目录

1 前言

大量数据代表了价值。数据背后通常隐含着客观规律，如果数据量足够大的话，其规律是可以被认知和学习的，其催生了机器学习的研究方向，研究如何用数据进行建模与变现。然而，由于数据量极大，而且所涉及的算法会很复杂，通常不可能进行人为的计算，即使是用计算机进行计算，也对计算机的处理速度，内存，储存空间提出了一定的要求。另一方面，如若要进行机器学习，除了计算机硬件的要求之外，还需要软件与算法的支持，其中，算法是机器学习的核心。历史发展来看，计算机硬件，用于机器学习的软件与算法的发展是相辅相成的。

在 20 世纪 40 年代，人们开始研究人工智能，由于生物学的发展，人们模仿人类的神经元运作而提出了神经网络的原型：M-P 神经元模型，并提出了激活函数的概念。在 20 世纪 50 年代到 60 年代，感知器算法、梯度下降法、最小二乘法等求解算法面世，而且提出了感知器，并开始应用在文字、语音、信号等领域。在 20 世纪 60 年代到 70 年代，神经网络算法因感知器的缺陷而衰落。在 70 年代到 80 年代，神经网络的种类变得丰富起来，涌现出 BP 神经网络，RBF 神经网络等各种网络，并提出了深度学习的概念与卷积神经网络（CNN）和循环神经网络（RNN）的结构。90 年代后，一些有别于神经网络的算法面世，如 SVM，决策树，boosting 与随机森林等方法，从不同的角度对机器学习算法进行丰富。在 2006 年，Hinton 提出了解决深度学习中梯度消失问题的解决方法之后，深度学习开始爆发。2012 年，ReLU 激活函数的提出，进一步抑制了梯度消失的问题，并且深度学习在语音和图像方面开始有惊人的表现。2012 年，在 ImageNet 图像识别比赛上，AlexNet 通过构建一定深度的 CNN 夺得冠军，其性能彻底击败了 SVM。需注意的是，AlexNet 首次使用了 ReLU 激活函数，Dropout 防止过拟合方法，以及 GPU 加速。之后，在 AlexNet 的结构上做优化，又提出了其他更强大的模型，如 VGGNet，Inception 系列，ResNet 等。强化学习和迁移学习的提出，进一步增强了模型的性能。

本论文基于 kaggle（全球数据科学平台）的花苗分类竞赛 (Plant Seedlings Classification[1]) 中的数据集，探究传统机器学习算法（SVM，决策树，随机森林与 boosting 等）、深度学习算法（AlexNet，VGGNet，InceptionV3）的原理与性能，并对其尝试做优化与结合（如 AlexNet+SVM 等）。

2 花苗分类问题

2.1 问题简述

该问题来自于 kaggle 的 Plant Seedlings Classification 竞赛。给定的 13 类花苗（有枝干，树叶，无花）的五千余张彩色图片用于训练、构建模型。

[1] https://www.kaggle.com/c/plant-seedlings-classification

2.2 数据样式

2.3 问题分析

3 数据预处理

3.1 掩模构建

对于花苗图像，我们可以看到，花苗的背景通常为黄土、砂砾或塑料箱等，而绿色的花苗则显得非常好辨认。而且我们面对的花苗是绿色的，因而考虑设置一个 hsv 范围，将绿色的部分从图像中剥离出来。于是我们首先将花苗图像进行颜色空间的转换，从 rgb 颜色空间转化为 hsv 颜色空间，之后设定 hsv 颜色空间为

!!!hsv 颜色空间

，从而筛选去绿色部分。如图所示

!!! 原图，过滤出来的绿色图像，mask

3.2 形态学去噪

对于用掩模处理后的花苗二值图像，考虑到在花苗所在盆栽可能会有一些小草，通过掩模处理后会有噪声。因而考虑用形态学方法去噪。

!!! 带噪声的图片

对于一个二值图像，比较常用的去噪方法是形态学去噪，而这通常涉及两种形态学转换，分别为腐蚀和膨胀，其涉及的原理较简单。对于腐蚀，先定义一个窗口，窗口将沿着图像滑动，以遍历整个图像。滑动过程中，窗口内所有像素不全为 1 时，则令窗口中的所有像素等于 0；若窗口内所有像素全为 1, 时，则不做操作。选用一个合适尺寸的窗口，对于腐蚀之后的图片，其白噪声点可以消除，但也会对物体的边缘进行腐蚀。膨胀则与腐蚀相反，区别在于滑动过程中，窗口内元素只要有 1，则整个窗口元素都令为 1，这样会增大物体的尺寸。通常对于有白噪声的图片，先腐蚀再膨胀可以消除白噪声，但一定程度会导致物体失真。但由于用掩模处理后的图像，其物体十分明显，用形态学方法去噪后失真的可能性不大。因而考虑用形态学方法去噪。

