# 最优化理论期末报告

数据科学与计算机学院　17大数据与人工智能

17341015　陈鸿峥

## 一、 问题一

### 1. 问题描述

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

$$\min\left(\frac{1}{2}\left\|A\mathbf{x}-\mathbf{b}\right\|_2^2+p\left\|\mathbf{x}\right\|_1\right)$$

其中$p$为非负的正则化参数。

请设计下述算法求解该问题：

1. 邻近点梯度下降法
2. 交替方向乘子法
3. 次梯度法

在实验中，设$\mathbf{x}$的真值中的非零元素服从均值为0方差为1的高斯分布，$A$中的元素服从均值为0方差为1的高斯分布，$\mathbf{e}$中的元素服从均值为0方差为0.1的高斯分布。对于每种算法，请给出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，请讨论正则化参数$p$对计算结果的影响。

### 2. 算法设计

#### (i) 邻近点梯度下降法

设

$$\begin{cases}s(\mathbf{x}):=\frac{1}{2}\left\|A\mathbf{x}-\mathbf{b}\right\|_2^2\\ r(\mathbf{x}):=p\left\|\mathbf{x}\right\|_1\end{cases}$$

其中，$A\in\mathbb{R}^{m\times n}$，$\mathbf{x}\in\mathbb{R}^n$，$\mathbf{b},\mathbf{e}\in\mathbb{R}^m$（在本题中$m=50,n=100$），则原式

$$f(\mathbf{x}):=\left(\frac{1}{2}\left\|A\mathbf{x}-\mathbf{b}\right\|_2^2+p\left\|\mathbf{x}\right\|_1\right)=s(\mathbf{x})+r(\mathbf{x})$$

先求$r(\mathbf{x})$的邻近点投影

$$\operatorname{prox}\hat{\mathbf{x}}=\arg\min_{\mathbf{x}}\left(p\left\|\mathbf{x}\right\|_1+\frac{1}{2\alpha}\left\|\mathbf{x}-\hat{\mathbf{x}}\right\|^2\right)\tag{1}$$

对1式右侧展开有

$$\arg\min_{\mathbf{x}} \left( p\sum_{i=1}^{n}|x_i| + \frac{1}{2\alpha}\sum_{i=1}^{n}(x_i-\hat{x}_i)^2 \right) \tag{2}$$

注意到(2)式对于下标$i$相互独立，故要求2的最小值，等价于对每一个下标$i$求最小值后求和，即

$$\arg\min_{x_i} \left( p|x_i| + \frac{1}{2\alpha}(x_i-\hat{x}_i)^2 \right),\ \forall i \tag{3}$$

由不可微函数的极值判断条件有

$$0 \in \partial_{x_i} p|x_i| + \frac{1}{\alpha}(x_i-\hat{x}_i) \tag{4}$$

对$x_i$进行分类讨论

- 若$x_i > 0$，则$|x_i|$对于$|x_i|$可微，即$\partial_{x_i}|x_i| = 1$，有

  $$p + \frac{1}{\alpha}(x_i-\hat{x}_i) = 0$$

  整理得

  $$x_i = \hat{x}_i - \alpha p$$

  由于$x_i > 0$，故$\hat{x}_i - \alpha p > 0$，即$\hat{x}_i > \alpha p$
- 若$x_i < 0$，则$|x_i|$对于$|x_i|$可微，即$\partial_{x_i}|x_i| = -1$，有

  $$-p + \frac{1}{\alpha}(x_i-\hat{x}_i) = 0$$

  整理得

  $$x_i = \hat{x}_i + \alpha p$$

  由于$x_i < 0$，故$\hat{x}_i + \alpha p < 0$，即$\hat{x}_i < -\alpha p$
- 若$x_i = 0$，则$|x_i|$对于$|x_i|$不可微，需要求次梯度，$\partial_{x_i}|x_i| = [-1, 1]$，即

  $$0 \in \left[ -p - \frac{\hat{x}_i}{\alpha}, p - \frac{\hat{x}_i}{\alpha} \right]$$

  那么，需要满足

  $$\begin{cases} p - \dfrac{\hat{x}_i}{\alpha} \geq 0 \\ -p - \dfrac{\hat{x}_i}{\alpha} \leq 0 \end{cases}$$

  推得

  $$\hat{x}_i \in [-\alpha p, \alpha p]$$

综上，有

$$x_i = \begin{cases} \hat{x}_i + \alpha p & \hat{x}_i < -\alpha p \\ 0 & \hat{x}_i \in [-\alpha p, \alpha p] \\ \hat{x}_i - \alpha p & \hat{x}_i > \alpha p \end{cases} \tag{5}$$

