PRML Assignment 2

Part 1

1.最小二乘法

最小平方模型描述：

$$\mathbf{y}(\mathbf{x}) = W^T\boldsymbol{x} + w_0 = \widetilde{W}^T\widetilde{\boldsymbol{x}}$$

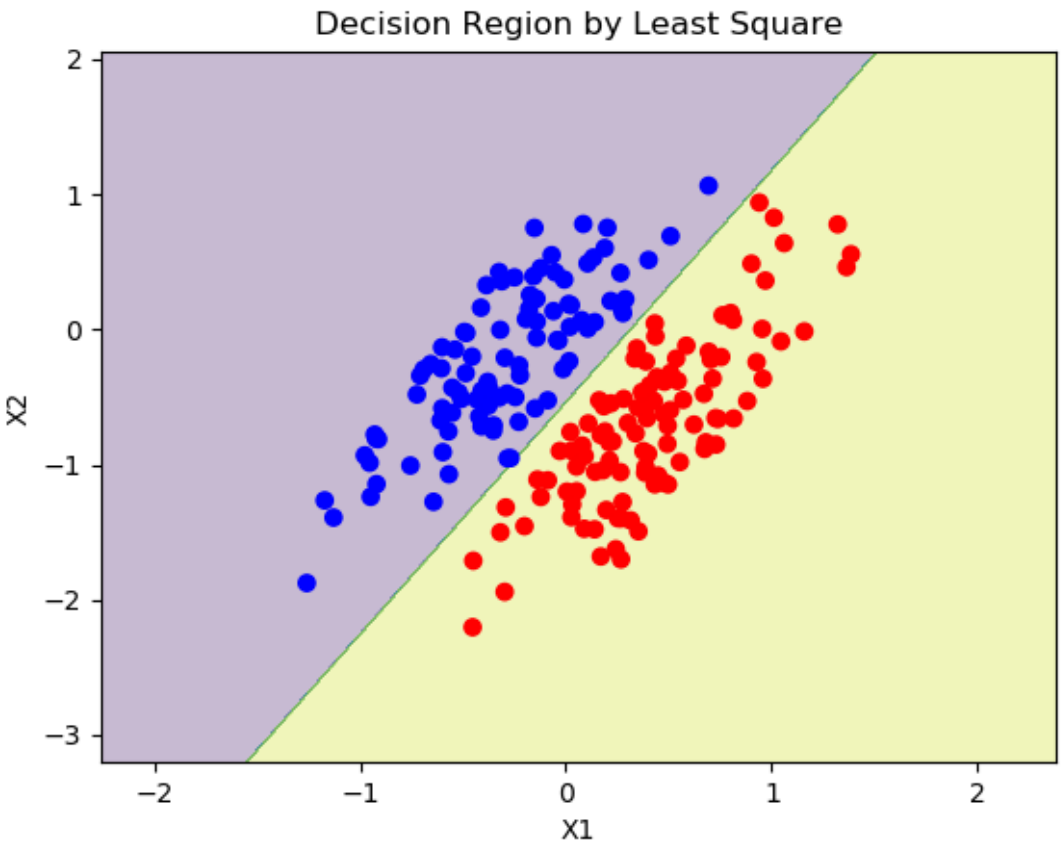
算法：

对训练数据集$\{\boldsymbol{x}_n, \boldsymbol{t}_n\}$，直接计算参数矩阵：

$$\widetilde{W} = \left(\tilde{X}^T\tilde{X}\right)^{-1}\tilde{X}^T T$$

实验结果：

分类准确率：100%



2.感知器算法

算法：

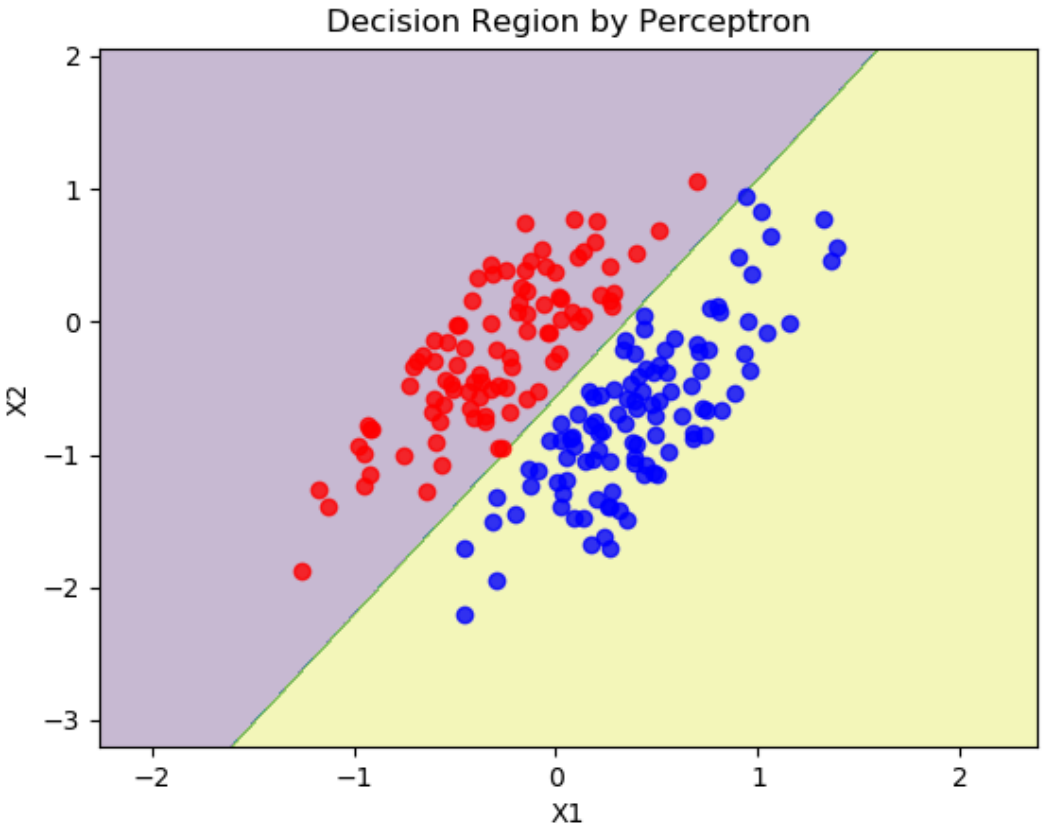
输入：训练集 $\{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$，迭代次数 T

初始化：$\mathbf{w}_0 \leftarrow 0, k \leftarrow 0$;
for $t = 1 \cdots T$ **do**
  随机对训练样本进行随机排序;
  for $n = 1 \cdots N$ **do**
    选取一个样本 $(\mathbf{x}^{(n)}, y^{(n)})$;
