

最优化理论大作业

分布式lasso问题求解与MNIST数据集分类

19335015 陈恩婷

概述

本实验探究了分布式lasso问题与MNIST数据集分类等两个问题，其中分布式lasso问题采用临近点梯度法、交替方向乘子法与次梯度法等算法进行解决，MNIST数据集分类问题利用前馈神经网络，采用梯度下降法、随机梯度法等算法进行解决。实验中还进行了数值实验，在MATLAB环境下实现了上述算法，探究了正则化系数 p 与小批量大小等参数对于实验结果的影响。

问题描述

分布式lasso问题

lasso问题介绍

lasso问题又名 ℓ_1 -范数规范化最小二乘优化问题（Least-squares approximation with ℓ_1 -norm regularization），它的标准形式如下：

$$\min \frac{1}{2} \|Ax - b\|_2^2 + p \|x\|_1$$

其中 $p > 0$ 被称为**正则化系数**，用来平衡模型的拟合程度和解的性质。如果 p 太小，那么对解的性质没有起到改善作用；如果 p 太大，则模型与原问题相差很大，可能是一个糟糕的逼近。

本实验的问题

考虑一个 20 节点的分布式系统。节点 i 有线性测量 $b_i = A_i x + e_i$ ，其中：

1. b_i 为 10 维的测量值；
2. A_i 为 10 x 300 维的测量矩阵， A_i 中的元素服从均值为 0，方差为 1 的高斯分布；
3. x 为 300 维的未知稀疏向量且稀疏度为 5，也就是说， x 中有五个非零元素。 x 的真值中的非零元素服从均值为 0 方差为 1 的高斯分布；
4. e_i 为 10 维的测量噪声， e_i 中的元素服从均值为 0 方差为 0.2 的高斯分布。

对于本实验的问题，如果直接求解一个最小二乘问题，最优解很可能不止一个，但不一定所有的解都是我们想要的。由于 x 真值是一个稀疏向量，我们可以采用 ℓ_0 范数规范化的最小二乘问题来得到一个稀疏的解。但是由于 ℓ_0 范数在实际中难以处理，我们往往使用 ℓ_1 范数来代替 ℓ_0 范数来保证稀疏性，即构造lasso问题

$$\min \frac{1}{2} \|A_1 x - b_1\|_2^2 + \dots + \frac{1}{2} \|A_{20} x - b_{20}\|_2^2 + p \|x\|_1$$

本次实验在单机版MATLAB上模拟实现了以下算法的分布式版本的来求解该问题：

1. 临近点梯度法；
2. 交替方向乘子法（ADMM）；
3. 次梯度法。

对于每种算法，给出了每步计算结果与真值的距离以及每步计算结果与最优解的距离。此外，也讨论了正则化参数 p 对计算结果的影响。

MNIST数据集分类

MNIST数据集来自美国国家标准与技术研究所, National Institute of Standards and Technology (NIST)。训练集 (training set) 由来自 250 个不同人手写的数字构成, 其中 50% 是高中学生, 50% 来自人口普查局 (the Census Bureau) 的工作人员。测试集(test set) 也是同样比例的手写数字数据组成。MNIST数据集中的所有图片都为28*28像素的灰度图像。

本实验采用CNN神经网络模型, 用下述算法求解在MNIST数据集上的分类问题:

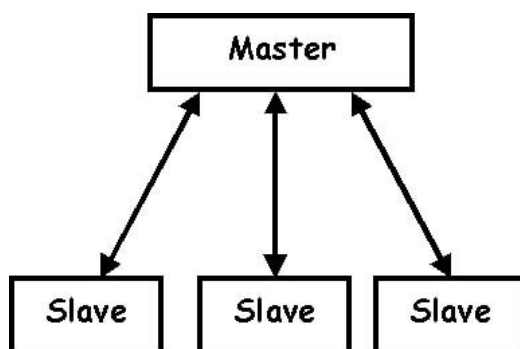
1. 梯度下降法;
2. 随机梯度法。

对于每种算法, 实验中给出了每步计算结果在测试集上所对应的分类精度。对于随机梯度法, 讨论了 mini-batch 大小的影响。

算法设计

分布式lasso问题

在并行优化中, master节点和slave节点的关系如下:



