机器学习实验报告

神经网络

朱天泽

(日期：2022 年 4 月 10 日)

摘要 在《机器学习》第 5 章中，我学习了神经网络。在此次实验中，我实现了标准 BP 算法和累积 BP 算法，在西瓜数据集 3.0 上分别用这两个算法训练了一个单隐层网络，并进行了比较。
关键词 神经网络；BP 算法；分类

1 习题 1

1.1 题目理解

题目要求：编程实现标准 BP 算法，在西瓜数据集 3.0 上训练一个单隐层网络，并做数据分析和结果评价。

1.2 标准 BP 算法原理阐述

给定训练集 $D=\{(\boldsymbol{x}_1,\boldsymbol{y}_i),(\boldsymbol{x}_2,\boldsymbol{y}_2),\cdots,(\boldsymbol{x}_m,\boldsymbol{y}_m)\}$，$\boldsymbol{x}_i\in\mathbb{R}^d$，$\boldsymbol{y}_i\in\mathbb{R}^l$。

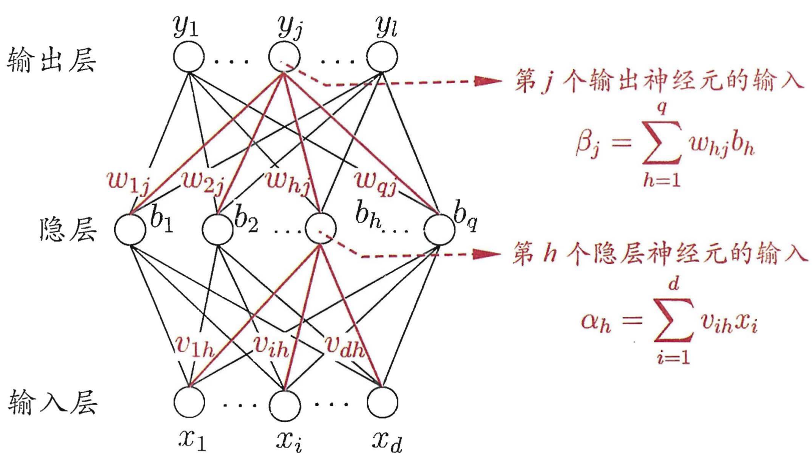

图 1: BP 网络及算法中的变量符号

为便于讨论，图1给出了拥有 d 个输入神经元、l 个输出神经元、q 个隐层网络的多层前馈网络结构，其中输出层第 j 个神经元的阈值用 θ_j 表示，隐层第 h 个神经元的阈值用 γ_h 表示。输入层第 i 个神经元与隐层第 h 个神经元之间的连接权为 v_{ih}，隐层第 h 个神经元与输出层第 j 个神经元之间的连接权为 w_{hj}。记隐层第 h 个神经元接收到的输入为 $\alpha_h=\sum\limits_{i=1}^{d}v_{ih}x_i$，输出层第 j 个神经元接收到的输入

为 $\beta_j = \sum\limits_{h=1}^{q} w_{hj} b_h$，其中 b_h 为隐层第 h 个神经元的输出。这里假设隐层和输出层神经元都使用 Sigmoid 函数 $f(z) = \frac{1}{1+e^{-z}}$。

对训练样本 $(\boldsymbol{x}_k, \boldsymbol{y}_k)$，假定神经网络的输出为 $\hat{\boldsymbol{y}}_k = (\hat{y}_1^k, \hat{y}_2^k, \cdots, \hat{y}_l^k)$，即

$$\hat{y}_j^k = f(\beta_j - \theta_j) \tag{1}$$

进一步可定义网络在 $(\boldsymbol{x}_k, \boldsymbol{y}_k)$ 上的均方误差为

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

BP 是一个迭代学习算法，在迭代的每一轮中采用广义的感知机学习桂策对参数进行更新估计，任意参数 χ 的更新估计式为

$$\chi \leftarrow \chi + \Delta\chi \tag{3}$$

BP 算法基于梯度下降策略，以目标的负梯度方向对参数进行调整。对式(2)的误差 E_k，给定学习率 η，有

$$\Delta w_{hj} = -\eta \frac{\partial E_k}{\partial w_{hj}} \tag{4}$$

注意到 w_{hj} 先影响到第 j 个输出层神经元的输入值 $\beta_j = \sum\limits_{h=1}^{q} w_{hj} b_h$，再影响到其输出值 $\hat{y}_j^k = f(\beta_j - \theta_j)$，然后影响到 $E_k = \frac{1}{2} \sum\limits_{j=1}^{l} \left(\hat{y}_j^k - y_j^k\right)^2$；根据链式法则，有

$$\frac{\partial E_k}{\partial w_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}} \tag{5}$$

根据 β_j 的定义，显然有

$$\frac{\partial \beta_j}{\partial w_{hj}} = b_h \tag{6}$$

基于 Sigmoid 函数良好的性质 $f'(x) = f(x)(1 - f(x))$，根据式(1)和式(2)，有

$$\begin{aligned} g_j &= -\frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \\ &= -\left(\hat{y}_j^k - y_j^k\right) f'\left(\beta_j - \theta_j\right) \\ &= \hat{y}_j^k \left(1 - \hat{y}_j^k\right) \left(y_j^k - \hat{y}_j^k\right) \end{aligned} \tag{7}$$

