Assignment 2 Report

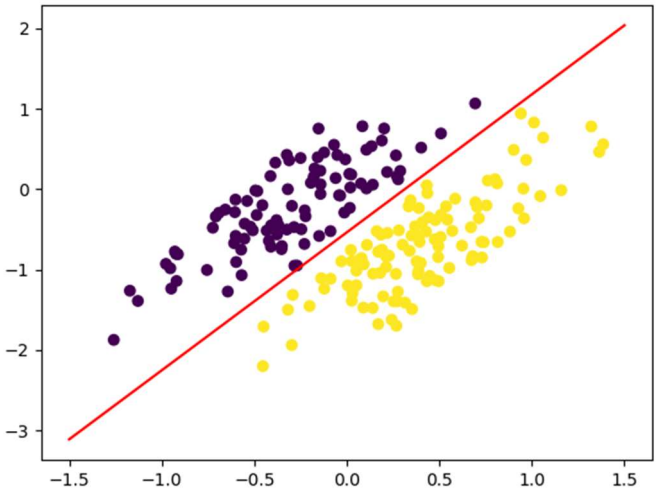
代码运行示例:

python linear.py -l (least square)
python linear.py -p (perceptron)
python logistic.py -optimizer=sgd
python logistic.py -optimizer=bgd -batch_size=128
python logistic.py -optimizer=fbgd -learning_rate=0.1
python check_gradient.py

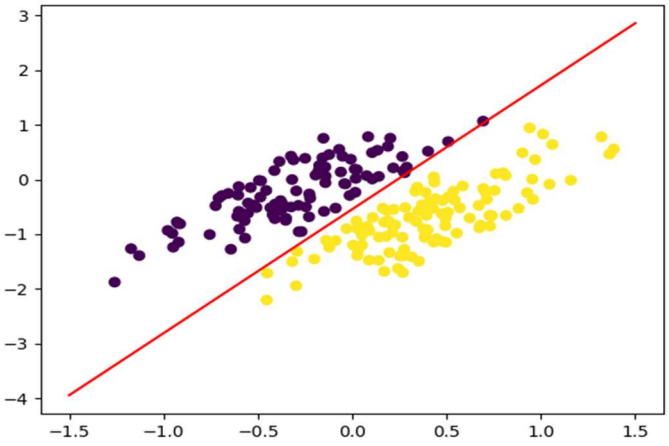
一、Part 1

代码见 linear.py。

least square model 和 perceptron algorithm 的正确率都是 100%。其结果如下:



least square



perceptron

二、Part 2

代码见 logistics.py。

1. 预处理

1.1 构建 vocabulary

首先获取 dataset_train.data，利用正则表达式忽略 string.puntuation，并将 string.whitespace 替换为空格。然后在将 data 中的字符串转换为小写的同时利用 split 空格将它们转换为单词。通过遍历统计每个单词出现的次数，再忽略掉次数不足 10 的单词，最后通过遍历构建一个字典 vocabulary，其键为单词、值为单词在原 list 中的索引。

1.2 构建 multi-hot vector

初试化一个长为 vocabulary 大小且值全为 0 的 array，将 data 转换为由单词组成的 list（见 1），遍历这个 list，若单词出现在 vocabulary 中，则将该 array 相应的位置置为 1。

1.2 构建 one-hot vector

初始化一个长为 target_nums 且值全为 0 的 array，然后将第 target 位置为 1 即可。

代码细节见 logistics.py 中的 Split_data、Generate_vocabulary、Get_X_Y 函数。

2. 微分

2.1 梯度的计算

$$z_{nm} = \sum_{t=1}^{T} X_{nt} * W_{tm} + b_m \tag{1}$$

$$\hat{y}_{nm} = \frac{e^{z_{nm}}}{\sum_{m=1}^{M} e^{z_{nm}}} \tag{2}$$

$$\mathrm{L} = -\frac{1}{\mathrm{N}} \sum_{n=1}^{N} (\sum_{m=1}^{M} y_{nm} ln\hat{y}_{nm}) + \lambda \|W\|^2 \tag{3}$$

