作业二报告

Part 1

数据处理：

给定的数据是二维点集，由 Dataset.y 标注 True/False 表示哪一类。首先 y 转化到{-1,+1}变成标准的二分类问题。

最小二乘法（least square model）

最小二乘法是希望使用线性模型，最小化平方误差：

$$min \sum_i (\mathrm{W}x_i + \mathrm{b} - \mathrm{yi})^2$$

将偏置项 b 与 W 合并，将上式写成矩阵形式即为：

$$\min \|WX - Y\|_2$$

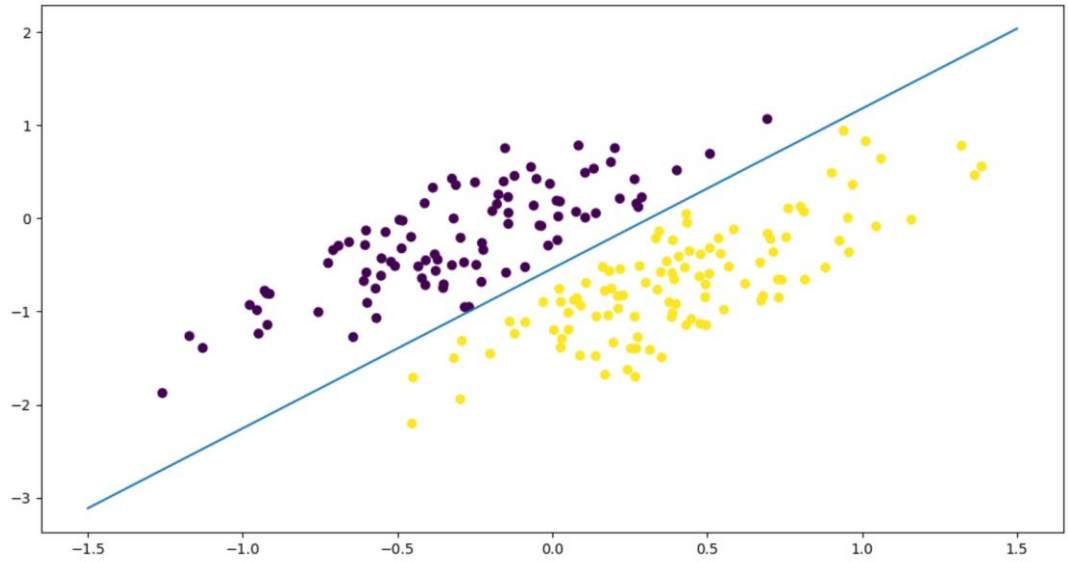
对于该问题的二维点集，X 矩阵应形如：

$$\begin{matrix} x_{1,1} & x_{1,2} & 1 \\ \vdots & \vdots & \vdots \\ x_{n,1} & x_{n,2} & 1 \end{matrix}$$