对于优化问题 $\min \sum_{i=1}^N f_i(x) + g(x)$, 典型的分布式算法的计算流程如下:

1. master节点将优化 $f_i(x)$ 的工作分配给Slave节点;
2. 各slave节点完成分配的工作, 向master节点返回数据;
3. master节点接收slave返回的数据;
4. master节点处理优化 $g(x)$ 和总目标函数的工作;
5. 不断重复1-4步骤, 直到结果收敛或者达到设定的最大迭代次数。

接下来分别介绍分布式的临近点梯度法、交替方向乘子法与次梯度法。

临近点梯度法

临近点梯度法的迭代格式如下:

$$x^{k+\frac{1}{2}} = x^k - \alpha \left(\sum_{i=1}^N \nabla f_i(x^k) \right)$$
$$x^{k+1} = \arg \min_x g(x) + \frac{1}{2\alpha} \|x - x^{k+\frac{1}{2}}\|_2^2$$

对于lasso问题, 一个更具体的迭代格式为:

$$x^{k+\frac{1}{2}} = x^k - \alpha \left(\sum_{i=1}^N A_i^T (A_i x^k - b_i) \right)$$
$$x^{k+1} = \text{sign}(x^{k+\frac{1}{2}}) \max\{|x^{k+\frac{1}{2}}| - \alpha p, 0\}$$

其中p为lasso问题的正则化系数。

在分布式的临近点梯度法中，计算的流程如下：

1. 初始化所有变量；
2. 中心将 x^k 发给节点；
3. 节点返回 $\nabla f_i(x^k)$ ，也就是 $A_i^T(A_i x^k - b_i)$ ；
4. 中心计算临近点投影；
5. 重复2至4步骤，直到达到最大迭代次数，或者 x 收敛。

交替方向乘子法 (ADMM)

考虑一个基础版的lasso问题：

$$\min \frac{1}{2} \|Ax - b\|_2^2 + \lambda \|x\|_1$$

这是一个典型的无约束复合优化问题，我们可以很容易地将其写成ADMM标准问题形式：

$$\begin{aligned} \min \quad & \frac{1}{2} \|Ax - b\|_2^2 + \lambda \|x\|_1 \\ \text{s.t.} \quad & x - z = 0 \end{aligned}$$

对于lasso问题，一个更具体的迭代格式为

$$\begin{aligned} x^{k+1} &= (A^T A + \rho)^{-1} (A^T b + \rho (z^k - u^k)) \\ z^{k+1} &= \text{sign}(x^{k+1} + u^k) \max(|x^{k+1} + u^k| - \lambda/\rho, 0) \\ u^{k+1} &= u^k + x^{k+1} - z^{k+1} \end{aligned}$$

ADMM的迭代格式如下：

$$\begin{aligned} x_i^{k+1} &= \arg \min_x f_i(x_i) + \frac{\rho}{2} \|x_i - z^k + u_i^k\|_2^2 \\ z^{k+1} &= \arg \min_z g(z) + \frac{\rho N}{2} \|z - (\overline{u^k} + \overline{x^{k+1}})\|_2^2 \\ u_i^{k+1} &= u_i^k + x_i^{k+1} - z^{k+1} \end{aligned}$$

其中

$$\begin{aligned} u^k &= \frac{1}{N} \sum_{i=1}^N u_i^k \\ x^{k+1} &= \frac{1}{N} \sum_{i=1}^N x_i^{k+1} \end{aligned}$$

对于lasso问题，迭代格式如下：

$$\begin{aligned} u_i^{k+1} &= u_i^k + x_i^k - z^k, \text{ for } i = 1, 2, \dots, N \\ x_i^{k+1} &= (A_i^T A_i + \rho I)^{-1} (A_i^T b + \rho (z^k - u_i^k)), \text{ for } i = 1, 2, \dots, N \\ z^{k+1} &= \text{sign}(\overline{x^{k+1}} + \overline{u^k}) \max(|\overline{x^{k+1}} + \overline{u^k}| - \lambda/N\rho, 0) \end{aligned}$$

