PRML Lab2 Report

Part 1

任务描述

分别使用`the least square model`和`perceptron model`对二维数据进行二分类处理。

线性模型

本报告所说的线性模型指扩展的线性模型，他是一个非线性函数对线性函数的嵌套。对于二分类问题，首先对数据集进行线性变换，然后使用一个非线性函数`f`将结果映射到两个类。

$$
y(x) = f(W^T x + w_0) \\
where: f(x) = \begin{cases} 1 & , x > 0.5 \\ 0 & , else \end{cases} \\
y(x)\, represents\, the\, class\, that\, x\, belongs\, to.
$$

为了方便处理，我们将对数据的维度进行延拓。

$$
y(x) = f(W^T x + w_0) = f(W'^T x') \\
where\ W' = [w_0, W]\ and\, x' = [1, x]
$$

经过这样的处理，我们将偏移整合到了W中，使得模型更加的简洁。下面的报告中，若不特指，模型的输入都经过了这种延拓处理。

The Least Square Model

模型简述

该模式使用平方误差作为模型的Loss函数，即：

$$
Loss(W) = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}(y_k(x_n) - t_{nk})^2 \\
and\, we\, want\, to\, minimaze\, Loss(W)
$$

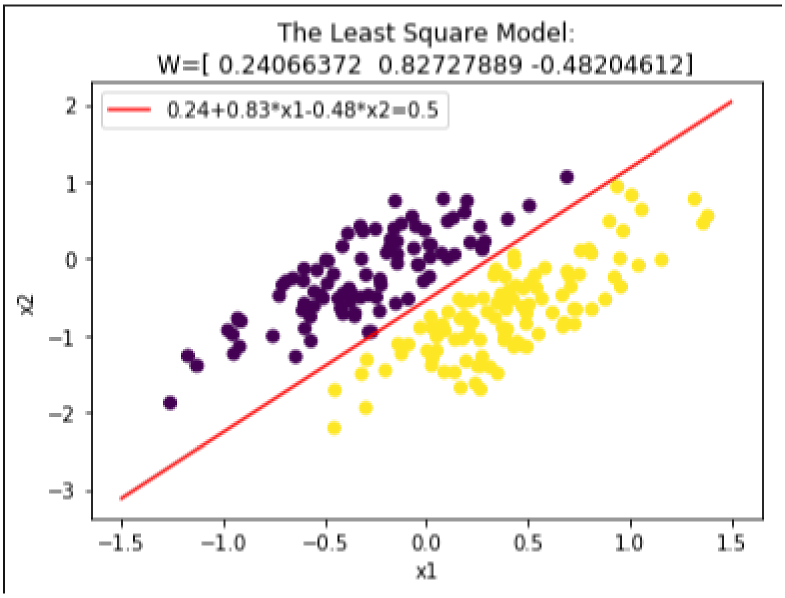
模型求解

这个问题的求解在本问题中实际上就是最小二乘法求解问题，他是有确定解的，即

$$
XW = Y, where\, X\, hasn't\, a\, reverse\, matrix. \\
=>\ \ X^T X W = X^T Y \\
=>\ \ W' = (X^T X)^{-1} X^T Y \\
W'\, is\, the\, closest\, solution\, to\, W
$$

于是直接通过数据集的X，Y坐标即可计算得出最优的W

分类结果



正确率

由于获取的数据集是线性可分的，所以测试的结果为100%。

Perceptron Model

模型简述

该模型与上一个模型最大的区别便是求解方法的不同，Perceptron模型使用梯度下降的方法进行求解，由于本次实验涉及的二分类问题是线性可分的，所以该方法一定会收敛。

模型求解

该模型求解的关键在于梯度的求解和W的更新公式，如下：

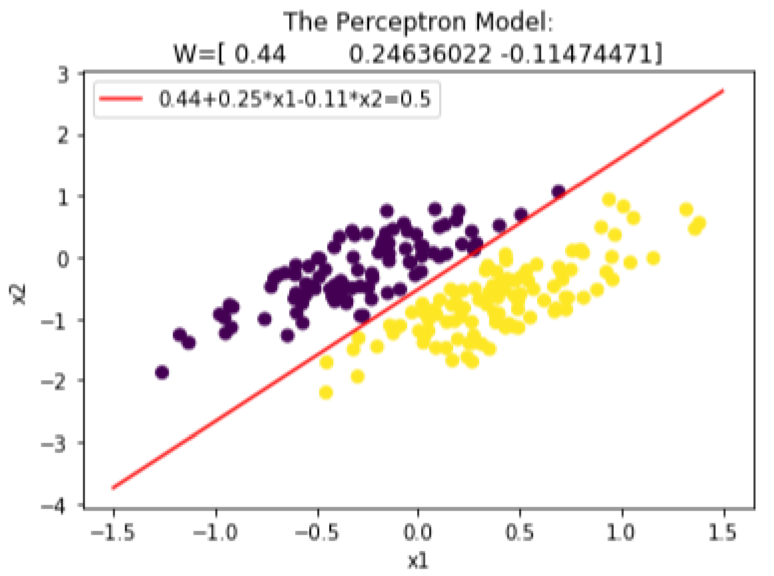
$$
\begin{aligned}
W_{next} &= W + \Delta W \\
&= W + \eta \nabla L \\
\end{aligned}
$$

$$where\, \eta\, is\, the\, learning\_rate$$

$$W_{next} = \eta \phi(x_n) t_n$$

$$where: \phi(x) = \begin{cases} x_N & , W^T x_n > 0.5 \\ -x_n & , else \end{cases} \quad and\ t_n = \begin{cases} 1 & , x_n\, belongs\, to\, class\, 1 \\ -1 & , x_n\, belongs\, to\, class\, 0 \end{cases}$$