将式(6)和式(7)代入式(5)，再代入式(3)，得到 BP 算法中关于 w_{hj} 的更新公式

$$\Delta w_{hj} = \eta g_j b_h \tag{8}$$

类似可得

$$\begin{aligned}
\frac{\partial E_k}{\partial \theta_j} &= \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \theta_j} \\
&= \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \left[f\left(\beta_j - \theta_j\right)\right]}{\partial \theta_j} \\
&= -\frac{\partial E_k}{\partial \hat{y}_j^k} \cdot f'\left(\beta_j - \theta_j\right) \\
&= \left(y_j^k - \hat{y}_j^k\right) \hat{y}_j^k \left(1 - \hat{y}_j^k\right)
\end{aligned} \tag{9}$$

$$\begin{aligned}
\frac{\partial E_k}{\partial v_{ih}} &= \sum_{j=1}^{l} \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h} \cdot \frac{\partial \alpha_h}{\partial v_{ih}} \\
&= \sum_{j=1}^{l} \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h} \cdot x_i \\
&= \sum_{j=1}^{l} \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} \cdot f'\left(\alpha_h - \gamma_h\right) \cdot x_i \\
&= \sum_{j=1}^{l} \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot w_{hj} \cdot f'\left(\alpha_h - \gamma_h\right) \cdot x_i \\
&= \sum_{j=1}^{l} \left(-g_j\right) \cdot w_{hj} \cdot f'\left(\alpha_h - \gamma_h\right) \cdot x_i \\
&= -f'\left(\alpha_h - \gamma_h\right) \cdot \sum_{j=1}^{l} g_j \cdot w_{hj} \cdot x_i \\
&= -b_h\left(1 - b_h\right) \cdot \sum_{j=1}^{l} g_j \cdot w_{hj} \cdot x_i
\end{aligned} \tag{10}$$

$$\begin{aligned}
\frac{\partial E_k}{\partial \gamma_h} &= \sum_{j=1}^{l} \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} \cdot \frac{\partial b_h}{\partial \gamma_h} \\
&= -\sum_{j=1}^{l} \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} \cdot f'\left(\alpha_h - \gamma_h\right) \\
&= -\sum_{j=1}^{l} \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot w_{hj} \cdot f'\left(\alpha_h - \gamma_h\right) \\
&= -\sum_{j=1}^{l} \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot w_{hj} \cdot b_h\left(1 - b_h\right) \\
&= \sum_{j=1}^{l} g_j \cdot w_{hj} \cdot b_h\left(1 - b_h\right)
\end{aligned} \tag{11}$$

令

$$e_h = b_h\left(1 - b_h\right) \sum_{j=1}^{l} w_{hj} g_j \tag{12}$$

进一步得到其余变量的增量

$$\Delta \theta_j = -\eta g_j \tag{13}$$

$$\Delta v_{ih} = \eta e_h x_i \tag{14}$$

$$\Delta \gamma_h = -\eta e_h \tag{15}$$

1.3 标准 BP 算法设计思路

将上述推导进一步写成矩阵形式，定义

$$\boldsymbol{g} = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_l \end{bmatrix} \quad \boldsymbol{e} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_q \end{bmatrix} \tag{16}$$

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_l \end{bmatrix} \quad \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_q \end{bmatrix} \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_q \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_l \end{bmatrix} \quad \hat{\boldsymbol{y}} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_l \end{bmatrix} \tag{17}$$

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_l \end{bmatrix} \quad \boldsymbol{\gamma} = \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_q \end{bmatrix} \quad \boldsymbol{V} = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1q} \\ v_{21} & v_{22} & \cdots & v_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ v_{d1} & v_{d2} & \cdots & v_{dq} \end{bmatrix} \quad \boldsymbol{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1l} \\ w_{21} & w_{22} & \cdots & w_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ w_{q1} & w_{q2} & \cdots & w_{ql} \end{bmatrix} \tag{18}$$

易得

$$\boldsymbol{g} = \hat{\boldsymbol{y}} \circ (\boldsymbol{I} - \hat{\boldsymbol{y}}) \circ (\boldsymbol{y} - \hat{\boldsymbol{y}}) \tag{19}$$

$$\boldsymbol{e} = \boldsymbol{b} \circ (\boldsymbol{I} - \boldsymbol{b}) \circ (\boldsymbol{W} \times \boldsymbol{g}) \tag{20}$$

$$\boldsymbol{\alpha} = \boldsymbol{V}^{\mathrm{T}} \boldsymbol{x} \tag{21}$$

$$\boldsymbol{\beta} = \boldsymbol{W}^{\mathrm{T}} \boldsymbol{b} \tag{22}$$

进一步可得

$$\Delta \boldsymbol{W} = \eta \boldsymbol{b} \boldsymbol{g}^{\mathrm{T}} \tag{23}$$

$$\Delta \boldsymbol{\theta} = -\eta \boldsymbol{g} \tag{24}$$

$$\Delta \boldsymbol{V} = \eta \boldsymbol{x} \boldsymbol{e}^{\mathrm{T}} \tag{25}$$