!!! 去噪后的图片

3.3 边框裁剪与尺寸归一化

4 机器学习理论

4.1 降维方法

4.2 Logistic 回归

4.3 SVM

4.4 决策树

决策树学习，假设给定训练数据集

$$D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(N)}, y^{(N)})\} \tag{1}$$

其中，$x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \cdots, x_n^{(i)})$，$n$ 为特征个数，$y^{(i)} \in \{1, 2, \cdots, K\}$，$K$ 为类别数目，$i = 1, 2, \cdots, N$，N 为样本容量。

4.4.1 特征选择

设 X 是一个取有限个值的离散随机变量，其概率分布为

$$P(X = x^{(i)}) = p_i, i = 1, 2, \cdots, n \tag{2}$$

则随机变量 X 的熵定义为

$$H(X) = -\sum_{i=1}^{n} p_i \log p_i \tag{3}$$

熵越大，随机变量的不确定性就越大。

设有随机变量 (X, Y)，其联合概率分布为

$$P(X = x^{(i)}, Y = y^{(i)}) = p_{ij}\ i, 2, \cdots, n; j = 1, 2, \cdots, m \tag{4}$$

条件熵 $H(Y|X)$ 为已知随机变量 X 的条件下随机变量 Y 的不确定性，随机变量 X 给定的条件下随机变量 Y 的条件熵 $H(Y|X)$ 定义如下

$$H(Y|X) = \sum_{i=1}^{n} p_i H(Y|X = x_i) \tag{5}$$

其中，$p_i = P(X = x_i), i = 1, 2, \cdots, n$。

信息增益（information gain）表示得知特征 X 的信息，是的类 Y 的信息的不确定性减少的程度，定义如下

信息增益 特征 A 对训练数据集 D 的信息增益 $g(D, A)$，定义为集合 D 的经验熵 $H(D)$ 与特征 A 给定条件下 D 的经验条件熵 $H(D|A)$ 之差，即

$$g(D, A) = H(D) - H(D|A) \tag{6}$$

根据信息增益准则的特征选择方法是：对训练数据集（或子集）D，计算其每个特征的信息增益，并比较它们的大小，选择信息增益最大的特征。

设训练数据为 D，$|D|$ 表示其样本容量，即样本个数，设有 K 个类 C_k，$k=1,2,\cdots,K$，$|C_k|$ 为属于类 C_k 的样本个数，$\sum_{k=1}^{K}|C_k|=|D|$。设特征 A 有 n 个不同的取值 $\{a_1,a_2,\cdots,a_n\}$，记特征集为 A，根据某一特征 a 的取值将 D 划分为 n 个子集 $D_1,D_2,\cdots,D_n$，$|D_i|$ 为 D_i 的样本个数，$\sum_{i=1}^{n}|D_i|=|D|$。记子集 D_i 中属于类 C_k 的样本的集合为 D_{ik}，即 $D_{ik}=D_i\cap C_k$，$|D_{ik}|$ 为 D_{ik} 的样本个数，信息增益算法如下

- 输入：训练数据集 D 和特征 a；
- 输出：特征 a 对训练数据集 D 的信息增益 $g(D,a)$

1 计算数据集 D 的经验熵 $H(D)$

$$H(D)=-\sum_{k=1}^{K}\frac{|C_k|}{|D|}\log_2\frac{|C_k|}{|D|} \tag{7}$$

2 计算特征 a 对数据集 D 的经验条件熵 $H(D|a)$

$$\begin{aligned}H(D|a)&=\sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D|a=a_i)\\&=\sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D_i)\\&=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\sum_{k=1}^{K}\frac{|D_{ik}|}{|D_i|}\log_2\frac{|D_{ik}|}{|D_i|}\end{aligned} \tag{8}$$

3 计算信息增益

$$g(D,A)=H(D)-H(D|a) \tag{9}$$

于是，在候选属性集合 A 中，选择使得划分后信息增益最大的属性作为最优划分属性，即

$$a_*=\arg\max_{a\in A}g(D,a) \tag{10}$$

该算法天生偏向选择分支多的属性，容易导致过拟合。

信息增益比 特征 a 对训练数据集 D 的信息增益比 $g_R(D,a)$ 定义为其信息增益 $g(D,a)$ 与训练数据集 D 关于特征 a 的值的熵 $H_a(D)$ 之比，即

$$g_R(D,a)=\frac{g(D,a)}{H_a(D)} \tag{11}$$

其中，$H_a(D)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\log_2\frac{|D_i|}{|D|}$，$n$ 是特征 a 取值的个数。

