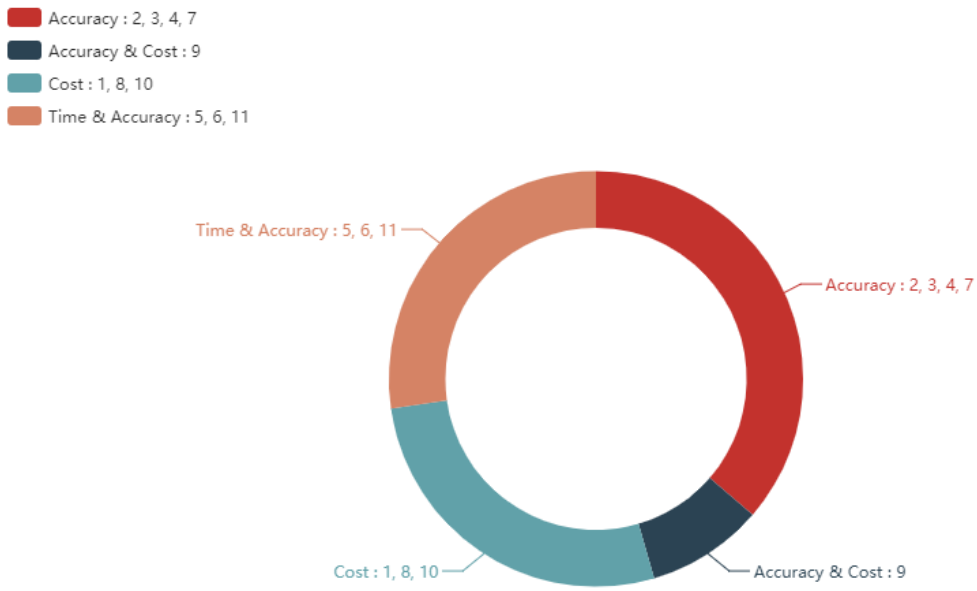
网络参数：可指网络层与层之间的交互方式（相加、相乘或者串接等）、卷积核数量和卷积核尺寸、网络层数（也称深度）和激活函数等。

优化参数：一般指学习率（learning rate）、批样本数量（batch size）、不同优化器的参数以及部分损失函数的可调参数。

正则化：权重衰减系数，丢弃法比率（dropout）

神经网络包含的超参数具体为以下十一个：

1. 学习率 η
2. 正则化参数 λ
3. 神经网络的层数 L
4. 每一个隐层中神经元的个数 j
5. 学习的回合数Epoch
6. 小批量数据 minibatch 的大小
7. 输出神经元的编码方式
8. 代价函数的选择
9. 权重初始化的方法
10. 神经元激活函数的种类
11. 参加训练模型数据的规模



在上图中可以看到超参数 2，3，4， 7 主要影响的时神经网络的分类正确率；9 主要影响代价函数曲线下降速度，同时有时也会影响正确率；1，8，10 主要影响学习速度，这点主要体现在训练数据代价函数曲线的下降速度上；5，6，11 主要影响模型分类正确率和训练用总体时间。这上面所提到的时某个超参数对于神经网络想到的首要影响，并不代表着该超参数只影响学习速度或者正确率。