$$\Delta \boldsymbol{\gamma} = -\eta \boldsymbol{e} \tag{26}$$

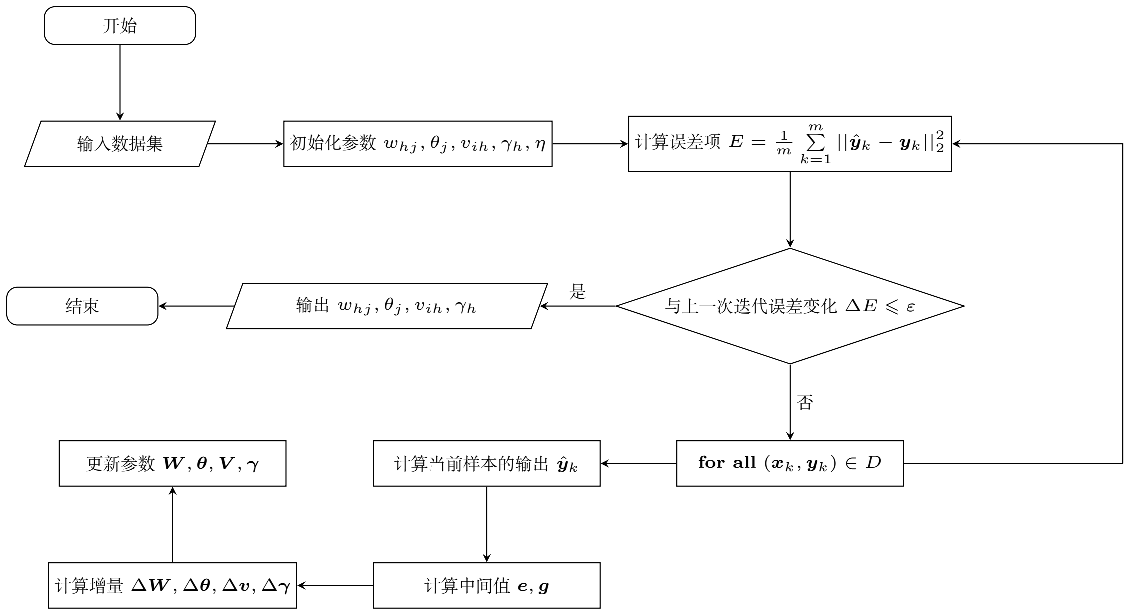

图 2: 标准 BP 算法流程

综上所述，将参数包装为向量和矩阵，参数之间的运算都为 Hadamard 积或矩阵乘法，在 NumPy 中可用运算符 “*” 计算 Hadamard 积、用函数 “dot” 或运算符 “@” 计算矩阵乘法。

标准 BP 算法的流程如图2，对每个训练样本，BP 算法执行以下操作：先将输入示例提供给输入神经元，信号将逐层前传，直到产生输出层的结果；然后计算输出层的误差，再将误差逆向传播至隐层神经元；最后根据隐层神经元的误差来对连接权和阈值进行调整。该迭代过程循环进行，直到达到设置的停止条件位置。

在实验中，我们使用西瓜数据集 3.0 训练单隐层神经网络。在西瓜数据集 3.0 中，“色泽”“根蒂”“敲声”“纹理”“脐部”“触感” 为离散值，而 “密度”、“含糖率” 为连续值；我们需要将离散值映射为连续值来方便计算；举例来说，色泽有三种取值，分别为 “青绿”“乌黑”“浅白”，可对三个取值分别赋值为 1、2、3；此外，还需要将结果是否为好瓜映射为 0、1 的布尔值。

1.4 核心代码分析

标准 BP 算法的核心代码如下。代码中用数组 errors 记录了每一次迭代的误差项。标准 BP 算法遍历每一个样本，每个样本都对参数值进行更新。

```
while True:
    error = E(X, Y, theta, gamma, V, W)
    errors.append(error)
    # 与上一次的误差相差在阈值内
    if  len(errors) > 1 and np. abs(error - errors[ len(errors) - 2]) <= eps:
        break
    for k in  range(m):
        x = X[k]
        y = Y[k]
        # 计算中间值
        alpha = np.dot(x, V)
        b = sigmoid(alpha - gamma)
```

```
        beta = np.dot(b, W)
        y_hat = sigmoid(beta - theta)
        g = y_hat * (1 - y_hat) * (y - y_hat)
        e = b * (1 - b) * np.dot(g, W.T)
        # 更新
        W += eta * (b.reshape((q, 1)) @ g.reshape((1, 1)))
        theta -= eta * g
        V += eta * (x.reshape((d, 1)) @ e.reshape((1, q)))
        gamma -= eta * e
```

1.5 实验结果分析

在本次实验中，我利用西瓜数据集 3.0，用标准 BP 算法训练了一个单隐层网络。

在标准 BP 算法中，参数 $\boldsymbol{W}$、$\boldsymbol{V}$、$\boldsymbol{\theta}$、$\boldsymbol{\gamma}$ 通过迭代得到，不同的训练样本集合 $\{(\boldsymbol{x}_k, \boldsymbol{y}_k)\}_{k=1}^m$ 会得到不同的参数值。不同于神经网络中的参数，学习率 η 是一个超参数，实际上指征了每一步迭代时向梯度负方向前进的速率。

参数 $\boldsymbol{W}$、$\boldsymbol{V}$、$\boldsymbol{\theta}$、$\boldsymbol{\gamma}$ 均随机初始化，而 η 可根据具体问题指定适合的值。在实验中，我随机初始化了 4 组参数，并对每一组参数取不同的 η 训练神经网络，得到其收敛速率。结合图3、图4、图5、图6，大约在前 50 次迭代，不论 η 值如何，总体均方误差均快速下降；随后，η 越大，收敛速度越快。η 并不是越大越好，可以发现当 η 较大时，曲线的抖动会更为剧烈，这可能是在极小值点附近跳跃造成的；但是这种抖动也可能带来意外收益，即跳出局部最优解，到达更优的情况，这就解释了当 η 较大时，会在一段时间的迭代后，均方误差再次快速下降的现象。

总结得出，对于西瓜数据集 3.0 来说，$\eta = 0.5$ 是较为适宜的值；如图 7，当 $\eta = 0.5$ 时，曲线跳跃区间较小，收敛速度也比较快，且与学习率更大时的均方误差收敛值相差不大。