于是，在候选属性集合 A 中，选择使得划分后信息增益最大比的属性作为最优划分属性，即

$$a_*=\arg\max_{a\in A}g_R(D,a) \tag{12}$$

分类问题中，假设有 K 个类，样本点属于第 k 类的概率为 p_k，则概率分布的基尼指数定义为

$$Gini(p) = \sum_{k=1}^{K} p_k(1-p_k) = 1 - \sum_{k=1}^{K} p_k^2 \tag{13}$$

对于样本集合 D，D 的纯度可以用 Gini 指数来度量

$$Gini(D) = 1 - \sum_{k=1}^{K} \left(\frac{|C_k|}{|D|}\right)^2 \tag{14}$$

其中，C_k 是 D 中属于第 k 类的样本子集，K 是类的个数。直观上，$Gini(D)$ 反映了 D 中随机抽取两个样本，其类别标记不一致的概率。因此，$Gini(D)$ 越小，则数据集 D 的纯度越高。

设特征 a 有 n 个不同的取值 $\{a_1, a_2, \cdots, a_n\}$,根据特征 a 的取值将 D 划分为 n 个子集 $D_1, D_2, \cdots, D_n$，$|D_i|$ 为 D_i 的样本个数，$\sum_{i=1}^{n} |D_i| = |D|$，则属性 a 的 Gini 指数定义为

$$Gini(D, a) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} Gini(D_i) \tag{15}$$

性 a 的 Gini 指数 $Gini(D, a)$ 表示经 a 分后集合 D 的不确定性，则 $Gini$ 指数值越大，样本集合的不确定性就越大。

于是，在候选属性集合 A 中，选择使得划分后 $Gini$ 指数最小的属性作为最优划分属性，即

$$a_* = \arg\min_{a \in A} Gini(D, a) \tag{16}$$

4.4.2 决策树的生成

从根节点开始，对结点计算所有可能的特征的信息增益，选择信息增益最大的特征作为节点的特征，由该特征的不同区直建立子节点，再递归地使用以上方法，构造决策树，直到所有特征的信息增益均最小或没有特征可以选择为止，最后得到一个决策树

ID3 算法

- 输入：训练数据集 D，特征集 A，阈值 ϵ
- 输出：决策树 T

1 若 D 中所有实例属于同一类 C_k，则 T 为单节点树，并将类 C_k 作为该结点的类标记，返回 T

2 若 $A = \emptyset$，则 T 为单节点树，并将 D 中实例数最大的类 C_k 作为该结点的类标记，返回 T

3 否则，计算 A 中各特征对 D 的信息增益 $g(D, A_i)$，选择信息增益最大的特征 A_g

4 如果 A_g 的信息增益小于阈值 ϵ，则置 T 为单结点树，并将 D 中实例数最大的类 C_k 作为该节点的类标记，返回 T

5 否则，对 A_g 的每一个可能取值 a_i，依 a_i 将 D 分割为若干非空子集 D_i，将 D_i 中实例数最大的类作为标记，构建子结点，由节点及其子结点构成树 T，返回 T

6 对第 i 个子结点，以 D_i 为训练集，以 $A-\{A_g\}$ 为特征集，递归地调用 $1\sim5$，得到子树 T_i，返回 T_i

C4.5 采用信息增益比来选择特征

C4.5 算法

- 输入：训练数据集 D，特征集 A，阈值 ϵ
- 输出：决策树 T

1 若 D 中所有实例属于同一类 C_k，则 T 为单节点树，并将类 C_k 作为该结点的类标记，返回 T

2 若 $A=\emptyset$，则 T 为单节点树，并将 D 中实例数最大的类 C_k 作为该结点的类标记，返回 T

3 否则，计算 A 中各特征对 D 的信息增益比 $g(D,A_i)$，选择信息增益比最大的特征 A_g

4 如果 A_g 的信息增益比小于阈值 ϵ，则置 T 为单结点树，并将 D 中实例数最大的类 C_k 作为该节点的类标记，返回 T

5 否则，对 A_g 的每一个可能取值 a_i，依 a_i 将 D 分割为若干非空子集 D_i，将 D_i 中实例数最大的类作为标记，构建子结点，由节点及其子结点构成树 T，返回 T