在分布式的ADMM中，计算的流程为：

1. 初始化所有变量；
2. 节点计算 u_i^{k+1}, x_i^{k+1} 发给中心；
3. 中心计算 z^{k+1} 发给节点；
4. 重复2，3步骤，直到达到最大迭代次数，或者 x 收敛。

次梯度法

临近点梯度法的迭代格式如下：

$$x^k = x^k - \alpha \left(\sum_{i=1}^N \nabla f_i(x^k) + \partial g(x^k) \right)$$

对于lasso问题，一个更具体的迭代格式为：

$$x^k = x^k - \alpha \left(\sum_{i=1}^N A_i^T (A_i x^k - b_i) + p * \text{sign}(x^k) \right)$$

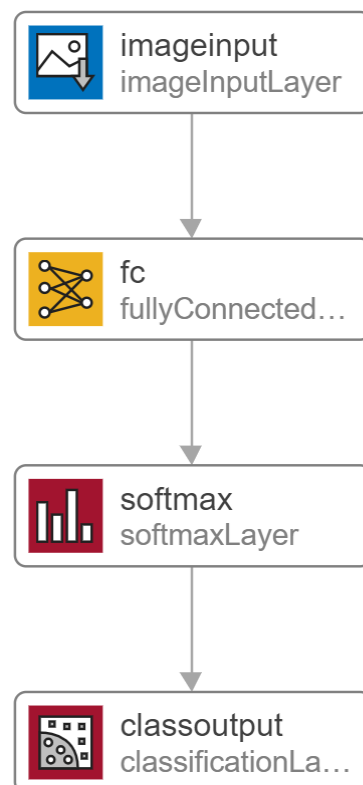
其中p为lasso问题的正则化系数。

在分布式的临近点梯度法中，计算的流程如下：

1. 初始化所有变量；
2. 中心将 x^k 发给节点；
3. 节点返回 $\nabla f_i(x^k)$ ，也就是 $A_i^T (A_i x^k - b_i)$ ；
4. 中心计算 $\partial g(x^k)$ ，更新 x ；
5. 重复2至4步骤，直到达到最大迭代次数，或者 x 收敛。

MNIST数据集分类问题

本实验采用前馈神经网络模型，用梯度下降法和随机梯度法求解MNIST数据集上的分类问题。实验中设计的神经网络结构如下：



梯度下降法

梯度下降法的迭代格式如下：

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

其中

$$\nabla f(x^k) = \frac{1}{N} \sum_{i=1}^N \nabla f_i(x^k)$$

在MATLAB环境下，`trainNetwork` 函数可以自动计算神经网络权值的梯度，只需设置步长 α 等参数即可。

随机梯度法

而在机器学习任务中，采集到的样本量是巨大的，因此在梯度下降法中计算梯度通常需要非常大的计算量。使用传统的梯度法求解机器学习问题并不是一个很好的做法。一个减少计算量的方法就是随机梯度法。

本实验采用小批量(mini-batch)随机梯度法，即随机选择一个元素个数很少的集合 $\mathcal{I}_k \subset \{1, 2, \dots, N\}$ ，然后执行迭代格式

$$x^{k+1} = x^k - \frac{\alpha_k}{|\mathcal{I}_k|} \sum_{s \in \mathcal{I}_k} \nabla f_s(x^k)$$

其中 $|\mathcal{I}_k|$ 表示 \mathcal{I}_k 中的元素个数。这样在每次迭代中，程序就无需计算整个训练集的梯度，顺序计算小批量的梯度即可。

数值实验与结果分析

本实验探究了分布式lasso问题和MNIST数据集分类问题的参数对于实验结果的影响，用MATLAB实现算法并进行测试，实验环境如下：

1. 处理器：Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz
2. 操作系统：Windows 10
3. 编程语言：MATLAB
4. 开发环境：MATLAB R2020b

分布式lasso问题

对于每种算法，给出了每步计算结果与真值的距离以及每步计算结果与最优解的距离。此外，也讨论了正则化参数 p 对计算结果的影响。

参数初始化

对于分布式lasso问题，各变量在程序中的初始化如下：

```
rng('default') % For reproducibility

x_true = sprandn(300,1,5/300);
x_true = full(x_true);

A = normrnd(0,1,10,300,20);
e = normrnd(0,0.2,10,1,20);
b = pagemtimes(A, x_true)+e;
```