    if $\mathbf{w}_k^{\mathrm{T}}(y^{(n)}\mathbf{x}^{(n)}) \leq 0$ **then**
      $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + y^{(n)}\mathbf{x}^{(n)}$;
      $k \leftarrow k + 1$;
    end
  end
end
输出：$\mathbf{w}_k$

（引用自：《神经网络与深度学习》，邱锡鹏）

实验结果：

分类准确率：100%



Part 2

1.文本预处理

在训练和测试中，我们需要将标记单词是否出现的多热点向量（multi-hot）和标记文本类别的单热点向量（one-hot）作为样本输入。因此我们要先制作一张合适的单词表，记录在整个训练集中至少出现 10 次的单词并编号（编号对应多热点向量中的下标）。再根据该单词表生成每篇文章的多热点向量。

处理过程（见源代码 vocabulary_init 函数和 pre_processing 函数）：

① 用 python 的 re 模块中的 sub 函数，**将文本中的标点符号删除，将空白字符和不可见字符替换为 "　"，将所有字母转换为小写字母，然后根据空白分割文本字符串为单词字符串列表**。

② **取训练集中的所有单词**，构建“单词，词频”词典（可以使用 collections 模块中的 Counter 函数），**取词频>=10 的单词**，加入单词表词典并编号。