可以得到图1的软门限图

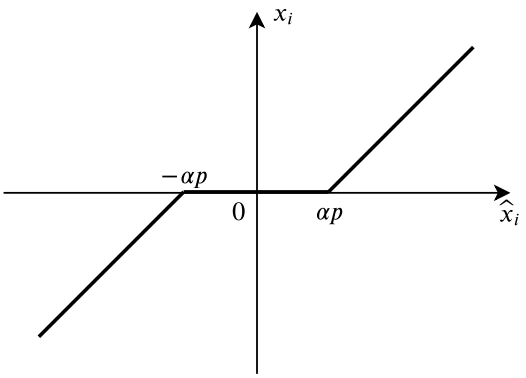


图 1: 关于$x_i$与$\hat{x}_i$的软门限图

进一步，得到邻近点梯度下降法的迭代式如下

$$\begin{cases} \mathbf{x}^{(k+\frac{1}{2})} = \mathbf{x}^{(k)} - \alpha \nabla s(\mathbf{x}^{(k)}) = \mathbf{x}^{(k)} - \alpha A^T(A\mathbf{x}^{(k)} - \mathbf{b}) \\ \mathbf{x}^{(k+1)} = \operatorname{prox} \mathbf{x}^{(k+\frac{1}{2})} = \arg\min_{\mathbf{x}} \left( p \|\mathbf{x}\|_1 + \dfrac{1}{2\alpha} \left\| \mathbf{x} - \mathbf{x}^{(k+\frac{1}{2})} \right\|_2^2 \right) \end{cases} \tag{6}$$

其中，$x^{(k+\frac{1}{2})}$可直接计算，$x^{(k+1)}$可由(5)式求得。

**(ii) 交替方向乘子法**

对原问题进行变形，等价于下述约束问题

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

构造增广拉格朗日函数

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

可以得到交替方向乘子法的迭代格式

$$\begin{cases} \mathbf{x}^{(k+1)} = \arg\min_{\mathbf{x}} L_c(\mathbf{x}, \mathbf{y}^{(k)}, \mathbf{v}^{(k)}) \\ \mathbf{y}^{(k+1)} = \arg\min_{\mathbf{y}} L_c(\mathbf{x}^{(k+1)}, \mathbf{y}, \mathbf{v}^{(k)}) \\ \mathbf{v}^{(k+1)} = \mathbf{v}^{(k)} + c(\mathbf{x}^{(k+1)} - \mathbf{y}^{(k+1)}) \end{cases} \tag{8}$$

对$\mathbf{x}^{(k+1)}$展开并配方，并将非主元项忽略，可求得(8)与下面的式子等价

$$\begin{cases} \mathbf{x}^{(k+1)} = \arg\min_{\mathbf{x}} \left( \frac{1}{2} \left\| A\mathbf{x} - \mathbf{b} \right\|_2^2 + \frac{c}{2} \left\| \mathbf{x} - \mathbf{y}^{(k)} + \frac{\mathbf{v}^{(k)}}{c} \right\|_2^2 \right) & \text{(9a)} \\ \mathbf{y}^{(k+1)} = \arg\min_{\mathbf{y}} \left( p \left\| \mathbf{y} \right\|_1 + \frac{c}{2} \left\| \mathbf{x}^{(k+1)} - \mathbf{y} + \frac{\mathbf{v}^{(k)}}{c} \right\|_2^2 \right) & \text{(9b)} \\ \mathbf{v}^{(k+1)} = \mathbf{v}^{(k)} + c(\mathbf{x}^{(k+1)} - \mathbf{y}^{(k+1)}) & \text{(9c)} \end{cases}$$

对于(9a)式，可直接通过求梯度的方法得到显式解，即

$$\mathbf{x}^{(k+1)} = (A^{\mathrm{T}}A + cI)^{-1}(A^{\mathrm{T}}\mathbf{b} + c\mathbf{y}^{(k)} - \mathbf{v}^{(k)})$$

对于(9b)式，由于涉及一范数，故需要求次微分，类似(5)式的方法，设$z_i = x_i^{(k+1)} + \frac{v_i^{(k)}}{c}$，可得到

$$y_i = \begin{cases} z_i - \frac{p}{c} & z_i > \frac{p}{c} \\ 0 & z_i \in \left[ -\frac{p}{c}, \frac{p}{c} \right] \\ z_i + \frac{p}{c} & z_i < -\frac{p}{c} \end{cases}$$

进而可以通过(9a)(9b)(9c)式迭代求解。

**(iii) 次梯度法**

对原式直接求次梯度有

$$\partial f(\mathbf{x}) = A^{\mathrm{T}}(A\mathbf{x} - \mathbf{b}) + p\partial \left\| \mathbf{x} \right\|_1 \tag{10}$$

其中

$$(\partial \left\| \mathbf{x} \right\|_1)_i = \partial |x_i| = \begin{cases} 1 & x_i > 0 \\ [-1, 1] & x_i = 0 \\ -1 & x_i < 0 \end{cases}$$