$$\frac{\boldsymbol{\partial \hat{y}_{nm}}}{\boldsymbol{\partial z_{nj}}} = \boldsymbol{\hat{y}_{nm}(I_{mj} - \hat{y}_{nj})} \tag{4}$$

$$\frac{\partial \mathrm{L}}{\partial W_{ij}} = \frac{\partial \{-\frac{1}{\mathrm{N}} \sum_{n=1}^{N} (\sum_{m=1}^{k} y_{nm} ln\hat{y}_{nm})\}}{\partial W_{ij}} + 2\lambda W_{ij}$$

$$= -\frac{1}{\mathrm{N}} \sum_{n=1}^{N} (\sum_{m=1}^{k} \frac{y_{nm}}{\hat{y}_{nm}} \frac{\partial \hat{y}_{nm}}{\partial z_{nj}} \frac{\partial z_{nj}}{\partial W_{ij}}) + 2\lambda W_{ij}$$

$$= -\frac{1}{\mathrm{N}} \sum_{n=1}^{N} \{\sum_{m=1}^{k} \frac{y_{nm}}{\hat{y}_{nm}} [\hat{y}_{nm}(I_{mj} - \hat{y}_{nj})] X_{ni}\} + 2\lambda W_{ij}$$

$$= -\frac{1}{\mathrm{N}}\sum_{n=1}^{N} X_{ni}\{\sum_{m=1}^{k} y_{nm}(I_{mj} - \hat{y}_{nj})\} + 2\lambda W_{ij}$$

$$= \frac{1}{\mathrm{N}}\sum_{n=1}^{N} X_{ni}(\hat{y}_{nj} - y_{nj}) + 2\lambda W_{ij} \tag{5}$$

同理可得: $$\frac{\partial \mathrm{L}}{\partial b_j} = \frac{1}{\mathrm{N}}\sum_{n=1}^{N}(\hat{y}_{nj} - y_{nj}) \tag{6}$$

2.2 计算的向量化

通过观察上式（5、6）可以知道:

$$\frac{\partial \mathrm{L}}{\partial W} = \frac{1}{\mathrm{N}} X^T\left(\hat{Y} - Y\right) - 2\lambda W \tag{7}$$

$$\frac{\partial \mathrm{L}}{\partial b} = \frac{1}{\mathrm{N}}\left(\hat{Y} - Y\right) \tag{8}$$

2.3 bias 是否需要正则化

L2 正则化的目的是限制模型参数值的大小以限制模型的复杂度（模型空间），来避免过拟合。而 bias 的作用仅仅是在预测结果中加上一个偏置，不会影响模型的复杂性，因此不需要正则化。

2.4 检验梯度的正确性

由导数的定义:

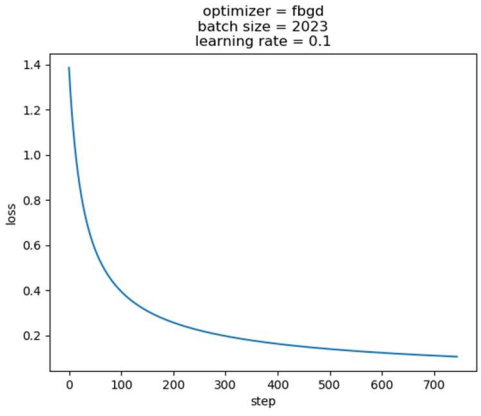
$$\frac{\partial \mathrm{y(x)}}{\partial x} = \lim_{\delta \to 0}\frac{y(x+\delta) - y(x)}{\delta}$$