分裂结果



正确率

同前一个模型，正确率为100%

Part 2

Requirement 1

multi-hot数据处理模型

输入：文本

输出：multihot vector (M维)

- Step1 tokenize：考虑到文本中会含有无词义的字符（如标点和空白），所以需要先将这二者去除。可以先将所有标点转换为空格符(`str.replace`)，然后去除多余的空格符再按空格分词(`str.split()`) *split方法在不传入参数时默认情况会按照string.whitespace分词*
- Step2 word count：对所有的问题进行词频统计，并过滤掉词频小于min_support的词(本实验中该值取10)
- Step3 multi-hot transformation：经过上一步的处理，可以获得一个M大小的字典，使用此字典可以将原文本映射到一个M维的multi-hot向量，如果文本中包含字典中的某一个词，那么在向量中对应的位置为1，否则为0.

one-hot数据预处理模型

输入：类编号t

输出：4维one-hot向量

过程：创建一个4维的全零向量，将t位置设置为1.

Requirement 2

符号说明

符号	含义	符号	含义
N	训练集大小（multi-hot vector)	L	Loss function
M	multi-hot vector 的维度	S	Sotfmax的输入 NxK
K	种类数量	P	Softmax的输出 NxK
X	训练集 MxN	D	矩阵求导符号
x	单个训练样本 Mx1	R	Softmax function
Y	训练集的分类情况（one-hot） KxN	F	现行变换函数
y	单个训练样本的分类情况 Kx1	\eta	学习率
W	线性变换参数矩阵MxK	\lambda	惩罚力度
b	偏置 1xK		

Loss Function Derivation

首先，我们先确定最普通的Loss函数（不加正则化）的形式：

$$\begin{aligned} L(W,b) &= -\frac{1}{N}\sum_{n=1}^{N} y_n^T log(p_n) && \text{where } y_n \text{ is 1xM and pn is Mx1} \\ &= -\frac{1}{N}\sum_{n=1}^{N} log(\hat{p}_n) && \text{where } \hat{p}_n \text{ is the value of } p_n \text{ on } \hat{y}_n\ dimension \\ &= -\frac{1}{N}\sum_{n=1}^{N} L_n(W,b) \end{aligned}$$

接下来使用chain rule对L(W,b)关于W求导

$$DL(W) = \sum_{n=1}^{N} DL_n(W,b)$$

$$\begin{aligned} D_W L_n(W,b) &= D_W(L_n(g(P_{nk}))) && \text{let's assume that the calss of } x_n \text{ is k} \\ &= g'(P_{nk}) D_W P_{nk} && \text{where } g(P_{nk}) = log(P_{nk}) \\ &= \frac{1}{P_{nk}} D_W(R(S))_{nk} \\ &= \frac{1}{P_{nk}} D_S P_{nk} \cdot D_W S \\ &= \frac{1}{P_{nk}} D_S P_{nk} \cdot D_W(F_n(W,b)) \end{aligned}$$

经过chain rule的处理后，分别对两个子问题进行求导。

对 $D_S P_{nk}$ 进行求导如下:

We know that in function R(:Softmax function:)

$$Sotfmax(S_{ni}) = \frac{e^{S_{ni}}}{\sum_{k=1}^{K} e^{S_{nk}}}$$

for convenient, we can write like this(where we assume that N=1):

$$F(a_i) = \frac{e^{a_i}}{\sum_{j=1}^{K} e^{a_j}} = \frac{e^{a_i}}{\sum}$$

Let's differeniate on a_i :

$$\frac{\partial F(a_i)}{\partial a_i} = \frac{e^{a_i}\sum - e^{a_i}e^{a_i}}{\sum^2} = \frac{a^{a_i}}{\sum}\frac{\sum - e^{a_i}}{\sum} = F(a_i)(1 - F(a_i))$$

Next, let's differentiate on a_j where $i \neq j$

$$\frac{\partial F(a_i)}{\partial a_i} = e^{a_i}\frac{-a_j}{\sum^2} = -\frac{a^{a_i}}{\sum}\frac{e^{a_j}}{\sum} = -F(a_i)F(a_j)$$

thus, we can express our problem's solution the same way:

$$D_{S_{nk'}} P_{nk} = \begin{cases} P_{nk}(1 - P_{nk}) & , k' = k \\ -P_{nk}P_{nk'} & , k' \neq k \end{cases}$$

(we can also know that for $m \neq n, D_{S_m} P_{nk} = \vec{0}$)

$$D_S P_{nk} \begin{aligned} &= \begin{bmatrix} -P_{nk}P_{n1} & -P_{nk}P_{n2} & \cdots & P_{nk}(1 - P_{nk}) & \cdots & -P_{nk}P_{nK} \end{bmatrix} \\ &= P_{nk}\begin{bmatrix} -P_{n1} & -P_{n2} & \cdots & (1 - P_{nk}) & \cdots & -P_{nK} \end{bmatrix} \end{aligned} \in R^{1\times K}$$