进而可以直接得到次梯度法的迭代格式

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha \partial f(\mathbf{x}^{(k)}) \tag{11}$$

## 3. 数值实验

采用Python进行编程[1]，对上述三种方法进行迭代计算，直到精度达到$10^{-8}$。

完整代码请见p1.py文件，这里只截取核心代码部分。

邻近点梯度法如下，设超参数$\alpha = 10^{-3}$。

```
class ProximalGradient():

    def __init__(self, alpha=1e-3):
        self.name = "Proximal Gradient"
        self.alpha = alpha

    def soft_thresholding(self, x, offset):
        if x < (-1) * offset:
            return x + offset
        elif x > offset:
            return x - offset
        else:
            return 0

    def prox(self, xk_old, offset):
        # v_soft_thresholding = np.vectorize(self.soft_thresholding)
        # return v_soft_thresholding(xk_old,offset)
        xk_new = np.zeros(xk_old.size)
        for i in range(xk_old.size):
            xk_new[i] = self.soft_thresholding(xk_old[i],offset)
        return xk_new

    def train(self,A,b,p):
        _, self.n = A.shape
        self.xk = np.zeros(self.n)

        res = []
        t = 0
        while True:
            xhat = self.xk - self.alpha * np.dot(A.T, np.dot(A, self.xk) - b)
            xk_new = self.prox(xhat, self.alpha * p)
            if np.linalg.norm(xk_new - self.xk, ord=2) < accuracy:
                break
```

[1]使用numpy进行数值计算，matplotlib进行画图

```
            res.append(xk_new)
            self.xk = xk_new.copy()
            t += 1

        print(t)
        return self.xk, res
```

交替方向乘子法如下，设超参数$c = 10^{-3}$。

```
class ADMM():

    def __init__(self, c=1e-3):
        self.name = "ADMM"
        self.c = c

    def soft_thresholding(self, x, offset):
        if x < (-1) * offset:
            return x + offset
        elif x > offset:
            return x - offset
        else:
            return 0

    def prox(self, xk_old, offset):
        xk_new = np.zeros(xk_old.size)
        for i in range(xk_old.size):
            xk_new[i] = self.soft_thresholding(xk_old[i],offset)
        return xk_new

    def train(self,A,b,p):
        _, self.n = A.shape
        self.xk = np.zeros(self.n)
        self.yk = np.zeros(self.n)
        self.vk = np.zeros(self.n)

        res = []
        t = 0
        while True:
            xk_new = np.dot(
                np.linalg.inv(np.dot(A.T, A) + self.c * np.eye(self.n,self.n)),
                np.dot(A.T, b) + self.c * self.yk - self.vk)
            self.yk = self.prox(xk_new + self.vk / self.c, p / self.c)
            self.vk = self.vk + self.c * (xk_new - self.yk)
            if np.linalg.norm(xk_new - self.xk, ord=2) < accuracy:
                break
            res.append(xk_new)
```

```
            self.xk = xk_new.copy()
            t += 1

        print(t)
        return self.xk, res
```

次梯度法如下，采用递减步长$\alpha^{(k+1)} = \alpha^{(k)}/(k+1)$，初始步长$\alpha^{(0)} = 10^{-3}$。

```
class Subgradient():

    def __init__(self,alpha=1e-3):
        self.name = "Subgradient"
        self.alpha = alpha

    def subgrad(self,x):
        # subgradient of |x|
        pdx = np.zeros(x.size)
        for i in range(x.size):
            if x[i] != 0:
                pdx[i] = 1 if x[i] > 0 else -1
            else: # pick a random float from [-1,1]
                pdx[i] = 2 * np.random.random() - 1
        return pdx

    def train(self,A,b,p):
        _, self.n = A.shape
        self.xk = np.zeros(self.n)

        res = []
        t = 0
        while True:
            alphak = self.alpha / (t + 1) # remember to decay the step
            pdf = np.dot(A.T, np.dot(A, self.xk) - b) + self.subgrad(self.xk)
            xk_new = self.xk - alphak * pdf
            if np.linalg.norm(xk_new - self.xk, ord=2) < accuracy:
                break
            res.append(xk_new)
            self.xk = xk_new.copy()
            t += 1

        print(t)
        return self.xk, res
```

