最优化作业四

18340047

2020 年 *7* 月 *16* 日

1 第一题

1.1 优化问题

考虑线性测量 $b = Ax + e$，其中 b 为 50 维的测量值，A 为 50 维的测量矩阵，x 为 100 维的未知稀疏向量且稀疏度为 5，e 为 50 维的测量噪声。从 b 与 A 中恢复 x 的一范数规范化最小二乘模型如下：

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

其中 p 为非负的正则化参数，x 中的非零元素服从标准整体分布 $N(0,1)$，A 中元素服从 $N(0,1)$，e 中元素服从 $N(0,0.1)$。

1.2 邻近点梯度下降法

该问题的目标函数中可求梯度部分为 $S(x) = \frac{1}{2}||Ax-b||_2^2$，不可求梯度部分为 $R(x) = p||x||_1$，使用邻近点梯度下降法的迭代过程如下：

$$\begin{cases} x^{k+\frac{1}{2}} = x^k - \alpha A^T(Ax^k - b) \\ x^{k+1} = arg\ min_x\ p||x||_1 + \frac{1}{2\alpha}||x - x^{k+\frac{1}{2}}||_2^2 \end{cases} \tag{2}$$

其中 α 为步长。

(2) 式迭代的第二步没有显式解，但可利用用一阶最优性条件求得闭式解。一阶最优性条件为：$\exists g_i \in \partial|x_i|$，使得

$$pg_i + \frac{1}{\alpha}(x_i - x_i^{k+\frac{1}{2}}) = 0 \tag{3}$$

其中 $\partial|x_i|$ 为 x 第 i 维的次梯度：

$$\partial|x_i| = \begin{cases} -1 & x_i < 0 \\ [-1,1] & x_i = 0 \\ 1 & x_i > 0 \end{cases} \tag{4}$$

结合 (3) 式有：

$$x_i^{k+\frac{1}{2}} \begin{cases} = p\alpha + x_i > p\alpha & x_i > 0 \\ \in [-p\alpha, p\alpha] & x_i = 0 \\ = -p\alpha + x_i < -p\alpha & x_i < 0 \end{cases}$$

即 x_i 应满足：

$$x_i = \begin{cases} = x_i^{k+\frac{1}{2}} - p\alpha & x_i^{k+\frac{1}{2}} > p\alpha \\ = 0 & x_i^{k+\frac{1}{2}} \in [-p\alpha, p\alpha] \\ = x_i^{k+\frac{1}{2}} + p\alpha & x_i^{k+\frac{1}{2}} < -p\alpha \end{cases} \tag{5}$$

这样就得到了 (2) 式迭代第二步的闭式解，将其写为：

$$x_{k+1} = S_{p\alpha}(x^{k+\frac{1}{2}}) = sign(x^{k+\frac{1}{2}})max(|x^{k+\frac{1}{2}}| - p\alpha, 0) \tag{6}$$

其中 $S_{p\alpha}(\cdot)$ 称为软门限算子。

结合 (2) 式迭代第一步和 (6) 式，我们就得到了问题 (1) 的邻近点梯度下降算法，其 Matlab 实现如下：

```
function x=pgd(A,b,p,xp,step,tol,maxiter)
%邻近点梯度法从Ax+e=b恢复x的一范数正则化问题
%输入
%   p：正则化参数
%   xp：解x的迭代初值
%   step：步长
%   tol：解的容差
%   maxiter：最大迭代次数
%输出
%   x：每一步迭代产生的解，包括最后找到的解
    x=zeros(length(xp),maxiter+1);
    x(:,1)=xp;
    %最多maxiter次迭代即停止算法
    for i=1:maxiter
        %迭代第一步：xh为x^{k+1/2}
        xh=x(:,i)-step*A'*(A*x(:,i)-b);
        %迭代第二步：x^{k}=sign(x ^{k+1/2})max(\A{x ^{k+1/2}-p\alpha,0)
        x(:,i+1)=sign(xh).*max(abs(xh)-p*step,0);
        %解的更新距离小于容差时 认为已经收敛 停止算法
        if norm(x(:,i+1)-x(:,i))<tol
            x=x(:,1:i+1);
            break;
        end
    end
end