对 $D_W F_n(W,b)$ 进行求导如下：

we know that $X \in R^{M\times N} W \in R^{M\times K}, b \in R^{N\times K}, S = F(W,b) \in R^{N\times K}$

$and\ F(W,b) = [F_1(W,b), F_2(W,b), \ldots F_N(W,b)]^T, F_n(W,b) \in R^{1\times K}$

so the function for all samples is: $F(W,b) = X^T \cdot W + b$

however, we transfered the origin question L to question L_n,

correspondly, we should concentrate on $F_n(W,b) = x_n^T \cdot W + b_n \in R^{1\times K}$

$$F_n(W,b) = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_M \end{bmatrix}^T \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1K} \\ w_{21} & w_{22} & \cdots & w_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MK} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_K \end{bmatrix}^T = \begin{bmatrix} f_1(x_n) \\ f_2(x_n) \\ \cdots \\ f_K(x_n) \end{bmatrix}^T$$

$$f_i(x_n) = w_{1i}x_1 + w_{2i}x_2 + \cdots w_{Mi}x_M + b_i$$

$$\frac{\partial f_i}{\partial w_{ji}} = x_j\ and\ \frac{\partial f_i}{\partial w_{jl}} = 0, where\ l \neq i$$

so the result of derivation can be write like following:

$$D_{W_{ij}} F_{nt} = \{ \begin{matrix} x_i & , j = t \\ 0 & , j \neq t \end{matrix}, F_{nt} = f_t$$

$$D_W F_n(W,b) = \begin{bmatrix} x_1 & x_2 & \cdots & x_M & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & x_1 & x_2 & \cdots & x_M \end{bmatrix} \in R^{K\times MK}$$

until now, we know both $D_S P_{nk}\ and\ D_W F_n(W)$

and the result will be $G_n = D_W L_n(W) \in R^{1\times MK}$

G_n is the gradient that contributed by $x_n, \nabla_n W_{ij} = G_n[(i-1)*M+j]$

Hence, we get the final gradient:

$$\nabla W_{ij} = \eta \sum_{n=1}^{N} G_n[(i-1)*M+j]$$

加上L2正则化项后，我们只需要再对正则化项进行单独求导即可：

$$L(W,b) = -\frac{1}{N}\sum_{n=1}^{N} L_n(W) + \lambda \| W \|^2$$

$$L2(W) = \lambda \sum_{i=1}^{M}\sum_{j=1}^{K} w_{ij}^2$$

$$D_W L2(W) = \lambda \frac{\partial \| W \|^2}{\partial W}$$

$$D_W L2(W) = \lambda \begin{bmatrix} 2w_{11} & 2w_{12} & \cdots & 2w_{1K} & \cdots & 2w_{M1} & 2w_{M2} & \cdots & 2w_{MK} \end{bmatrix} \in R^{1\times MK}$$

所以原来的梯度公式可以改为:

$$\nabla W_{ij} = \eta(2\lambda w_{ij} + \sum_{n=1}^{N} G_n[(i-1)*M+j])$$

同理我们可以求对偏置b的梯度

$$D_b L_n(W,b) = \frac{1}{P_{nk}} D_S P_{nk} \cdot D_b(F_n(W,b))$$
$$\frac{\partial f_i}{\partial b_i} = 1$$
$$\frac{\partial f_i}{\partial b_j} = 0, where\, j \neq i$$
$$let\, B_n = D_b L_n(W,b) = \frac{1}{P_{nk}} D_S P_{nk}$$
$$\nabla b_{ij} = \eta(\sum_{n=1}^{N} B_n[(i-1)*M+j])$$