### 4. 结果分析

记最优解$\mathbf{x}^\star$为最后一次迭代获得的$\mathbf{x}$值，真值为$\mathbf{x}_{\text{true}}$。

运行结果如图2、图3和图4所示，该次实验$p = 0.1$。

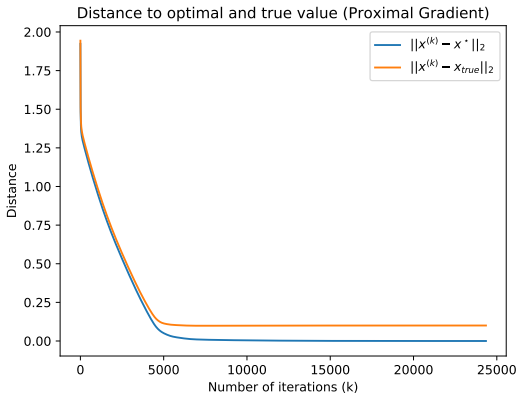


图 2: 邻近点梯度下降法

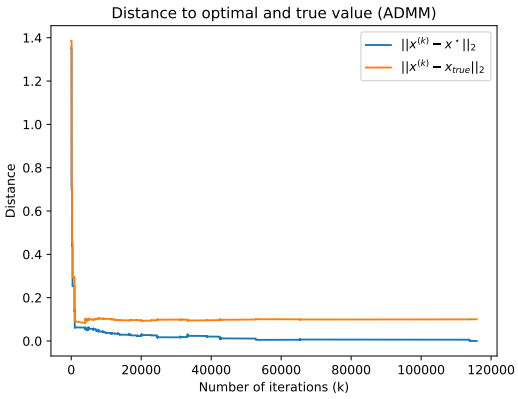


图 3: 交替方向乘子法

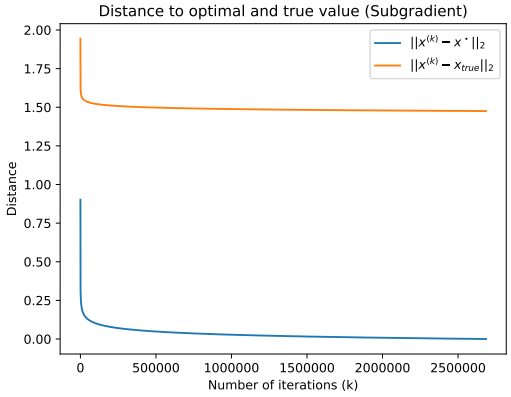


图 4: 次梯度法

由实际的运行时间可以得出，邻近点梯度法和交替方向乘子法达到目标精度需要的时间都比较短，次梯度法所需的时间最长。从迭代的次数也可以看出，邻近点梯度下降法需要24349轮迭代，交替方向乘子法需要115901轮迭代，次梯度法需要2684797轮迭代，都相差了一个数量级。（此实验仅仅展示了一种情况，实际上迭代次数与超参数的选择有关，见下面的实验。）

不仅如此，从上述三幅图中还可以看出，交替方向乘子法和邻近点梯度下降法都能达到很高的精度（即与真值相差不多），且收敛速度块，但次梯度法迭代次数多且精度非常低，是一种非常糟糕的算法。