```

pgd.m：邻近点梯度法

1.3 交替方向乘子法

在交替方向乘子法中，两个优化变量交替更新，而在问题 (1) 中只有一个优化变量，所以按以下方式引入新变量 $y = x$：

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

写出其增广拉格朗日函数：

$$L_c(x, y, v) = \frac{1}{2}||Ax-b||_2^2 + p||y||_1 + v^T(x-y) + \frac{c}{2}||x-y||_2^2 \tag{7}$$

其中 v 为拉格朗日乘子，c 取大于 0 的常数。这样交替方向乘子法的迭代过程为：

$$\begin{aligned} x^{k+1} &= arg\ min_x\ L_c(x, y^k, v^k) \\ y^{k+1} &= arg\ min_y\ L_c(x^{k+1}, y, v^k) \\ v^{k+1} &= v^k + c(x^{k+1} - y^{k+1}) \end{aligned}$$

利用一阶最优性条件，可得到第一步的和第二步的闭式解（第二步需要优化一范数，参考邻近点梯度法中的推导），可以推导出：

$$
\begin{aligned}
x^{k+1} &= (A^TA + cI)^{-1}(A^Tb + c(y^k - w^k)) \\
y^{k+1} &= S_{\frac{p}{c}}(w^k + x^{k+1}) \\
w^{k+1} &= w^k + x^{k+1} - y^{k+1}
\end{aligned}
$$

其中 $w = \frac{v}{c}$，$S_{p\alpha}(\cdot)$ 为软门限算子。

这样就得到了求解问题 (1) 的交替方向乘子法，其 Matlab 实现如下：

```
function x=admm(A,b,p,xp,step,tol,maxiter)
%交替方向乘子法从Ax+e=b恢复x的一范数正则化问题
%输入
%   p：正则化参数
%   xp：解x的迭代初值
%   step：步长 此处用于增广拉格朗日惩罚项中的正常数系数c
%   tol：解的容差adm
%   maxiter：最大迭代次数
%输出
%   x：每一步迭代产生的解，包括最后找到的解
    x=zeros(length(xp),maxiter+1);
    x(:,1)=xp;
    y=xp;%引入的优化变量y
    w=xp;%w=拉格朗日乘子v/c
    c=step;%c
    I=eye(length(A));%单位阵
    %最多maxiter次迭代即停止算法
    for i=1:maxiter
        x(:,i+1)=(A'*A+c*I)^(-1)*(A'*b+c*(y-w));%更新x
        y=sign(w+x(:,i+1)).*max(abs(w+x(:,i+1))-p/c,0);%更新y
        %解的更新距离小于容差时 认为已经收敛 停止算法
        if norm(x(:,i+1)-x(:,i))<tol
            x=x(:,1:i+1);
            break;
        end
        w=w+x(:,i+1)-y;%更新w
    end
end

```

admm.m：交替方向乘子法

1.4 次梯度法

次梯度法在目标函数不可微的地方使用次梯度代替梯度进行梯度下降：

$$
x^{k+1} = x^k - \alpha g(x^k)
$$

其中 $g(x^k)$ 为梯度或次梯度，在问题 (1) 中，有：

$$
g(x^k)) = -A^T(b - Ax^k) + p \cdot \partial||x^k||_1
$$

其中 $\partial||x^k||_1$ 为一范数的次梯度，其取值见前面的 (4) 式。

当次梯度的取值不唯一时，随机选取一个进行梯度下降。这样我们就得到了问题 (1) 的次梯度法，其 Matlab 实现如下：

```
function x=ssgd(A,b,p,xp,step,tol,maxiter)
%次梯度法从Ax+e=b恢复x的一范数正则化问题
```

```
%输入
%   p：正则化参数
%   xp：解x的迭代初值
%   step：步长 此处用于增广拉格朗日惩罚项中的正常数系数c
%   tol：解的容差
%   maxiter：最大迭代次数
%输出
%   x：每一步迭代产生的解，包括最后找到的解
    x=zeros(length(xp),maxiter+1);
    x(:,1)=xp;
    %最多maxiter次迭代即停止算法
    for i=1:maxiter
        %g为次梯度 xi=0时为[-1,1]随机选取
        g=sign(x(:,i));
        idx=find(x(:,i)==0);
        g(idx)=-1+2*rand([length(idx) 1]);
        %次梯度下降
        x(:,i+1)=x(:,i)-step*(-A'*(b-A*x(:,i))+p*g);
        %解的更新距离小于容差时 认为已经收敛 停止算法
        if norm(x(:,i+1)-x(:,i))<tol
            x=x(:,1:i+1);
            break;
        end
    end
end