这里为了可复现性，将随机数种子设置为 'default'。

临近点梯度法

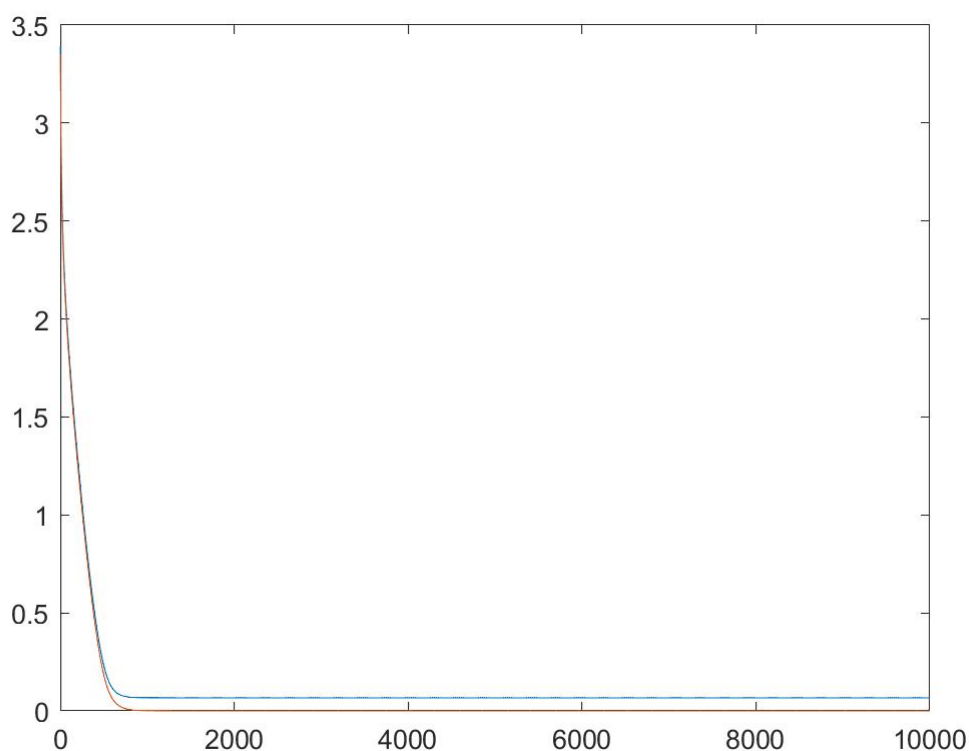
正则化系数p对最优解的影响

对于临近点梯度法，本实验探究了对于不同的正则化参数p对求得的最优解对与x的真值距离的影响：

正则化参数p	0.1	0.01	0.001	0.0006	0.0001
最优解与x真值的欧式距离	3.4748	0.8125	0.0794	0.0665	0.1898
迭代次数	0	666	986	1501	4056
是否全零向量	是	否	否	否	否

其中步长t固定为0.000125，解的精度要求为 $1 * 10^{-8}$ ，初始解为全零向量。可见对于临近点梯度法，正则化系数并不是越大越好，这是因为如果p太小，那么对解的性质没有起到改善作用；如果p太大，则模型与原问题相差很大，可能是一个糟糕的逼近。

当正则化系数为0.1时，最优解为全零向量，且迭代次数为0，这是因为初始化的解与最优解相同，程序在尝试第一次迭代时就会检测到算法已收敛。当正则化系数为0.0006时，最优解与x真值最接近，此时每步计算结果与真值距离，以及每部计算结果与最优解距离的变化过程如下图：



其中每步计算结果与真值距离为蓝色曲线，每步计算结果与最优解距离为红色曲线，可见两条曲线的下降速率基本一致，程序最终收敛到的x为最优解。

关于 $p > p_{max}$ 的情况

在之前的作业当中，我们曾经证明了对于lasso问题

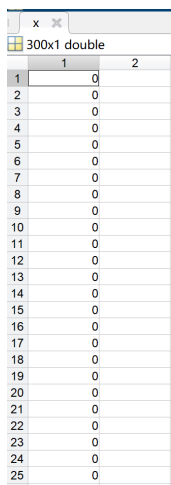
$$\min \frac{1}{2} \|Ax - b\|_2^2 + p \|x\|_1$$

