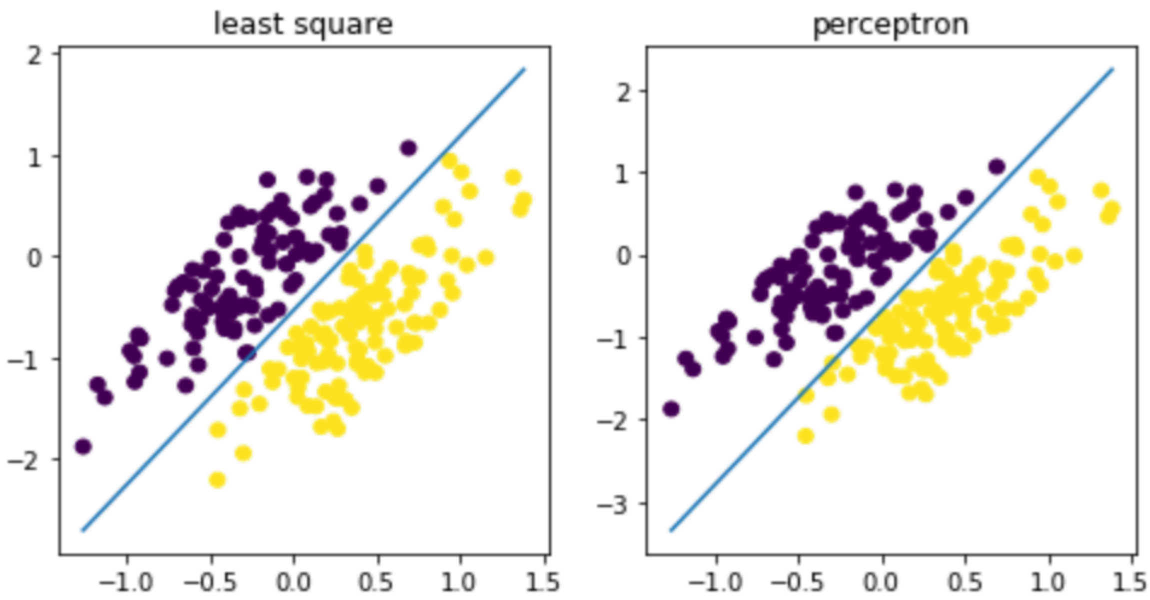
lab2报告

Part1

最小平方算法和感知器算法都取得了100%的分类准确率，图如下所示：



最小平方算法的主要思路为最小化平方误差：

$$\min \frac{1}{2}\sum_{n=1}^{N}\{w^T x_n - y_n\}^2$$

对于二分类问题，$y_n \in \{-1, 1\}$，令上式对w求导等于0，得到：

$$w = (X^T X)^{-1} X^T y$$

这里X按照行方向存储样本，且第一列都置为1，分界面为$w^T x = 0$.

对于每个样本x，感知器的输出为：

$$y(x) = f(w^T x)$$

其中函数f当输入非负时输出1，否则输出-1.

感知器算法定义另外一个误差函数，即感知器准则：

$$E(w) = -\sum_{n\in\mathcal{M}} w^T x_n y_n$$

其中$\mathcal{M}$为分类错误的样本集合，对于每个分类错误的样本更新权重：

$$w^{(t+1)} = w^{(t)} + \eta x_n y_n$$

实现时η可设为1，循环遍历所有样本，直到都被分类正确为止。根据感知器收敛定理，只要样本线性可分，则算法一定可在有限步内找到解。分界面和最小平方算法相同。

Part2

Q1

数据预处理

首先对文本进行处理，删除所有的string.punctuation，并将string.whitespace替换为空格后split，所有大写字母转换为小写，得到word_list，并删去所有出现次数小于10次的单词。用一个字典记录每个单词在word_list中的位置，对于数据中的每一个单词，查找字典得到它在word_list中的位置pos，将pos位置为1。将target按照one_hot编码为一个四维向量，例如target为1，则编码为[0,1,0,0].