```

ssgd.m：次梯度法

1.5 计算结果与讨论

首先生成测试数据：

```
%生成50*100的测量矩阵A 其元素服从标准正态分布N(0,1)
%和100维的未知稀疏向量x，其稀疏度为5，其元素服从N(0,1)
%和50维的测量噪声e，其元素服从N(0,0.1)
function [A,x,e]=getAxe()
A=randn(50,100);

i=randi([1,100],5,1);
x=zeros(100,1);
x(i)=randn(5,1);

e=normrnd(0,sqrt(0.1),50,1);%注意0.1是方差，标准差是sqrt(0.1)
end

```

getAxe.m：测量问题数据生成

完整测试脚本如下：

```
%产生符合题意的数据
[A,x,e]=getAxe();
b=A*x+e;
%参数
xp=zeros(length(x),1);
p=0.1;
step=0.005;
maxiter=20000;
tol=1e-5;
%算法求解与结果展示
```

```
x_pgd=pgd(A,b,p,xp,step,tol,maxiter);
showres(x_pgd,x,'PGD');
x_admm=admm(A,b,p,xp,step,tol,maxiter);
showres(x_admm,x,'ADMM');
x_ssgd=ssgd(A,b,p,xp,step,tol,maxiter);
showres(x_ssgd,x,'SSGD');

%结果展示：每一步到最优解、真实解的距离
function showres(x_f,x,fname)
x_f_iter=length(x_f);
x_f_converr=arrayfun(@(xi)norm(x_f(:,xi)-x_f(:,x_f_iter)),1:x_f_iter);
x_f_realerr=arrayfun(@(xi)norm(x_f(:,xi)-x),1:x_f_iter);
figure
xlabel('iteration');
ylabel('error');
title(fname);
hold on;
plot(1:1:x_f_iter,x_f_converr,'DisplayName','converr')
hold on;
plot(1:1:x_f_iter,x_f_realerr,'DisplayName','realerr');

legend;
end

