哈爾濱工業大學

人工智能数学基础实验报告

题　　目	优化方法的实现与实验
学　　院	计算机科学与技术
专　　业	人工智能
学　　号	2021112845
学　　生	张智雄
任 课 教 师	刘绍辉

哈尔滨工业大学计算机科学与技术学院

2023.5

实验三：优化方法的实现与实验

1、 实验内容或者文献情况介绍

1.1 优化算法的背景

数学优化研究如何寻找函数的最小值或最大值的方法。而在机器学习中，模型的目标是通过从训练数据中学习到的参数来最小化或最大化一个特定的目标函数。这些函数通常是非凸的、高维的，并且可能存在大量的参数。因此，传统的优化方法往往在处理机器学习模型的训练过程中效率低下或无法应对复杂的模型结构。

优化算法(Optimization Algorithm) 能够通过迭代调整模型的参数从而逐步逼近最优解，加快训练过程的收敛速度，降低训练时间和计算资源的消耗。具体可以根据问题的特性选择不同的策略，例如梯度下降、牛顿法、拟牛顿法等。

1.2 实验内容

掌握基本的优化思想及算法，利用梯度下降法、牛顿法求解二项逻辑回归模型，了解常用的各种加速技术的思想。

使用 MNIST 或 CIFAR 数据集用于训练，在训练的优化算法中，采用梯度下降，牛顿法和各种加速技术进行训练速度和轮次，以及精度的比较。

2、 算法简介及其实现细节

2.1 算法简介

梯度下降法(Gradient Descent) 的基本思想是通过迭代地更新模型参数，沿着目标函数的负梯度方向，按照一定步长逐步逼近最优解。

ALGORITHM 1 Gradient Descent (梯度下降)

1：**input** $J(\theta)$ ←目标优化函数，α ←学习率，$iter$ ←迭代次数，θ_0 ←参数初值；

2：$\theta \leftarrow \theta_0$；

3：**while** $i < iter$ **do**

4：　　$gradient \leftarrow \nabla J(\theta)$；

5：　　$\theta \leftarrow \theta - \alpha \times gradient$；

6：**end while**

7：**return** θ //返回最优参数；

而牛顿法(Newton's method)则是利用目标函数的一阶导数（梯度）和二阶导数（海森矩阵）信息在每一次迭代中近似目标函数的局部曲线，来进行参数更新，从而逐步逼近最优解。

ALGORITHM 2 Newton's method (牛顿法)

1：**input** $J(\theta)$ ←目标优化函数， $iter$ ←迭代次数，θ_0 ←参数初值；

2：$\theta \leftarrow \theta_0$；

3：**while** $i < iter$ **do**

4：　　$gradient \leftarrow \nabla J(\theta)$；

5：　　$Hessain \leftarrow \nabla^2 J(\theta)$；

6：　　$direction \leftarrow -Hessain^{-1} \times gradient$

7：　　$\theta \leftarrow \theta + direction$；

9：**end while**

10：**return** θ//返回最优参数；

2.2 逻辑回归理论推导

二项逻辑回归可用于一些简单的二分类问题，不妨设$Z \in R^n$是问题的输入，$X \in \{0,1\}$是输出。则逻辑回归假设$X|Z$服从伯努利分布。具体模型为

$$P(X=1|Z) = \frac{1}{1+e^{-\theta^T z}};\ P(X=0|Z) = 1 - \frac{1}{1+e^{-\theta^T z}}$$

其中，θ^T为模型的参数，$\theta^T z = \theta_0 + \theta_1 z_1 + \cdots + \theta_n z_n$。应用时，根据两个条件概率值的大小将实例分到概率值较大的一类。

和线性回归不同的是，逻辑回归通过$Sigmoid$函数引入了非线性因素，将实数域内的值约束到$(0,1)$的取值范围内，从而可以较为地轻松处理二分类问题。

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

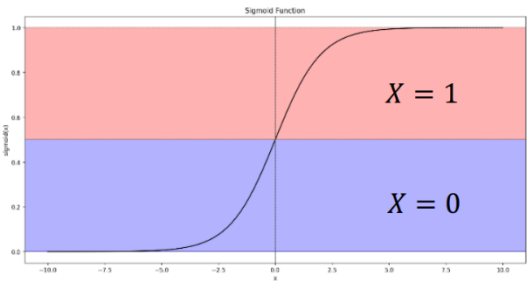

Figure 1 $Sigmoid$函数

令观测函数为$h_\theta(z) = 1/(1+e^{-\theta^T z})$，于是，可以整合上述逻辑回归的概率模型得到输入集合的最大似然估计为$L(\theta) = \prod_{i=1}^{m} P(x_i|z_i;\theta) = (h_\theta(z))^x(1-$

$h_\theta(z))^{1-x}$，取对数，乘$-1/m$将最大值问题转化为最小值问题。

$$L(\theta)=\left(-\frac{1}{m}\right)\prod_{i=1}^{m}\log P(x_i|z_i;\theta)=-\frac{1}{m}\sum_{i=1}^{m}[x_i\log h_\theta(z_i)+(1-x_i)\log\big((1-h_\theta(z_i)\big)]$$

$L(\theta)$是凸的、可微的，因此可以运用梯度下降法、牛顿法及其他方法对此问题进行求解此优化问题。

2.2.1 梯度下降法

对上述$L(\theta)$进行求导，得到其梯度各方向为

$$\frac{\partial L(\theta)}{\partial\theta_j}=\left(-\frac{1}{m}\right)\sum_{i=1}^{m}\left[x_i\frac{1}{h_\theta(z_i)}\cdot\frac{\partial h_\theta(z_i)}{\partial\theta_j}-(1-x_i)\frac{1}{(1-h_\theta(z_i)}\cdot\frac{\partial h_\theta(z_i)}{\partial\theta_j}\right]$$