6 对第 i 个子结点，以 D_i 为训练集，以 $A-\{A_g\}$ 为特征集，递归地调用 $1\sim5$，得到子树 T_i，返回 T_i

CART 对回归树用平方误差最小化准则，对分类树用 Gini 指数最小化准则，进行特征选择，生成二叉树。

假设 X 和 Y 分别为输入和输出变量,并且 Y 是连续变量,给定训练数据集 $D=\{(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),\cdots,(x^{(N)},y^{(}$ 回归树将输入空间划分为 M 个单元 $D_1,D_2,\cdots,D_M$，且在每个单元上有一个固定值 c_m，因此回归树模型表示为

$$f(x)=\sum_{m=1}^{M}c_m1(x\in D_m) \tag{17}$$

其中，$1(x)$ 为示性函数。具体的，为求解

$$\min_{j,s}\left(\min_{c_1}\sum_{x_i\in D_1(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in D_2(j,s)}(y_i-c_2)^2\right) \tag{18}$$

其中，j 为最优划分变量，s 为最优划分点。$D_1(j,s)=\{x|x_j\le s\}$，$D_2(j,s)=\{x|x_j>s\}$。对于 j 和 s 的选取，采用遍历的方法。遍历划分变量 j，再以步长 Δs 扫描划分点 s。对于 j，s 都固定，且用平方误差 $\sum_{x_i\in D_m}(y^{(i)}-f(x^{(i)}))^2$ 来表示回归树对于训练数据的预测误差，则可求得在单元 D_m 上的 c_m 的最优值 $\hat{c_m}$ 为

$$\hat{c_m}=\frac{1}{N_m}\sum_{x_i\in D_m(j,s)}y_i,\ m=1,2 \tag{19}$$

其中，N_m 为单元 D_m 中的样本个数。其表示对 D_m 的所有样本的 y 取均值。

经过两轮遍历之后，即可选出最优划分变量和最优划分点，以及计算出对应的 $\hat{c_m}$。算法如下

最小二乘回归树生成算法

- 输入：训练数据集 D，特征集 A，阈值 ϵ
- 输出：回归树 $f(x)$

1 若 D 中所有实例的输出均为 y_0，则 T 为单节点树，并将 y_0 作为该结点的输出值，返回 T

2 否则，采用遍历的方法。遍历划分变量 j，对于固定的划分变量 j 再以步长 Δs 扫描划分点 s，对于 j，s 都固定，求 c_1，c_2

$$c_m = \frac{1}{N_m} \sum_{x_i \in D_m(j,s)} y_i,\ m = 1, 2 \tag{20}$$

来求解

$$\min_{j,s} \left(\min_{c_1} \sum_{x_i \in D_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in D_2(j,s)} (y_i - c_2)^2 \right) \tag{21}$$

并记损失函数

$$L = \sum_{x_i \in D_1(j,s)} (y_i - c_1)^2 + \sum_{x_i \in D_2(j,s)} (y_i - c_2)^2 \tag{22}$$

3 如果 L 小于阈值 ϵ，则置 T 为单结点树，并将 D 中实例的输出的均值 $\hat{c} = \frac{1}{N_{|D|}} \sum_{x_i \in D} y_i$ 作为该结点的输出值，返回 T

4 否则，用选定的对 (j, s) 划分区域，并决定相应的输出值

$$D_1(j,s) = \{x | x_j \le s\} \tag{23}$$

$$D_2(j,s) = \{x | x_j > s\} \tag{24}$$

$$\hat{c_m} = \frac{1}{N_m} \sum_{x_i \in D_m(j,s)} y_i,\ m = 1, 2 \tag{25}$$

5 对这两个子区域 $D_i,\ i = 1, 2$，以 D_i 为训练集，以 A 为特征集，递归地调用 $1 \sim 5$，得到子树 T_i，返回 T_i

6 将输入空间划分为 M 个区域 $D_1, D_2, \cdots, D_M$，生成决策树

$$f(x) = \sum_{m=1}^{M} c_m 1(x \in D_m) \tag{26}$$

注：对比 ID3 和 C4.5 算法，由于回归树少了将特征集进行剔除，即少了第 5 步以 $A - \{A_g\}$ 为特征集，因而少了这一步：

2 若 $A = \emptyset$，则 T 为单节点树，并将 D 中实例的输出的均值 $\hat{c} = \frac{1}{N_{|D|}} \sum_{x_i \in D} y_i$ 作为该结点的输出值，返回 T