根据上式进行数值计算，检查误差大小是否合理即可。（代码见 check_check_gradient.py，样例误差在 1e-8 左右）

3. 训练

3.1 loss 曲线

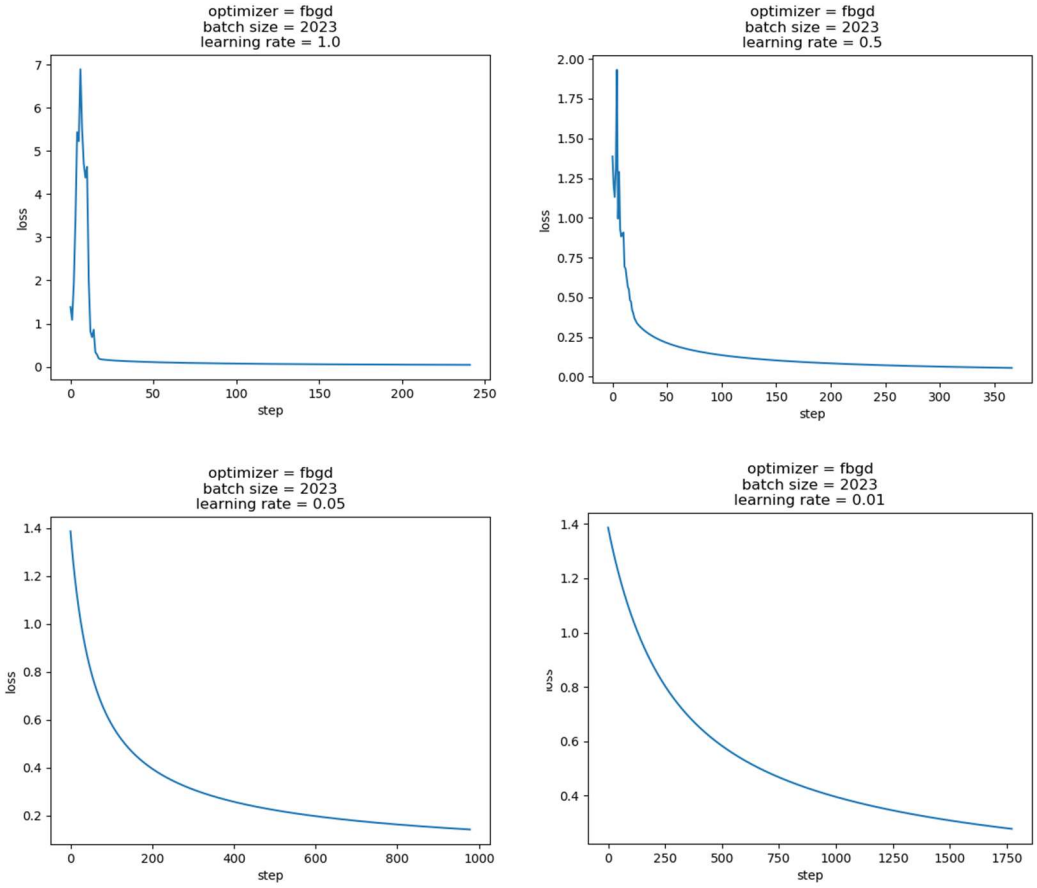
当 learning rate=0.1，正则化系数λ=1e-4 时，loos 曲线如下:



可以看到，loss 一开始下降得较快，并逐渐趋于收敛。

3.2 learning rate 的选择

固定正则化系数λ=1e-4，改变 learning rate 获得的 loss 曲线如下：



通过以上 loss 曲线可以看到，learning rate 过大会导致曲线出现一定程度的波动（甚至可能无法收敛，当我把 learning rate 设为 1000 时，出现了这种情况），而 learning rate 过小则导致收敛的速度过慢。**因此我们需要寻求一个适中的 learning rate，它得到的 loss 既稳定，又能较快收敛**。

由于 learning rate 是一个超参数，因此可能只能通过尝试找到合适的值。一个可能的方法是设定一系列 learning rate（如：1，0.1，0.01，0.001 等），然后在较小的数据集上进行训练并绘制 loss 曲线，再通过 loss 曲线的表现中选择一个合适的值。

3.3 何时终止训练

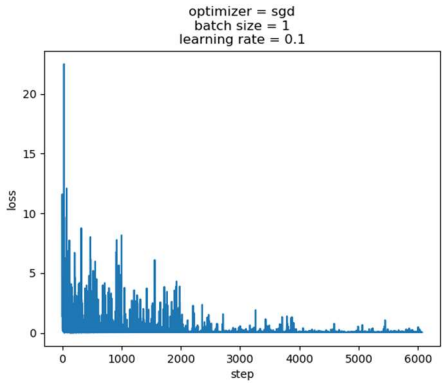
我认为有三种限制训练次数（终止训练）的方式：

- 设定训练周期数(epoch)的阈值，当训练周期数超过此阈值时停止训练
- 设定 loss 的阈值，当 loss 小于此阈值时停止训练
- 考虑相邻两次训练得到的 loss 的差的绝对值，当其小于给定阈值时停止训练（一个可能的改进是考虑相邻多个这样的值的平均值，避免偶然性）