其中，根据观测函数的定义可以展开$\frac{\partial h_\theta(z_i)}{\partial\theta_j}$为

$$\frac{-1}{\left(1+e^{-\theta^Tz}\right)^2}(e^{-\theta^Tz})\frac{\partial\theta^Tz}{\partial\theta_j}=\text{sigmoid}(\theta^Tz)\cdot\big(1-\text{sigmoid}(\theta^Tz)\big)\frac{\partial\theta^Tz}{\partial\theta_j}$$

将其代入原式，最终可以化简得到$\frac{\partial L(\theta)}{\partial\theta_j}=\frac{1}{m}\sum_{i=1}^{m}(h_\theta(z_i)-x_i)\,z_i^j$。

而后更新参数$\theta_j=\theta_j-\alpha\times\frac{1}{m}\sum_{i=1}^{m}(h_\theta(z_i)-x_i)\,z_i^j,j=1,2,\dots,n$，重复上述过程，即可逐步向局部最优解逼近。

2.2.2 牛顿法

$L(\theta)$二阶导数的泰勒展开式为

$$L(\theta)=L(\theta_k)+(\theta-\theta_k)L'(\theta)+\frac{1}{2}(\theta-\theta_k)^2L''(\theta)$$

函数的极值点在$L'(\theta)=0$时取得，于是上式可以化简为$\theta_{k+1}=\theta_k-L'(\theta)/L''(\theta)$。$L'(\theta)$的具体求解同 2.2.1 节中梯度求值，而$L''(\theta)$则需要对$L(\theta)$的 Hessian 矩阵进行求解，对于函数$L(\theta_1,\theta_2,\dots,\theta_n)$，其 Hessian 矩阵为

$$Hessain=\begin{bmatrix}\frac{\partial^2L}{\partial\theta_1^2} & \cdots & \frac{\partial^2L}{\partial\theta_1\partial\theta_n}\\ \vdots & \ddots & \vdots\\ \frac{\partial^2L}{\partial\theta_n\partial\theta_1} & \cdots & \frac{\partial^2L}{\partial\theta_n^2}\end{bmatrix}$$

如果函数$L(\theta)$二阶连续可导，且$L''(\theta)\neq0$，则 Hessian 矩阵为对称矩阵，求导顺序不影响结果，因而可以变换为如下形式

$$Hessain=\frac{1}{m}(Z^T\cdot diag\big(h_\theta(z_i)\big)\cdot diag\big(1-h_\theta(z_i)\big)\cdot Z^T)$$

而后更新参数$\theta=\theta-\left(\frac{1}{m}(h_\theta(z)-x)z\right)/Hessain$，重复迭代，相较于梯度下降法，牛顿法能够更加快速的收敛到极值点。

2.3 其他加速算法

在 2.1 节中，梯度下降的迭代格式为$x^{k+1} = x^k - \alpha_k \nabla f(x^k)$，其中$\nabla f(x^k) = \sum_{i=1}^{N} \nabla f_i(x^k)/N$，在绝大部分情况计算梯度需要计算所有的$\nabla f_i(x^k), i = 1,2,\dots,N$并相加，在面对海量的数据时运算速度缓慢，不适合完成一些机器学习的任务。因而提出了其他一些的加速算法，如动量方法 (Momentum)、随机梯度下降法 (Stochastic Gradient Descent)、Nesterov 加速算法等。

2.3.1 随机梯度下降法

随机梯度下降法(SGD)使用元素个数较少的集合$\mathcal{T}_k$的梯度代替了全梯度，并且每次迭代选取的样本点是随机的，这使得计算得到的梯度期望 $\mathbb{E}_{s_k}$等于全梯度。而每次迭代时计算梯度的复杂度变为了原先的k/N，极大地减小了时间开销。

$$x^{k+1} = x^k - \frac{\alpha_k}{|\mathcal{T}_k|} \sum_{s \in \mathcal{T}_k} \nabla f_s(x^k),\ \mathbb{E}_s[\nabla f_s(x^k)|x^k] = \nabla f(x^k)$$

2.3.2 动量方法

动量方法(Momentum)在算法迭代时一定程度上保留之前更新的方向，同时利用当前计算的梯度调整最终的更新方向，可以在一定程度上增加稳定性，在处理高曲率或是带噪声的梯度上非常有效，并且还有一定摆脱局部最优解的能力。

具体而言，动量方法引入了一个速度变量v，它代表参数移动的方向和大小。$\mu_k \in [0,1)$，表示迭代点带有的惯性，当$\mu_k = 0$时退化为随机梯度下降法。

$$v^{k+1} = \mu_k v^k - \alpha_k \nabla f(x^k)$$
$$x^{k+1} = x^k + v^{k+1}$$

直观上看，当许多连续的梯度指向相同的方向时，步长就会很大，因而收敛速度更快；而梯度存在噪声时，动量法也能够在一定程度上进行修正。

2.3.3 Nesterov 加速算法

Nesterov 加速算法也是一种动量方法，其先对点施加速度的作用，再求梯度，这可以理解为对标准动量方法做了一个校正。其中，参数$\mu_k = (k-1)/(k+2)$。其具体迭代形式为

$$v^{k+1} = \mu_k v^k - \alpha_k \nabla f(x^k + \mu_k v^k)$$
$$x^{k+1} = x^k + v^{k+1}$$