Q2

偏导计算

令第k个样本的loss为$\mathcal{L}_k$ ，则$\mathcal{L} = \frac{1}{N}\sum_{k=1}^{n}\mathcal{L}_k + \lambda\|W\|_2$，考虑$\mathcal{L}_k$对$W$和$b$求偏导，记真实的类别为$l$，共有$C$类，单词总数为$m$, 令$p = W^Tx + b$，对于$W$则：

$$
\begin{aligned}
\mathcal{L}_k &= -log(\hat{y}_l) \\
&= -log(\frac{e^{p_l}}{\sum_{u=1}^{C} e^{p_u}}) \\
&= -p_l + log(\sum_{u=1}^{C} e^{p_u}) \\
-\frac{\partial p_l}{\partial W_{ij}} &= -\frac{\partial \sum_{i=1}^{m} W_{il}x_i}{\partial W_{ij}} \\
&= -1\{j = l\}x_i \\
&= -y_j x_i \\
\frac{\partial log(\sum_{u=1}^{C} e^{p_u})}{\partial W_{ij}} &= \frac{1}{\sum_{u=1}^{C} e^{p_u}} \frac{\partial e^{p_j}}{\partial W_{ij}} \\
&= \frac{\partial e^{p_j}}{\sum_{u=1}^{C} e^{p_u}} x_i \\
&= \hat{y}_j x_i \\
\frac{\partial \mathcal{L}_k}{\partial W_{ij}} &= (\hat{y}_j - y_j)x_i \\
\frac{\partial \mathcal{L}}{\partial W_{ij}} &= \frac{1}{N}\sum_{k=1}^{N}(\hat{y}_{jk} - y_{jk})x_{ik} + 2\lambda W_{ij}
\end{aligned}
$$

同理，对于b则有：

$$
\frac{\partial \mathcal{L}}{\partial b_i} = \frac{1}{N}\sum_{k=1}^{N}(\hat{y}_{ik} - y_{ik})
$$

具体实现在train函数中，利用numpy矩阵操作很容易得到代码，**注意到这里的x, y的第k列对应第k个样本。**

是否要正则化b

不应该对b进行正则化。正则化是为了减小模型的复杂性，防止模型过拟合的一种手段。从贝叶斯的角度出发，权重项W应该满足均值为0的高斯分布，所以对其中较大的值进行惩罚，减小复杂性。而偏置项b的大小并不影响整体模型的复杂性，如果对b进行正则化，不仅不能防止过拟合，还有可能降低分类正确率。

如何检查梯度计算是否正确

根据导数的定义：

$$\frac{\partial \mathcal{L}(\theta)}{\partial \theta}=\lim_{h\to 0}\frac{\mathcal{L}(\theta+h)-\mathcal{L}(\theta)}{h}$$

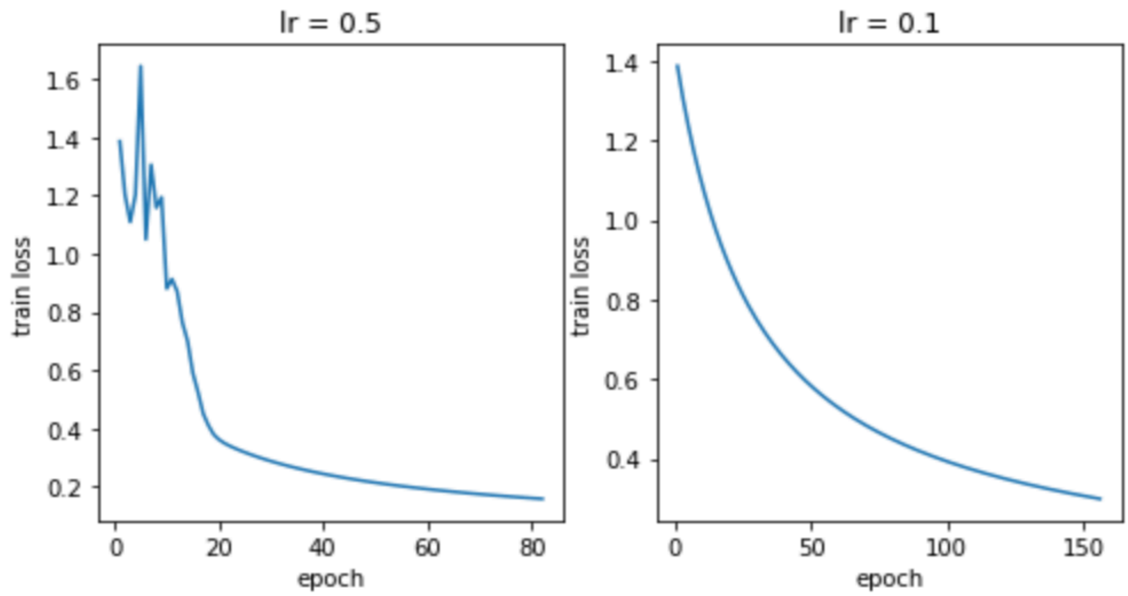
故当h足够小时，我们可以认为：

$$\frac{\partial \mathcal{L}(\theta)}{\partial \theta}\approx\frac{\mathcal{L}(\theta+h)-\mathcal{L}(\theta)}{h}$$

