哈爾濱工業大學

人工智能数学基础实验报告

题　　目　梯度下降法、牛顿法，以及 Lion
基于学习的加速方法的实现与实验

计算机科学与技术

人工智能

学　　号

生

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：梯度下降法、牛顿法，以及 Lion基于学习的加速方法的实现与实验

1 实验内容或者文献情况介绍

- 掌握基本的优化思想及算法
- 掌握加速的思想
- 如何使用这些基本的方法来求解实际问题
- 逼近的思想的体现

2 算法简介及实现细节

2.1 梯度下降法

梯度下降法（*Gradient Descent*）是一种求解最优化问题的一阶迭代算法。它的核心思想是沿着目标函数梯度的方向更新参数值，以期望达到目标函数最小值。

在机器学习中，梯度下降法常被用来求解损失函数的最小值。损失函数越大，意味着模型拟合效果越差。在机器学习中，梯度下降法的根本目是在迭代中寻找损失函数的最小值，并通过反向传播的方式调整各个权重的大小，知道模型拟合程度达到指定要求。

梯度下降法的更新公式如下：

$$\theta_{i+1} = \theta_i - a\nabla J(\theta_i)$$

其中，θ_i 表示第 i 次迭代的参数，a 表示学习率，$\nabla J(\theta_i)$ 表示 $J(\theta_i)$ 的梯度。梯度下降法的优点是简单易懂，容易实现，但是其缺点是容易陷入局部最优解，同时需要手动调节学习率。

下面是梯度下降法的具体实现

```
def gradient_descent(X, y, theta, alpha, num_iters):
    m = len(y)
    J_history = np.zeros((num_iters, 1))
    for i in range(num_iters):
        h = np.dot(X, theta)
        theta = theta - alpha / m * np.dot(X.T, h - y)
        J_history[i] = compute_cost(X, y, theta)
    return theta, J_history
```

其中，X表示特征矩阵，y表示标签向量，θ表示参数向量，a表示学习率，num_iters表示迭代次数。在每次迭代中，首先计算预测值h，然后根据梯度下降公式更新参数θ，最后计算代价函数$J(\theta)$的值并保存到$J_history$中。

2.2 牛顿法

牛顿法（Newton's method）是一种求解最优化问题的迭代算法。它的基本思想是在当前位置对目标函数做二阶泰勒展开，进而找到下一个位置的估计值。

牛顿法不仅可以用来求解函数的极值问题，还可以用来求解方程的根，二者在本质上是一个问题，因为求解函数极值的思路是寻找导数为0的点，这就是求解方程。

与梯度下降法相比，牛顿法的特点是迭代次数少、收敛速度快；得到的最小值点比较准确；没有选取步长的麻烦。但缺点是需要计算目标函数的二阶梯度，也就是Hessian矩阵，可能计算量较大。

牛顿法的更新公式如下：

$$\theta_{i+1} = \theta_i - H^{-1}\nabla J(\theta_i)$$

其中，θ_i表示第 i 次迭代的参数，H 表示 $J(\theta_i)$ 的海森矩阵，$\nabla J(\theta_i)$ 表示 $J(\theta_i)$ 的梯度。牛顿法的优点是收敛速度快，但是其缺点是需要计算海森矩阵，计算量较大，同时可能会出现矩阵不可逆的情况。

下面是牛顿法的具体实现：

```
def newton_method(X, y, theta, num_iters):
    m = len(y)
    J_history = np.zeros((num_iters, 1))
    for i in range(num_iters):
        h = np.dot(X, theta)
        grad = 1 / m * np.dot(X.T, h - y)
        H = 1 / m * np.dot(X.T, X)
        theta = theta - np.dot(np.linalg.inv(H), grad)
        J_history[i] = compute_cost(X, y, theta)
    return theta, J_history
```

其中，X 表示特征矩阵，y 表示标签向量，θ 表示参数向量，num_iters 表示迭代次数。在每次迭代中，首先计算预测值 h 和梯度 $\nabla J(\theta_i)$，然后计算海森矩阵 H，最后根据牛顿法公式更新参数 θ，并计算代价函数 $J(\theta)$ 的值并保存到 $J\_history$ 中。