3.2.2 超参数重要性顺序

- 首先， 学习率，损失函数上的可调参数。在网络参数、优化参数、正则化参数中最重要的超参数可能就是学习率了。学习率直接控制着训练中网络梯度更新的量级，直接影响着模型的有效容限能力；损失函数上的可调参数，这些参数通常情况下需要结合实际的损失函数来调整，大部分情况下这些参数也能很直接的影响到模型的的有效容限能力。这些损失一般可分成三类，第一类辅助损失结合常见的损失函数，起到辅助优化特征表达的作用。例如度量学习中的Center loss，通常结合交叉熵损失伴随一个权重完成一些特定的任务。这种情况下一般建议辅助损失值不高于或者不低于交叉熵损失值的两个数量级；第二类，多任务模型的多个损失函数，每个损失函数之间或独立或相关，用于各自任务，这种情况取决于任务之间本身的相关性，目前笔者并没有一个普适的经验由于提供参考；第三类，独立损失函数，这类损失通常会在特定的任务有显著性的效果。例如RetinaNet中的focal loss，其中的参数γ，α，对最终的效果会产生较大的影响。这类损失通常论文中会给出特定的建议值。
- 其次，批样本数量，动量优化器（Gradient Descent with Momentum）的动量参数β。批样本决定了数量梯度下降的方向。过小的批数量，极端情况下，例如batch size为1，即每个样本都去修正一次梯度方向，样本之间的差异越大越难以收敛。若网络中存在批归一化（batchnorm），batch size过小则更难以收敛，甚至垮掉。这是因为数据样本越少，统计量越不具有代表性，噪声也相应的增加。而过大的batch size，会使得梯度方向基本稳定，容易陷入局部最优解，降低精度。一般参考范围会取在[1:1024]之间，当然这个不是绝对的，需要结合具体场景和样本情况；动量衰减参数β是计算梯度的指数加权平均数，并利用该值来更新参数，设置为 0.9 是一个常见且效果不错的选择；
- 最后，Adam优化器的超参数、权重衰减系数、丢弃法比率（dropout）和网络参数。在这里说明下，这些参数重要性放在最后并不等价于这些参数不重要。而是表示这些参数在大部分实践中不建议过多尝试，例如Adam优化器中的β1，β2，ϵ，常设为 0.9、0.999、10−8就会有不错的表现。权重衰减系数通常会有个建议值，例如0.0005 ，使用建议值即可，不必过多尝试。dropout通常会在全连接层之间使用防止过拟合，建议比率控制在[0.2,0.5]之间。使用dropout时需要特别注意两点：一、在RNN中，如果直接放在memory cell中,循环会放大噪声，扰乱学习。一般会建议放在输入和输出层；二、不建议dropout后直接跟上batchnorm，dropout很可能影响batchnorm计算统计量，导致方差偏移，这种情况下会使得推理阶段出现模型完全垮掉的极端情况；网络参数通常也属于超参数的范围内，通常情况下增加网络层数能增加模型的容限能力，但模型真正有效的容限能力还和样本数量和质量、层之间的关系等有关，所以一般情况下会选择先固定网络层数，调优到一定阶段或者有大量的硬件资源支持可以在网络深度上进行进一步调整。

3.3 模型超参数确定

3.3.1 超参数调优的原因

本质上，这是模型优化寻找最优解和正则项之间的关系。网络模型优化调整的目的是为了寻找到全局最优解（或者相比更好的局部最优解），而正则项又希望模型尽量拟合到最优。两者通常情况下，存在一定的对立，但两者的目标是一致的，即最小化期望风险。模型优化希望最小化经验风险，而容易陷入过拟合，正则项用来约束模型复杂度。所以如何平衡两者之间的关系，得到最优或者较优的解就是超参数调整优化的目的。