于是我们可以随机找到一个W_{ij}或b_i，将其加上一个较小的值eps，比较loss前后的差值除以eps是否等于所求的梯度（绝对值误差小于一个阈值），重复随机多次即可判定梯度计算是否正确。

Q3

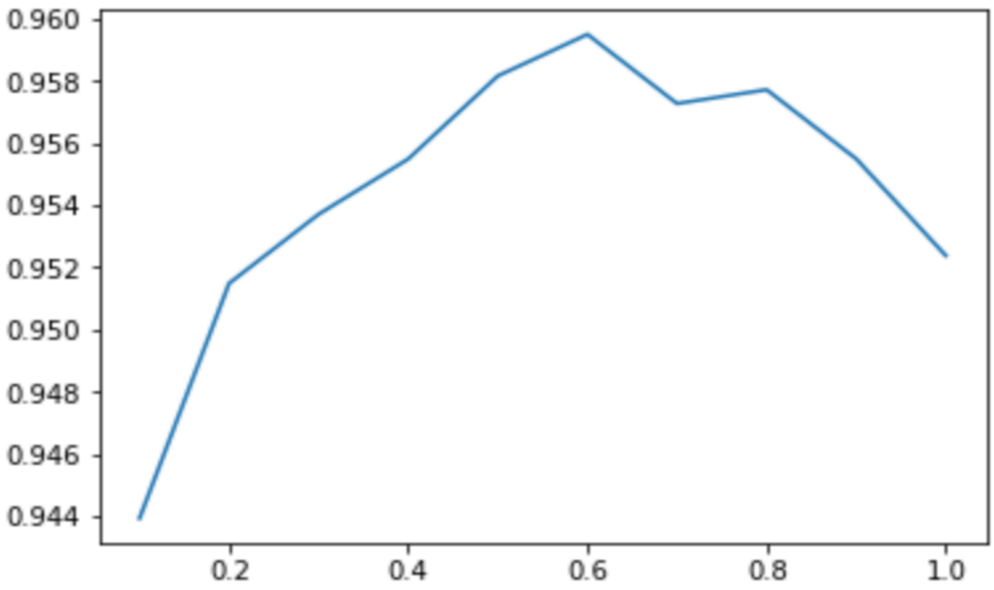
loss曲线



左图中由于学习率设置为0.5，值较大，在前几个epoch中loss的值上下起伏，后面的epoch中便平缓下降。右图中的学习率为0.1，学习率变小后loss曲线平缓下降。

如何确定学习率

对于学习率这种超参的选取，我们可以考虑在训练集上进行5折交叉验证，取验证时平均准确率最高的值。首先通过尝试不同数量级的学习率确定大致的范围为0.1～1.0，再取点画图：