3 实验设置及结果分析（包括实验数据集）

3.1 pytorch 简介

本实验主要使用 pytorch 库进行训练。PyTorch 是一个基于 Python 的科学计算库，它主要提供了两个高级功能：张量计算和深度学习。张量计算是一种基于数组的计算，可以在 GPU 上加速运算。深度学习是一种机器学习技术，可以用于图像识别、自然语言处理等领域。PyTorch提供了丰富的工具和接口，可以方便地进行模型构建、训练和部署。

3.2 数据集生成

前期计划使用 MNIST 数据集进行拟合，但训练时经常出现损失值爆炸的情况，而调小学习率又会出现损失值不下降的情况。

因此最终使用了一个较为简单的函数 $\sin x$ 作为拟合对象。构建了一个 Dataset 子类作为数据集，取 $[-2\pi, 2\pi]$ 范围内的数据。

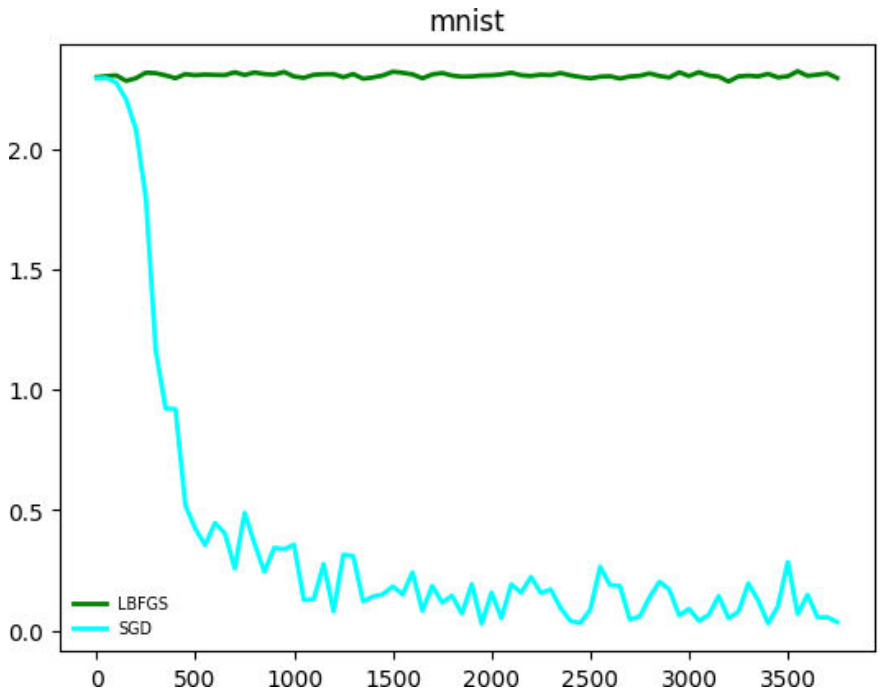

图 1: MNIST 数据集上的一种训练结果

```
class SinDataset(Dataset):
    def __init__(self, size=1000):
        self.x = torch.linspace(-2*torch.pi, 2*torch.pi,size).unsqueeze(1)
        self.y = torch.sin(self.x)

    def __getitem__(self, index):
        return self.x[index], self.y[index]

    def __len__(self):
        return len(self.x)
```

3.3 梯度下降法

实验中使用了 torch 库中已经实现的 SGD 优化器。随机梯度下降（Stochastic Gradient Descent，SGD）是一种常用的优化算法，用于求解无约束优化问题。其基本思想是在每次迭代中随机选择一个样本来计算梯度，然后根据梯度下降公式更新参数。SGD 的更新公式如下：

$$\theta_{i+1} = \theta_i - a\nabla J(\theta_i; x^{(i)}, y^{(i)})$$

其中，θ_i 表示第 i 次迭代的参数，a 表示学习率，$\nabla J(\theta_i; x^{(i)}, y^{(i)})$ 表示在样本 $(x^{(i)}, y^{(i)})$ 上的梯度。SGD 的优点是计算速度快，但是其缺点是收敛速度慢，容易陷入局部最优解。为了解决这个问题，可以使用带动量的 SGD（Momentum SGD）或自适应学习率的 SGD (Adagrad、 Adadelta、Adam 等)。

```
# 使用SGD优化器
model = Net().to(device)
criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=learning_rate)
sgd_losses = train(model, optimizer, criterion, train_loader, epochs)
```