1.6 学习收获

在此次实验中，我实现了标准 BP 算法，并对学习率参数 η 进行了一些研究，得到了迭代收敛速率关于学习率的一些性质。

在实际中，很难使用一个确定的值为最佳的学习率。在误差变化平坦区内，我们希望学习率增大的，因为太小会使得训练次数增加，增大会加速脱离平坦区。在误差变化很大的区域，太大会容易跨过较窄的最低点，这个最低点可能是全局最优点，同时会产生振荡，反而是迭代次数增加。因此为了加速收敛，一个较好的解决思路就是让学习率根据具体情况进行动态调整。查阅资料可知，有 AdaGrad、Adadelta、RMSProp、Adam 等方法来动态调整学习率。

还有一些内容并未在实验中涉及，未来可进一步研究隐层维度、参数初始化方式等因素对算法的影响。

1.7 参考资料

- 《机器学习》5.3；周志华；清华大学出版社。
- 《机器学习公式详解》5.12,5.13,5.14,5.15；谢文睿，秦州；人民邮电出版社。
- 深度学习 — BP 算法详解（BP 算法的优化）

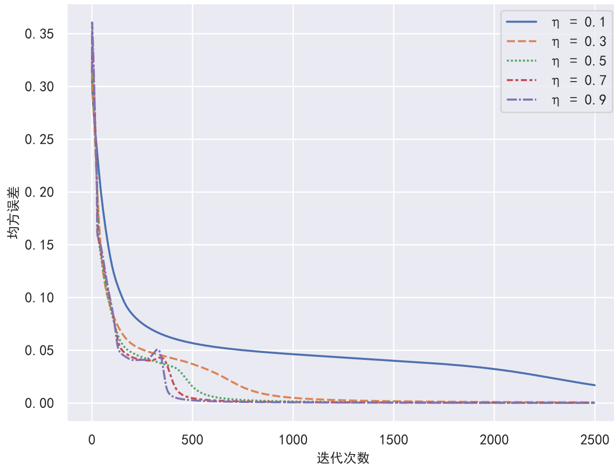

图 3: 标准 BP 算法取不同 η 时的迭代速度（1）

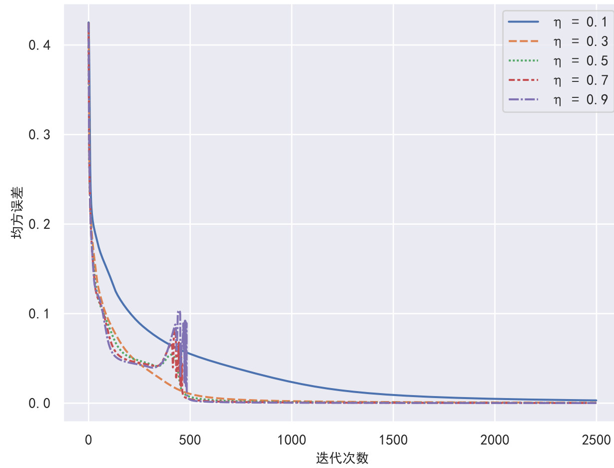

图 4: 标准 BP 算法取不同 η 时的迭代速度（2）

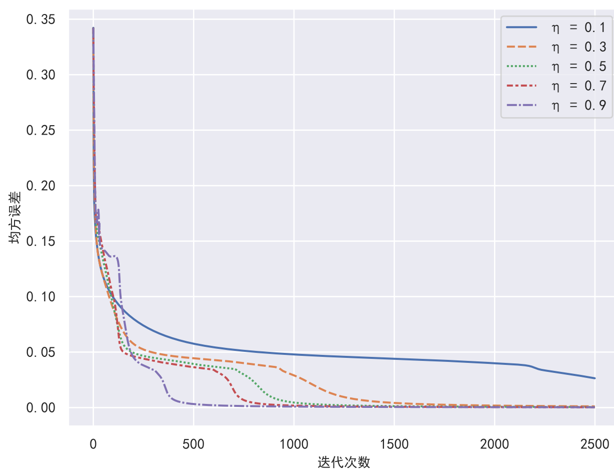

图 5: 标准 BP 算法取不同 η 时的迭代速度（3）

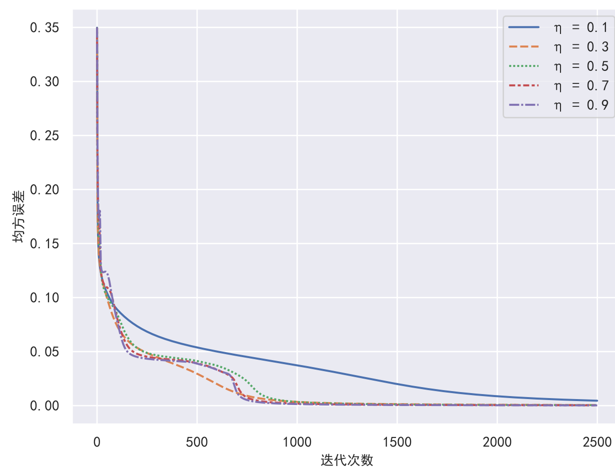

图 6: 标准 BP 算法取不同 η 时的迭代速度（4）

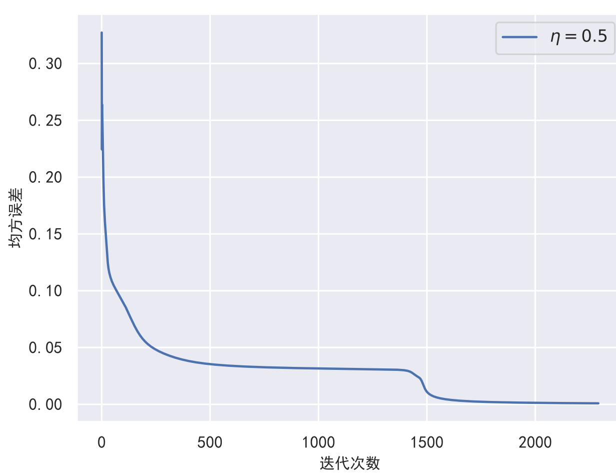

图 7: 标准 BP 算法超参数 η 在西瓜数据集 3.0 上的适宜值

2 习题 2

2.1 题目理解

题目要求：编程实现累积 BP 算法，在西瓜数据集 3.0 上训练一个单隐层网络，做数据分析和结果评价，并和标准 BP 算法进行比较。

2.2 累积 BP 算法原理阐述

需要注意的是，BP 算法的目标是要最小化训练集 D 上的累积误差

$$E = \frac{1}{m}\sum_{k=1}^{m} E_k \tag{27}$$

上面介绍的"标准 BP 算法"每次仅针对一个训练样例更新连接权和阈值，也就是说，"标准 BP 算法"的更新规则是基于单个的 E_k 推导而得，如果类似地推导出基于累积误差最小化的更新规则，就得到了"累积 BP 算法"。

对于神经网络中任意结点上的一个参数 χ

$$\frac{\partial E}{\partial \chi} = \frac{1}{m}\sum_{k=1}^{m} \frac{\partial E_k}{\partial \chi} \tag{28}$$

给定学习率 η，有

$$\Delta\chi = \frac{1}{m}\sum_{k=1}^{m}\left(-\eta\frac{\partial E_k}{\partial \chi}\right) \tag{29}$$

式(29)中的 $\left(-\eta\frac{\partial E_k}{\partial \chi}\right)$ 一项，实际上就是标准 BP 算法中每个参数每一步迭代的增量；累积 BP 算法直接针对累积误差最小化，参数增量为每一个样本求得增量的平均值。

2.3 累积 BP 算法设计思路

仿照标准 BP 算法的思路，我们可以将推导写成矩阵形式方便程序处理；由于累积 BP 算法在读取整个训练集 D 一遍后才对参数进行更新，我们不妨将所有样本及每个样本在迭代时产生的向量 $\boldsymbol{g}$ 和 $\boldsymbol{e}$ 组合成矩阵

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{g}_1 \\ \boldsymbol{g}_2 \\ \vdots \\ \boldsymbol{g}_m \end{bmatrix} \quad \boldsymbol{E} = \begin{bmatrix} \boldsymbol{e}_1 \\ \boldsymbol{e}_2 \\ \vdots \\ \boldsymbol{e}_m \end{bmatrix} \tag{30}$$