关于正则化参数$p$对最终结果的影响可见图5、图6和图7。

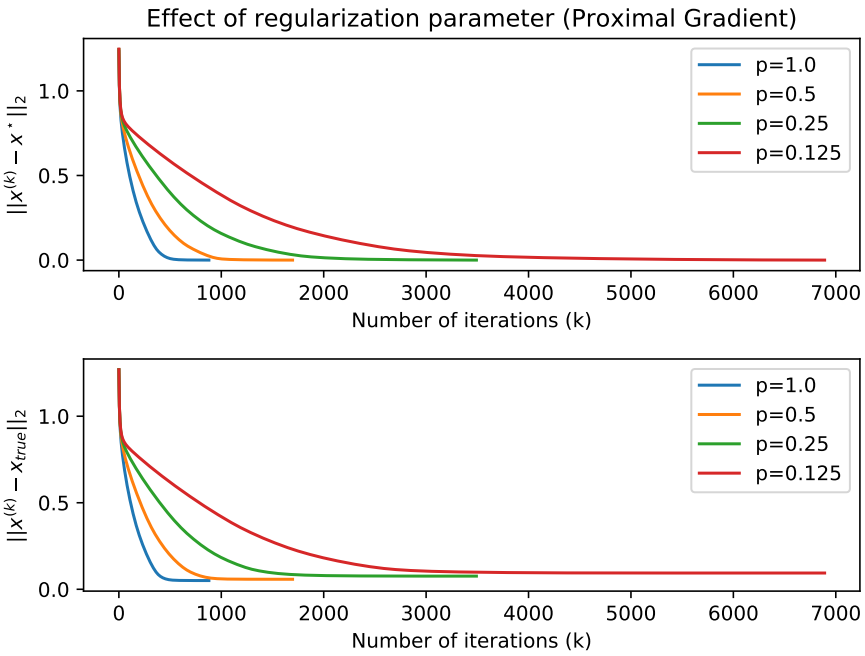


图 5: 邻近点梯度下降法

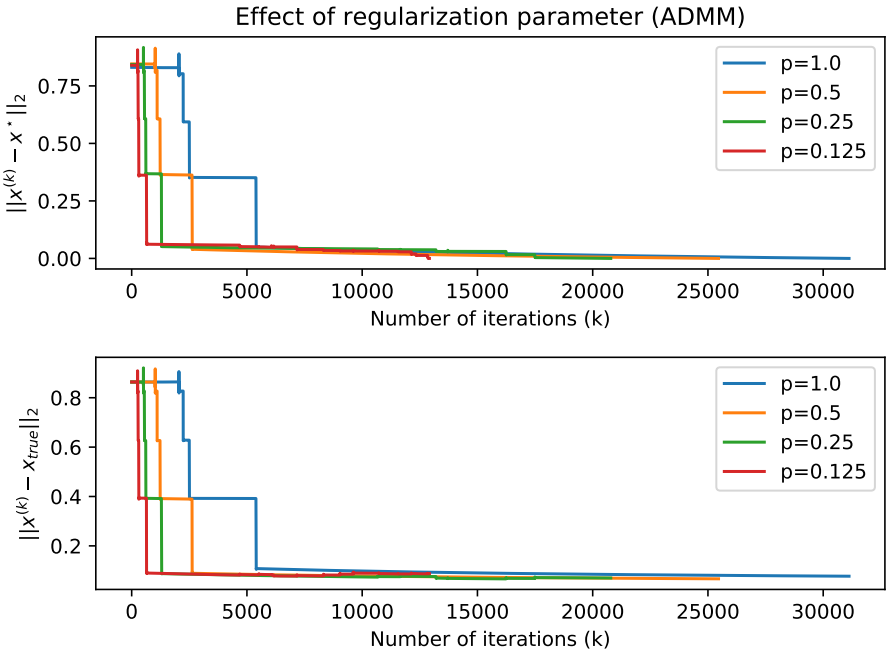


图 6: 交替方向乘子法

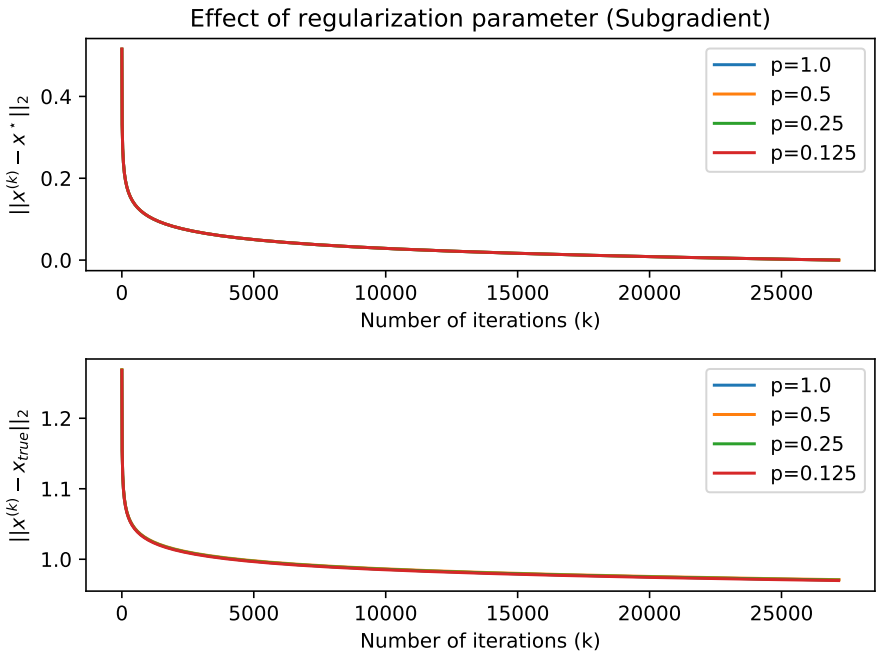


图 7: 次梯度法

从这三幅图中可以看出，当正则化参数$p$对迭代次数和收敛精度都有一定的影响。对于邻近点梯度法，$p$取较大的值时，收敛速度比较快，且精度也较高。对于交替方向乘子法，$p$取较小的值时，收敛速度较快，但精度差异没有邻近点梯度法那么明显。而对于次梯度法，$p$对收敛速度和收敛精度几乎没有影响。