存在 $p_{max} = \|A^T b\|_\infty$ ，当 $p > p_{max}$ 时，lasso问题的最优解为全零向量。

在本实验中，对于分布式的lasso问题， p_{max} 可推广为

$$p_{max} = \left\| \sum_{i=1}^N A_i^T b_i \right\|_{\infty}$$

对于本实验的初始化参数，可以求得 p_{max} 为593.6131，取 $p=600$ 代入程序，可得 x 的最优解确为零向量：



	1	2
1	0	
2	0	
3	0	
4	0	
5	0	
6	0	
7	0	
8	0	
9	0	
10	0	
11	0	
12	0	
13	0	
14	0	
15	0	
16	0	
17	0	
18	0	
19	0	
20	0	
21	0	
22	0	
23	0	
24	0	
25	0	

实际上，在前面的表格当中，由于 x 的初始解就为最优的全零向量，当 $p=0.1$ 时，程序只迭代了0次就已收敛，就已经出现了最优解为零向量的情况。

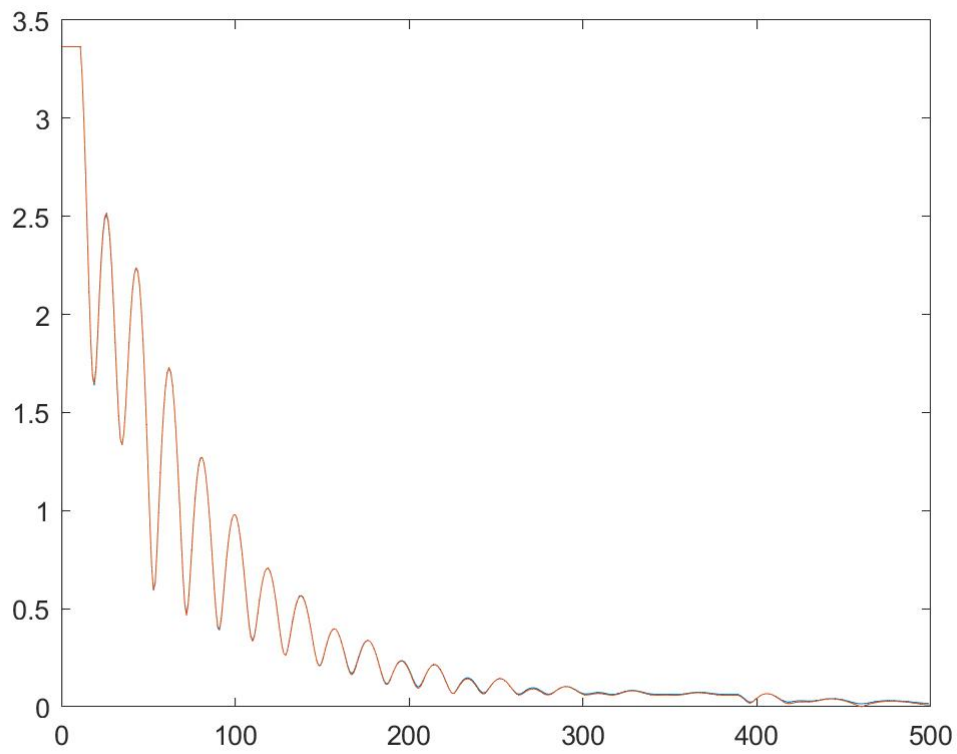
交替方向乘子法

对于次梯度法，本实验探究了不同的正则化参数 p 对求得的最优解（与 x 真值最接近的解）与 x 的真值距离的影响：

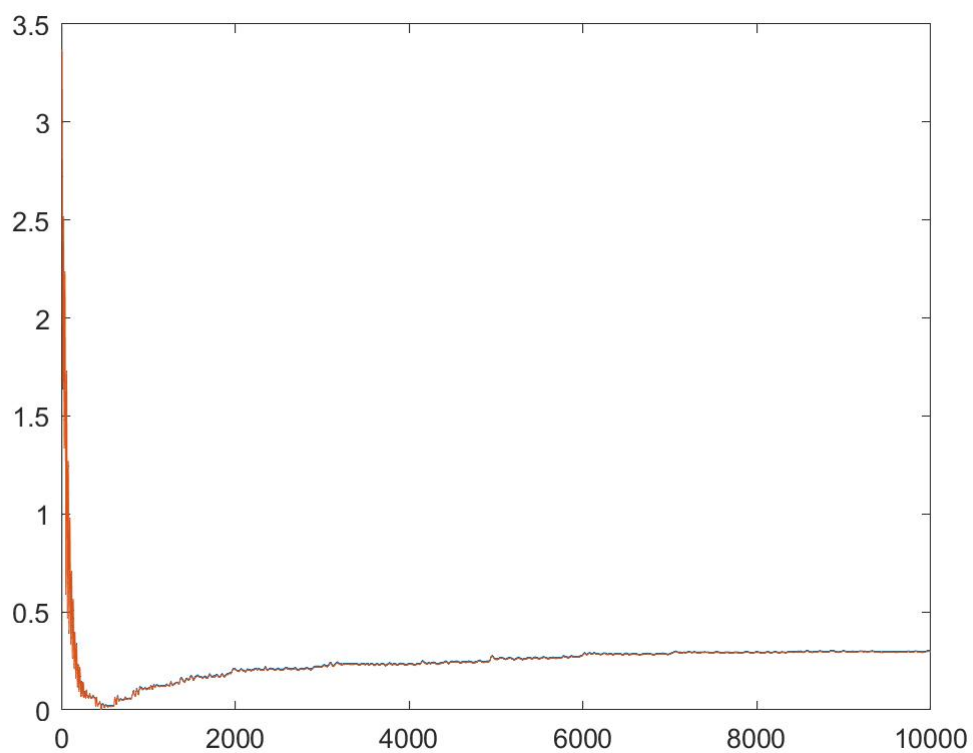
正则化参数 p	1	0.1	0.02	0.01
最优解与 x 真值的欧式距离	0.0654	0.0163	0.0155	0.0232
最优解所需迭代次数	2441	2006	460	291
是否全零向量	否	否	否	否

其中二次罚项系数 ρ 为0.001，初始解为零向量。可见对于交替方向乘子法，正则化系数并不是越大越好，这是因为如果 p 太小，那么对解的性质没有起到改善作用；如果 p 太大，则模型与原问题相差很大，可能是一个糟糕的逼近。

当正则化系数 p 越小，程序越快迭代到最优解，当 p 为0.02时，最优解与 x 真值的欧式距离最小，为0.0155，此时每步计算结果与真值距离，以及每部计算结果与最优解距离的变化过程如下图：