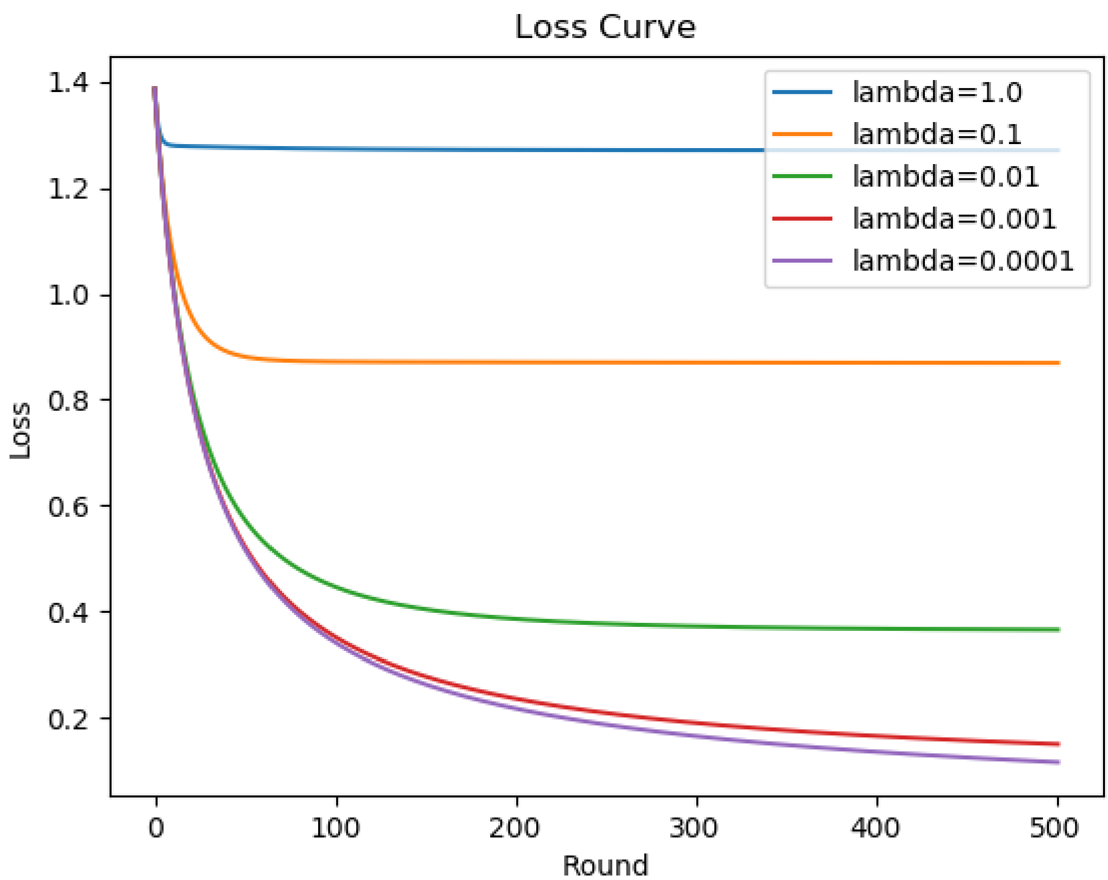
正则化与偏置

一般情况下我们不会把偏置加入正则化项。原因有两点:

- 对偏置的惩罚对模型的正确性没有影响，对w惩罚已经可以避免过拟合。
- 对偏置的惩罚有可能会增强模型对训练集的敏感性。（如需要的偏置极大的模型)

惩罚力度

加入正则化项时，惩罚力度是一个重要的超参，它的选择会影响模型的效果。实验中，通过改变惩罚力度(lamda)的大小，观察模型收敛情况，如下：(learning_rate=0.1,epoch = 500)



可以发现当lambda大于0.01的时候，模型在200周期以后基本便不再下降，lambda=0.001或0.0001是一个不错的选择，后面的实验中若不指明，lambda取0.001

梯度正确性检验

检验方法：有限差分近似法

$$\boldsymbol{d}^{\top}\nabla f(\boldsymbol{x}) \approx \frac{1}{2\varepsilon}(f(\boldsymbol{x}+\varepsilon\cdot\boldsymbol{d})-f(\boldsymbol{x}-\varepsilon\cdot\boldsymbol{d}))$$

检验原则

- 尽可能的测试更多的方向
- 尽可能的测试更多的样本点

伪代码

Parameter: threshold,theta
Input: Round (how many start points the prog will check.)
Output: True or False (whether the gradient is calculated correctly.)
Progress:
for r from 1 to Round:
 get a random W (here W is combined with b)
 gw = calculate gradient()
 for i from 1 to MxN:
  gwi = gw[i]
  dwi = calculate_approximate_gradient(w,i,theta)
  if abs(dwi-gwi)>threshold:
   return False
return True

$$\text{calculate\_approximate\_gradient(w,i,}\varepsilon\text{):}$$
$$dw_i = \frac{L(w_0, w_1, \ldots, w_i + \varepsilon, w_{i+1}, \ldots) - L(w_0, w_1, \ldots, w_i - \varepsilon, w_{i+1}, \ldots)}{2 * \varepsilon}$$