4.5 提升方法

4.6 交叉熵

4.7 one-hot 编码与 softmax

5 BP 神经网络

5.1 神经网络的构造与前向传播

神经网络是由单个或多个神经元组成。下面是单个神经元的构造。

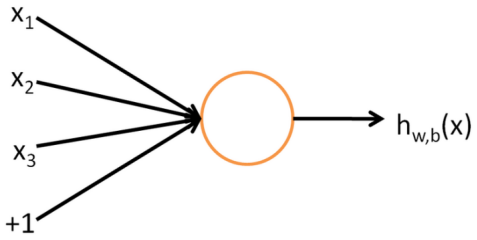


图 1：神经元

该神经元的输入由三个数据 x_1, x_2, x_3 以及偏置项 (bias)+1 组成，通过神经元后输出的表达式为

$$h_{W,b}(x) = f(W^T x + b) = f(\sum_{i=1}^{3} W_i x_i + b) \tag{27}$$

其中 f 为激活函数。激活函数是为了将线性项 $W^T x$ 变换为非线性。在 BP 中，较常用的激活函数为 sigmoid 函数，其表达式如下

$$f(z) = \frac{1}{1 + \exp(-z)} \tag{28}$$

另外，令 $b = w_0$，则可重新定义 $W = (w_0, w_1, w_2, w_3)^T$，$x = (1, x_1, x_2, x_3)$，于是可将上式写为

$$h_{W,b}(x) = f(W^T x) \tag{29}$$

下面讨论神经网络。多个神经元可以组成一个层，多个层互相连接可以组成神经网络。其中，接受数据输入的层为输入层，数据计算后的数据的输出层，中间的层则称为隐含层。为下图是含有两个隐含层的神经网络。

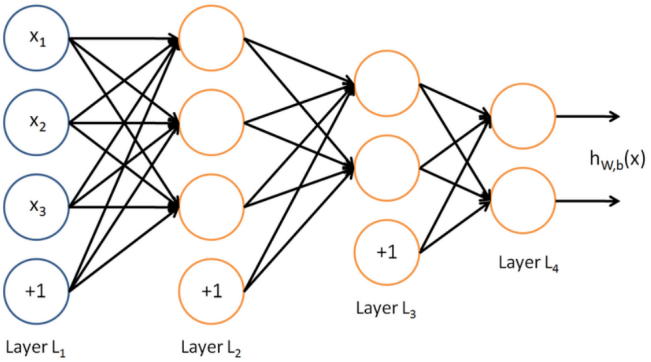


图 2：含有两个隐含层的神经网络

如图，最左边的为输入层，即图中的 Layer L1，最右边的为输出层，即图中的 Layer L4，中间的所有层，即图中的 Layer L2，Layer L3 为隐含层。

我们用 n_l 来表示网络的层数，记第 i 层为 L_i，于是输入层为 L_1，输出层为 L_{n_l}。由于神经网络可以有任意多的隐层以及隐藏神经元，则我们记 $W_{ij}^{(l)}$ 为第 l 层第 j 单元以及第 $l+1$ 层第 i 单元之间的连接权重，$b_i^{(l)}$ 为第 $L+1$ 层第 i 单元的偏执。我们用 $a_i^{(l)}$ 表示第 l 层第 i 单元的激活值（输出值），则有

$$a_i^{(l+1)} = f(\sum_{j=1}^{S_l} W_{ij}^{(l)} a_j^{(l)} + b_i^{(l)}) \tag{30}$$

其中当 $l=1$ 时，$a^{(l)} = x$，x 为输入向量 $(x_1, x_2, \cdots, x_{S_l})$，$S_l$ 指第 l 层的神经元个数，我们用 $z_i^{(l+1)}$ 表示第 $l+1$ 层第 i 单元输入加权和（包括偏置），即

$$z_i^{(l+1)} = \sum_{j=1}^{S_t} W_{ij}^{(l)} a_j^{(l)} + b_i^{(l)} \tag{31}$$

则有

$$a_i^{(l+1)} = f(z_i^{(l+1)}) \tag{32}$$

$$h_{W,b}(x) = a^{(n_l)} = f(z^{(n_l)}) \tag{33}$$

上述过程称为神经网络的前向传播。

5.2 神经网络的反向传播

根据上面的前向传播，我们设神经网络的各层表示为 $L_1, L_2, \cdots, L_{n_l}$，其中，$L_{n_l}$ 为输出层，对于输出层，假设输出层输出为 $t = a^{(n_l)}$，y 为标签，则若为回归问题，则代价函数使用 MSE，即

$$J(W, b; x, y) = \frac{1}{2}||t - y||^2 \tag{34}$$

接下来计算输出层的残差

$$\begin{aligned}
\delta_i^{(n_l)} &= \frac{\partial}{\partial z_i^{(n_l)}} J(W, b; x, y) \\
&= \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} ||y - h_{W,b}(x)||^2 \\
&= \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \sum_{j=1}^{S} {}_{n_l}(y_i - a_j^{(n_l)})^2 \\
&= \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \sum_{j=1}^{S} {}_{n_l}(y_i - f(z_i^{(n_l)}))^2 \\
&= -(y_i - f(z_i^{(n_l)})) \cdot f'(z_i^{(n_l)}) \\
&= -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})
\end{aligned} \tag{35}$$