最优解为：

$$\widehat{\mathrm{W}} = (\mathrm{X}^T X)^{-1} X^T Y$$

求得对应 W 矩阵后，显然决策面为$\mathrm{WX} = 0$，在给定数据集上可绘出如下分界线：

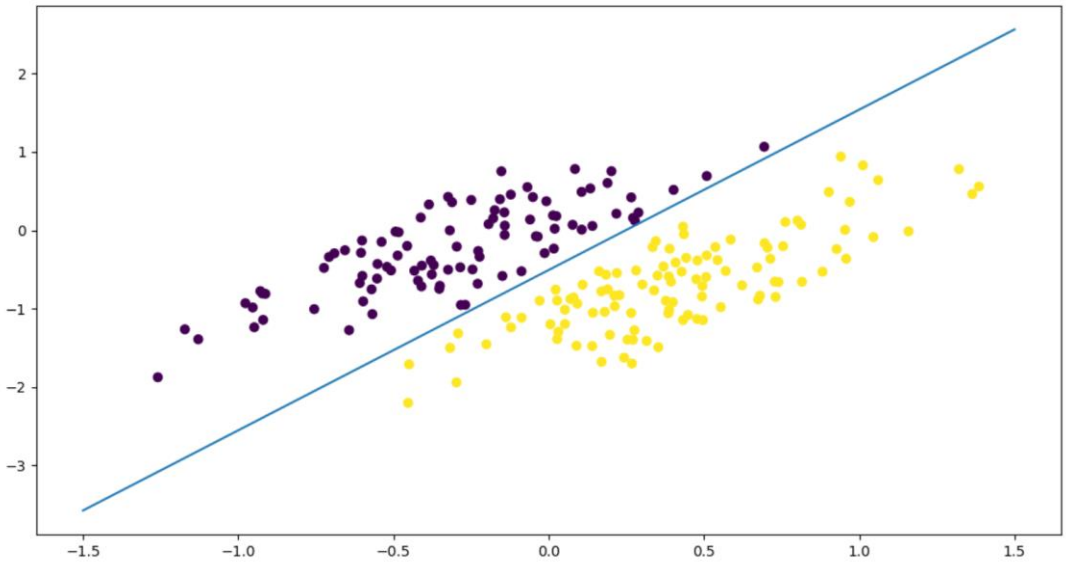


感知机算法（perceptron algorithm）

感知机算法希望学习得到 W,b，使得 f(x)=sign(Wx+b)可对点集进行分类。

感知机的学习算法通过迭代完成：当一个实例被误分类时，调整 w 和 b 的值，使分离超平面向该误分类点的一测移动，以减少该误分类点与超平面间的距离，直至超平面越过该点使其正确分类。

感知机学习率 η 设置会影响达到正确分类所需的迭代次数，当 η 较小时会需要更多次迭代。实际中，我采用 η=0.1，得到 W,b，最终分类决策面为 Wx+b=0：



当训练数据集线性可分时，感知机学习算法是收敛的，而且感知机学习算法存在无穷多个解，其解由于不同的初值或不同的迭代顺序而可能有所不同。

Part 2

Requirement 1:

数据处理即对照作业中的指引，分为如下几步，具体实现细节见 part2.py 中相应部分：

1. 对于原始数据的特殊字符进行处理、大小写转换的工作（deal 函数），主要利用字符串的 replace。
2. 对于训练数据利用字典统计单词出现次数，并根据 mini_count=10 筛选建立此表（Vocab 类）
3. 根据词表，将数据中的单词转为 index
4. 根据 index 建立 multi-hot 表示（transform 函数）

Requirement 2:

损失函数偏导数的推导：

在本问题下对于单个样本特征为 m，输出类别为 4，令：

$$\boldsymbol{h} = \mathrm{W}\mathbf{x} + \mathbf{b}$$

$$\widehat{y_i} = \frac{e^{h_i}}{\sum_{k=1}^{4} e^{h_k}}$$

其中，**h,b** 是 4 维列向量，W 是 4×m 的矩阵，**x** 是 m 维列向量

根据链式法则易得：

$$\frac{\partial \mathrm{L}}{\partial W_{i,j}} = \frac{\partial \mathrm{L}}{\partial h_i} \cdot \frac{\partial h_i}{\partial W_{i,j}} = \frac{\partial \mathrm{L}}{\partial h_i} \cdot x_j$$

$$\frac{\partial \mathrm{L}}{\partial h_i} = -\sum_{k=1}^{4} \frac{\partial \mathrm{L}}{\partial \widehat{y_k}} \cdot \frac{\partial \widehat{y_k}}{\partial h_i} = -\sum_{k=1}^{4} \frac{y_k}{\widehat{y_k}} \cdot \frac{\partial \widehat{y_k}}{\partial h_i}$$

$$\frac{\partial \widehat{y_k}}{\partial h_i} = \frac{e^{h_k} \cdot \frac{\partial h_k}{\partial h_i} \cdot \left(\sum_{j=1}^{4} e^{h_j}\right) - e^{h_k} e^{h_i}}{(\sum_{j=1}^{4} e^{h_j})^2} = \widehat{y_k} \cdot \frac{\frac{\partial h_k}{\partial h_i} \cdot \left(\sum_{j=1}^{4} e^{h_j}\right) - e^{h_i}}{\sum_{j=1}^{4} e^{h_j}}$$

$$= \widehat{y_k} \cdot \frac{\partial h_k}{\partial h_i} - \widehat{y_k} \cdot \widehat{y_i}$$

$$\therefore \frac{\partial \mathrm{L}}{\partial h_i} = -\sum_{k=1}^{4} y_k \left(\frac{\partial h_k}{\partial h_i} - \widehat{y_i}\right) = \widehat{y_i} - y_i$$

$$\therefore \frac{\partial \mathrm{L}}{\partial W_{i,j}} = (\widehat{y_i} - y_i) \cdot x_j$$

写成矩阵形式即为：

$$\frac{\partial \mathrm{L}}{\partial W} = (\hat{\mathbf{y}} - \mathbf{y})\boldsymbol{x}^T$$