2.3.4 AdaGrad 方法

AdaGrad(Adaptive Subgradient Methods)能够在运行的过程中，根据当前情况自发地调整参数α_k，适应当前下降的梯度，能够一定程度上减小参数设置对算法性能的影响，提高收敛速度。

令$g^k = \nabla f_s(x^k)$，为了记录整个迭代过程中梯度各个分量的累积情况，引入向量G^k，其分量表示梯度在该处的累积平方和。

$$G^k = \sum_{i=1}^{k} g^i \odot g^i$$

则当G^k的某分量较大时，我们认为该分量变化比较剧烈，因此应采用小步长，反之逆然。因此 AdaGrad 的迭代格式为

$$x^{k+1} = x^k - \frac{\alpha_k}{\sqrt{G^k + \varepsilon_n}} \odot g^k$$

$$G^{k+1} = G^k + g^{k+1} \odot g^{k+1}$$

直观上理解，AdaGrad 的步长Δx大致反比于历史梯度累计值G^k的算术平方根，但实际应用中，AdaGrad 从训练开始就积累梯度平方会导致步长Δx过早或过多减小，增大了计算负荷。

AdaGrad 也可以看成是一种介于一阶和二阶的优化算法，考虑$f(x)$在点x^k处的二阶泰勒展开

$$f(x) \approx f(x^k) + \nabla f(x^k)^T (x - x^k) + \frac{1}{2}(x - x^k)^T B^k (x - x^k)$$

选取不同的B^k可以导出不同的优化算法。与一阶近似的梯度法和二阶近似的牛顿法做比较，梯度法选用常数倍单位矩阵近似B^k；牛顿法B^k为海瑟矩阵；而 AdaGrad 则是使用一个对角矩阵来作为B^k，具体地，取

$$B^k = \frac{1}{\alpha} diag(\sqrt{G^k + \varepsilon_n})$$

时导出的算法就是 AdaGrad。

2.3.5 RMSProp 方法

RMSProp(Root Mean Square Propagation)是对 AdaGrad 的一个改进，引入了衰减参数ρ，主要考虑离当前迭代点比较近的项，而忽略距离迭代点较远项的影响。即将G^k修正为M^k，而其他部分未做修改。

$$M^{k+1} = \rho M^k + (1 - \rho) g^{k+1} \odot g^{k+1} (\rho < 1)$$

2.3.6 AdaDelta 方法

AdaDelta 在 RMSProp 的基础之上，对历史的Δx^k也同样累积平方D^k并求均方根，并使用D^{k-1}和M^k平方根的商对梯度进行校正，其具体迭代形式为

$$\begin{cases} M^k = \rho M^{k-1} + (1 - \rho) g^k \odot g^k \\ \Delta x^k = -\dfrac{\sqrt{D^{k-1} + \varepsilon_n}}{\sqrt{M^k + \varepsilon_n}} \odot g^k \\ D^k = \rho D^{k-1} + (1 - \rho) \Delta x^k \odot \Delta x^k \\ x^{k+1} = x^k + \Delta x^k \end{cases}$$

需要注意的是，由于Δx^k的产生滞后梯度一步，因而计算步长时D^{k-1}和M^k平方根的下标相差 1。AdaDelta 的特点是步长选择较为保守，同时也改善了 AdaGrad 步长单调下降的缺陷。

2.3.7 Adam 方法

Adam(Adaptive Moment Estimation)本质上是带动量项的 RMSProp，可以看成是带修正的动量法和 RMSProp 法的结合，它利用梯度的一阶矩估计和二阶矩估计动态调整每个参数步长。

在 Adam 方法中不直接使用随机梯度作为基础的更新方向，而是选择了一个动量项进行更新。

$$S^k = \rho_1 S^{k-1} + (1-\rho_1) g^k$$

同时也会记录迭代过程中梯度的二阶矩

$$M^k = \rho_2 M^{k-1} + (1-\rho_2) g^k \odot g^k$$

但是，由于S^k和M^k本身带有偏差，需要对其进行修正

$$\hat{S}^k = \frac{S^k}{1-\rho_1^k}, \qquad \widehat{M}^k = \frac{M^k}{1-\rho_2^k}$$

其中ρ_1^k, ρ_2^k分别表示ρ_1, ρ_2的k次方，Adam 最终使用修正后的一阶矩和二阶矩进行迭代点的更新。

$$x^{k+1} = x^k - \frac{\alpha_k}{\sqrt{\widehat{M}^k + \varepsilon_n}} \odot \hat{S}^k$$

2.3.8 Lion 方法

除了上述的人工引入优化器方法外，还有一个方向是程序自动发现优化算法，即通过训练神经网络来发现优化器。

Lion(EvoLved Sign Momentum)即为程序自动发现的一种优化方法。与 AdamW（带有正则项的 Adam 方法）需要同时保存一阶和二阶矩相比，Lion 只需要跟踪动量并利用符号操作来计算更新，能够有效减少内存占用，提高收敛速度。其具体迭代形式如下，

$$\begin{cases} C^k = \rho_1 M^{k-1} + (1-\rho_1) g^k \\ x^k = x^{k-1} - \eta^k (\mathrm{sign}(C^k) + \lambda x^{k-1}) \\ M^k = \rho_2 M^{k-1} + (1-\rho_2) g^k \end{cases}$$