## 二、 问题二

### 1. 问题描述

请设计下述算法，求解MNIST数据集上的Logistic Regression问题：

1. 梯度下降法
2. 随机梯度法

对于每种算法，请给出每步计算结果与最优解的距离以及每步计算结果在测试集上所对应的分类精度。此外，请讨论随机梯度法中Mini Batch大小对计算结果的影响。

可参考：http://deeplearning.net/tutorial/logreg.html

### 2. 算法设计

由于MNIST手写数据集为多分类问题，需要将输入的图片映射到对应的10个数字上，故采用多类别的logistic回归，即softmax回归进行求解。

设训练集$\{(\mathbf{x}_i, y_i)\}_{i=1}^M$，其中$M$为训练集数目，$|\mathcal{C}|$为类别数目，$\mathbf{x}_i \in \mathbb{R}^{n+1}$为输入样本，$y_i \in \{0, 1, \ldots, |\mathcal{C}| - 1\}$为对应的标签。在本问题中$|\mathcal{C}| = 10$，$M = 60000$，$n = 28 \times 28 = 784$，这里

已经将二维的图片数组展平为一维的向量。

输入$\mathbf{x}_i$属于第$y_i$类的条件概率可用softmax函数进行计算

$$p(y_i \mid \mathbf{x}_i; W) = \frac{\mathrm{e}^{\mathbf{w}_{y_i}^{\mathrm{T}} \mathbf{x}_i}}{\sum_{c=0}^{|\mathcal{C}|-1} \mathrm{e}^{\mathbf{w}_c^{\mathrm{T}} \mathbf{x}_i}} \tag{12}$$

其中，

$$W = \begin{bmatrix} \mathbf{w}_0^{\mathrm{T}} \\ \mathbf{w}_1^{\mathrm{T}} \\ \vdots \\ \mathbf{w}_{|\mathcal{C}|-1}^{\mathrm{T}} \end{bmatrix} \in \mathbb{R}^{|\mathcal{C}| \times (n+1)}$$

为需要训练的参数，$\mathbf{w}_i \in \mathbb{R}^{n+1}$为每一个类别对应的权重向量[2]。

从而得到数据集的似然函数为

$$\mathcal{L}(Y \mid X; W) = \prod_{i=1}^{m} p(y_i \mid \mathbf{x}_i; W)$$

对应的对数似然函数为

$$\ln \mathcal{L}(Y \mid X; W) = \sum_{i=1}^{m} \ln p(y_i \mid \mathbf{x}_i; W)$$

极大化对数似然函数，相当于极小化负对数似然函数(negative log likelihood, NLL)[3]，故得到最优化问题

$$\min_{W} \ \ell(W) := -\ln \mathcal{L}(Y \mid X; W) \tag{13}$$

$\ell(W)$其实也是深度学习领域常说的损失函数(loss)。

对$\ln p(y_i \mid \mathbf{x}_i; W)$求梯度，有

$$\frac{\partial \ln p(y_i \mid \mathbf{x}_i; W)}{\partial \mathbf{w}_j} = \begin{cases} \mathbf{x}(1 - p(y_i \mid \mathbf{x}_i; W)) & y_i = j \\ \mathbf{x} p(y_i \mid \mathbf{x}_i; W) & y_i \neq j \end{cases}, \ j \in \{0, 1, \ldots, |\mathcal{C}| - 1\}$$

进而

$$\nabla_{\mathbf{w}_j} \ell(W) = -\sum_{i=1}^{m} \left[ \mathbf{x}_i \left( \mathbb{1}(y_i = j) - p(y_i \mid \mathbf{x}_i; W) \right) \right] \tag{14}$$

其中，$\mathbb{1}(\cdot)$为示性函数，当输入为真时返回1，输入为假时返回0。

---

[2]注意到$\mathbf{w}^{\mathrm{T}}\mathbf{x} + b = \begin{bmatrix} \mathbf{w}^{\mathrm{T}} & b \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$，故将最后的偏置项归入$\mathbf{w}$中，并在$\mathbf{x}$中添加为1的一个维度，进而偏置量和权重可以一起运算，$\mathbf{w}_c^{\mathrm{T}}$和$\mathbf{x}_i$都有$n+1$个维度。

[3]由于已经取了softmax函数，并且在实施时采用独热码(one-hot encoding)方便矩阵运算，所以这里的NLL也可以看作是交叉熵函数。

最终得到梯度下降的表达式

$$\mathbf{w}_j^{(k+1)} = \mathbf{w}_j^{(k)} - \frac{\alpha^{(k)}}{m}\nabla_{\mathbf{w}_j}\ell(W) \tag{15}$$

其中，$m$为小批量大小。当$m$选取不同值时，(15)式为不同的优化方法：

- 当$m = M$时，梯度下降法，每次选取**所有样本**用于更新权重
- 当$m = 1$时，随机梯度下降法(SGD)，每次选取**一个样本**用于更新权重
- 当$m \in [2, M-1]$时，小批量梯度下降法，每次只选取**部分样本**用于更新权重

### 3. 数值实验

MNIST输入数据以numpy压缩格式存储，从https://s3.amazonaws.com/img-datasets/mnist.npz获取。可从中直接读出`x_train`、`y_train`、`x_test`和`y_test`。