```

test.m：测试脚本

运行测试脚本，结果如下：

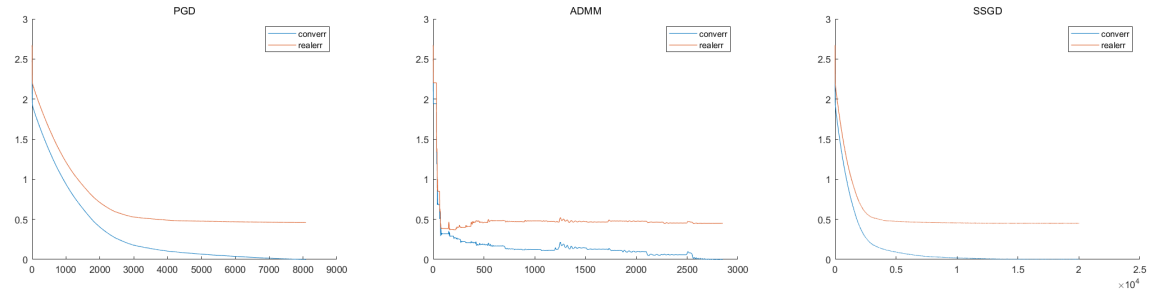

图 1: 正则化参数 p=0.1，步长 =0.005，容差 =1e-5，最大迭代次数 =20000

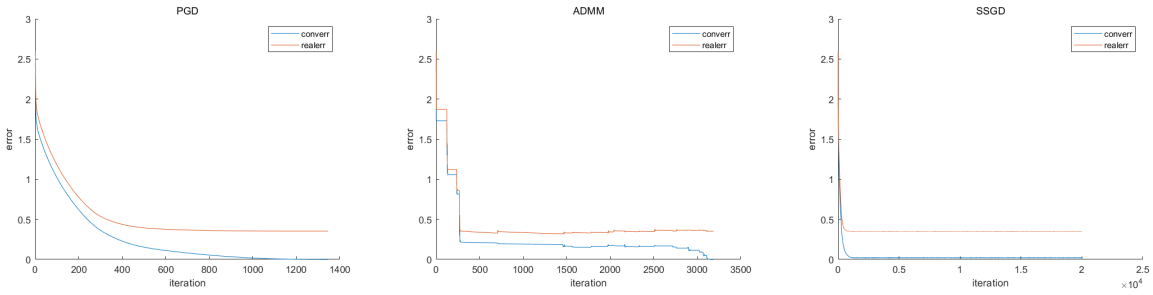

图 2: 正则化参数 p=0.5，步长 =0.005，容差 =1e-5，最大迭代次数 =20000

可以看到在图 1 到 3 中，三种算法都收敛到距离真实解 0.5 左右的位置。从收敛速度（只考虑迭代次数，不考虑实际运行时间）上看，次梯度法实际上没有真正收敛（放大可以看到曲线的震荡），其最终迭代次数为允许的最大迭代次数；在正则化参数较小时，交替方向乘子法比邻近点梯度法的收敛更快，在正则化参数较大时，后者收敛更快。

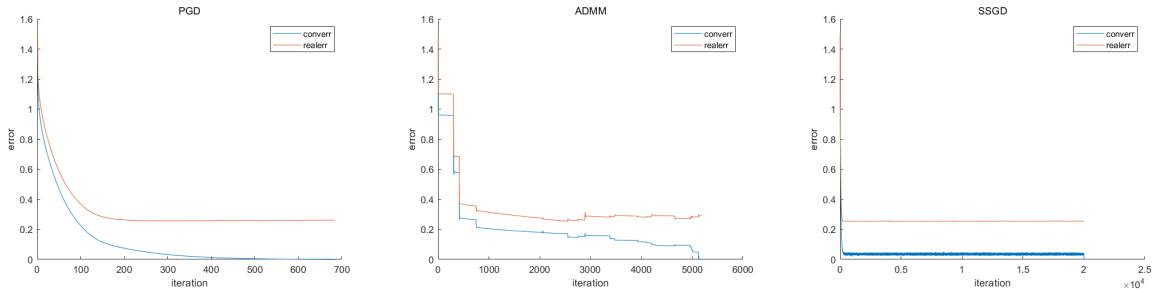

图 3: 正则化参数 p=0.8，步长 =0.005，容差 =1e-5，最大迭代次数 =20000

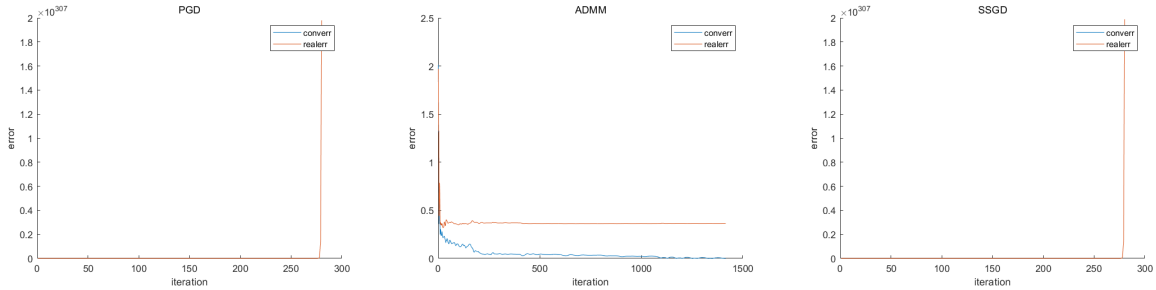

图 4: 正则化参数 p=0.1，步长 =0.05，容差 =1e-5，最大迭代次数 =20000

可以发现较大的正则化参数对于邻近点梯度法和次梯度法可以加快收敛，对交替方向乘子法则相反。另外发现，改变正则化参数对于提升最终解的质量没有太大帮助。

图 4 展示了提高步长时邻近点梯度法和次梯度法失效（发散）的情况，而交替方向乘子法反而可以更快收敛。

综上，对于此问题，交替方向乘子法是一个比较稳定、快收敛的算法，当正则化参数较小（如 0.1，参考 0.5 较小），步长较大（如 0.05 参考 0.01 较大）时，有比较好的效果，但收敛时离真实解还有一定距离。