下面考虑残差的递推算法，以输出层前一层为例。由前向传播我们可以推导出

$$z_i^{(l+1)} = \sum_{j=1}^{S_l} W_{ij}^{(l)} f(z_i^{(l)}) + b_i^{(l)} \tag{36}$$

则有

$$z_i^{(n_i)} = \sum_{j=1}^{S_l} W_{ij}^{(n_l-1)} f(z_i^{(n_l-1)}) + b_i^{(n_l-1)} \tag{37}$$

于是有

$$\frac{\partial z_i^{(n_l)}}{\partial z_i^{(n_l-1)}} = \sum_{j=1}^{S_l} W_{ij}^{(n_l-1)} f'(z_i^{(n_l-1)}) \tag{38}$$

则可以得到输出层前一层的残差

$$\begin{aligned} \delta_i^{(n_l-1)} &= \frac{\partial}{\partial z_i^{(n_l-1)}} J(W,b;x,y) \\ &= \frac{\partial J(W,b;x,y)}{\partial z_i^{(n_l)}} \cdot \frac{\partial z_i^{(n_l)}}{\partial z_i^{(n_l-1)}} \\ &= \sum_{j=1}^{S_l} \delta_j^{(n_l)} W_{ij}^{(n_l-1)} f'(z_i^{(n_l-1)}) \end{aligned} \tag{39}$$

将 $n_l - 1$ 与 n_l 的关系替换为 l 与 $l+1$ 的关系，则可得到

$$\delta_i^{(l)} = \frac{\partial}{\partial z_i^{(l)}} J(W,b;x,y) = \left(\sum_{j=1}^{S_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)}) \tag{40}$$

若取函数 f 为 sigmoid 函数，则有

$$f'(z_i^{(l)}) = f(z_i^{(l)}) \circ (1 - f(z_i^{(l)})) = a_i^{(l)} \circ (1 - a_i^{(l)}) \tag{41}$$

其中 $\circ$ 代表点乘。于是可得到 $\sigma_j^{(l+1)}$ 到 $\sigma_j^{(l)}$ 的递推式：

$$\delta_i^{(l)} = \left(\sum_{j=1}^{S_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) (a_i^{(l)} \circ (1 - a_i^{(l)})) \tag{42}$$

反向传播，一般采用梯度下降法对每一层的权重进行调整，即

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) \tag{43}$$

其中，α 是学习率。因而需要求权重 $W_{ij}^{(l)}$ 对于代价函数的偏导，此时可使用当前层的残差来进行计算，即

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = \frac{\partial J(W,b;x,y)}{\partial z_i^{(l+1)}} \frac{z_i^{(l+1)}}{W_{ij}^{(l)}} \tag{44}$$

又有

$$\frac{z_i^{(l+1)}}{W_{ij}^{(l)}} = \frac{\left(\sum_{j=1}^{S_l} W_{ij}^{(l)} f(z_i^{(l)}) \right)}{W_{ij}^{(l)}} = f(z_i^{(l)}) = a_i^{(l)} \tag{45}$$

于是可得

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)} \tag{46}$$

综上，可以总结 BP 神经网络算法

BP 神经网络算法

输入：训练输入 x，训练输出 y，学习率 α

while 未达到收敛条件

输入训练输入，训练输出，学习率

1. 初始化神经网络的权重与偏置

2. 对输入进行前向传播，得到除输入层外每一层（$L_2, \cdots, L_{n_l}$）的激活值 $a^{(2)}, \cdots, a^{(n_l)}$

3. 计算各层残差：

(1) 对输出层（第 n_l 层）

$$\delta^{(n_l)} = -(y - a^{(n_l)}) \cdot (a^{(l)} \circ (1 - a^{(l)})) \tag{47}$$

(2) 对于 $l = n_l - 1, \cdots, 2$ 各层，可递推得出残差值

$$\delta^{(l)} = ((W^{(l)})^T \delta^{(l+1)}) \cdot (a^{(l)}) \tag{48}$$

(3) 计算损失函数对每一层权重的偏导数值

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l)})^T \tag{49}$$

(4) 更新参数

$$W^{(l)} = W^{(l)} - \alpha \nabla_{W^{(l)}} J(W, b; x, y) \tag{50}$$

end

若为多分类问题，先对 y 进行 one-hot 处理得到 p 维向量 $(y_1, y_2, \cdots, y_p)$（假设 y 有 p 种取值），并将输出层的激活函数选为 softmax，即

$$a_i^{(n_l)} = f_s(z_i^{(n_l)}) = \frac{e^{z_i^{(n_l)}}}{\sum_j e^{z_j^{(n_l)}}} \tag{51}$$