同理可得：

$$\frac{\partial \mathrm{L}}{\partial \mathbf{b}} = \hat{\mathbf{y}} - \boldsymbol{y}$$

对于同时训练 N 个样本来说：

$$\frac{\partial \mathrm{L}}{\partial W} = \frac{1}{N}(\widehat{\mathrm{Y}} - \mathrm{Y})X^T$$

$$\frac{\partial \mathrm{L}}{\partial \mathbf{b}} = \frac{1}{N}\sum_{i=1}^{N} \widehat{\mathrm{Y}}_i - Y_i$$

L2 正则化不需要包括偏置项，这是因为正则化是为了降低模型的复杂度以防止过拟合，而偏置 b 大小并不影响模型的复杂度。相反，有时很需要一个较大的偏置达到收敛。

加入 L2 正则项后容易得梯度下降公式为：

$$\frac{\partial \mathrm{L}}{\partial W} = \frac{1}{N}(\widehat{\mathrm{Y}} - \mathrm{Y})X^T + 2\lambda W$$

使用 Numpy 只需要几行即可实现：

```
1. delta=np.dot(a-
   Y,X.transpose())/len(batch)+2*lamda*self.w # a is softmax output
2. self.w-=eta*delta
```

```
3.  self.b-=eta*np.mean(a-Y,axis=1,keepdims=True)
```

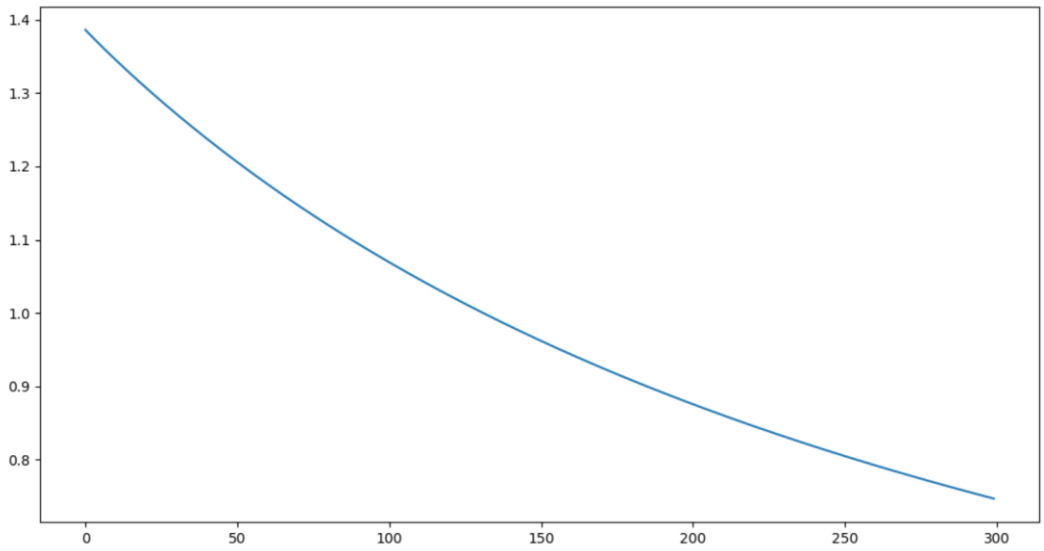
至于如何检测正确性，可以调试输出 loss 函数在相邻近的两个输入下的变化值与手算梯度值对比看一下是否近似相等即可。

Requirement 3：

在训练的时候，为了更好的观察训练，需要从原始训练集中随机选取 10%数据作为验证集。通过观察 loss 和在验证集上准确率的变化，大致得出超参的选择方向。

对于学习率的选择，一开始可以根据经验定一个初始学习率（先小一点）；如果在训练过程中，loss 无法收敛则可以适当调小学习率；如果 loss 下降过慢，则可以增大学习率。一般而言，学习率较小需要更多次迭代收敛，但下降比较稳定；较大时 loss 下降更快但有可能导致无法收敛。