3.3.2 模型超参数的确定

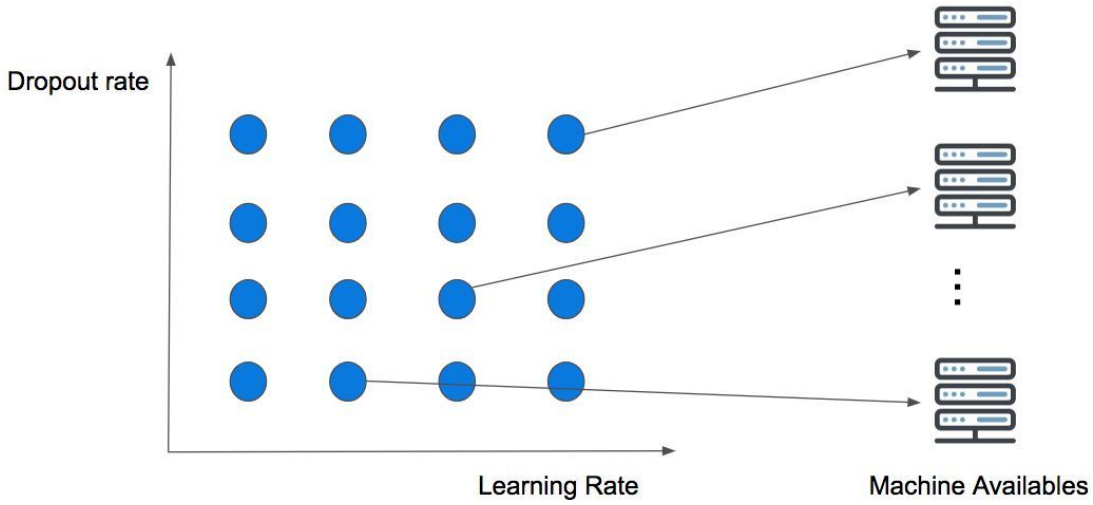
四种主流超参数调优技术：

1. 传统或手动调参
2. 网格搜索
3. 随机搜索
4. 贝叶斯搜索

在传统的调优中，我们通过手动检查随机超参数集来训练算法，并选择最适合我们目标的参数集。但这种方法不能保证得到最佳的参数组合，反复试验会消耗更多的时间。

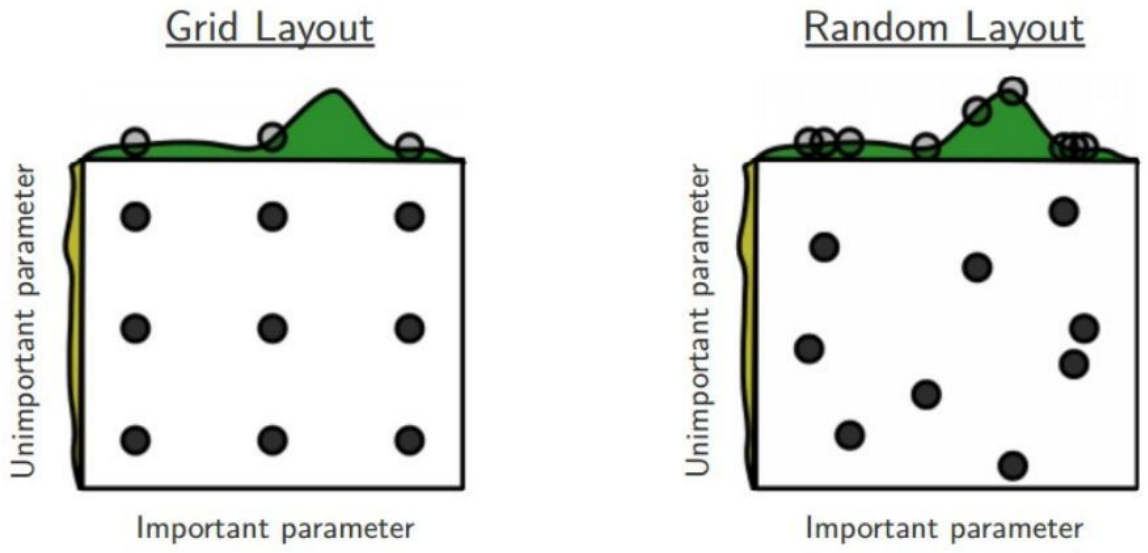
3.3.2.1 网格搜索

- 网格搜索是一种基本的超参数调整技术。它类似于手动调优，为网格中指定的所有给定超参数值的每个排列建立模型，并评估和选择最佳模型。由于它尝试每一种超参数组合，并根据交叉验证分数选择最佳组合，这使得 GridsearchCV 极其缓慢。
- 这种启发式的搜索算法对超参数搜索算法，被称之为网格搜索。(如果人工处理所有可能的超参数组合，通常的办法是，根据超参数的维度，列成相应的表格，比如说k的取值有[2，3，4，5，6，7，8]，另一个系数比如λ取值有[0.01,0.03,0.1,0.3]等，这样就可以列出一个二维表格，组合出7*4种可能性的超参数组合，再对每一个格子中具体的超参数组合，通过交叉验证的方式进行模型性能的评估，然后通过验证性能的比较，最终筛选出最佳的超参数数据组合)
- 网格搜索采用交叉验证的方法，来寻找更好的超参数组合的过程非常耗时，由于各个新模型在执行交叉验证的过程中是相互独立的，那么我们可以充分利用多核处理器甚至是分布式的计算资源来从事并行搜索，从而成倍的节省运算时间。



3.3.2.2 随机搜索

使用随机搜索代替网格搜索的动机是，在许多情况下，所有的超参数可能并非同等重要。随机搜索从超参数空间中随机选择参数组合，参数按 n_iter 给定的迭代次数进行选择。随机搜索已经被实践证明比网格搜索得到的结果更好，但随机搜索的问题是它不能保证给出最佳的参数组合。



3.3.2.3 贝叶斯优化

贝叶斯优化属于一类被称为*sequential model-based optimization*(SMBO)的优化算法。这些算法使用先前对损失 f 的观测，来确定下一个(最佳)点来取样 f。该算法大致可以概括如下：