并且代价函数使用交叉熵损失函数

$$J(W, b; x, y) = -\sum_i y_i \log a_i^{(n_l)} \tag{52}$$

则输出层残差为

$$\begin{aligned} \delta_i^{(n_l)} &= \frac{\partial J}{\partial z_i^{(n_l)}} \\ &= \sum_i \frac{\partial J}{a_i^{(n_l)}} \cdot \frac{\partial a_i^{(n_l)}}{\partial z_i^{(n_l)}} \\ &= \sum_i \frac{\partial - \sum_i y_i \log a_i^{(n_l)}}{a_i^{(n_l)}} \cdot \frac{\partial a_i^{(n_l)}}{\partial z_i^{(n_l)}} \\ &= -\sum_i \frac{y_i}{a_i^{(n_l)}} \frac{\partial a_i^{(n_l)}}{\partial z_j^{(n_l)}} \end{aligned} \tag{53}$$

当 $i=j$ 时，记 $e^{z_j^{(n_l)}}=e^A$，$\sum_{k\neq j}e^{z_k^{(n_l)}}=e^B$，显然有 $e^A+e^B=\sum_i e^{z_i^{(n_l)}}$，于是

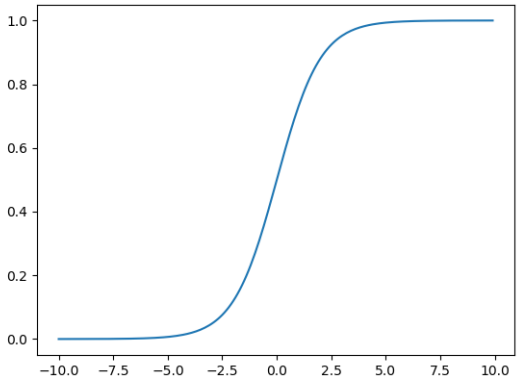
$$\begin{aligned}\frac{\partial a_i^{(n_l)}}{\partial z_j^{(n_l)}}&=\frac{\partial a_j^{(n_l)}}{\partial z_j^{(n_l)}}\\&=\frac{\partial \frac{e^A}{e^A+e^B}}{\partial A}\\&=\frac{e^A(e^B+e^A)-e^{2A}}{(e^A+e^B)^2}\\&=\frac{e^Ae^B}{(e^A+e^B)^2}\\&=\frac{e^A}{e^A+e^B}\frac{e^B}{e^A+e^B}\\&=\frac{e^A}{e^A+e^B}(1-\frac{e^A}{e^A+e^B})\\&=a_j^{(n_l)}(1-a_j^{(n_l)})\end{aligned}\tag{54}$$

5.3 激活函数

sigmoid sigmoid 函数表达式如下

$$f(x)=\frac{1}{1+e^{-x}}\tag{55}$$

其图像如下图所示



sigmoid 激活函数考虑将输入值映射到 $(0,1)$ 的区间中，该函数在定义域内连续，且导数大于 0。它也有较为简单的求导结果

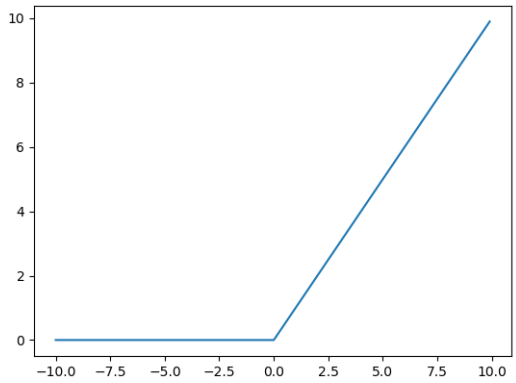
$$f'(x)=f(x)(1-f(x))\tag{56}$$

但是在神经网络中，特别是对于层数较多的网络，通常不采用 sigmoid 作为激活函数，主要是因为其容易产生梯度消失的情况。当输入非常大或非常小的时候，其梯度趋近于 0，反向传播的过程中直接导致梯度无法传播，无法有效地调整权重。虽然做标准化可以让数据近似服从正态分布，但梯度消失仍有可能产生，在学习过程中可能会产生输入较大或较小的情况。或许这个问题可以用 batch-normalization 来缓解，但明显采取一种更佳的激活函数是较为可取的做法。