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^{\mathrm{T}} \\ \boldsymbol{x}_2^{\mathrm{T}} \\ \vdots \\ \boldsymbol{x}_m^{\mathrm{T}} \end{bmatrix} \quad \boldsymbol{Y} = \begin{bmatrix} \boldsymbol{y}_1^{\mathrm{T}} \\ \boldsymbol{y}_2^{\mathrm{T}} \\ \vdots \\ \boldsymbol{y}_m^{\mathrm{T}} \end{bmatrix} \quad \boldsymbol{A} = \begin{bmatrix} \boldsymbol{\alpha}^{\mathrm{T}} \\ \boldsymbol{\alpha}^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\alpha}_m^{\mathrm{T}} \end{bmatrix} \quad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{b}_1^{\mathrm{T}} \\ \boldsymbol{b}_2^{\mathrm{T}} \\ \vdots \\ \boldsymbol{b}_m^{\mathrm{T}} \end{bmatrix} \quad \boldsymbol{\Xi} = \begin{bmatrix} \boldsymbol{\beta}_1^{\mathrm{T}} \\ \boldsymbol{\beta}_2^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\beta}_m^{\mathrm{T}} \end{bmatrix} \quad \hat{\boldsymbol{Y}} = \begin{bmatrix} \hat{\boldsymbol{y}}_1^{\mathrm{T}} \\ \hat{\boldsymbol{y}}_2^{\mathrm{T}} \\ \vdots \\ \hat{\boldsymbol{y}}_m^{\mathrm{T}} \end{bmatrix} \tag{31}$$

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_l \end{bmatrix} \quad \boldsymbol{\gamma} = \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_q \end{bmatrix} \quad \boldsymbol{V} = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1q} \\ v_{21} & v_{22} & \cdots & v_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ v_{d1} & v_{d2} & \cdots & v_{dq} \end{bmatrix} \quad \boldsymbol{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1l} \\ w_{21} & w_{22} & \cdots & w_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ w_{q1} & w_{q2} & \cdots & w_{ql} \end{bmatrix} \tag{32}$$

易得

$$\boldsymbol{G}=\hat{\boldsymbol{Y}}\circ\left(\boldsymbol{I}-\hat{\boldsymbol{Y}}\right)\circ\left(\boldsymbol{Y}-\hat{\boldsymbol{Y}}\right) \tag{33}$$

$$\boldsymbol{E}=\boldsymbol{B}\circ(\boldsymbol{I}-\boldsymbol{B})\circ\left(\boldsymbol{G}\times\boldsymbol{W}^{\mathrm{T}}\right) \tag{34}$$

$$\boldsymbol{A}=\boldsymbol{X}\boldsymbol{V} \tag{35}$$

$$\boldsymbol{\Xi}=\boldsymbol{B}\boldsymbol{W} \tag{36}$$

进一步可得

$$\Delta\boldsymbol{W}=\frac{\eta}{m}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{G} \tag{37}$$

$$\Delta\boldsymbol{\theta}=-\frac{\eta}{m}\sum_{k=1}^{m}\boldsymbol{g}_k \tag{38}$$

$$\Delta\boldsymbol{V}=\frac{\eta}{m}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{E} \tag{39}$$

$$\Delta\boldsymbol{\gamma}=-\frac{\eta}{m}\sum_{k=1}^{m}\boldsymbol{e}_k \tag{40}$$