2 第二题

2.1 优化问题

在 MNIST 数据集的 Logistic regression 模型中，通过 softmax 对输入进行分类：

$$P(Y=i|x,W,b)=softmax_i(s)=\frac{e^{s_i}}{\sum_{k=1}^{N}e^{s_k}}$$
$$y_{pred}=arg\ max_i\ P(Y=i|x,W,b) \tag{8}$$

其中 $s=Wx+b$ 为线性变换结果，x 为输入数据，W 为线性变换权值矩阵，b 为线性变换平移参数，y_{pred} 为预测的分类（这里用下标 1 到类总数 N 对应不同分类，s_k 可以看作第 k 类的得分，得分相对于其他分类越高，则越有可能属于该类）。

为了得到好的分类结果，需要对参数 W 和 b 进行优化，优化的目标函数为分类的损失函数：

$$l(W,b,D)=-\frac{1}{|D|}L(W,b,D)=-\frac{1}{|D|}\sum_{i=1}^{|D|}log(P(Y=y^{(i)}|x^{(i)},W,b) \tag{9}$$

其中 L 为对数似然函数，将其相反数作为损失函数，则数据集上的平均损失函数为 l。这样平均损失 l 越小，说明似然度越高，即分类模型（8）是对输入数据 D 所服从分布的极大似然估计。

这样我们就得到了分类模型的优化问题：

$$min\ l(W,b,D) \tag{10}$$

下面我们将分别用梯度下降法和随机梯度法求解该问题。这里先对目标函数的梯度进行推导，设 W_j 和 b_j 分别为 W 的第 j 行和 b 的第 j 元素，则有：

$$\begin{aligned}
\frac{\partial l}{\partial W_j} &= \frac{1}{|D|}\cdot\frac{\partial(\sum_{i=1}^{|D|}(-log(e^{s_i})+log(\sum_{k=1}^{N}e^{s_k})))}{\partial W_j} \\
&= \frac{1}{|D|}\sum_{i=1}^{|D|}(-I_{\{i=j\}}\frac{\partial s_j}{\partial W_j}+\frac{e^{s_j}}{\sum_{k=1}^{N}e^{s_k}}\cdot\frac{\partial s_j}{\partial W_j}) \\
&= \frac{1}{|D|}\sum_{i=1}^{|D|}(\frac{e^{s_j}}{\sum_{k=1}^{N}e^{s_k}}-I_{\{i=j\}})x_i^T \\
\frac{\partial l}{\partial b_j} &= frac1|D|\sum_{i=1}^{|D|}(\frac{e^{s_j}}{\sum_{k=1}^{N}e^{s_k}}-I_{\{i=j\}})\frac{\partial s_j}{\partial b_j}) \\
&= \frac{1}{|D|}\sum_{i=1}^{|D|}(\frac{e^{s_j}}{\sum_{k=1}^{N}e^{s_k}}-I_{\{i=j\}})
\end{aligned} \tag{11}$$

其中 $I_{\{i=j\}}$ 为示性函数，当 i 等于 j 时为 1，当 i 不等于 j 时为 0。

2.2 梯度下降法

问题 (10) 的梯度下降法迭代过程如下：

$$\begin{aligned}
W_j^{k+1} &= W_j^k-\alpha\frac{\partial l}{\partial W_j}, \\
b_j^{k+1} &= b_j^k-\alpha\frac{\partial l}{\partial b_j}, \quad j\in[1,N]
\end{aligned} \tag{12}$$