ReLU ReLU 函数表达式如下

$$f(x) = \max\{0, x\} \tag{57}$$

图像如下



其决定它有非常简单的求导结果

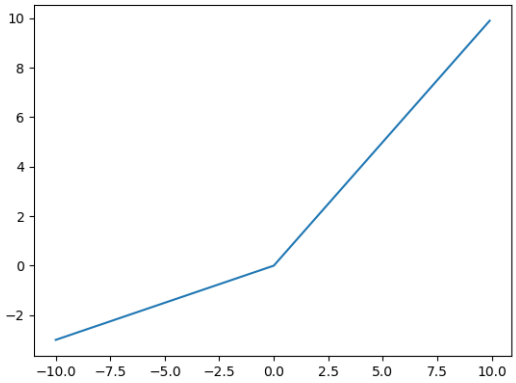
$$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases} \tag{58}$$

RuLU 收敛能比 sigmoid 快的多，一方面其计算快，比起 sigmoid 函数的导数需要指数运算，RuLU 只需要做大小的比较。另一方面，其梯度经过多个层传播之后，多数能够保持原汁原味，比起 sigmoid 会梯度消失要好得多。然而，RuLU 也有弱点，当 $x < 0$ 时 $f(x)$ 为 0，梯度为 0，这直接导致该神经元失活。因而在训练过程中，要注意取较小的学习率。

Leaky ReLU Leaky ReLU 是针对 RuLU 的弱点而改进的，其考虑用一个比较小的数去替代 $x < 0$ 时的 $f(x) = 0$，即

$$f'(x) = \begin{cases} x, & x > 0 \\ ax, & x < 0 \end{cases} \tag{59}$$

图像如下



其求导结果为

$$f'(x) = \begin{cases} 1, & x > 0 \\ a, & x < 0 \end{cases} \tag{60}$$

这个方法可以使 $x < 0$ 处避免失活，但是额外引入了超参数 a。

PReLU PReLU 是针对 Leaky ReLU 的进一步优化，其考虑在反向传播过程中，也对 a 进行学习，从而避免引入超参数 a。一些实验 [1] 证明这种优化能取到好的学习效果。

5.4 梯度下降方法

梯度下降法的选取能影响收敛速度与质量，它也是模型构成的一部分。在应用中一般有如下的梯度下降法可供选择

批量梯度下降法 批量梯度下降法（Batch Gradient Descent ）考虑在计算了所有样本之后再对参数进行更新，即

$$W^{(l)} = \sum_{i=1}^{m} W^{(l)} - \alpha \nabla_{W^{(l)}} J(W, b; x^{(i)}, y^{(i)}) \tag{61}$$

由于通常训练的样本非常大，若在计算所有样本之后再进行参数更新，会让更新的速度减慢。另外，模型实现一般会采用矩阵运算，BGD 占的内存会非常多，从而影响计算速度。

随机梯度下降法 随机梯度下降法（Stochastic Gradient Descent ）的想法与 BGD 截然不同，计算每一个样本之后便进行一次反向传播，对参数进行更新，即

$$W^{(l)} = W^{(l)} - \alpha \nabla_{W^{(l)}} J(W, b; x, y) \tag{62}$$

相比之下，SGD 的训练速度比 BGD 快得多，在 BGD 进行一次反向传播的时间内，SGD 已经进行过多次传播。但是在梯度下降过程中，SGD 容易出现震荡，由于单个样本并不能代表梯度最大的方向，也有可能导致解非最优的情况。

小批量梯度下降法 小批量梯度下降法（Mini-batch Gradient Descent ）考虑了批量梯度下降法和随机梯度下降法的优缺点，并进行结合，考虑将数据集划分成多个含有较小数据的 batch，然后对这些 batch 分别采用 BGD。下面给出第 i 个 batch 的训练公式

$$W^{(l)} = \sum_{(x,y) \in b_i}^{m} W^{(l)} - \alpha \nabla_{W^{(l)}} J(W, b; x, y) \tag{63}$$

其中，b_i 代表当前 batch 所包含的训练样本 (x, y) 的集合。

动量梯度下降法 无论是 SGD 还是 MGD，即便 MGD 已在 SGD 上做了优化，在训练过程中仍可能会有振荡的风险。一种优化的方法是基于 SGD，在对参数 $W^{(l)}$ 进行更新时，会考虑上一次的更新幅度，若是当前的梯度方向与上一次的相同，则能够加速收敛，反之则能抑制更新，这也是采用了动量的想法。其算法如下

1 输入：学习率 ϵ，动量参数 α，1

5.5 学习率

5.6 正则化

5.7 BP 神经网络 +

6 卷积神经网络

6.1 卷积神经网络概述

6.2 AlexNet

6.3 VGGNet

6.4 Inception

6.5 ResNet

6.6 迁移学习

6.7 CNN+

6.8 Batch Normalization

7 参考文献

[1] Kaiming He,Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,https://arxiv.org/abs/1502.01852