③ 对每篇文章的单词列表，根据**单词表中单词编号**生成**多热点向量**；根据其**类别**生成**单热点向量**。用上述方法分别生成训练集和测试集中每篇文章的多热点向量和单热点向量。

2.计算损失函数梯度

损失函数梯度计算过程：

（见代码中的 loss_grad 函数）

已知损失函数$\mathrm{L}=-\frac{1}{\mathrm{N}}\sum_{n=1}^{N}(y_n)^T log\hat{y}_n+\lambda\|W\|^2$

其中：

$$\hat{\mathrm{y}}=\mathrm{softmax(z)},\quad \mathrm{z}=\mathrm{W}^T x+b$$

则对第 n 个样本，由链式法则：

$$\frac{\partial \mathrm{L}^{(\mathrm{n})}(W,b)}{\partial \mathrm{W}}=-\frac{\partial (y_n)^T log\hat{y}_n}{\partial W}+\frac{\partial \lambda\|W\|^2}{\partial \mathrm{W}}=-\frac{\partial z_n}{\partial \mathrm{W}}\frac{\partial \hat{y}_n}{\partial z_n}\frac{\partial log\hat{y}_n}{\partial \hat{y}_n}y_n+2\lambda W$$

$$\frac{\partial \mathrm{L}^{(\mathrm{n})}(W,b)}{\partial \mathrm{b}}=-\frac{\partial (y_n)^T log\hat{y}_n}{\partial b}=-\frac{\partial z_n}{\partial \mathrm{b}}\frac{\partial \hat{y}_n}{\partial z_n}\frac{\partial log\hat{y}_n}{\partial \hat{y}_n}y_n$$

先计算各项偏导数：

$$\frac{\partial \hat{y}_n}{\partial z_n}=\mathrm{diag}(\hat{y}_n)-\hat{y}_n{\hat{y}_n}^T$$

$$\frac{\partial log\hat{y}_n}{\partial \hat{y}_n}=(\mathrm{diag}(\hat{y}_n))^T$$

$$\frac{\partial z_n}{\partial \mathrm{W}}=\mathrm{x}_n$$

$$\frac{\partial z_n}{\partial \mathrm{b}}=1$$

累乘后整理得：

$$\frac{\partial \mathrm{L}^{(\mathrm{n})}(W,b)}{\partial \mathrm{W}} = -\mathrm{x}_n(y_n - \hat{y}_n)^T + 2\lambda W$$

$$\frac{\partial \mathrm{L}^{(\mathrm{n})}(W,b)}{\partial \mathrm{b}} = -(y_n - \hat{y}_n)$$

于是在所有 N 个样本上：

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

$$\frac{\partial \mathrm{L}(W,b)}{\partial \mathrm{b}} = -\frac{1}{N}\mathbf{1}\left(Y - \hat{Y}\right)^T$$

（1）正则化项

在损失函数中加入 L2 正则化项的目的是降低模型的复杂度，避免过拟合；实际效果是使模型偏好学习更小的权值。

在贝叶斯统计的观点中，**正则化项是由对权重 w 的先验分布假设推导出的，因此正则化项包含了对权重 w 的先验信息，使得模型倾向于学习更小的权值。**

然而，对于偏置 b 则没有类似的先验信息，因为偏置 b 只包含模型的位置信息，而与模型的复杂度（方向信息）基本无关，所以一般不对偏置 b 加正则化项。

即使对偏置 b 进行先验分布假设，由于目的不同，先验信息不同，偏置 b 与权重 w 不应置于同一正则化系数下，而应为偏置 b 单独设置系数和约束方法。

（2）梯度检验

将**数值方法求得的偏导数与解析法梯度中的对应项相比较**，判断相对误差是否在合理范围内。

求偏导数的数值方法:

$$\frac{\partial J}{\partial \theta} = \lim_{\varepsilon \to 0} \frac{J(\theta + \varepsilon) - J(\theta - \varepsilon)}{2\varepsilon}$$

