哈爾濱工業大學

人工智能数学基础实验报告

题　　目	优化算法的实现
学　　院	计算机科学与技术
专　　业	人工智能
学　　号	2022113416
学　　生	刘子康
任 课 教 师	刘绍辉

哈尔滨工业大学计算机科学与技术学院

2024.3

实验三：利用梯度下降法、牛顿法求解无约束优化问题

一、 实验内容或者文献情况介绍

最优化问题在机器学习中有着非常重要的地位，很多机器学习算法最后都归结为求解最优化问题。最优化问题即求解函数极值的问题，包括极大值和极小值。掌握梯度下降法和牛顿法的优化思想及算法，掌握 Lion 基于学习的加速方法的思想，掌握函数逼近的思想，并使用这些基本的方法来求解实际问题。

梯度下降法和牛顿法均是通过迭代的方式找到目标函数的最小值（极小值）的方法，分别采用函数的一阶近似和二阶近似，而函数的 n 阶近似基于函数在某点的泰勒展开式 $f(x)=\frac{f(x_0)}{0!}+\frac{f'(x_0)}{1!}(x-x_0)+\frac{f''(x_0)}{2!}(x-x_0)^2+\cdots+\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n+R_n(x)$实现，其中$f^{(n)}(x)$表示函数$f(x)$的 n 阶导数，$R_n(x)$是泰勒公式的余项，表示$(x-x_0)^n$的高阶无穷小。

泰勒公式利用一个函数在某点的信息，描述其附近取值。如果函数足够平滑，且函数在某点的各阶导数值已知，便可以将这些导数值作为系数，构建一个多项式近似函数，求得在这一点的邻域中的值，即用一个多项式函数逼近给定的函数。泰勒公式在机器学习中常用于梯度迭代。

由于机器学习中存在很多复杂的难求解方程（如超越方程），直接计算目标函数的导数或梯度，令其为 0 并解方程求得精确解的方法在机器学习中并不适用，因此常采用迭代法求得近似解。

二、 算法简介及其实现细节

2.1 梯度下降法

2.1.1 算法简介

梯度下降法是机器学习中求解无约束优化问题最常采用的方法之一，基于一阶泰勒展开来近似x_0附近的目标函数$f(x)$，基本思想是：通过不断的梯度下降找到目标函数最小化时取值对应的自变量 x，在迭代过程中，它从一个初始点x_0开始，反复根据某种规则移动到下一个点，直到收敛到梯度为 0 的极小值点处。

梯度下降法好比下山，从山顶开始向下走，每次都选择最陡峭的方向，直到抵达山谷，而频繁选择方向会耗费大量时间，选择方向过少又可能会偏离轨道，因此需要找到一个合适的频率，即为梯度下降法中的“步长”。

2.1.2 实现细节

首先确定迭代停止准则，当目标函数在当前迭代点x_k处的梯度向量模长满足$\|\nabla f(x_k)\|\leq\varepsilon$时停止迭代并返回结果。然后选定初始点$x_0$，设定最大迭代次数，

通过迭代式$x_{k+1} = x_k - \lambda_k H_k g_k$，其中$g_k = \nabla f(x_k)$表示目标函数在迭代点$x_k$处的梯度向量，$H_k$表示对梯度向量做偏转的矩阵，一般为对称正定矩阵，梯度下降法中H_k为单位矩阵。

在每次迭代中，计算目标函数在x_k点的梯度向量，并选择可行下降方向p_k，p_k可选为负梯度方向或任意与负梯度方向夹角小于 90 度的方向，然后通过线搜索或某规则（如 Goldstein 规则）选取步长λ_k，使得$\begin{cases} x_{k+1} = x_k + \lambda_k p_k \\ f(x_k + \lambda_k p_k) < f(x_k) \end{cases}$，判断是否满足停止条件，并进入下一次迭代。

2.2 牛顿法

2.2.1 算法简介

牛顿法是求解无约束优化问题最早使用的经典方法之一，其基于二阶泰勒展开来近似x_0附近的目标函数$f(x)$，基本思想是：用迭代点x_k处的一阶导数（梯度）和二阶导数（Hessian 矩阵）的逆矩阵对目标函数进行二次函数近似，然后把二次模型的极小值点作为新的迭代点，并不断重复这一过程，直至求得满足精确度的近似极小值点。

相较于梯度下降法，牛顿法采用二阶收敛，不仅会考虑坡度是否够大，还会考虑走一步后坡度是否会变得更大，因此求解无约束优化问题需要的迭代次数更少，速度更快。

2.2.2 实现细节

牛顿法的实现与梯度下降法类似，区别在于迭代式$x_{k+1} = x_k - \lambda_k H_k g_k$中$H_k = (\nabla^2 f(x_k)^{-1}$，是目标函数在迭代点$x_k$处的二阶导数矩阵（即 Hessian 矩阵）的逆矩阵。