直观上看，Lion 算法通过符号操作在所有维度上产生了具有统一幅度的更新，并且符号操作为更新添加了噪声，作为了一种正则化形式并有助于泛化。

3、 实验设置及结果分析（包括实验数据集）

3.1 Iris 数据集上的二项逻辑回归

鸢尾花数据集(Iris dataset)包含了来自三个不同种类(Iris setosa, Iris versicolor, Iris virginica)的鸢尾花(Iris)的观测数据，每个种类有 50 个样本。每个样本由包括花萼长度(sepal length)、花萼宽度(sepal width)、花瓣长度(petal length)和花瓣宽度(petal width)四个特征描述。

实验保留了 Iris-setosa 和 Iris-versicolor 两个类别的 100 组数据进行二项逻辑回归。首先按照 8:2 的比例对数据进行训练集和测试集的划分，而后分别使用梯度下降法和牛顿法对训练集进行近似。

实验观察损失函数下降的曲线发现，选取学习率为 0.001 时，梯度下降的效果较为明显。而牛顿法使用了二阶导数的信息来确定参数更新的方向和步长，可以根据损失函数的曲率来自适应地调整步长，因此不需要学习率来控制收敛速度。

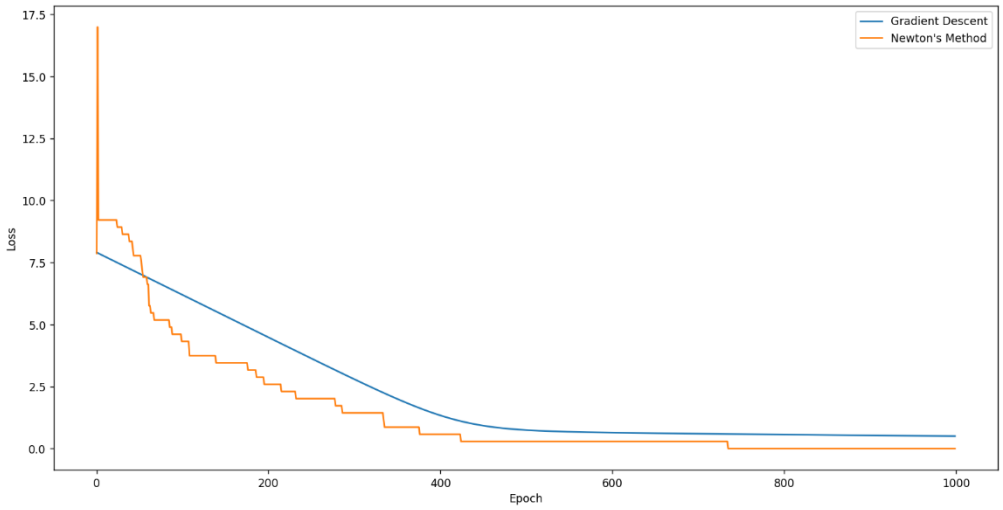

Figure 2 损失函数随迭代次数的变化
（蓝线为梯度下降，橙色为牛顿法）

从图像中显而易见可以看到，牛顿法收敛的速度相较于梯度下降更快，在迭代次数为 1000 时，梯度下降的损失函数仍大于牛顿法。

观察曲线还发现，使用牛顿法初期还存在不稳定的情况，这可能是由于初始参数选择的不好的缘故。而其损失函数变化曲线呈阶梯式的原因可能是数据集较小，参数已经接近最优解，梯度变得非常小，导致更新的幅度减小。

而后观察在测试集上的正确率随迭代变化的曲线，由于测试集只有 20 组，数量较少，因而正确率呈现阶梯式上升。但依然可以观察到，牛顿法能够更快地到达极值点，在第 500 次迭代时牛顿法就有 95%的正确率，而此时梯度法的正确率只有 50%左右。

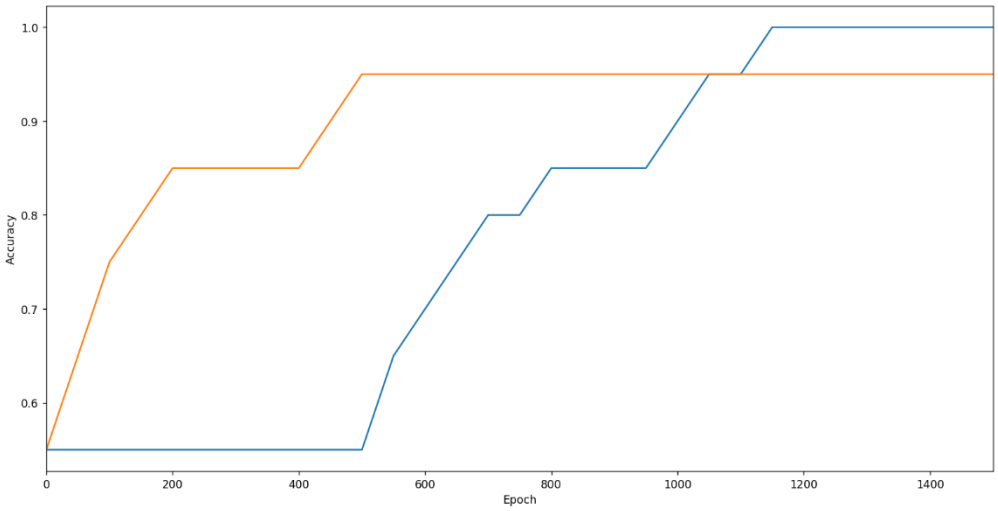

Figure 3 正确率随迭代次数的变化

但是，最终牛顿法的正确率无法到达 100%的原因可能是，数据样本较少，且可能存在噪声和异常值。而牛顿法由于使用了二阶导数的信息，对数据样本的依赖更强，因而鲁棒性弱于梯度法。