具体做法（见源代码中 grad_check 函数）：

①对权重矩阵 W 和偏置项 b 进行随机初始化，在数据集上求出 W 和 b 的解析梯度。

②随机选取 W 中的一项 w_{ij}，在损失函数上对 w_{ij} 求数值偏导：

取 w_{ij} 附近的一段很小的区间，用差商代替微分求得数值近似偏导数。

③将 W 解析梯度中的第(i,j)项与数值偏导数比较，判断相对误差是否在合理范围内。重复若干次。

④对 b 做类似的操作，判断相对误差是否在合理范围内。重复若干次。

经验证，该模型的梯度解析解正确。

3. 模型训练

（1）确定学习率

先明确我们选择**合适学习率的评判方法**：合适的学习率使得

①模型在训练过程中能收敛。

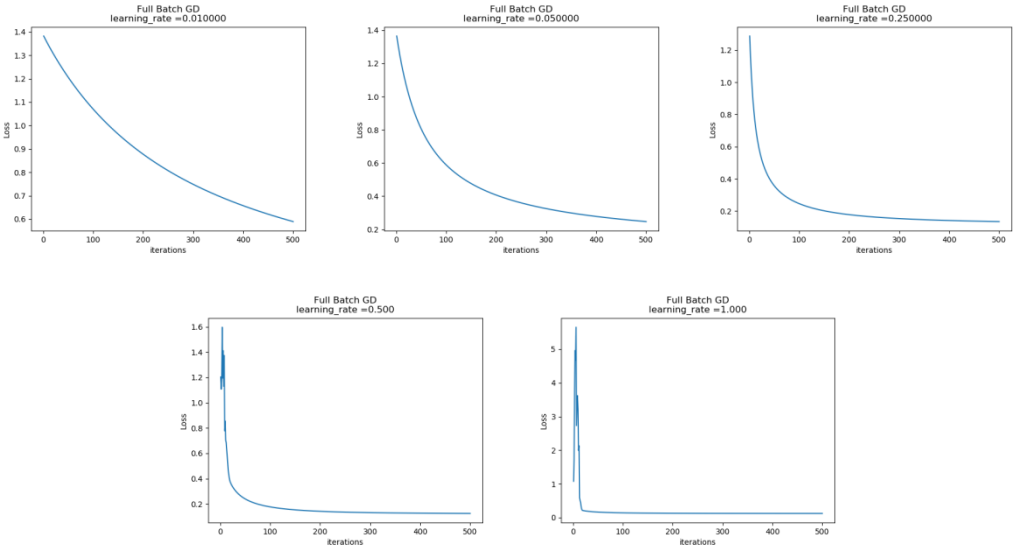
②训练完成后的模型在数据集上有**足够高的预测准确率**。

③在满足①②后，模型的训练过程应该**更短**（迭代次数更少，收敛速度更快）。

在我们的分类模型中，除了学习率 η，还存在其他超参如正则化系数 λ。它们会互相影响。一般来说，存在多个超参时，我们采用**正交实验法**确定最优的方案。比如取 N 个 η 值，M 个 λ 值，做 N*M 次训练，用交叉验证法确定最佳的参数取值。

为了方便，这里我们直接确定（事先粗略试验过的）λ 的值为1×10^{-3}。然后先对学习率 η 取不同值，观察其训练过程中的收敛情况，大致确定合适学习率的范围。

全批量梯度下降（Full Batch Gradient Descent）



取学习率 η 为[0.01, 1]上的若干离散值，观察模型的收敛速率。可以发现，**当学习率较小时，虽然损失值下降十分平滑，但模型收敛速度很慢；当学习率较大时，虽然收敛速度变快，但损失值变化会出现震荡的情况，这会使得训练过程变得不稳定。**因此我们需要做权衡。

由于全批量梯度下降在理论上能一直朝着极小值方向更新参数，最后能保证收敛于极小值点，且该模型的损失函数是凸函数，因此如果训练中损失值出现明显震荡，那一般是更新步长过大导致越过极小值点。我们可以认为这时的学习率过大。