1. 使用先前计算过的点 X1: n，计算损失 f 的后验期望值。
2. 在一个新的点 Xnew取样损失 f ，它最大化了 f 的期望的某些效用函数。该函数指定 f 域的哪些区域是最适合采样的。

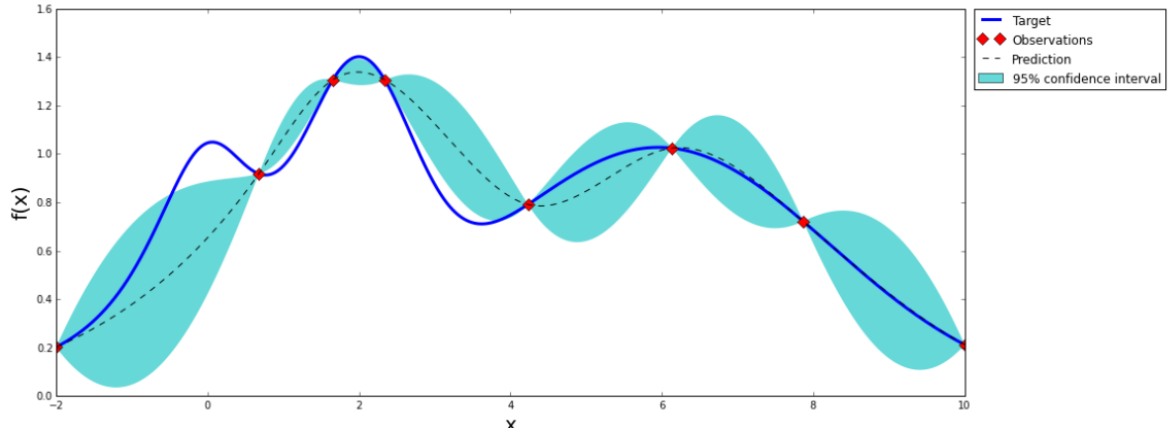
重复这些步骤，直到达到某种收敛准则。

3.3.2.3.1 高斯过程

在贝叶斯调参过程中，假设一组超参数组合是X=x1,x2,...,xn(xn表示某一个超参数的值)，而这组超参数与最后我们需要优化的损失函数存在一个函数关系，最终的评估结果为Y，通过什么样的X可以取得最优的Y，我们假设是f(X)， Y=F(X)

而目前机器学习其实是一个黑盒子(black box),即我们只知道input和output，所以上面的函数f(x)很难确定。所以我们需要将注意力转移到一个我们可以解决的函数上去.