观察分类的具体结果（选取第 1、2 个特征进行可视化），发现分类结果基本一致，决策面直观上差距不大，符合两种方法均对梯度方向进行收敛的论断。

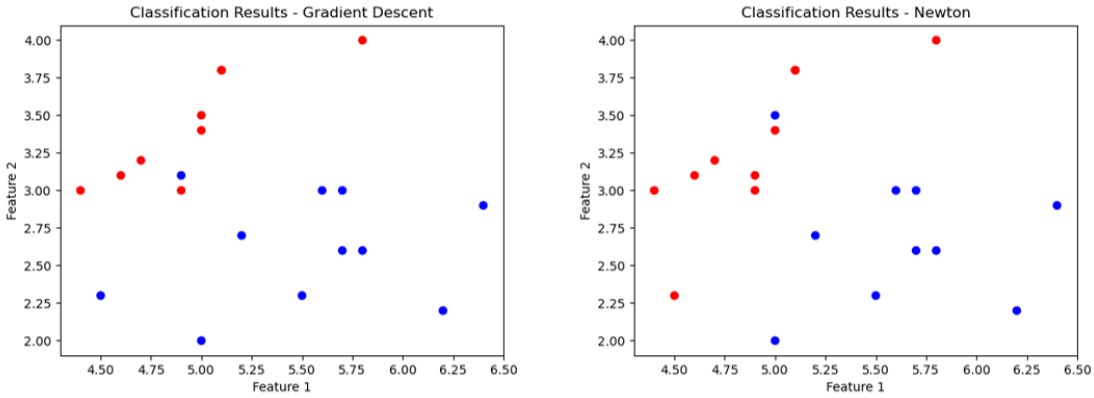

Figure 4 两种方法在测试集上的预测结果（epoch=1000）

3.2 Dry-Beans 数据集上的二项逻辑回归

Dry-Beans 数据集是 Kaggle 网站上的关于菜豆的一个数据集，包括菜豆的区域，周长，长轴长、短轴长等指标，在本节中选取 SEKER 和 BARBUNYA 两个品种（3349 组），选择 MajorAxisLength 和 MinorAxisLength 两项指标对菜豆进行二项逻辑回归分析，同样选择 20%的数据作为测试集。

由于牛顿法在此问题中求解 hessian 矩阵时会遇到奇异矩阵的情况，故本节中只有梯度下降的相关分析，也可换用拟牛顿法，如 BFGS 算法来解决此问题。

首先观察不同学习率下的 loss 下降曲线，当学习率为 0.01 时出现明显振荡现象，而当学习率为 0.0001 时，则出现长时间不变化的现象。原因有可能是 0.01