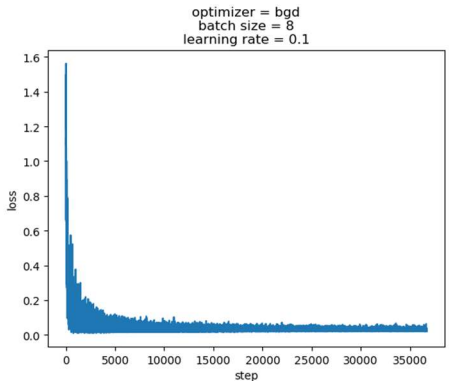
本次实验我结合了这三钟方式，但从结果来看，起作用的只有第三种（这跟阈值的设定有关）。

4. 三种优化器（fbgd、bgd、sgd）

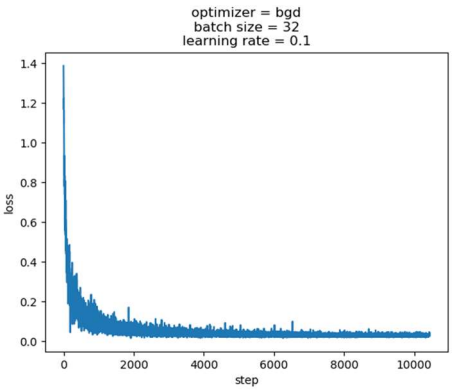
4.1 bgd 与 sgd 的 loss 曲线



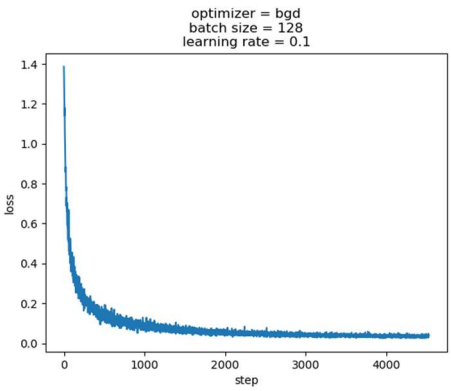
epoch≈6000/2000=3



epoch≈8*35000/2000=140



epoch≈32*10000/2000=160



epoch≈128*4500/2000=288

4.2 观察结果

从 loss 曲线可以看到，sgd 的 loss 曲线波动较大，但收敛速度比较快。而对于 bgd 来说，随着 batch size 的增大，loss 曲线的波动性会逐步降低，收敛速度也会变慢。而 fbgd 的 loss 几乎没有波动，收敛速度也最慢。**因此观察到 bgd 是 sgd 与 fbgd 的一个折中**。

4.3 三种优化器的优缺点

4.3.1 fbgd

优点：

- 损失函数稳定

缺点：

- 收敛速度慢
- 占用大量内存

4.3.2 sgd

优点：

- 收敛速度快

- 占用内存少

缺点：

- 损失函数不稳定

4.3.3 bgd

优点：

- 平衡了 sgd 与 fbgd 的收敛速度
- 占用内存较少

缺点：

- 需要额外调整 batch size 这一超参数

5. 测试结果

在测试集上的准确率如下表：

优化器	fbgd	bgd			sgd
batch size	2023	128	32	8	1
准确率	92.98%	92.58%	92.25%	92.31%	92.38%

需要思考的是，选择什么样的参数（W,b）对测试集进行预测呢？

我采取的策略是从初始的训练集中划分一部分为验证集（本次实验使用了1/10），剩下的部分成为真正的训练集。在训练的过程中，每隔一段时间使用训练得到的参数（W,b）计算验证集上的准确率，保存使得准确率最大的参数。这样做能够减小使用过拟合参数的可能性。（也可以通过在验证集上的表现调整正则化系数等超参数）