于是可以假设这个寻找最优化参数的过程是一个高斯过程。高斯过程有个特点，就是当随机遍历一定的数据点并拿到结果之后，可以大致绘制出整个数据的分布曲线。



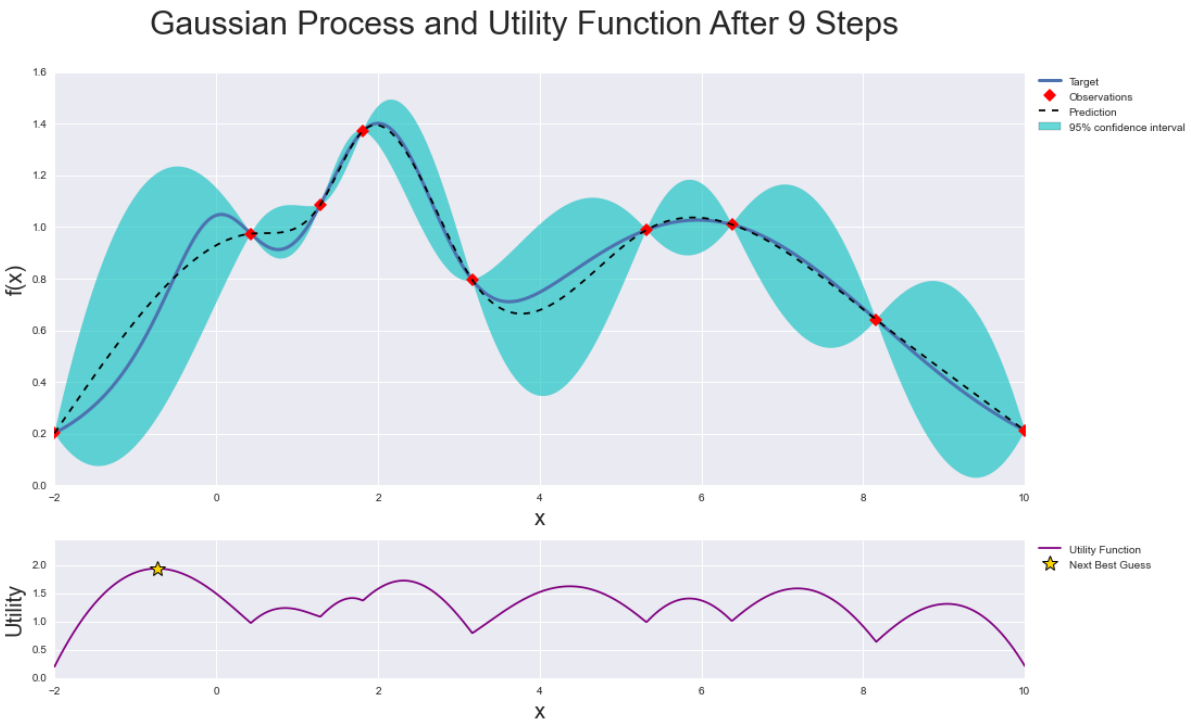
3.3.2.3.2 贝叶斯优化理论

还是这张图，把横轴看作是参数组合X，纵轴看作是这个参数的结果Y。可以通过已经构建的曲线，找到曲线上升的方向，从而在这个方向上继续探索，这样就可以大概率拿到更好的结果。在生活的轨迹上，如果找到一条明确通往幸福的路，可以继续向前探索，因为大概率可以成功，但也许也有会错过更好的机会，陷入局部最优解。请看上图中的五角星，如果我们处于它的位置，继续向上走会迎来一个高峰，但是如果后退，在下降一段时间之后可能会迎来更高的波峰，你该如何选择。

于是，在参数的探索中要掌握一个平衡：

开发：在明确的曲线上扬方向继续走，大概率获得更好的结果，但是容易陷入局部最优。

探索：除了在曲线上扬的方向，在其它的区域也不忘寻找



3.4 结果分析

3.4.1 MAPE

平均绝对百分比误差（Mean Absolute Percentage Error）

$$MAPE = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{\hat{y}_i - y_i}{y_i}\right|$$

范围[0,+∞)，MAPE 为0%表示完美模型，MAPE 大于 100 %则表示劣质模型。

注意：当真实值有数据等于0时，存在分母0除问题，该公式不可用！

3.4.2 调参前结果及分析

3.4.2.1 代码

```
mlp = MLPRegressor(hidden_layer_sizes=(100), activation='relu', solver='adam',
alpha=0.0001, batch_size='auto', learning_rate='constant',
learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True,
random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9,
nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1,
beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000) #所
有参数默认
mlp.fit(X_std, Y)
MAPE = -1*cross_val_score(mlp, X_std, Y,
cv=rkf,scoring='neg_mean_absolute_percentage_error').mean()
print('MAPE:',MAPE)
```

3.4.2.2 结果及分析

首先我们使用`sklearn.neural_network`.MLPRegressor中的所有默认参数设置来训练模型，五次五折交叉验证的平均MAPE为：0.2036462204885882