值得一提的几个实现细节：

- 读入`x_train`和`x_test`后，增加一个全为1的维度，用于与偏移量$b$点乘
- 训练前先进行归一化(normalization)防止计算softmax时数值上溢(overflow)
- 充分利用了numpy的广播(broadcast)、扩展(keepdim)等技巧，使得梯度计算可以直接运用矩阵乘法，而不需逐元素相乘
- 步长/学习率设置为$\alpha = 0.1$

核心代码如下，完整代码请见`p2.py`。

```
class SoftmaxRegression():

    def train(self, X, y_true, n_classes, n_iters=10, learning_rate=0.1,
        ↪ batch_size=1):
        self.n_samples, n_features = X.shape # (M, C)
        self.n_classes = n_classes
        self.batch_size = batch_size
        self.weights = np.random.rand(self.n_classes, n_features)
        all_losses = []
        all_accuracy = []
        all_weights = []

        for i in range(n_iters):
            batch_index = np.array(random.sample(range(self.n_samples),self.
                ↪ batch_size))
            X_batch, y_batch = X[batch_index], y_true[batch_index]
            scores = self.compute_scores(X_batch) # w^T.x
            probs = self.softmax(scores)
            y_one_hot = self.one_hot(y_batch)

```

```
            loss = self.nll_loss(y_one_hot, probs) # target function
            all_losses.append(loss)

            # gradient descent -> update weights
            old_weights = self.weights.copy()
            dw = (1 / self.batch_size) * np.dot(X_batch.T, (probs - y_one_hot))
            self.weights = self.weights - learning_rate * dw.T
            all_weights.append(self.weights)

            if i % 100 == 0 or i == n_iters - 1:
                y_predict = self.predict(X_test)
                all_accuracy.append((np.sum(y_predict == y_test) / X_test.shape[0])
                    ↪ * 100)
                print(f'Iteration number: {i}, loss: {np.round(loss, 4)}, accuracy:
                    ↪ {all_accuracy[-1]}%')

        return all_weights, all_losses, all_accuracy

    def predict(self, X):
        scores = self.compute_scores(X)
        probs = self.softmax(scores)
        return np.argmax(probs, axis=1)[:, np.newaxis]

    def softmax(self, scores):
        exp = np.exp(scores) # (n_samples, n_classes)
        sum_exp = np.sum(np.exp(scores), axis=1, keepdims=True) # sum along classes
        softmax = exp / sum_exp
        return softmax

    def compute_scores(self, X):
        # X: (n_samples, n_features)
        # scores: (n_samples, n_classes)
        return np.dot(X, self.weights.T)

    def nll_loss(self, y_true, probs):
        loss = - (1 / self.batch_size) * np.sum(y_true * np.log(probs))
        return loss

    def one_hot(self, y):
        one_hot = np.zeros((y.size, self.n_classes))
        one_hot[np.arange(y.size), y.T] = 1
        return one_hot
```