3.4 牛顿法

由于 torch 中没有标准的牛顿法，故使用 LBFGS 优化器代替。LBFGS（Limited-memory Broyden-Fletcher-Goldfarb-Shanno）是一种基于拟牛顿法的优化算法，用于求解无约束优化问题。其基本思想是通过近似海森矩阵来逼近目标函数，然后求解该逼近函数的最小值。LBFGS的优点是收敛速度快，但是其缺点是需要存储一定的历史信息，同时可能会出现矩阵不可逆的情况。

```
# 使用LBFGS优化器
model = Net().to(device)
criterion = nn.MSELoss()
optimizer = optim.LBFGS(model.parameters(), lr=learning_rate
    )
            lbfgs_losses = train(model, op
timizer,    criterion,train_loader,
```

3.5 训练过程

实验使用了简单的三层全连接神经网络作为模型，损失值使用均方误差，训练 100 个epoch。两组损失值变换如下：

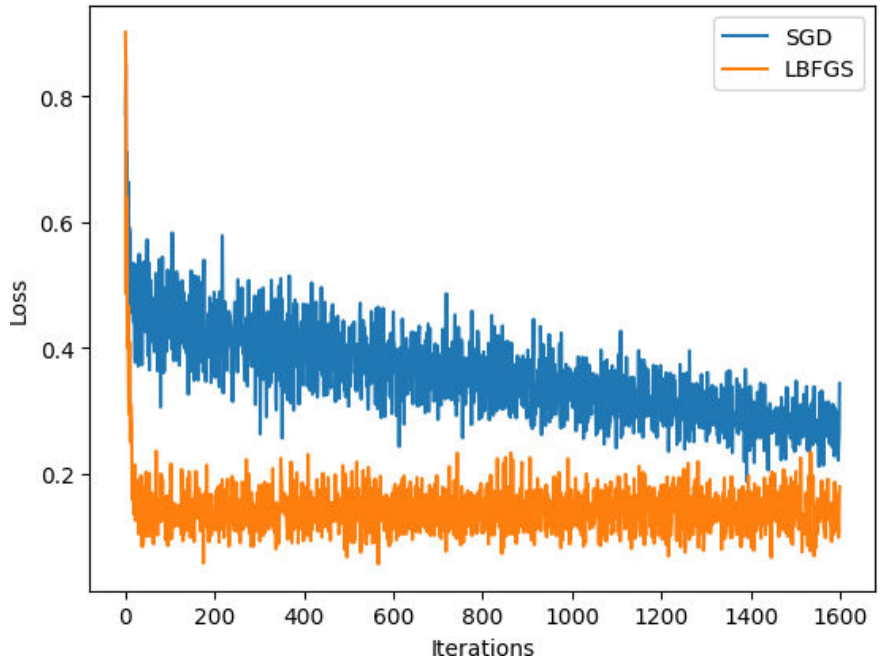

图 2: 训练结果

可见，LBFGS 优化器的收敛速度快于 SGD 优化器。但在训练时，使用 LBFGS 优化器的模型仍有可能出现损失值爆炸或为 nan 的情况，稳定性较差。对于 5 层的 CNN 来说，LBFGS 优化器几乎无法使用。

4 结论

梯度下降法和牛顿法都是常用的优化方法，它们各有优缺点。

梯度下降法的优点是实现简单，计算量相对较小。但缺点是迭代次数多，收敛速度慢；需要选取合适的步长，否则可能不收敛；容易陷入局部最优解。为了解决这个问题，可以使用带动量的 SGD（Momentum SGD）或自适应学习率的 SGD（Adagrad、Adadelta、Adam 等)。

牛顿法的优点是迭代次数少、收敛速度快；得到的最小值点比较准确；没有选取步长的麻烦。但缺点是需要计算目标函数的二阶梯度，也就是*Hessian*矩阵，可能计算量较大。实际中常用拟牛顿法，同时需要存储一定的历史信息，同时可能会出现矩阵不可逆的情况。

综上所述，对于简单的模型和小规模数据集，可以使用牛顿法进行优化。对于复杂的模型和大规模数据集，可以使用 SGD 优化器或其变种进行优化。在实际应用中，可以根据具体情况选择不同的优化算法进行训练。