其中的梯度见 (11) 式，且每次迭代计算梯度的数据集 D 等于全集。

Matlab 实现如下：

```
function [W,b]=gd(tranimg,tranlabl,step,tol,maxiter)
%Logistic regression模型的梯度下降法
%输入
%   tranimg：训练数据
%   tranlabl：训练数据标签
%   step：步长
%   tol：解的容差
%   maxiter：最大迭代次数
%输出
%   W，b：每一步迭代产生的参数，包括最后找到的参数
    DATASIZE=length(tranlabl);%数据份数
    N_IN=size(tranimg,1);%每个x的维度
    N_OUT=10;%分类数目

    W=zeros(N_OUT,N_IN,maxiter);%线性分类Wx+b的权值矩阵W
    b=zeros(N_OUT,maxiter);%线性分类Wx+b的权值矩阵b

    %最多maxiter次迭代即停止算法
    for i=1:maxiter
        %更新Wj bj
        for j=1:N_OUT
            grad_wj=zeros(1,N_IN);
            grad_bj=0;
```

```
            %求Wj bj在整个数据集上的梯度
            for k=1:DATASIZE
                es=exp(W(:,:,i)*tranimg(:,k)+b(:,i));
                grad_sj=es(j)/sum(es)-(j==tranlabl(k));
                grad_wj=grad_wj+grad_sj*tranimg(:,k)';
                grad_bj=grad_bj+grad_sj;
            end
            %梯度下降
            W(j,:,i+1)=W(j,:,i)-step/DATASIZE*grad_wj;
            b(j,i+1)=b(j,i)-step/DATASIZE*grad_bj;
        end
        %参数的更新距离小于容差时 认为已经收敛 停止算法
        if norm(W(:,:,i+1)-W(:,:,i))+norm(b(:,i+1)-b(:,i))<tol
            W=W(:,:,1:i+1);
            b=b(:,1:i+1);
            break;
        end
    end
end

```

gd.m：梯度下降法

2.3 随机梯度法

问题 (9) 的随机梯度下降法迭代过程同 (11) 式，但每次迭代计算梯度的数据集 D 等于全集的一个随机子集，称为 Batch。

Matlab 实现如下：

```
function [W,b]=msgd(tranimg,tranlabl,step,BATCHSIZE,tol,maxiter)
%Logistic regression模型的随机梯度法
%输入
%   tranimg：训练数据
%   tranlabl：训练数据标签
%   step：步长
%   BATCHSIZE：随机梯度下降的Batch的大小
%   tol：解的容差
%   maxiter：最大迭代次数
%输出
%   W，b：每一步迭代产生的参数，包括最后找到的参数
    DATASIZE=length(tranlabl);%数据份数
    N_IN=size(tranimg,1);%每个x的维度
    N_OUT=10;%分类数目

    W=zeros(N_OUT,N_IN,maxiter);%线性分类Wx+b的权值矩阵W
    b=zeros(N_OUT,maxiter);%线性分类Wx+b的权值矩阵b

    %最多maxiter次迭代即停止算法
    for i=1:maxiter
        %随机小批量采样 随机梯度下降
        batch=randi([1,DATASIZE],BATCHSIZE,1);
        %用Batch更新Wj bj
        for j=1:N_OUT
            grad_wj=zeros(1,N_IN);
            grad_bj=0;
            %求Wj bj在Batch上的梯度
            for ki=1:BATCHSIZE
                k=batch(ki);
                es=exp(W(:,:,i)*tranimg(:,k)+b(:,i));
                grad_sj=es(j)/sum(es)-(j==tranlabl(k));
```

```
                grad_wj=grad_wj+grad_sj*tranimg(:,k)';
                grad_bj=grad_bj+grad_sj;
            end
            %随机梯度下降
            W(j,:,i+1)=W(j,:,i)-step/BATCHSIZE*grad_wj;
            b(j,i+1)=b(j,i)-step/BATCHSIZE*grad_bj;
        end
        %参数的更新距离小于容差时 认为已经收敛 停止算法
        if norm(W(:,:,i+1)-W(:,:,i))+norm(b(:,i+1)-b(:,i))<tol
            W=W(:,:,1:i+1);
            b=b(:,1:i+1);
            break;
        end
    end
end