可见两条曲线几乎完全重叠，呈波浪线状。增加迭代次数，可见 x 最终会慢慢收敛，但是不收敛到最优解：



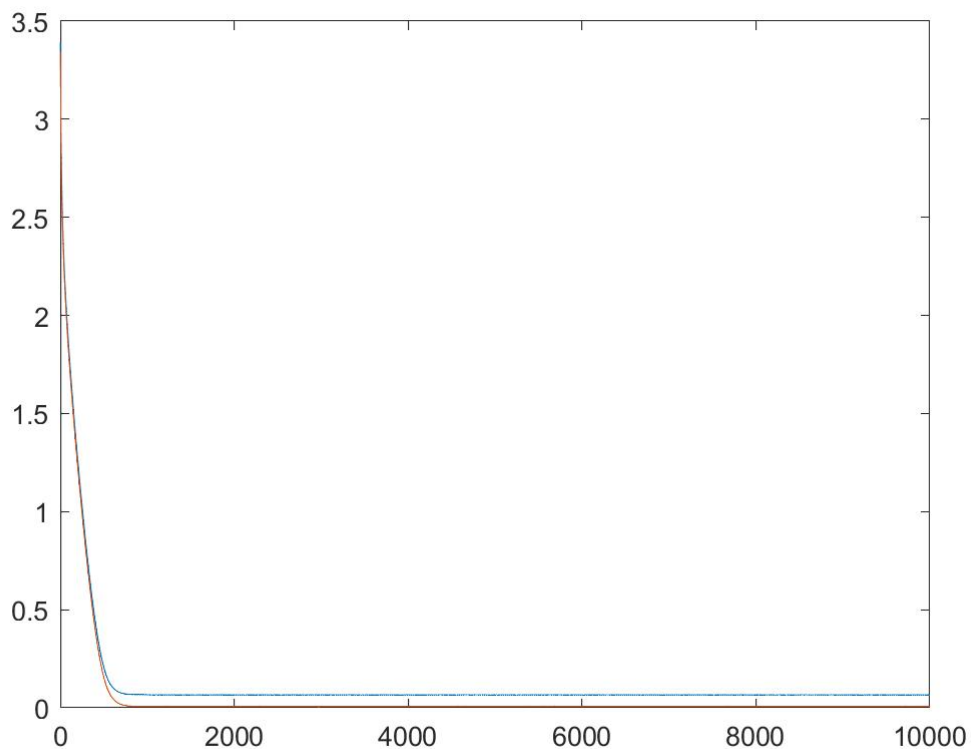
次梯度法

对于次梯度法，本实验探究了对于不同的正则化参数 p ，求得的最优解对与 x 的真值距离的影响：

正则化参数p	100	10	5	1	0.1	0.01
最优解与x真值的欧式距离	1.0849	0.1059	0.0664	0.1699	0.2956	1.0617
迭代次数	47361	631	1054	3221	35020	100000
是否全零向量	否	否	否	否	否	否

其中步长 t 固定为0.000125，解的精度要求为 1×10^{-7} ，初始解为全零向量，最大迭代次数为100000。可见正则化系数并不是越大越好，这是因为如果 p 太小，那么对解的性质没有起到改善作用；如果 p 太大，则模型与原问题相差很大，可能是一个糟糕的逼近。

当正则化系数较大或较小时，算法收敛非常慢，当正则化系数为5时，最优解与x真值最接近，此时每步计算结果与真值距离，以及每部计算结果与最优解距离的变化过程如下图：



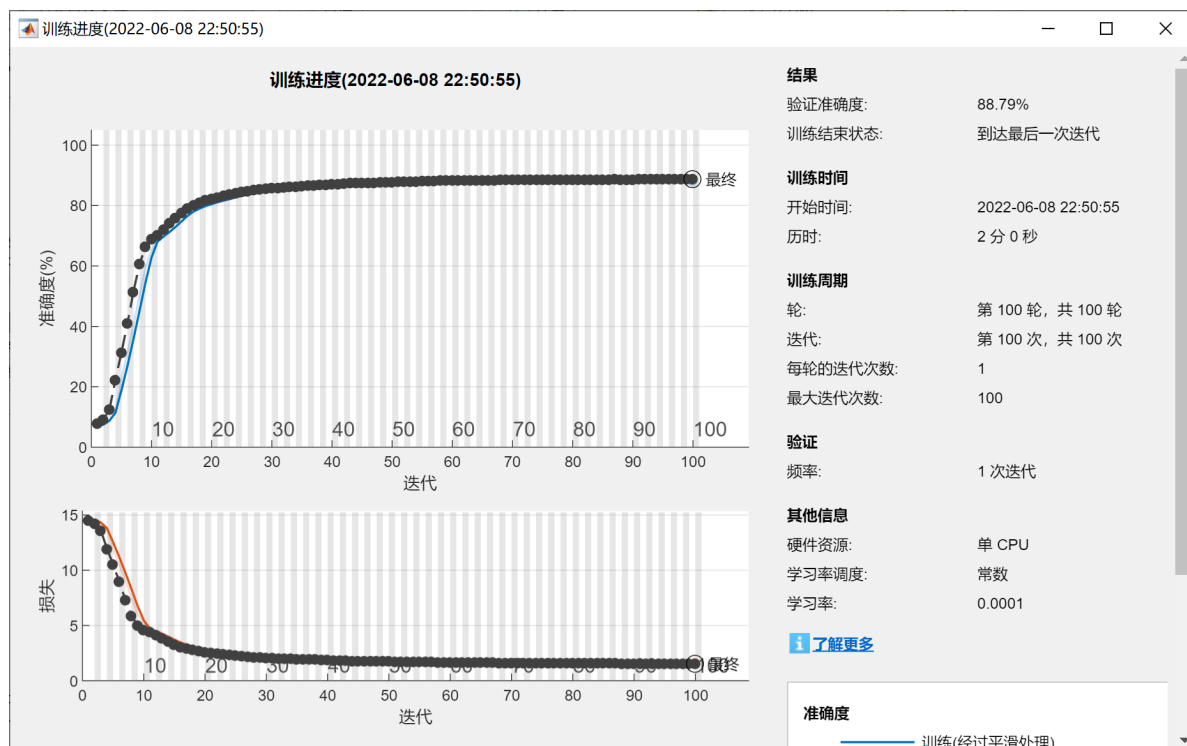
可以看到，两种距离的下降速度基本保持一致，大致在600次迭代处开始变得缓慢，和临近点梯度法的曲线大致相同。可见在正则化系数为最优时，两种方法的性能基本一致，程序最终收敛到的 x 为最优解。