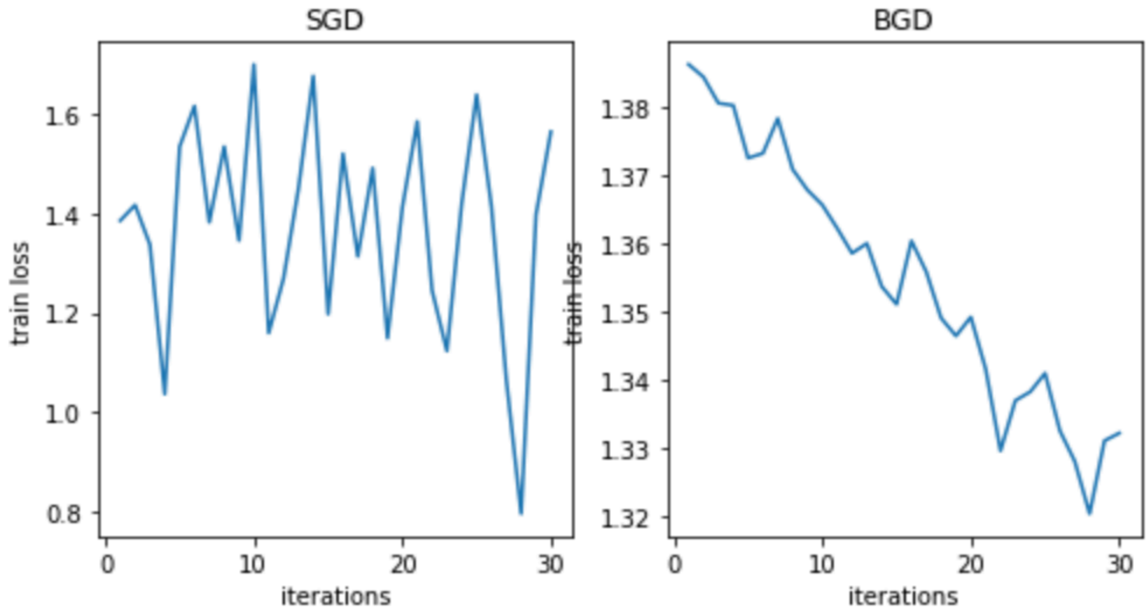
最终选取0.5为学习率的值，正则化参数λ也可以同理得到。交叉验证的代码在cross_val_lr函数中。

何时终止训练

如果训练的epoch太少，模型会欠拟合，在训练集和测试集上表现都很差，训练的epoch太多则容易造成过拟合，模型虽然在训练集上取得了很高的准确率，但泛化性很差，在测试集上表现不好。为了更好地衡量模型的泛化能力，我把训练数据根据9:1的比例分为训练集和验证集，验证集仅用来衡量模型的泛化性，不参与训练。当验证集的loss不再明显下降后终止训练（当前loss减去前10个epoch的平均值小于一个阈值）。**注意到这里的终止策略不需要人为设定超参，大大减少了工作量。**

Q4

对另外两种梯度下降的观察



首先分别观察SGD和BGD的loss曲线(此处横坐标为迭代次数)，由于每次只对一个样本更新权重，而单个样本可能具有特殊性，不足以反映整体的性质，所以SGD的loss曲线上下起伏。而通过每次取一个batch的样本更新权重(batch_size=64)，较好地体现了样本整体的性质，BGD的loss曲线较为平缓。

另外打印epoch的运行时间可以发现，SGD每个epoch运行时长0.8s, 共运行21个epoch，BGD每个epoch运行时长0.05s，共运行50个epoch。虽然SGD算法收敛所需的epoch较少，但每个epoch内的迭代次数较多，故总时长和总迭代次数也多于BGD。

三种梯度下降算法的利弊

SGD的优点：占用空间小，单次迭代更新速度快。

SGD的缺点：每次只对一个样本更新，收敛所需的迭代次数多，总用时长，并且难以训练，容易受到噪声样本的影响，鲁棒性差。

BGD的优点：占用空间较小，运行速度快。

BGD的缺点：需要人为设定超参batch_size。

FBGD的优点：迭代次数最少，loss曲线平稳下降，不容易受到噪声样本的影响，鲁棒性好。

FBGD的缺点：占用空间大，当数据集较大时需要大量内存（显存）资源。运行速度也慢于BGD。

Q5

算法	学习率	正则化参数	运行epoch数量	测试集准确率
SGD	0.006	0.001	21	92.6%
BGD	0.06	0.001	50	92.8%
FBGD	0.5	0.001	82	92.8%

自由探索

传统的梯度下降算法收敛速度较慢，且容易受到局部最小值的影响，因此考虑用带动量的梯度下降算法进行改进。基本思路是更新此次参数时不仅要考虑当前梯度，还要考虑之前梯度加权后对现在的影响，原理如下所示：

$$
\begin{aligned}
v_w &= \beta v_w + (1-\beta)dW \\
v_b &= \beta v_b + (1-\beta)db \\
W &= W - lr * v_w \\
b &= b - lr * v_b
\end{aligned}
$$

参数β即为我们的动量，注意到当$\beta = 0$时即变为普通梯度下降算法。相关代码已实现在train函数中。

算法	学习率	正则化参数	运行epoch数量	测试集准确率
FBGD	0.4	0.001	93	92.8%
FBGD+动量(0.8)	0.4	0.001	90	92.9%

在其它超参相同的情况下，加入动量=0.8后，运行的epoch数量相应减少，测试集准确率也有所提升。