Requirement 3

How to choose learning rate

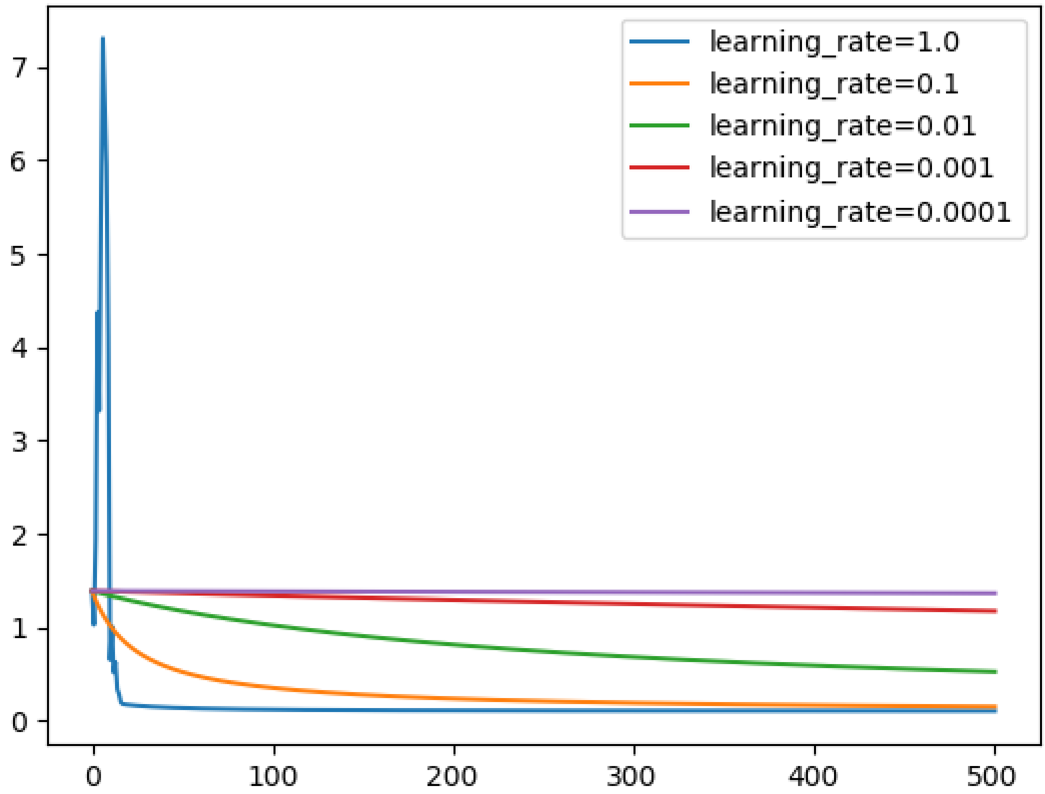
选择学习率有两类方法，一种是静态的，一种是动态的。

静态的方法有：Grid Search 以及Lipschitz constant方法。

动态的方法有：Linear Search、Cauchy、Barzilai and Borwein、Backtracking。

由于本实验数据样本不大，且比较简单，训练时间很短，所以选择Grid Search（预先设定好几组值，然后一个个试，选择最好的那一个即可）即可。

下图展示了不同learning_rate情况下模型的收敛情况（epoch=500,lamba=0.001,full_batch）：



从曲线中可以观察出，learning_rate越大，模型收敛越快，但是learning_rate>=1并不是一个很好的选择，因为它引起了严重的抖动，这种影响时不可预估的，有些情况下会导致抖动很久。learning_rate=0.1是一个不错的选择，而且当到达500的周期的时候，可以观察到它和learning_rate=1的曲线已基本重合。所以后面的实验中，本报告一律取learning_rate=0.1.

When to terminate

终止条件有三种基本方法：

- 梯度阈值法：如果本轮的所有梯度的绝对值都很小，则终止
- 错误率阈值法：如果本轮模型使得训练集的错误率小于一个阈值，则终止
- 轮次阈值法：指定模型训练的轮次，达到即终止

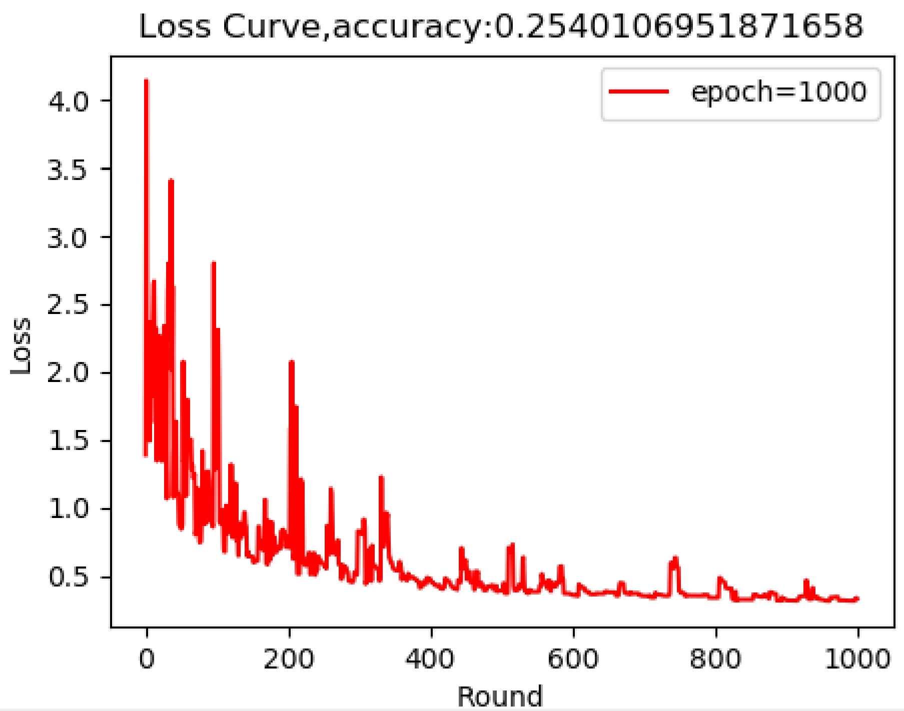
三种方法中，第二种的计算代价较大，第一种其次，第三种最小。但是第三种可能会带来冗余的计算（相当于为模型添加了一个新的超参）。为了充分观察Loss的变化情况，本实验种采用了第三种方法。