三、 实验设置及结果分析（包括实验数据集）

3.1 实验设置

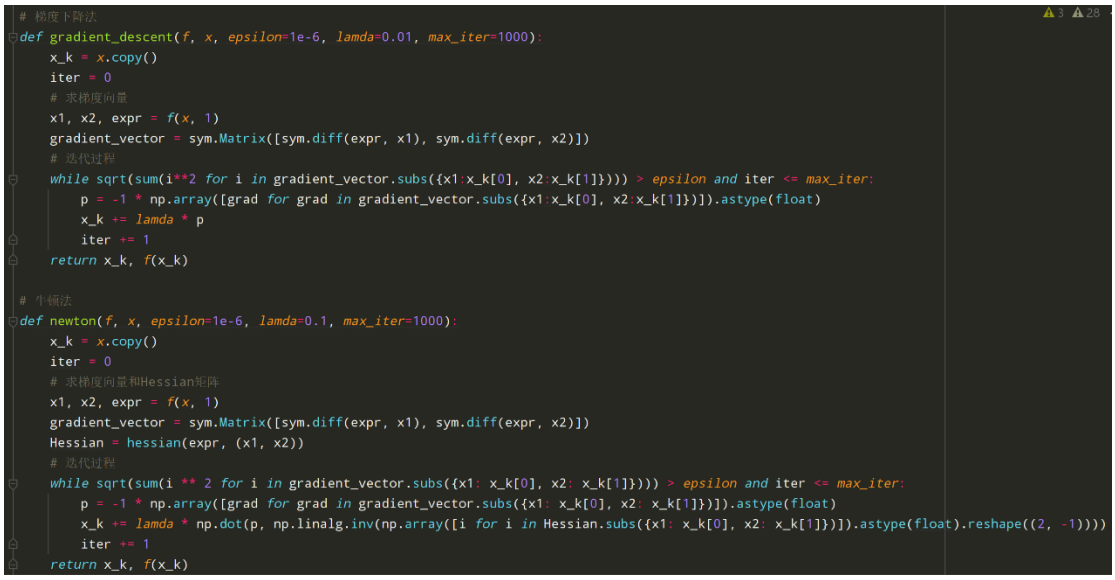
实验使用的 Python 库为 Numpy 库、SymPy 库、math 库、Scipy 库和 Torch 库等，测试函数为 Booth 函数、Rosenbrock Banana 函数，使用 Numpy 库函数生成随机初始点x_0，然后运行梯度下降法、牛顿法、Scipy 库优化函数和 PyTorch 库优化函数 SGD。

Booth 函数：$f(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$

Rosenbrock Banana 函数：$f(x) = (1 - x_1)^2 - 5(x_2 - x_1^2)^2$

3.1.1 梯度下降法和牛顿法

以测试 Booth 函数为例，设定步长 λ 为 0.01（梯度下降法）/0.1（牛顿法），最大迭代次数 max_iter 为 1000，传入目标函数$f(x)$和初始点x_0。使用 SymPy 库函数求解梯度向量（牛顿法中还需求解 Hessian 矩阵），构造一个 while 循环体，退出条件为当前迭代点的梯度向量模长小于给定精度 ε（默认值为 1e-6）或达到最大迭代次数，每次循环中下降方向为负梯度方向，并更新迭代点。



```python
# 梯度下降法
def gradient_descent(f, x, epsilon=1e-6, lamda=0.01, max_iter=1000):
    x_k = x.copy()
    iter = 0
    # 求梯度向量
    x1, x2, expr = f(x, 1)
    gradient_vector = sym.Matrix([sym.diff(expr, x1), sym.diff(expr, x2)])
    # 迭代过程
    while sqrt(sum(i**2 for i in gradient_vector.subs({x1:x_k[0], x2:x_k[1]}))) > epsilon and iter <= max_iter:
        p = -1 * np.array([grad for grad in gradient_vector.subs({x1:x_k[0], x2:x_k[1]})]).astype(float)
        x_k += lamda * p
        iter += 1
    return x_k, f(x_k)

# 牛顿法
def newton(f, x, epsilon=1e-6, lamda=0.1, max_iter=1000):
    x_k = x.copy()
    iter = 0
    # 求梯度向量和Hessian矩阵
    x1, x2, expr = f(x, 1)
    gradient_vector = sym.Matrix([sym.diff(expr, x1), sym.diff(expr, x2)])
    Hessian = hessian(expr, (x1, x2))
    # 迭代过程
    while sqrt(sum(i ** 2 for i in gradient_vector.subs({x1: x_k[0], x2: x_k[1]}))) > epsilon and iter <= max_iter:
        p = -1 * np.array([grad for grad in gradient_vector.subs({x1: x_k[0], x2: x_k[1]})]).astype(float)
        x_k += lamda * np.dot(p, np.linalg.inv(np.array([i for i in Hessian.subs({x1: x_k[0], x2: x_k[1]})]).astype(float).reshape((2, -1))))
        iter += 1
    return x_k, f(x_k)
```