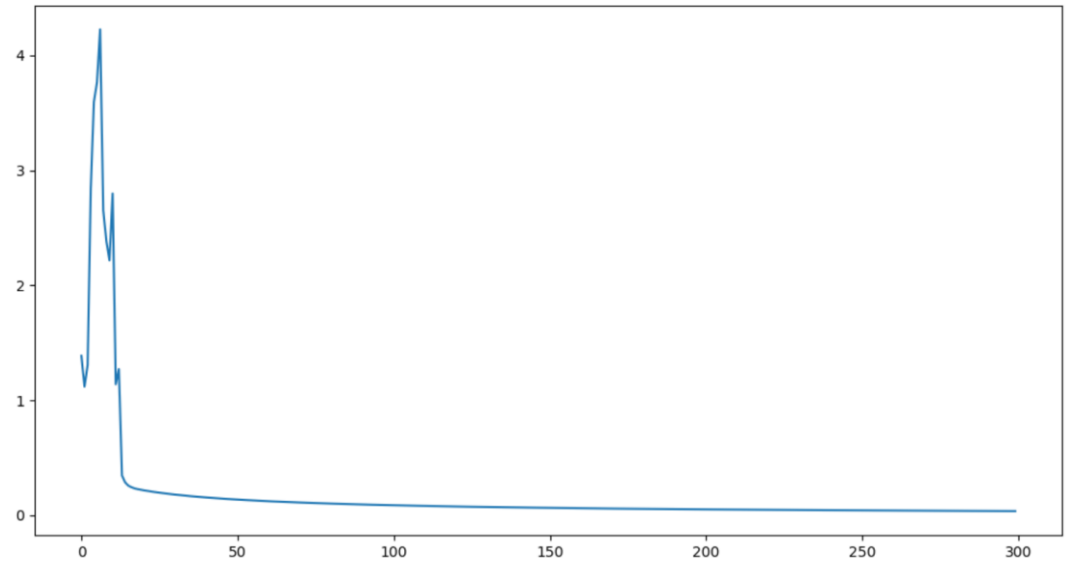
例如用 full batch gradient descent 训练，一开始设定学习率为 0.01，loss 曲线图如下：



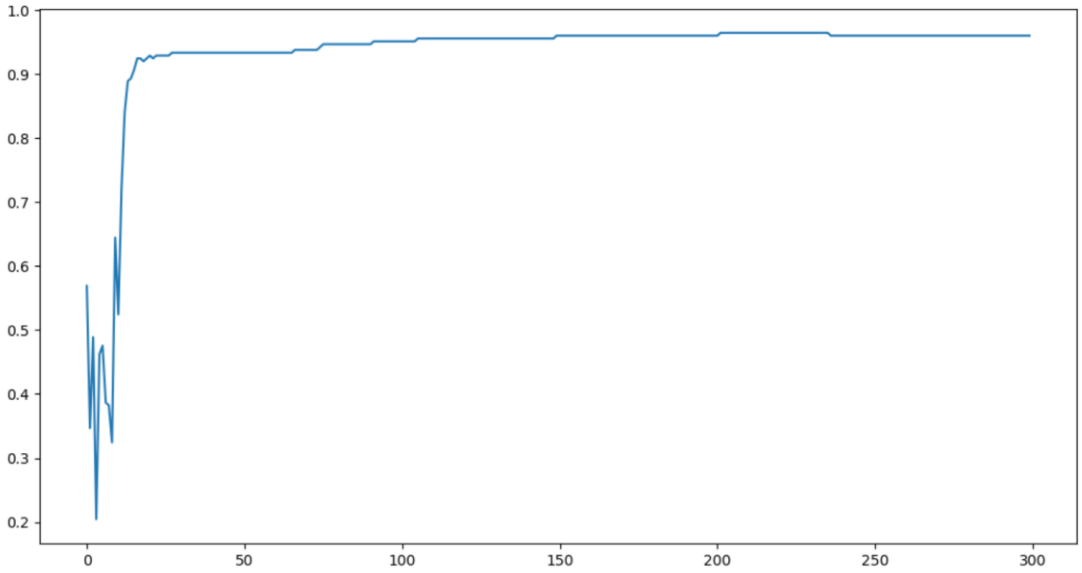
可以看出，loss 下降的比较缓慢，且 300 个 epoch 后还有继续下降的趋势，这时候就应该增大学习率，重新进行训练。

关于何时终止训练，在训练时，如果发现经过一定的迭代更新后 loss 不再有明显的下降同时验证集上的准确率不再明显的上升，即可终止训练。由于在这个问题上，训练很快，因此可以直接观察。当数据较大时，可以对 loss 和验证集准确率设置阈值以结束训练。