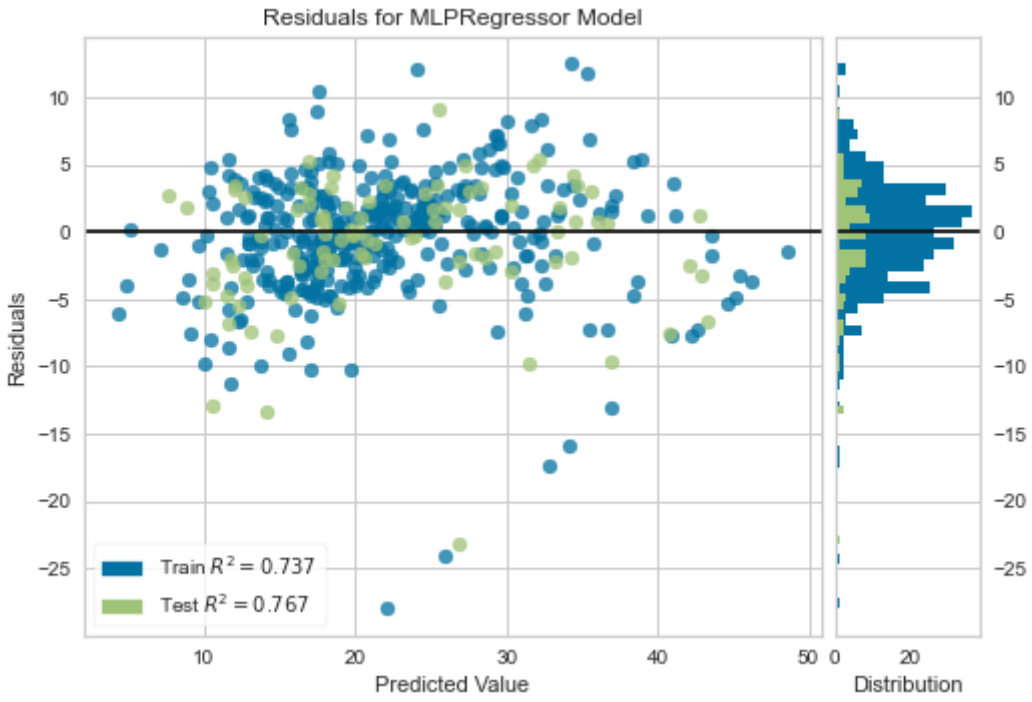


Fig.1 调参前残差图

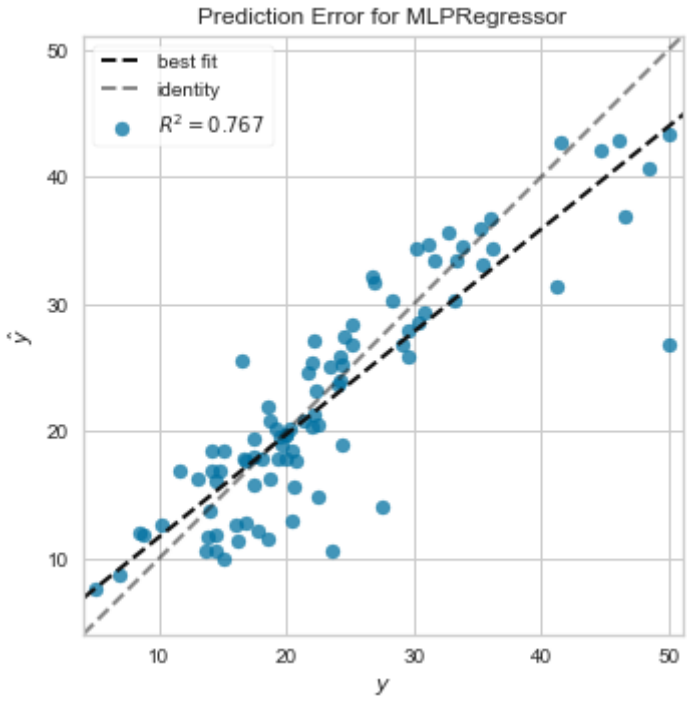


Fig.2 调参前预测误差图

残差是因变量未被自变量解释的部分，线性模型要求残差服从独立同分布，且分布类型为正态分布。通过一系列方法判断残差是否符合这一要求，可以达到检验模型是否符合相应假设的目的。从上图可以看出，我们的训练集和测试集的R2在0.75左右，说明我们的模型训练结果具有一定的可信度，但并不理想。下面我们进行调参，尝试提高准确率。

3.4.3 网格搜索调参

接下来我们手动调试模型，将各个超参数逐一修改并查看MAPE的变化结果，最终得出 '**hidden_layer_sizes**'，'**activation**'，'**solver**'，'**alpha**'，'**learning_rate**'

这五个对结果影响较大的参数。

最后我们利用GridSearchCV结合一些"经验结论"来搜索出最优的超参数。

Hyperparameter	Typical value
# input neurons	One per input feature (e.g., 28 x 28 = 784 for MNIST)
# hidden layers	Depends on the problem, but typically 1 to 5
# neurons per hidden layer	Depends on the problem, but typically 10 to 100
# output neurons	1 per prediction dimension
Hidden activation	ReLU (or SELU, see Chapter 11)
Output activation	None, or ReLU/softplus (if positive outputs) or logistic/tanh (if bounded outputs)
Loss function	MSE or MAE/Huber (if outliers)