η =0.25 时收敛速率比较适当，损失下降过程比较平滑，因此 0.25 附近的数值可作为模型的学习率。

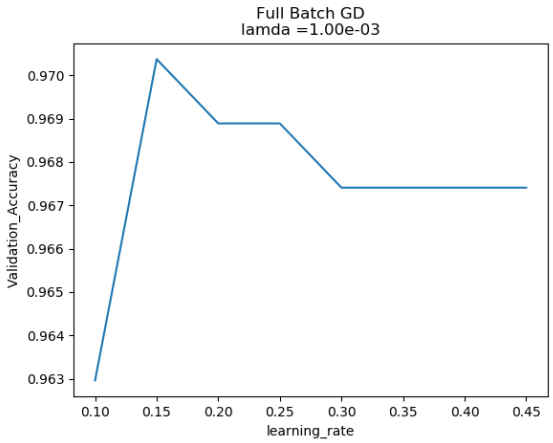
上述方法只能近似地确定一个**收敛速度较快且震荡很小**，训练过程比较稳定的学习率。为了得到一个最终测试结果比较好的模型，我们还会把训练集拆分为训练集和验证集，**用交叉验证的方法，根据模型在验证集上的预测准确率，确定超参**。

代码中实现了 2 种交叉验证法：

①简单交叉验证法，随机地对训练集进行 7:3 的简单分割。其中 30%作为验证集。

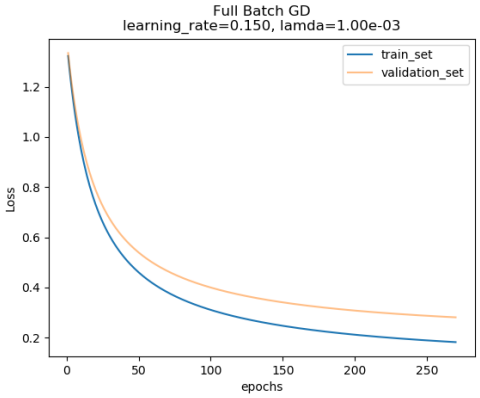
②K 折交叉验证法，随机地将数据集切分为 K 个互不相交的大小相同的子集，每次用 K-1 个子集作为训练集，余下 1 个子集作为验证集，重复 K 次，使用平均准确率进行评估。

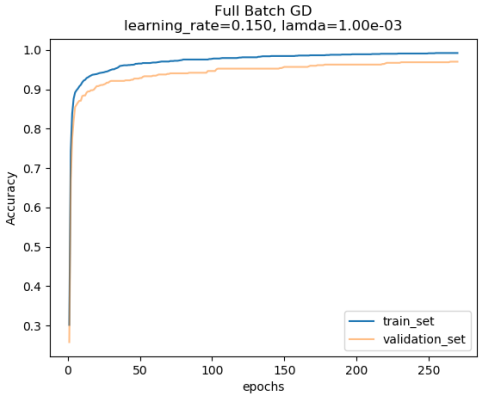
在 η =0.25 附近取一个区间[0.10, 0.45]，以步长 0.05 进行交叉验证确定学习率。我们得到**训练完成的模型在验证集上的预测准确率随学习率变化的曲线**。



由于学习率=0.15 时训练完成的模型在验证集上的预测准确率最高，因此我们最终确定学习率 η =0.15。

（下图为学习率 η =0.15 的模型训练过程）





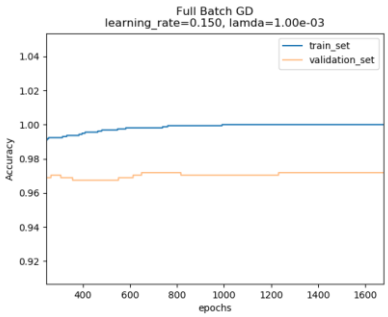
（2）训练终止条件

当模型接近收敛，损失函数值趋于稳定时，训练应该终止。由于我们通过交叉熵损失函数评价模型。因此我们在训练过程中要**记录每次迭代后模型在训练集上的交叉熵损失**（Loss），并在每几轮迭代结束后，**对模型在最后 k 轮迭代时的交叉熵损失求方差**。当**方差小于我们预设的阈值**时，我们便可**终止训练**。