例如在划分 10%数据为验证集的情况下，以学习率 0.8 训练了 300 个 epoch loss 曲线图：

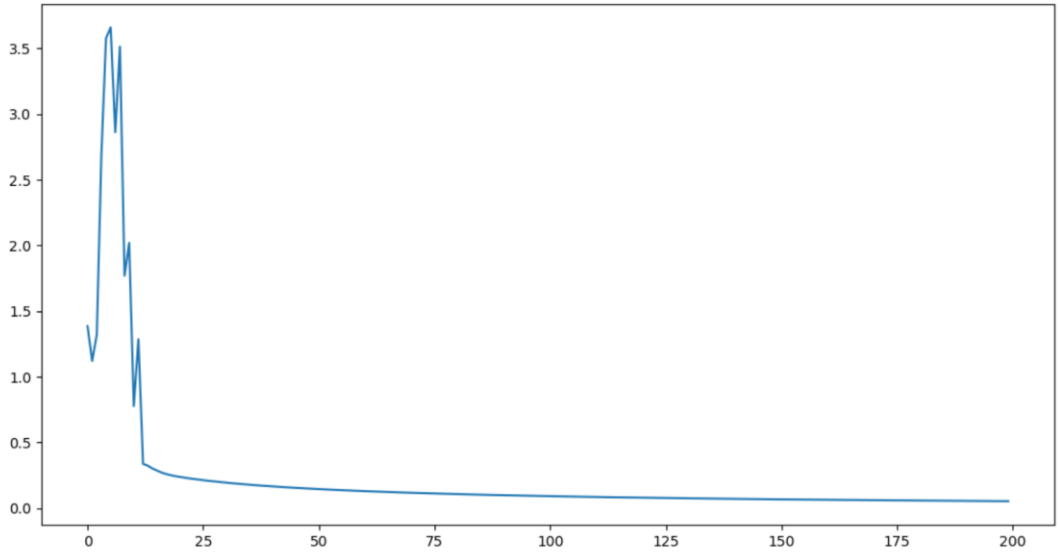


验证集 accuracy 曲线如下：



从训练结果可以看出，200 个 epoch 后 loss 虽然一直在下降，但是在验证集上的准确率提升已经微乎其微甚至已经开始上下波动，说明出现了过拟合的情况，应终止训练。

所以最终选定以学习率 0.8 训练了 200 个 epoch，其在全部训练数据上的 loss 曲线图如下：



Requirement 4：

全批量梯度下降（full batch gradient descent）每一次迭代时使用所有样本来进行梯度的更新，其优缺点为：

优点：

1. 一次迭代根据所有样本进行计算，利用矩阵运算实现并行，在更新一次全体样本速度最快。
2. 由全数据集确定的方向能够更好地代表样本总体，从而更准确地朝向极值所在的方向下降。

缺点：

1. 当训练集较大空间可能无法承载
2. 一次迭代更新的速度较慢

随机梯度下降（stochastic gradient descent）每次迭代使用一个样本来对参数进行更新。

优点：

1. 一次更新迭代参数的速度大大加快

缺点：

1. 准确度下降，由于只使用单个样本更新，梯度下降的方向会很不问会需要更多次迭代才能找到准确的下降方向，收敛难度大大增加。
2. 由于单个样本并不能代表全体样本的趋势，可能会收敛到局部最优。
3. 基于前两点，设置学习率应相应减小。

批量梯度下降（batched gradient descent）是上述两种方法的折中，即每次迭代都使用 batch_size 个样本进行训练。

优点：

1. 节省了空间，同时对收敛速度和准确率也有一定的保证。

缺点：

1. 需要适当的选择 batch_size 大小。batch_size 过小时会遇到随机梯度下降的收敛性差的问题。

在实验中，由于最先进行了全批量梯度下降学习率的设置实验，随机梯度下降和批量梯度下降的学习率可以根据 batch 的大小做相应的调整。

Requirement 5：

最终在测试集的训练结果见下表：

Method	Learning_rate	Epoch	Test accuracy
全批量梯度下降	0.8	200	92.9%
随机梯度下降	1e-4	100	92.6%
批量 batch=100	0.04	300	92.7%

自由探索：

在训练过程中我发现，参数的初始化对最终的训练结果影响很大。一开始我使用 np.random.randn 初始化参数，但是在验证集上准确率较低，很难收敛，使用全批量梯度下降也很容易出现 divide by zero 的情况。后来发现使用全零初始化或较小的随机数初始化可以大大提高训练效果。

在训练中我也尝试了不同的 L2 正则化系数，但好像并没有明显的效果提升。

程序说明

源代码见 source.py 和 part2.py。运行方式采用 argparse 传参：

```
python source.py
--part #表示第几部分
--lr #表示学习率
--batch #表示 batch_size 大小
--epoch #表示 epoch 数
--skip-dev #表示是否使用验证集
--full #表示是否为全批量梯度下降训练
```