图 1：梯度下降法和牛顿法部分代码

3.1.2 Python 库函数

使用 Scipy 库的 minimize 函数进行优化，传参为目标函数$f(x)$和初始点x_0，返回结果信息，输出极小值点和最优值。

使用 PyTorch 的 SGD 优化器进行优化，传参为目标函数$f(x)$，设定学习率 lr 为 0.01 和最大迭代次数为 1000（以 Booth 函数为例），根据目标函数变元数量创建张量作为初始值，每次迭代先清除之前累积的梯度，然后计算梯度并更新模型参数，最后使模型向前传播。

```python
# 使用PyTorch的SGD优化器
def pytorch_SGD(f, lr=0.01, max_iter=1000):
    x1 = torch.tensor([np.random.uniform(-10, 10)], requires_grad=True)
    x2 = torch.tensor([np.random.uniform(-10, 10)], requires_grad=True)
    params = [x1, x2]
    y = f(params)
    optimizer = optim.SGD(params, lr=lr)
    # 梯度下降迭代过程
    for i in range(max_iter):
        optimizer.zero_grad()
        y.backward()
        optimizer.step()
        y = f(params)
        # if (i + 1) % 10 == 0:
        #     print(f"Iteration {i + 1}, x = {x.item()}, f(x) = {y.item()}")

    return np.array([x1.item(), x2.item()]), y.item()
```

图 2：PyTorch 的 SGD 优化器代码

3.2 结果分析

3.2.1 Booth 函数（极小值点(1, 3)，最优值 0）

·梯度下降法步长$\lambda = 0.01$，最大迭代次数$max_iter = 1000$

·牛顿法步长$\lambda = 0.1$，最大迭代次数$max_iter = 1000$

·随机梯度下降(SGD)学习率$lr = 0.01$，最大迭代次数$max_iter = 1000$

可以看到，四种方法的测试结果均很接近真实值，且在当前设定参数下梯度下降法和牛顿法的实现效果比库函数更好。

```
初始点：[-4.10669995  0.61173511]
梯度下降法 极小点：[0.99999965 3.00000035]，最优值：2.4500425150393143e-13
牛顿法 极小点：[0.99999995 2.99999998]，最优值：2.477034905727751e-14
Scipy库函数 极小点：[1.00000096 2.99999899]，最优值：1.9422930337097594e-12
PyTorch库函数(SGD) 极小点：[1.00000429 2.99999547]，最优值：3.728928277269006e-11

进程已结束,退出代码0
```

图 3：Booth 函数测试结果

3.2.2 Rosenbrock Banana 函数（极小值点(1, 1)，最优值 0）

·梯度下降法步长 λ=0.001，最大迭代次数 max_iter=10000

·牛顿法步长 λ=0.01，最大迭代次数 max_iter=10000

·随机梯度下降(SGD)学习率 lr=0.001，最大迭代次数 max_iter=10000

可以看到，四种方法的测试结果均很接近真实值，且在当前设定参数下牛顿法的实现效果比库函数更好。

```
初始点：[-4.10669995  0.61173511]
梯度下降法 极小点：[0.9912954  0.98196091]，最优值：7.825983616231804e-05
牛顿法 极小点：[0.99999898 0.99999787]，最优值：1.0818367562963634e-12
Scipy库函数 极小点：[0.99999881 0.99999732]，最优值：1.8677115442751064e-12
PyTorch库函数(SGD) 极小点：[0.99272132 0.98490655]，最优值：5.471422628033906e-05

进程已结束,退出代码0
```

图 4：Rosenbrock Banana 函数测试结果

四、 结论

梯度下降法和牛顿法广泛应用于求解无约束优化问题，也是众多机器学习算法中最常用的优化方法，几乎当前每一个先进的机器学习库或者深度学习库都会包括梯度下降算法的不同变种实现。

梯度加速方法对学习率（步长）较为敏感，学习率过高或过低均会对算法性能产生显著影响。相比梯度下降法，牛顿法求解高维问题更具有优势，其基于二阶泰勒展开，收敛速度更快，结果更加精确。

在本次实验中，实现的梯度下降法和牛顿法对于测试函数均取得很好的效果，但也存在超参数设定较难（步长的设定需要经验和尝试），计算效率不高、运行时间长的不足，需要进一步改进。

五、 参考文献

[1] taoKingRead.机器学习-梯度下降算法原理及公式推导[EB/OL].CSDN 博客,2020-07-10. https://blog.csdn.net/iqdutao/article/details/107174240

[2] ha_lee.最优化方法——梯度下降法、牛顿法、LM 算法[EB/OL].CSDN 博客, 2022-01-08. https://blog.csdn.net/ha_lee/article/details/122363325

[3] 星水天河.python sympy 求多元函数的梯度、Hessian 矩阵[EB/OL].CSDN 博客, 2022-11-10. https://blog.csdn.net/weixin_44509533/article/details/127795438

[4] 夏日清风有你.python 中的求导和偏导——diff 函数和 symbols 函数[EB/OL].CSDN 博客, 2023-08-21.Zhttps://blog.csdn.net/fanlily913/article/details/109297994