MNIST数据集分类问题

对于每种算法，实验中给出了每步计算结果在测试集上所对应的分类精度。对于随机梯度法，讨论了mini-batch 大小的影响。

梯度下降法

本实验采用了四层的前馈神经网络，梯度下降法在训练过程中的损失与正确率如下：



如图所示, 可见采用梯度下降法时, 训练集和测试集上的损失和正确率都基本一致, 没有出现过拟合的情况, 最终的验证准确率为88.79%。这是因为训练集和测试集的数据都由相同的人书写而成并按相同比例混合, 数据分布基本一致, 所以当模型在训练集上经过训练逐渐改善的同时, 在测试集上的改善也保持了一样的损失和精度。

随机梯度法

对于随机梯度法, 神经网络在1-2轮内训练就收敛了。本实验探究了小批量大小对于最优测试精度的影响, 如下表

小批量大小 (minibatchsize)	16	32	64	128
最大训练轮数 (epoch)	1	2	2	2
最优测试精度	89.36%	89.14%	88.60%	88.77%

如表格所示, 小批量大小对最优测试精度并没有显著的影响, 最优测试精度都达到了89%左右。实验中遇到的问题与解决方法

ADMM运行速度很慢

在采用交替方向乘子法(ADMM)求解分布式lasso问题时, 发现运行速度明显低于其他两种算法

解决方法: 猜想这是由于程序中有一些复杂的步骤被重复了。通过仔细观察迭代格式发现, 分布式ADMM的迭代格式中的这一步

$$x_i^{k+1} = (A_i^T A_i + \rho I)^{-1} (A_i^T b + \rho (z^k - u^k)), \text{ for } i = 1, 2, \dots, N$$

需要对矩阵进行求逆的操作, 而该矩阵对于所有的节点都是固定不变的, 所以可以将它保存起来在后面使用

```
Inv = zeros(300, 300, 20);
for i = 1:20
    Inv(:, :, i) = inv(A(:, :, i)'*A(:, :, i) - rou*eye(300));
end
```

这样就避免了重复的求逆运算，效率有很大的提升。

实验总结与讨论

本实验探究了分布式lasso问题与MNIST数据集分类等两个问题，并在MATLAB环境下进行了数值实验。对于分布式lasso问题，实验中发现正则化系数并不是越大越好，这是因为如果 p 太小，那么对解的性质没有起到改善作用；如果 p 太大，则模型与原问题相差很大，可能是一个糟糕的逼近。其中交替方向乘子法在正则化系数为0.1时最优解最接近 x 真值，达到0.0163的欧氏距离。对于MNIST数据集分类问题，梯度下降法和随机梯度法性能接近，都达到了89%的验证精度。

参考链接

1. https://www.caam.rice.edu/~optimization/L1/optseminar/DistrADMM_Peng.pdf
2. <http://yann.lecun.com/exdb/mnist/>
3. https://ww2.mathworks.cn/help/deeplearning/ref/trainnetwork.html?s_tid=doc_ta
4. <https://bicmr.pku.edu.cn/~wenzw/optbook/opt2.pdf>