Requirement 4

观察

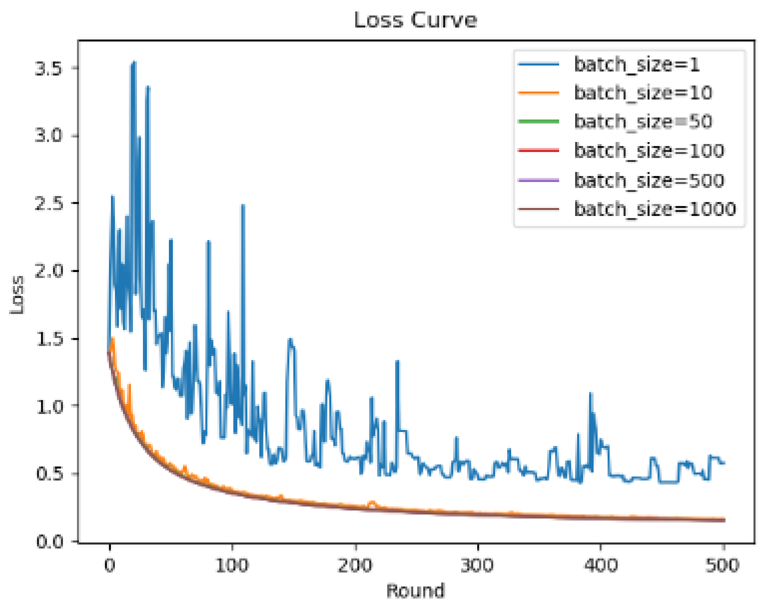
`stochastic gradient descent`

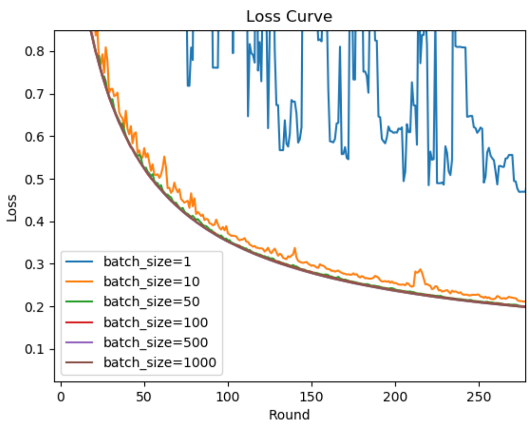
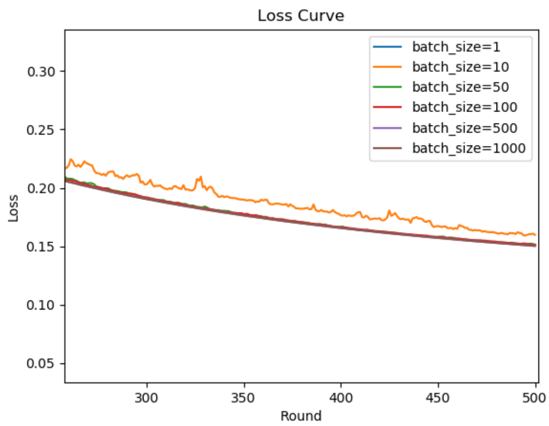
它的Loss curve有很明显的抖动，但是整体的趋势是下降的，而且抖动的幅度也在不断减小。



`batched gradient descent`

它受到batch_size的影响，通过调节batch_size（从1到1000），可以观察到：



局部放大1	局部放大2
	

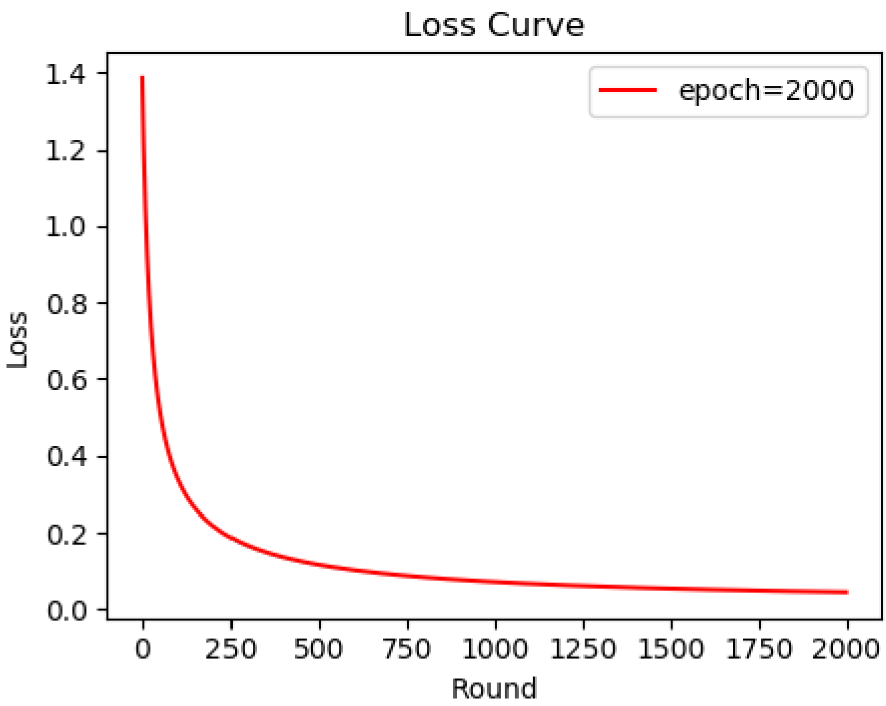
从曲线中中也可以推断出，batch_size主要影响的是抖动的程度，但不会影响其收敛性，如果给定足够大的epoch，即时是batch_size=1(即`stochastic gradient descent`)也可以收敛，并且随着周期的增长，抖动的程度也整体下降。当batch_size >=50时，可以观察到曲线其实已经没有明显的抖动（见局部放大图）。