当然我们也要**设置迭代次数的上限**，因为有时方差并不能下降到阈值以下。

对于全批量梯度下降，因为训练过程中，参数更新一般一直向着极小值方向进行，所以损失函数值一直下降且模型在**训练集上的准确率**一直上升，且这个准确率可以达到 100%。由于损失函数中含有正则化项，因此即使模型在训练集上已经有 100%的准确率，参数还是会向着权重矩阵范数更低的方向更新，所以损失函数值不会降低到 0。我们固然可以在训练

集准确率达到 100%时停止训练，但**观察准确率变化曲线后发现，模型在训练集上的预测准确率接近 100%时，在验证集上的准确率基本不变**，所以我们**不需要牺牲更多的训练时间来达到 100%的训练集预测准确率**。



如果我们在训练集中划分出验证集，那么在训练的过程中，我们还可以**用验证集来辅助判断训练终止的时机及挑选训练过程中产生的泛化性能最好的参数**：

①每完成 1 次或若干次迭代后，用当前模型在验证集上计算损失函数值。

②若验证损失达到新低，则记录此时的验证损失和模型的参数。

③接近训练结束时，训练损失逐渐降低，方差降到阈值以下，但验证损失却开始逐渐上升。当验证损失已经连续 m 轮比之前记录的最低验证损失高，终止训练。

④将最低验证损失对应的模型参数作为此次训练的最终模型参数。

（在本实验中，模型在验证集上的准确率更适合代替验证损失作为判断依据）

实际实验中，由于在大数据集上重新求交叉熵损失比较耗时，且整个训练过程的时间往往很长，我们往往还需要**监控模型的损失函数值变化情况和收敛情况**，在适当的时刻终止训练。

4. 3 种梯度下降方法

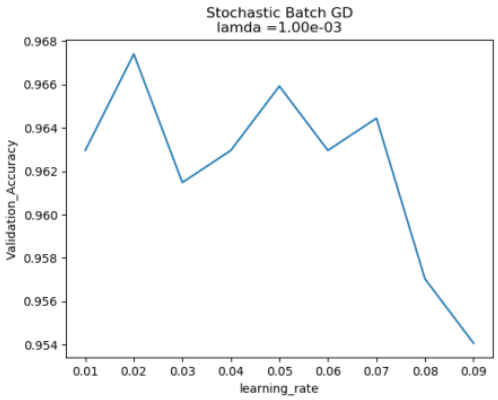
在实验之前，我们便可预知 3 种梯度下降方法所需的迭代次数肯定有较大差别，因此我们将自动判断训练终止条件的功能开启，这样我们就能知道模型接近收敛大概需要几轮迭代。之前我们已经在全批量梯度下降上做了实验，接下来我们试验另外 2 种梯度下降方法。