的步长较大，函数在极值点附近反复逼近；而 0.0001 时步长较小，在初期初值选取的不好时，参数的变化对模型的影响不大。

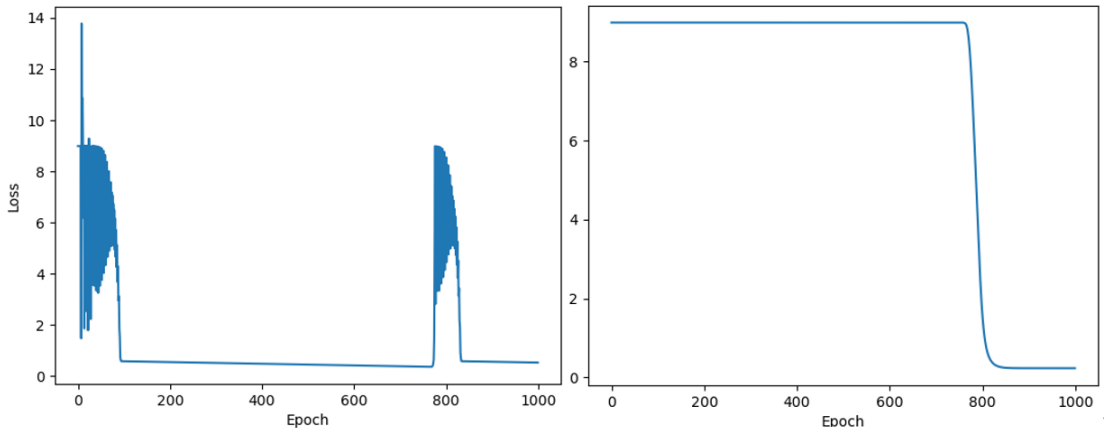

Figure 5 损失函数随迭代次数的变化
（左图 lr=0.01，右图 lr=0.0001）

观察最后分类结果，正确率为 93.43%，在图中也能较为直观的找到决策面，具体预测结果如下：

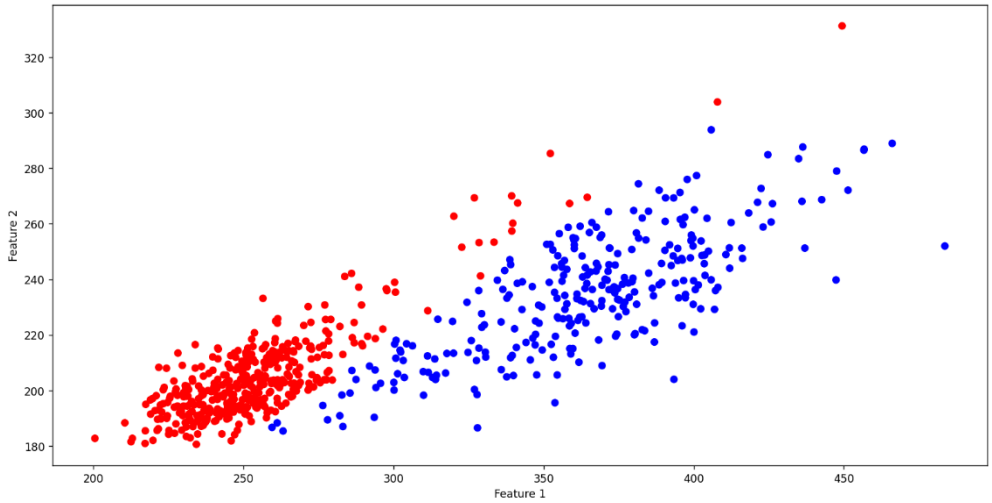

Figure 6 测试集上的预测结果（epoch=1000）

3.3 牛顿法对于非线性问题的求解

牛顿法还可以用以求解无约束非线性规划问题，或者包含等式约束的非线性规划问题（可以使用拉格朗日乘子法转化为一个无约束优化问题）。而对于包含不等式约束的非线性规划问题，则考虑采用其他优化算法。

其基本思想都是采用目标函数的一阶和二阶导数，通过迭代逐步逼近最优解。

$$L(\theta) = L(\theta_k) + (\theta - \theta_k)L'(\theta) + \frac{1}{2}(\theta - \theta_k)^2 L''(\theta)$$

本节中，首先对单变元函数如$0.5x^3 + 4x^2 + 3x + 1$，随机给一个初值，使用牛顿法对其极大（小）值进行求解，标记函数在迭代过程中的轨迹如下。为了方便观察，缩小其步长为计算得到的$1/10$。

可以发现，牛顿法总是会选择逼近距离初值最近的一个极值点。

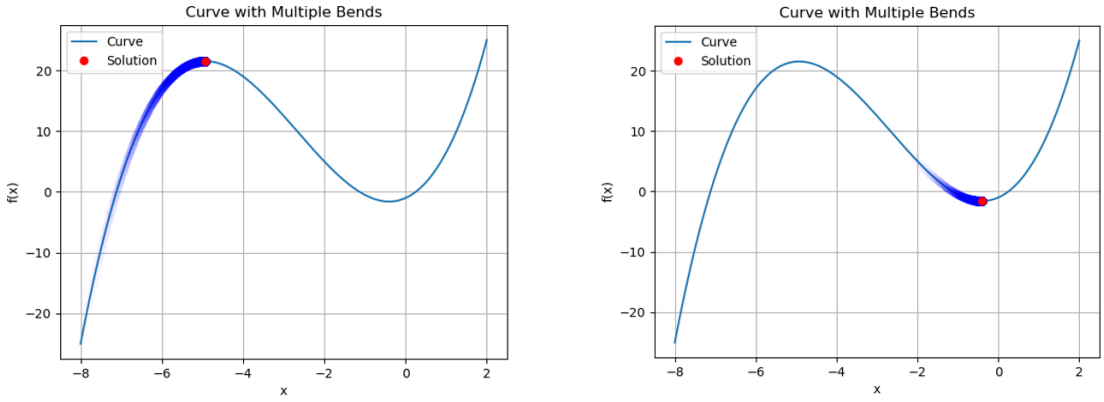

Figure 7 高次方程的牛顿法逼近

（左图$x_0 = -8$，右图$x_0 = -2.5$）

再在$\sin x + \sin 2x$函数曲线上使用牛顿法进行极值逼近，结果如下。

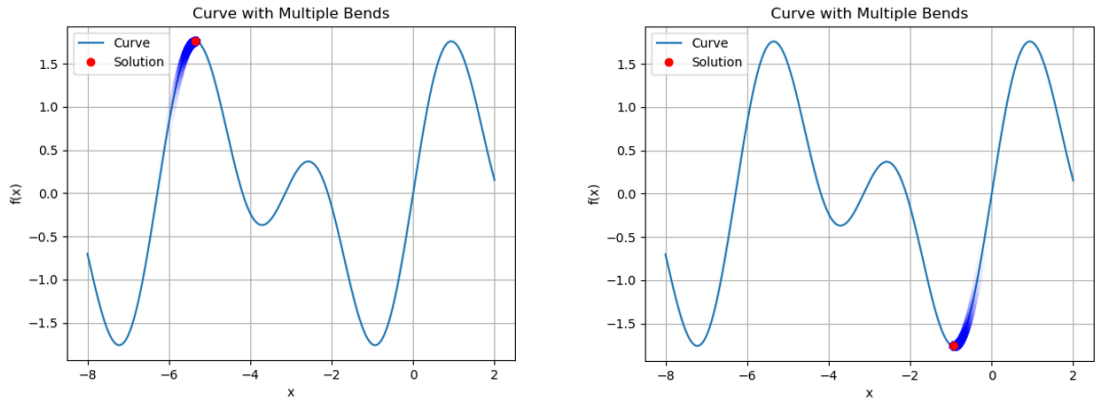

Figure 8 *sin* 函数方程的牛顿法逼近

（左图$x_0 = -6$，右图$x_0 = -0.1$）

同理，对于有多个变元的非线性规划问题，也可以选用牛顿法对其优化解进行求解。而一般需要求解其一阶导数 Jacobian 矩阵和二阶导数 Hessian 矩阵，实际应用中可以直接给出求导定义，也可以进行数值近似计算。