综上所述，将参数包装为向量和矩阵，参数之间的运算都为 Hadamard 积或矩阵乘法，在 NumPy 中可用运算符“*”计算 Hadamard 积、用函数“dot”或运算符“@”计算矩阵乘法。

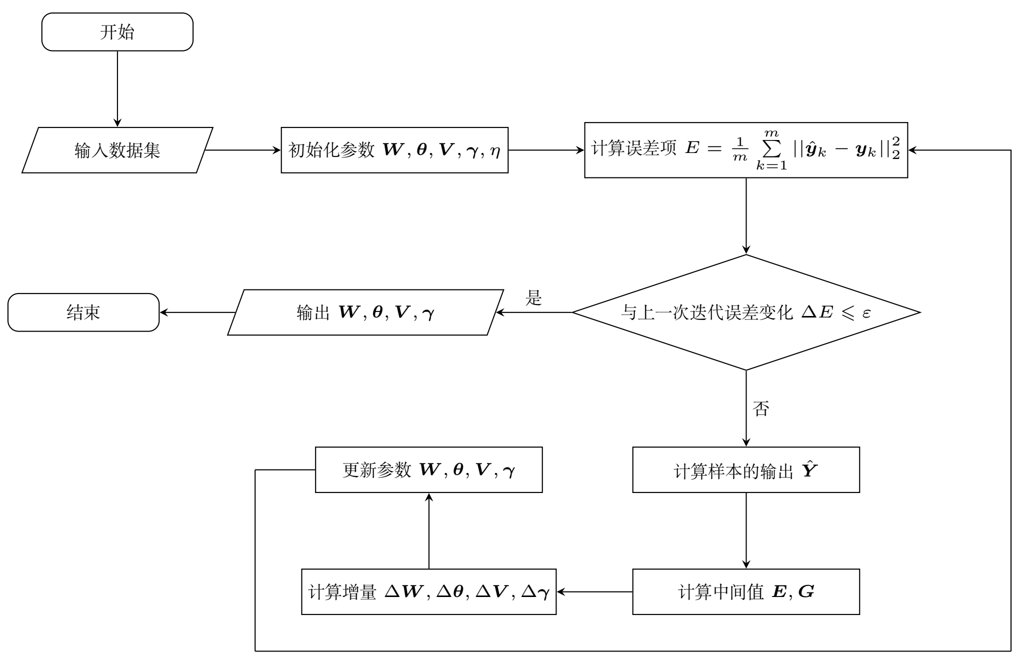

图 8: 累积 BP 算法流程

累积 BP 算法的流程如图8，累积 BP 算法执行以下操作：将输入样本一次性提供给输入神经元，信号将逐层前传，直到产生输出层的结果；然后计算输出层的误差，再将误差逆向传播至隐层神经元；最

后根据隐层神经元的误差来对连接权和阈值进行调整。该迭代过程循环进行，直到达到设置的停止条件为止。

在实验中，我们使用西瓜数据集 3.0 训练单隐层神经网络。在西瓜数据集 3.0 中，“色泽”“根蒂”“敲声”“纹理”“脐部”“触感”为离散值，而“密度”、“含糖率”为连续值；我们需要将离散值映射为连续值来方便计算；举例来说，色泽有三种取值，分别为“青绿”“乌黑”“浅白”，可对三个取值分别赋值为 1、2、3；此外，还需要将结果是否为好瓜映射为 0、1 的布尔值。

2.4 核心代码分析

累积 BP 算法的核心代码如下。代码中用数组 errors 记录了每一次迭代的误差项。累积 BP 算法一次性遍历样本，再对参数值进行更新。

```
while True:
    # 计算均方误差
    error = get_error(X, Y, theta, gamma, V, W)
    errors.append(error)
    if  len(errors) > 1 and np. abs(error - errors[ len(errors) - 2]) <= eps:
        break
    # 得到预测值
    Alpha = np.dot(X, V)
    B = sigmoid(Alpha - gamma)
    Beta = np.dot(B, W)
    Y_hat = sigmoid(Beta - theta)
    # 更新参数
    G = Y_hat * (1 - Y_hat) * (Y - Y_hat)
    E = B * (1 - B) * np.dot(G, W.T)
    W += eta * np.dot(B.T, G) / m
    theta -= eta * np. sum(G, axis=0) / m
    V += eta * np.dot(X.T, E) / m
    gamma -= eta * np. sum(E, axis=0) / m
```

2.5 实验结果分析

在本次实验中，我利用西瓜数据集 3.0，用标准 BP 算法训练了一个单隐层网络。

在标准 BP 算法中，参数 $\boldsymbol{W}$、$\boldsymbol{V}$、$\boldsymbol{\theta}$、$\boldsymbol{\gamma}$ 通过迭代得到，不同的训练样本集合 $\{(\boldsymbol{x}_k, \boldsymbol{y}_k)\}_{k=1}^{m}$ 会得到不同的参数值。不同于神经网络中的参数，学习率 η 是一个超参数，实际上指征了每一步迭代时向梯度负方向前进的速率。