## 4. 结果分析

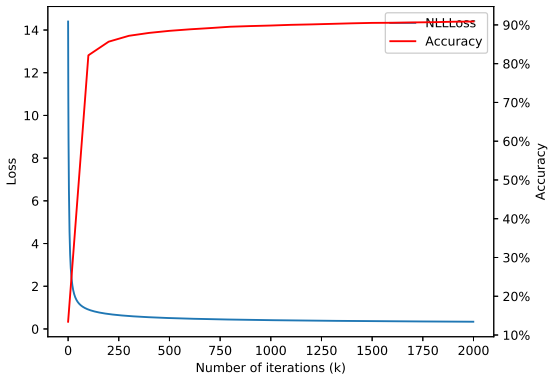


图 8: 梯度下降法损失函数及分类精度变化

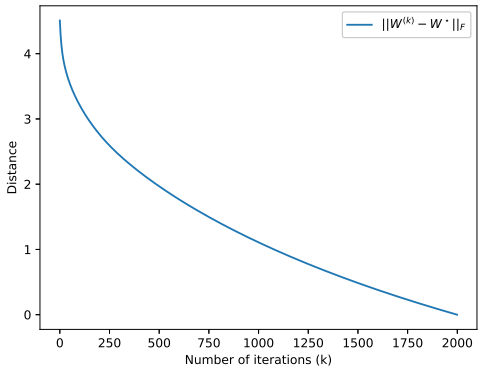


图 9: 梯度下降法与最优解的距离变化

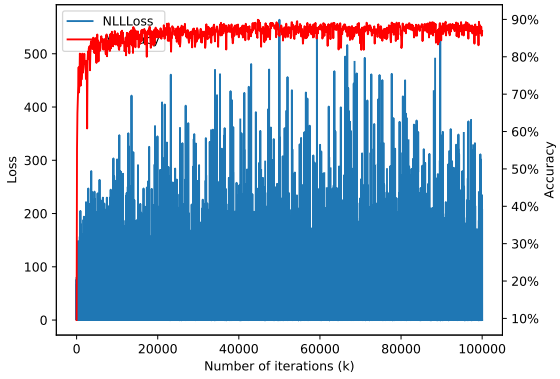


图 10: SGD损失函数及分类精度变化

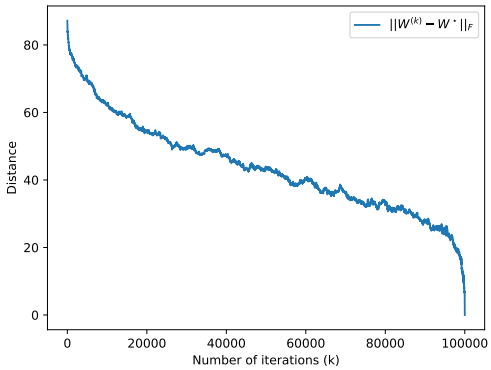


图 11: SGD与最优解的距离变化

从图8中可以看出，运用梯度下降法求解，损失函数不断下降，在测试集上的分类精度不断上升；图9中可以看出，迭代过程确实是在往最优解方向逼近[4]，最终收敛在91.4%的精度。

同样，从图10和图11中也可以得出对随机梯度下降法同样的结论，但是随机梯度下降法明显要不稳定得多，分类精度与损失函数都有较大波动（因为受单一特殊样本影响非常大），最终收敛在88%左右的精度，没有梯度下降法优。

批量大小、精度和训练时间的比较如表1所示。从表中可以看出，当`batch_size`较小时，参数更新的速度快，也能较快收敛，但是最终收敛到的精度往往较低。而`batch_size`较大时，迭代速度非常慢，但往往几轮迭代就能得到很好的效果，而且随着迭代轮数的不断增加，分类的精度也不断在提升。如此例中，100到1000之间的`batch_size`就是比较好的选择。

[4]注：这里采用了矩阵的F范数作为距离度量。

表 1: 批量大小、精度和训练时间的比较

| 批量大小 | 迭代次数 | 精度 | 训练时间 |
|---|---|---|---|
| 1 | 10000 | 86.67% | 2.38s |
| 10 | 10000 | 88.08% | 3.21s |
| 100 | 5000 | 91.04% | 2.97s |
| 1000 | 5000 | 91.63% | 38.81s |
| 10000 | 1000 | 89.46% | 69.88s |
| 60000 | 1000 | 89.86% | 404.68s |
| 1 | 2500 | 82.6% | 0.58s |
| 10 | 2500 | 87.69% | 0.64s |
| 100 | 2500 | 91.0% | 1.30s |
| 1000 | 2500 | 91.31% | 17.64s |
| 10000 | 2500 | 90.89% | 170.88s |
| 60000 | 2500 | 90.88% | 1010.26s |

图12展示了批量大小(`batch_size`)对最终分类精度的影响。我们也可以得到同样的结论，因此`batch_size`的选取一定要适中，太小影响精度，太大影响训练时间。通过多个参数的测试，就能快速找到比较合适的`batch_size`，进而加速训练并且训练的模型具有非常好的性能。

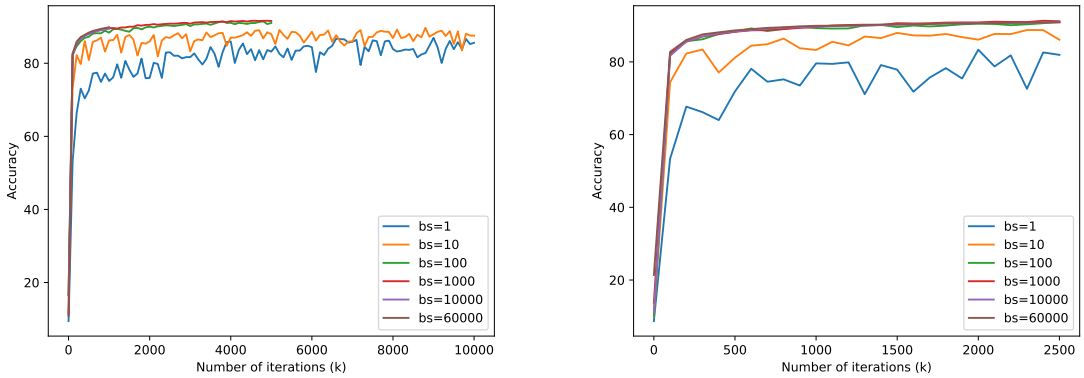


图 12: 批量大小对分类精度的影响

## 三、 参考资料

1. Stanford CS231n, http://cs231n.github.io/
2. Stanford UFLDL Tutorial - Softmax Regression, http://deeplearning.stanford.edu/tutorial/supervised/SoftmaxRegression/