```

msgd.m：随机梯度法

2.4 计算结果与讨论

在测试中，对两种算法进行测试，为了减少计算，只对每隔若干步迭代得到的模型进行测试，计算参数的收敛情况和模型的精度，测试精度时在测试集中随机抽取足够数量的测试数据进行测试。测试脚本如下：

```
%MNIST数据来自http://yann.lecun.com/exdb/mnist/
%MNIST数据读取方法来自https://blog.csdn.net/tracer9/article/details/51253604
tranimg=loadMNISTImages('train-images.idx3-ubyte');%MNIST数据读取方法
tranlabl=loadMNISTLabels('train-labels.idx1-ubyte');%MNIST数据读取方法
tsimg=loadMNISTImages('t10k-images.idx3-ubyte');%NIST数据读取方法
tslabl=loadMNISTLabels('t10k-labels.idx1-ubyte');%NIST数据读取方法

%测试随机梯度法
[W,b]=msgd(tranimg,tranlabl,0.1,10,1e-4,100);
testres(tsimg,tslabl,W,b,'MSGD');
%测试梯度下降法
[W,b]=gd(tranimg,tranlabl,0.1,1e-4,100);
testres(tsimg,tslabl,W,b,'GD');

%测试每一步的收敛情况和模型的准确率
function testres(tsimg,tslabl,W,b,fname)
    n_iter=size(b,2);
    %对每经过若干步更新的模型进行测试
    SAMPLE_SIZE=min(n_iter,400);
    s_iter=round(linspace(1,n_iter,SAMPLE_SIZE));
    %随机测试数据（类似batch）
    TEST_SIZE=200;
    converr=zeros(n_iter,1);%收敛情况
    accura=zeros(n_iter,1);%准确率
    for ii=1:SAMPLE_SIZE
        i=s_iter(ii);
        converr(i)=norm(W(:,:,i)-W(:,:,end))+norm(b(:,i)-b(:,end));
        testbatch=randi([1,length(tslabl)],TEST_SIZE,1);
        %在随机测试数据上测试当前模型准确率
        for ki=1:TEST_SIZE
            k=testbatch(ki);
            es=exp(W(:,:,i)*tsimg(:,k)+b(:,i));
            es=es/sum(es);
            [~,out]=max(es);
            accura(i)=accura(i)+(out==tslabl(k));
```

```
        end
        accura(i)=accura(i)/TEST_SIZE;
    end
    figure
    title([fname,' converge error']);
    ylabel('converge error');
    xlabel('iteration');
    hold on;
    plot(s_iter,converr(s_iter),'DisplayName','converge error')
    figure
    title([fname,' accuracy']);
    ylabel('accuracy');
    xlabel('iteration');
    hold on;
    plot(s_iter,accura(s_iter),'DisplayName','accuracy');
end