对比

	full_batch(原始方法)	batch_size=1(SGD)	batch_size = n
优点	每轮迭代，所有的点都对梯度改变有贡献，梯度下降的方向总是最优的	只选取一个样本点计算梯度，每周期计算代价小，速度很快	若n选取的恰当，可以兼顾前两者的优点。（从下面的实验也可以看出，batch_size为1和100的运行时间差别很小）
缺点	要对所有的样本点计算，代价大，速度慢	曲线有较大的抖动	若n选取的恰当，基本可以避免前面两者的问题，不过n的值需要提前设定

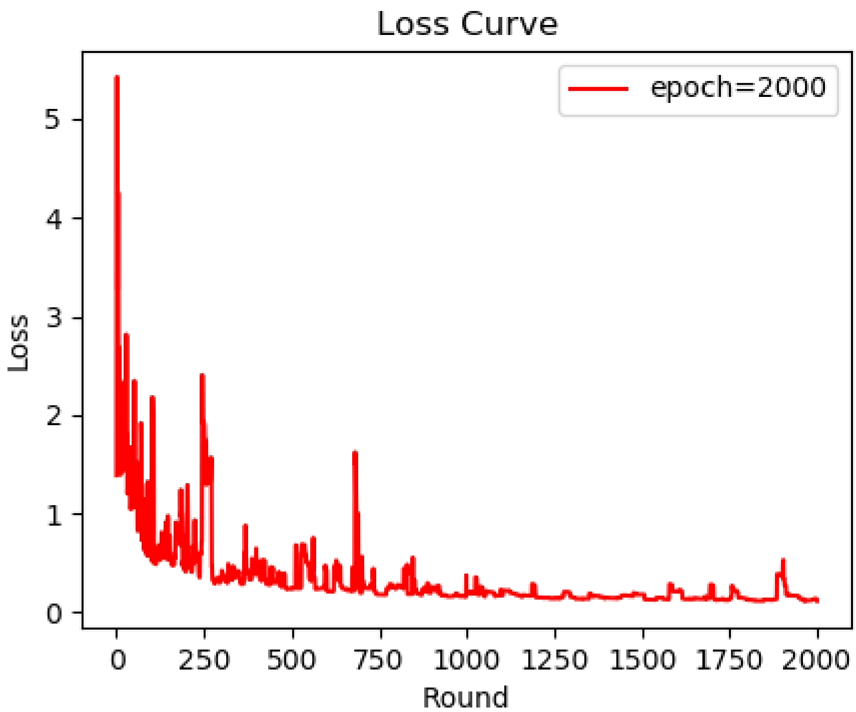
Requirement 5

full_batch



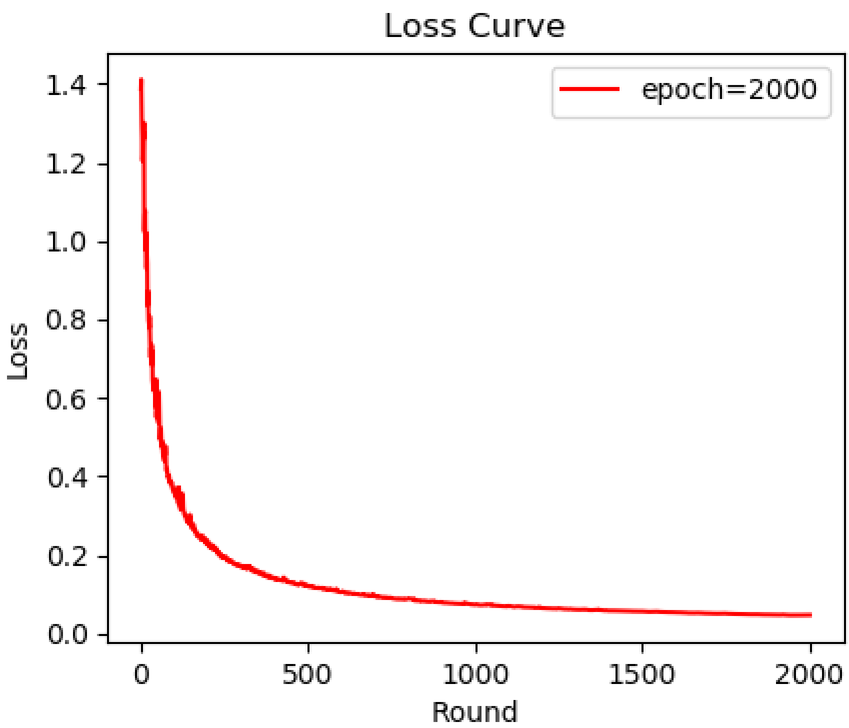
learning_rate	lambda	epoch	accuracy	time cost /s
0.1	0.0001	2000	0.9298128342245989	961.290281938