3.4.3.1 代码

```
# 超参数调优
from sklearn.model_selection import GridSearchCV
parameters = {'hidden_layer_sizes': [(10,10,10,10,10),(20,20,20,20,20),
(30,30,30,30,30),(40,40,40,40,40),(50,50,50,50,50),(60,60,60,60,60),
(70,70,70,70,70),(80,80,80,80,80),(90,90,90,90,90),(100,100,100,100,100)],
                'activation': ['identity', 'logistic','tanh', 'relu'],
                'solver': ['adam','lbgfs','sgd'],
                'alpha': [0.0001, 0.001, 0.01, 0.1, 1,10,100],
                'learning_rate': ['constant', 'invscaling', 'adaptive']}
grid = GridSearchCV(mlp, parameters, cv=rkf,
scoring='neg_mean_absolute_percentage_error',n_jobs=-1)
grid.fit(X_std, Y)
print('最优参数：',grid.best_params_)
print('最优模型得分：',grid.best_score_)
```

3.4.3.2 结果及分析

- 最优参数： {'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (100, 100, 100, 100, 100), 'learning_rate': 'adaptive', 'solver': 'sgd'}
- 最优模型得分： 0.10669120458358475

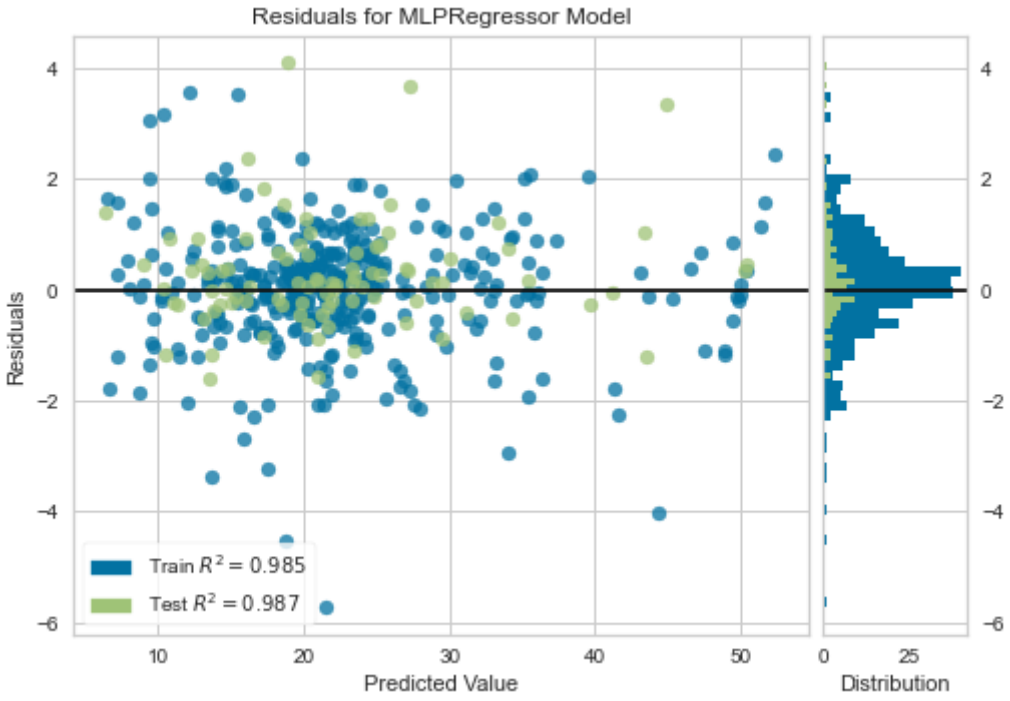


Fig.3 调参后残差图

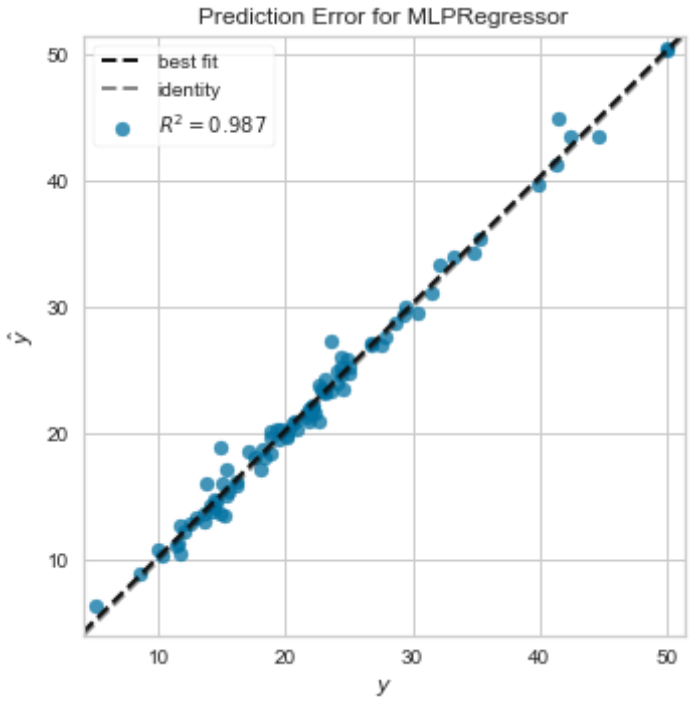


Fig.4 调参后预测误差图

经过网格搜索最优参数后，我们的模型得到大幅度提升。从上可见，我们的训练和测试的R^2均在0.98以上，说明模型对训练集的拟合效果和泛化能力都很强。

四、总结模型训练过程中的收获

4.1 神经网络的训练过程

简单的神经网络的训练过程包括以下几个步骤：

1. 定义一个包含多个可学习参数（权重）的神经网络；
2. 对输入的数据集进行迭代计算；
3. 通过多层网络结构来处理输入数据；
4. 计算损失值（输出值与目标值的差值）；
5. 反向传播梯度到神经网络的参数中；
6. 根据更新规则来更新网络中的权重值。

4.2 确定超参数

其中，如何定义一个包含多个可学习参数的神经网络（即如何确定模型的超参数）是重点，会影响神经网络学习速度和最后结果。我们确定超参数的步骤如下：