对函数$x_1^4 + 25x_2^4 - 2x_1^2x_2^2$求其极小值，标记函数在迭代过程中的轨迹如下。

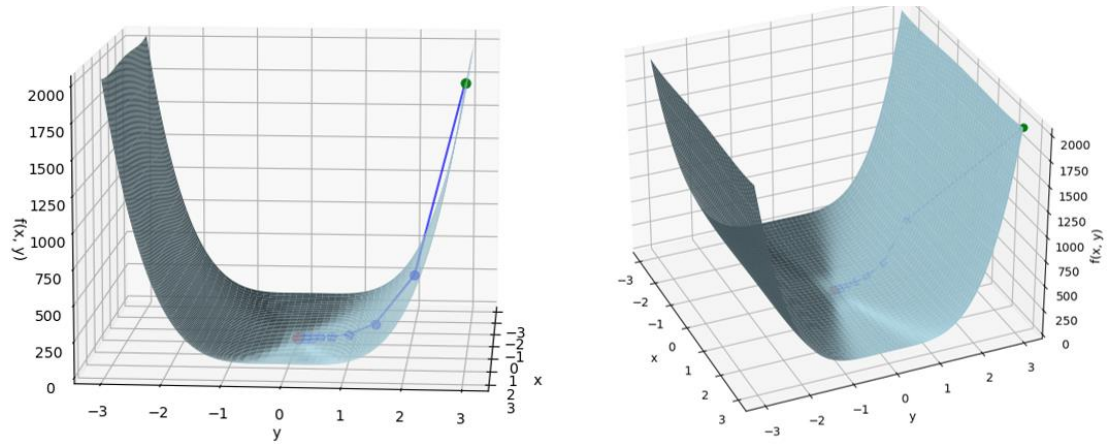

Figure 9 二元非线性规划的牛顿法逼近

（绿点为初始点，红点为最终取值，蓝点为迭代过程轨迹）

不难发现，牛顿法会先对各变量的梯度步长都进行调整，选择最优最快的收敛路径。但当目标函数存在奇点或 Hessian 矩阵不可逆时，牛顿法常常表现出不稳定性，这时候可以考虑采用 BFGS、L-BFGS 等拟牛顿法近似。

3.4 不同优化器性能的比较

在 2.3 节中对不同优化器进行了理论上的分析，而在本节对 MNIST 数据集使用一个简单的线性神经网络模型进行训练，分析并比较不同优化器下的训练速度和效果。

实验比较了 SGD、ASGD、Adagrad、RMSprop、Adadelta、Adam、AdamW 七种优化器在学习率为 0.01、0.001 和 0.0001 时的训练速度和效果，比较 loss 损失函数随迭代次数的变化结果如下