batch_size = 1



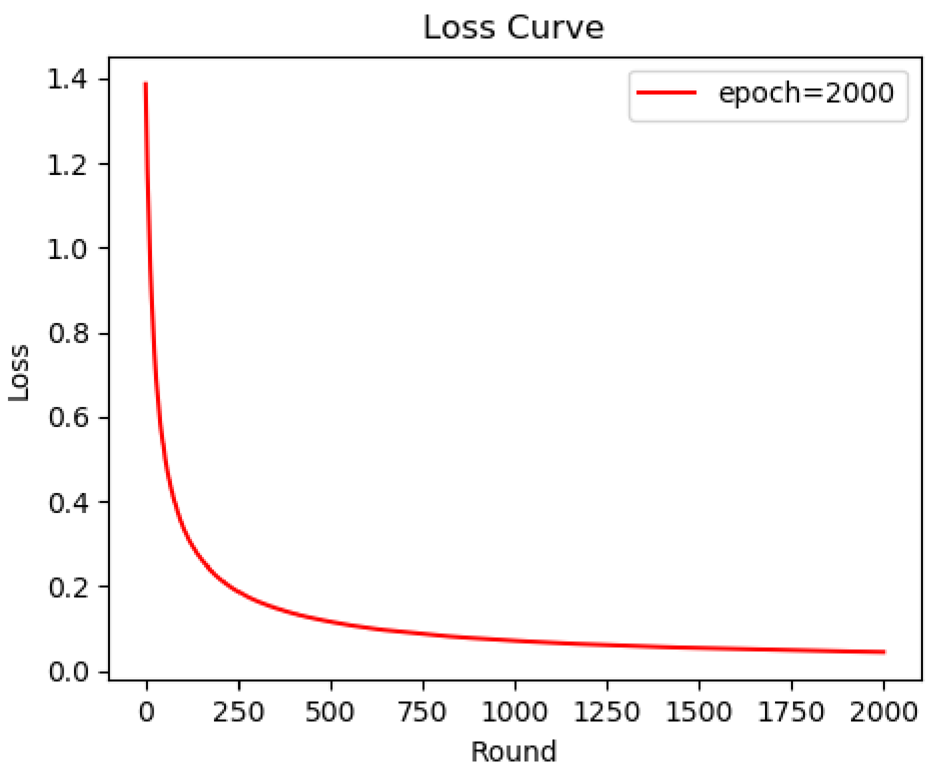
learning_rate	lambda	epoch	accuracy	time cost/s
0.1	0.0001	2000	0.8957219251336899	290.79498786

batch_size = 10



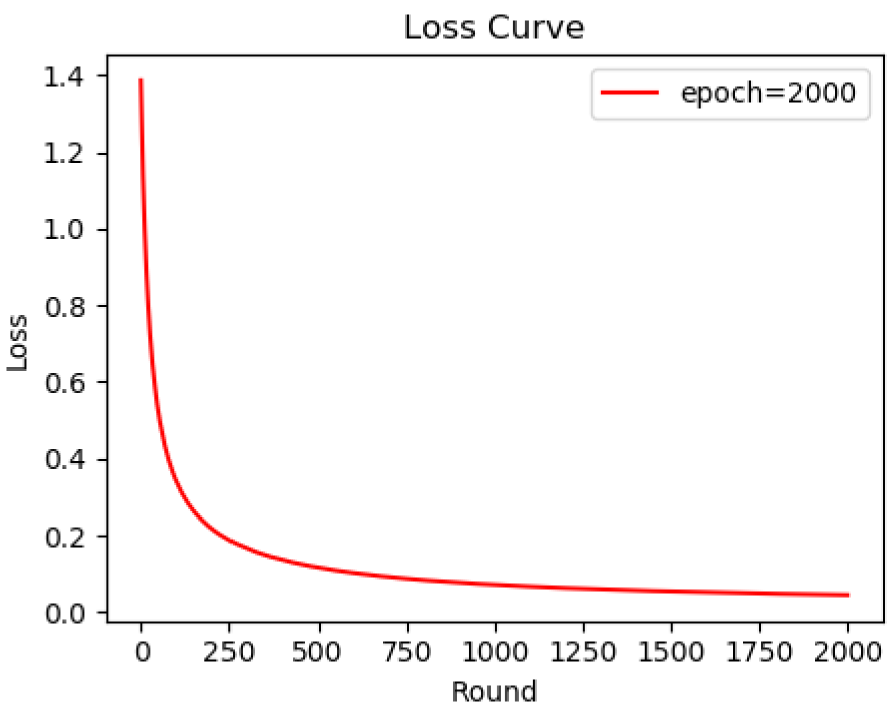
learning_rate	lambda	epoch	accuracy	time cost/s
0.1	0.0001	2000	0.9311497326203209	293.579905528

batch_size = 100



learning_rate	lambda	epoch	accuracy	time cost/s
0.1	0.0001	2000	0.929144385026738	344.588357371

batch_size = 500



learning_rate	lambda	epoch	accuracy	time cost/s
0.1	0.0001	2000	0.9311497326203209	531.7518501339999

Reference

1. https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/
2. https://towardsdatascience.com/coding-neural-network-gradient-checking-5222544ccc64
3. Pattern Recognition and Machine Learning