参数 $\boldsymbol{W}$、$\boldsymbol{V}$、$\boldsymbol{\theta}$、$\boldsymbol{\gamma}$ 均随机初始化，而 η 可根据具体问题指定适合的值。在实验中，我随机初始化了 4 组参数，并对每一组参数取不同的 η 训练神经网络，得到其收敛速率。结合图9、图10、图11、图12，大约在前 50 次迭代，不论 η 值如何，总体均方误差均快速下降；随后，η 越大，收敛速度越快。与标准 BP 算法相比，累积 BP 算法可以使用更大的学习率而不至于抖动，均方误差下降曲线更为平滑，不同学习率的收敛曲线相比标准 BP 算法的差距更小。

总结得出，对于西瓜数据集 3.0 来说，$\eta = 2$ 是较为适宜的值；如图 13，当 $\eta = 2$ 时，曲线跳跃区间较小，收敛速度也比较快，且与学习率更大时的均方误差收敛值相差不大。

在实验中，我随机初始化了 4 组参数，并取超参数 $\eta = 0.5$，对每一组参数用标准 BP 算法和累积 BP 算训练神经网络，得到其收敛速率。结合图14、图15、图16、图17，对于同一训练集、超参数相同

的情况下，有如下现象：

1. 标准 BP 算法收敛速度比累积 BP 算法更快；
2. 标准 BP 算法在迭代时抖动比累积 BP 算法更剧烈；
3. 累积误差下降到一定程度后，使用累积 BP 算法进行迭代进一步下降会非常慢，这时标准 BP 算法会更快获得更好的解。

造成上述现象的原因主要是：标准 BP 算法每次针对单个样本进行更新，参数更新得非常频繁，但是对不同样本进行更新的效果可能出现“抵消”的情况，而累积 BP 算法读取整个训练集后才针对总体均方误差进行更新。

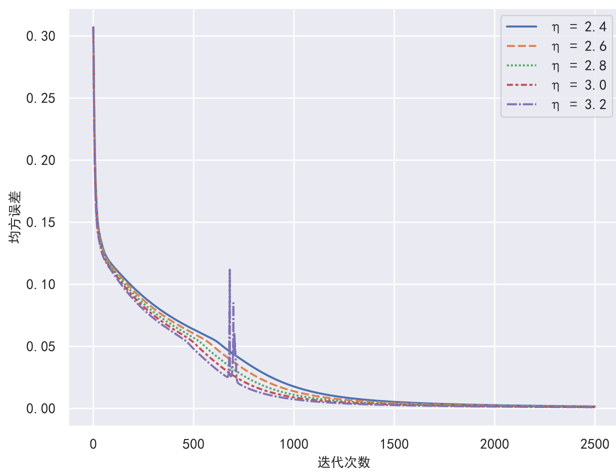

图 9: 累积 BP 算法取不同 η 时的迭代速度（1）

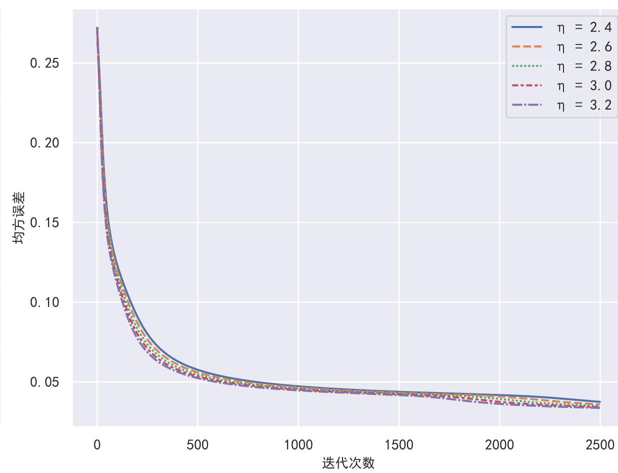

图 10: 累积 BP 算法取不同 η 时的迭代速度（2）

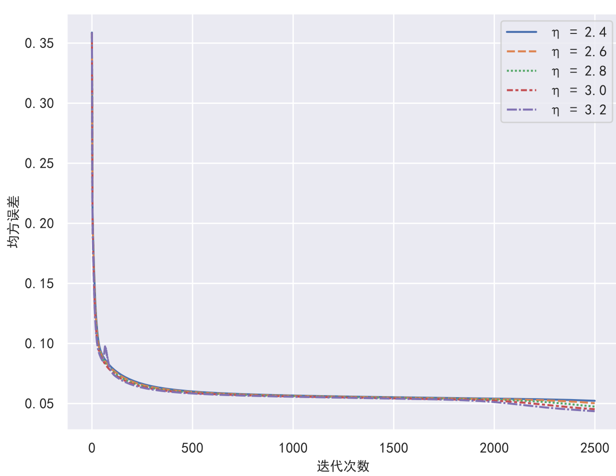

图 11: 累积 BP 算法取不同 η 时的迭代速度（3）

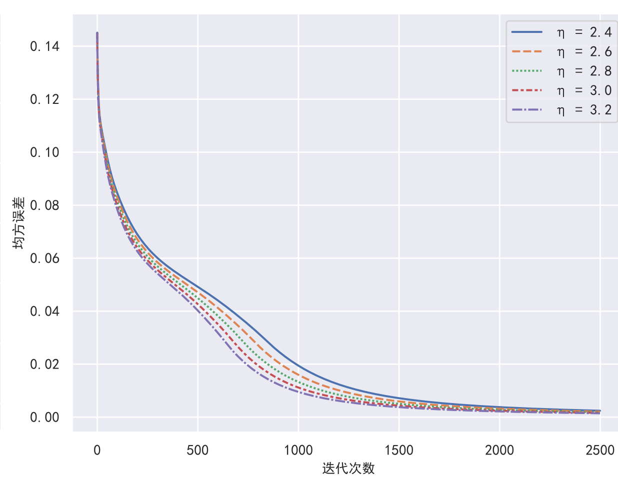

图 12: 累积 BP 算法取不同 η 时的迭代速度（4）

2.6 学习收获

在此次实验中，我实现了累积 BP 算法，并对学习率参数 η 进行了一些研究，得到了迭代收敛速率关于学习率的一些性质。此外，我还对标准 BP 算法和累积 BP 算法进行了比较，其差异类似于随机梯度下降与标准梯度下降之间的区别。