可以发现各优化器均能够有效下降 loss 函数，但是在不同学习率下，下降的速率不同，最终的精度也不同。

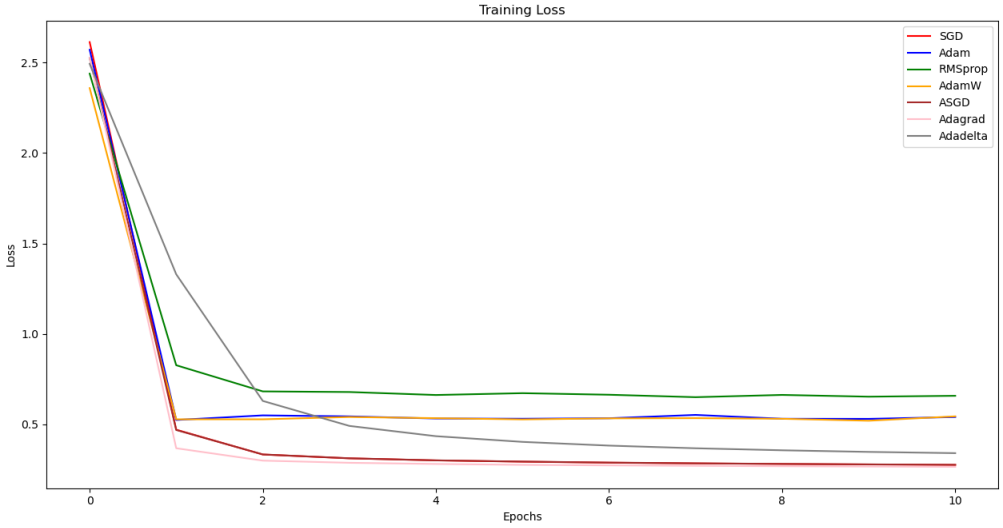

Figure 10 优化器性能比较（lr=0.01）

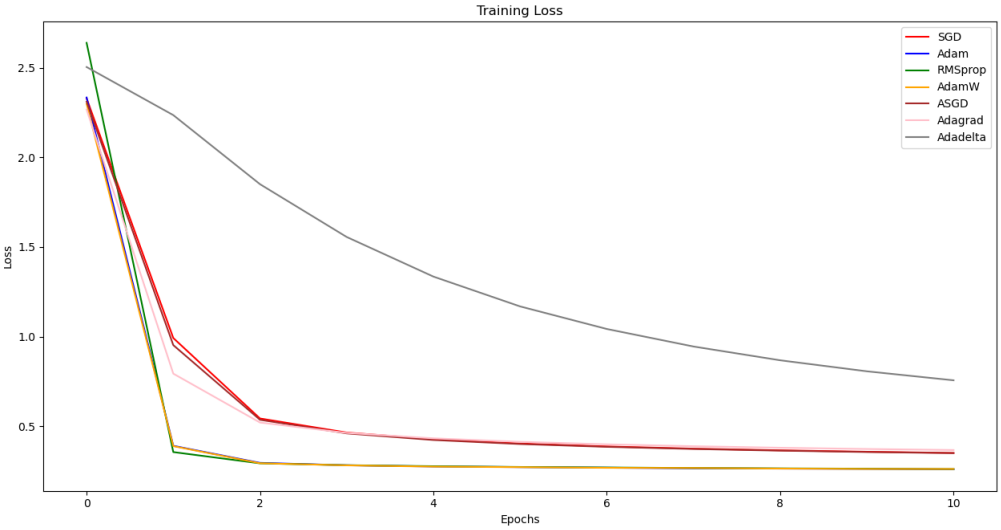

Figure 11 优化器性能比较（lr=0.001）

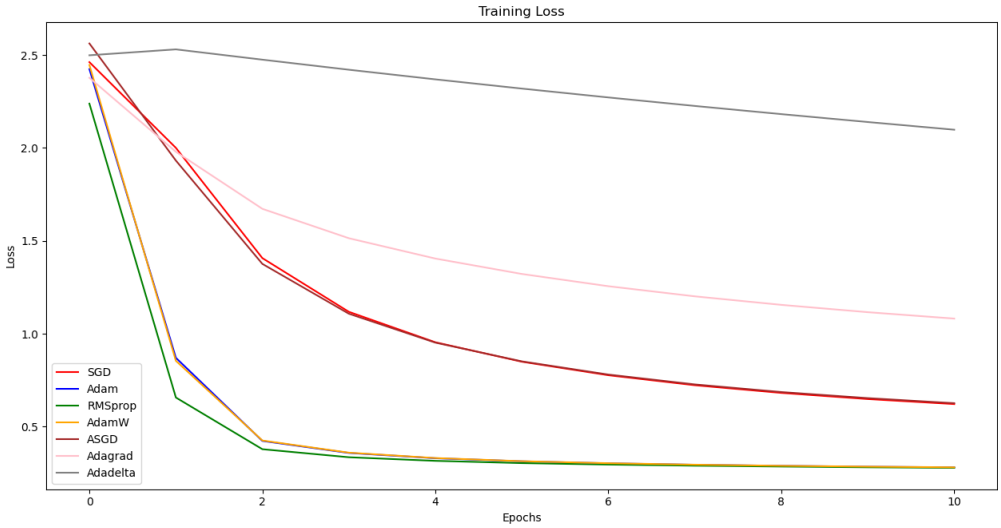

Figure 12 优化器性能比较（lr=0.0001）

比较优化器在不同学习率下的最终精度（epoch=10）如下表，发现不同优化器可能对学习率的变化有不同的敏感度，如 SGD、ASGD、Adagrad、Adadelta 随学习率增大而效果变差，其中 Adagrad 和 Adadelta 尤为明显，可能是因为它们步长选择较为保守，并且步长随迭代单调减少的特性；而 RMSprop、Adam、AdamW 在学习率为 0.001 时效果最佳。

Table 1 优化器性能比较

	lr=0.01	**lr=0.001**	**lr=0.0001**
SGD	0.2755	0.3510	0.6218
ASGD	0.2756	0.3499	0.6265
Adagrad	0.2637	0.3671	1.0817
RMSprop	0.6494	0.2609	0.2785
Adadelta	0.3388	0.7562	2.0980
Adam	0.5377	0.2595	0.2811
AdamW	0.5132	0.2614	0.2814

并且由上述图表还能发现，不同优化器最终收敛时的 loss 值仍存在一定差距，这可能是由于有些优化器可能更容易陷入局部最优解，而有些优化器可能能够更好地逃离鞍点并找到更好的解。

并且，优化器的性能可能还由问题本身和数据集的特点决定，不同的算法可能对于不同类型的问题和数据集表现更好或更差。

4、 结论

梯度下降法在求解二项逻辑回归问题上有着不错的表现，但是可能会陷入到局部最优解而非全局最优解。牛顿法能有更快的收敛速度以及更精确的求解，但是对噪声较为敏感，并且可能会遇到奇异矩阵等问题难以求解，实际应用中常使用 L-BGFS 等拟牛顿法。

牛顿法还可以用于求解无约束或等式约束的非线性规划问题，能够较为迅速地收敛到问题的极值点。

针对模型的训练，不同优化器在损失函数的下降速率和效果上表现不同，原因包括不同优化器可能对学习率的变化有不同的敏感度、有些优化器可能更容易陷入局部最优解而有些优化器可能能够更好地逃离鞍点并找到更好的解以及不同优化器适合求解不同的问题和数据集。

5、 参考文献

[1]周志华. 机器学习[M]. 清华大学出版社, 2016.

[2]杜子芳. 多元统计分析[M]. 清华大学出版社, 2016.

[3]陈宝林. 最优化理论与算法[M].清华大学出版社,2005.

[4]Chen X, Liang C, Huang D, et al. Symbolic discovery of optimization algorithms[J]. arXiv preprint arXiv:2302.06675, 2023.

[5]模式识别——牛顿法实现逻辑回归 https://zhuanlan.zhihu.com/p/63305895

[6]机器学习(超详细讲解)：利用 Logistic Regression(逻辑回归)实现多分类 https://blog.csdn.net/qq_44350242/article/details/112372860