```

mtest.m：测试脚本

运行脚本文件，结果如下：

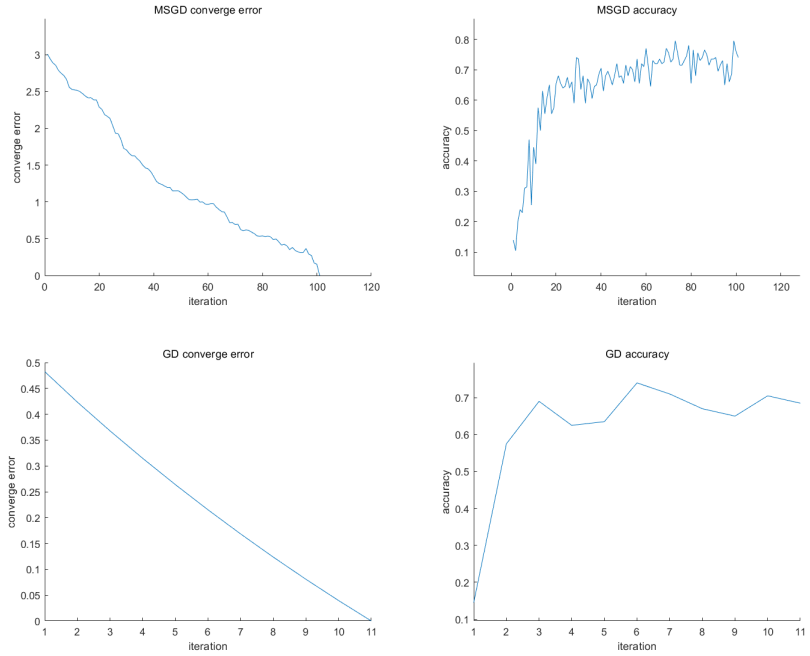

图 5: 梯度下降法步长 =0.1，最大迭代次数 =10；随机梯度法 batch 大小 =10，步长 =0.1，最大迭代次数 =100

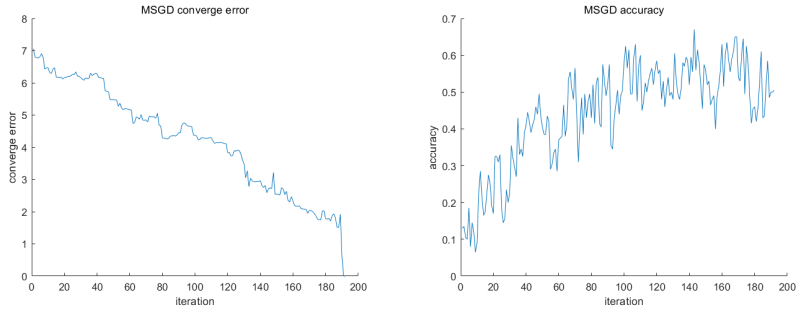

图 6: 随机梯度法 batch 大小 =1，步长 =0.1，最大迭代次数 =1000

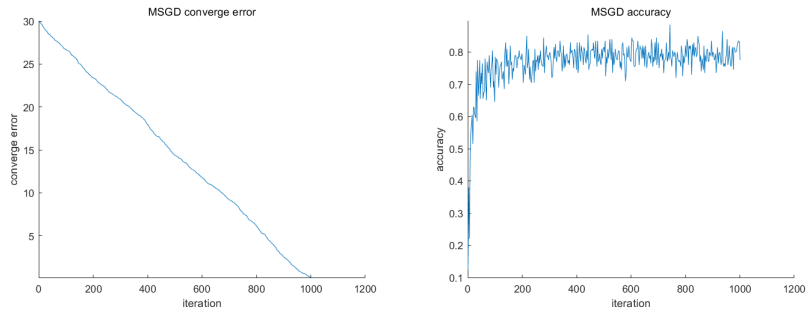

图 7: 随机梯度法 batch 大小 =100，步长 =0.1，最大迭代次数 =1000

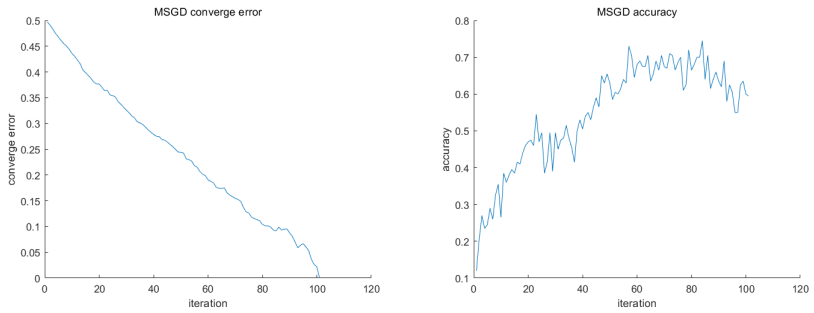

图 8: 随机梯度法 batch 大小 =10，步长 =0.01，最大迭代次数 =100

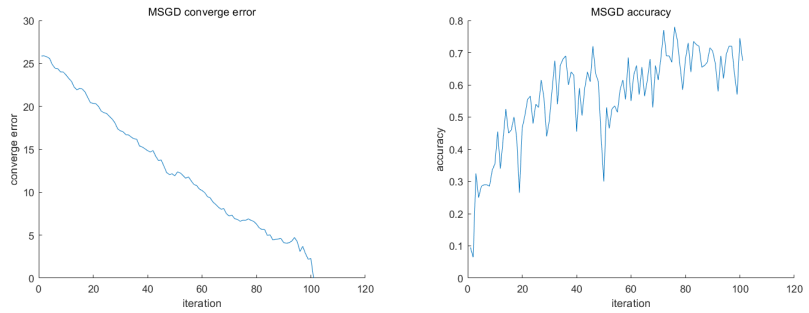

图 9: 随机梯度法 batch 大小 =10，步长 =1，最大迭代次数 =100

如图 5 所示，相比于随机梯度法，梯度下降法的收敛更平稳，所需迭代次数更少（但实际上每次迭代运行时间远高于随机梯度法），因为计算量大，所以在一定时间内连 100 次迭代都做不了，只做了 10 次迭代。

如图 5-7 所示，当随机梯度法的 batch 很小时（如为 1），收敛情况很不稳定，最后的收敛是由突变引起的，而此时的模型精度只有 0.5 左右。当 batch 增大时（如为 10，100），收敛情况逐渐平稳、加快，且收敛到的模型精确度有一定提升。

如图 5、8-9 所示，减小、增大随机梯度法固定步长对结果影响不大，适中的步长（如 0.1）效果较好。

综上，在计算量允许范围内提高随机梯度法的 batch 大小，采取适中的步长（如果是固定的话），对于问题 (10) 的求解效果较好。