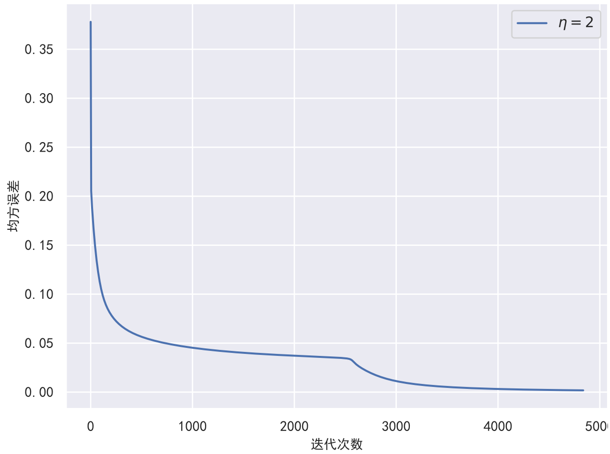

图 13: 累积 BP 算法超参数 η 在西瓜数据集 3.0 上的适宜值

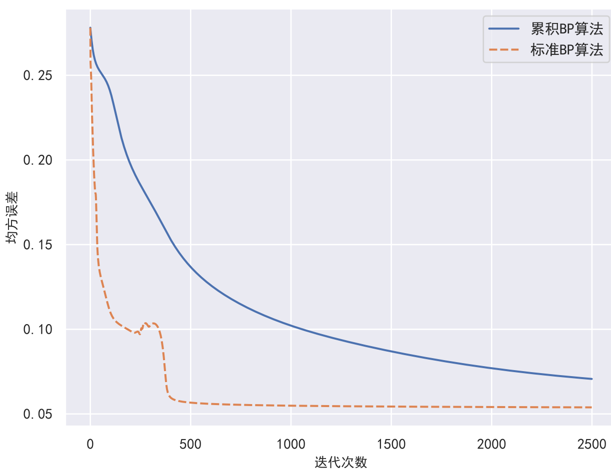

图 14: 比较标准 BP 算法与累积 BP 算法（1）

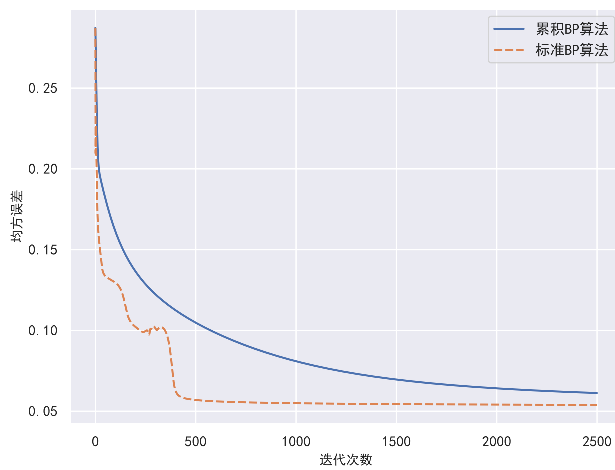

图 15: 比较标准 BP 算法与累积 BP 算法（2）

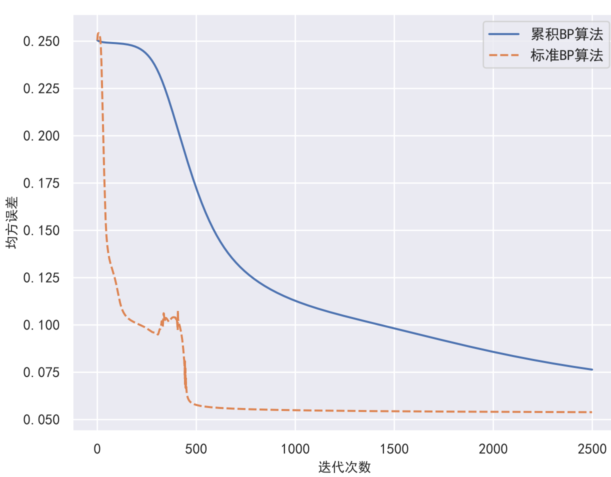

图 16: 比较标准 BP 算法与累积 BP 算法（3）

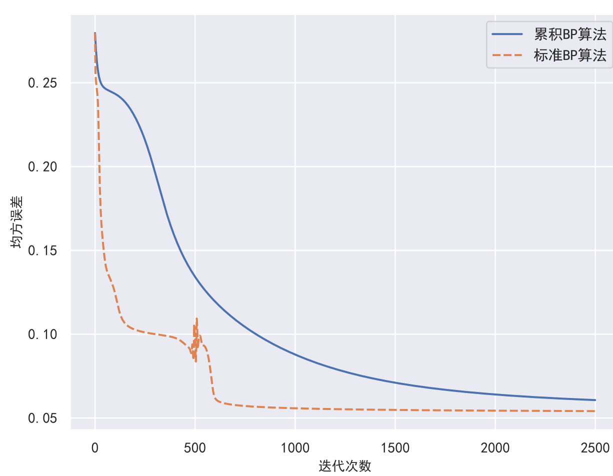

图 17: 比较标准 BP 算法与累积 BP 算法（4）

2.7 参考资料

- 《机器学习》5.3；周志华；清华大学出版社。