①我们根据经验结论手动调试模型，将各个超参数逐一修改并查看MAPE的变化结果，最终得出'**hidden_layer_sizes**'，'**activation**'，'**solver**'，'**alpha**'，

'**learning_rate**'这五个对结果影响较大的参数。

②搜集“经验总结”的资料后，我们用网格搜索法对下列超参数进行排列组合，得到10*4*3*7*3=2520种超参数的排列组合方式。

```
parameters = {'hidden_layer_sizes': [(10,10,10,10,10),(20,20,20,20,20),
(30,30,30,30,30),(40,40,40,40,40),(50,50,50,50,50),(60,60,60,60,60),
(70,70,70,70,70),(80,80,80,80,80),(90,90,90,90,90),(100,100,100,100,100)],
                'activation': ['identity', 'logistic','tanh', 'relu'],
                'solver': ['adam','lbgfs','sgd'],
                'alpha': [0.0001, 0.001, 0.01, 0.1, 1,10,100],
                'learning_rate': ['constant', 'invscaling', 'adaptive']}
```

③使用五次五折将数据划分为25份，把上述2520种超参数的组合都跑一遍数据（计算神经网络中的最佳的参数用的是误差逆传播算法），每一个组合都会得到25个MAPE值，取平均；之后2520份MAPE中的最小值对应的超参数组合即为我们选定的最优超参数组合。

4.3 防止过拟合的方法

在机器学习模型（特别是深度学习模型）的训练过程中，模型是非常容易过拟合的。深度学习模型在不断的训练过程中训练误差会逐渐降低，但测试误差的走势则不一定。

①正则化方法。正则化方法包括L0正则、L1正则和L2正则，而正则一般是在目标函数之后加上对于的范数。但是在机器学习中一般使用L2正则。

②数据增强（Data augmentation），增大数据的训练量；还有一个原因就是我们用于训练的数据量太小导致的，训练数据占总数据的比例过小。

③重新清洗数据，导致过拟合的一个原因也有可能是数据不纯导致的，如果出现了过拟合就需要我们重新清洗数据。

④提前终止法（Early stopping），对模型进行训练的过程即是对模型的参数进行学习更新的过程，这个参数学习的过程往往会用到一些迭代方法，如梯度下降（Gradient descent）学习算法。提前终止法便是一种迭代次数截断的方法来防止过拟合的方法，即在模型对训练数据集迭代收敛之前停止迭代来防止过拟合。

⑤丢弃法（Dropout）。这个方法在神经网络里面很常用。丢弃法是ImageNet中提出的一种方法，通俗一点讲就是丢弃法在训练的时候让神经元以一定的概率不工作。具体看下图：