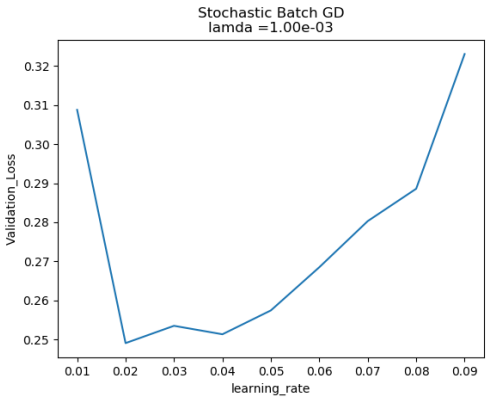
（1）观察另外 2 种方法

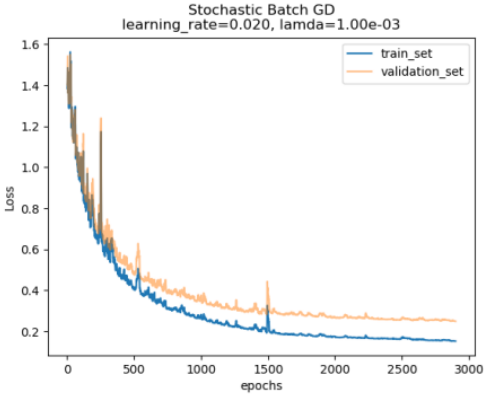
随机梯度下降（Stochastic Gradient Descent）：

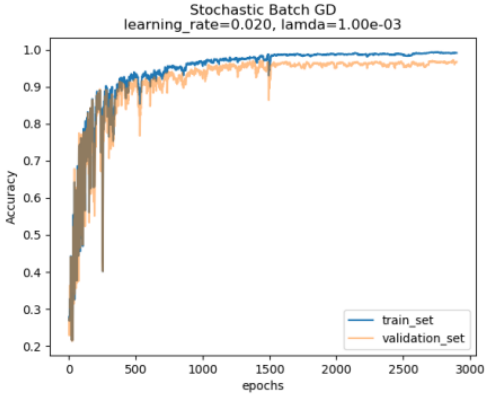
随机梯度下降算法每次从训练集中随机选择一个样本来学习，计算目标函数的梯度并更新参数。显然，**每次迭代中的参数更新不一定向着极小值方向进行**。所以随机梯度下降过程中，**损失值会有较大的波动，且对参数更新步长（学习率）更加敏感**。之前在全批量梯度下降中确定的学习率已经不适用于此。

经过交叉验证后，我们通过验证集准确率-学习率曲线和验证集损失-学习率曲线，确定**随机梯度下降方法的学习率 η =0.02**：





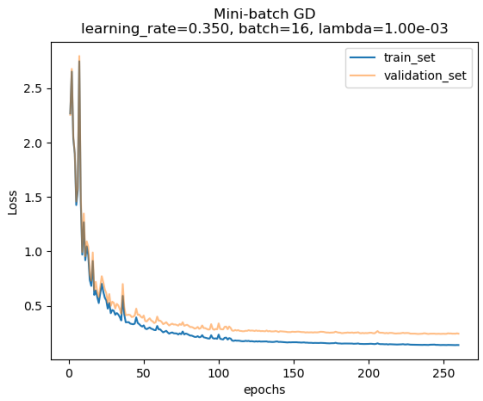


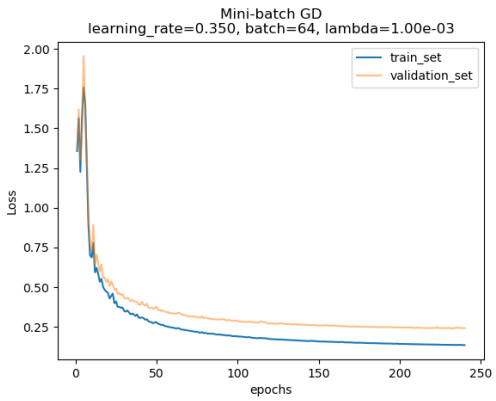


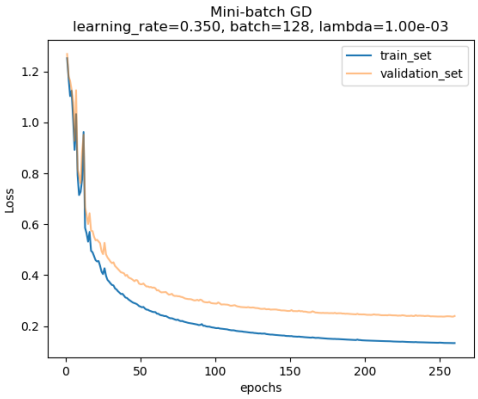
可以发现，随机梯度下降**单次迭代时间很短**，只需要处理单个样本，但由于单个样本产生的更新不一定朝着正确的方向进行，在训练过程中损失函数值会**剧烈波动**，因此训练的**迭代次数会增加，收敛速度变慢**。如果我们**选取比较低的学习率，或使用衰减法**在训练过程中逐渐降低学习率，则随机梯度下降**最终会和批量梯度下降具有相同的收敛性**，即凸函数收敛于全局极值点，非凸函数收敛于局部极值点。

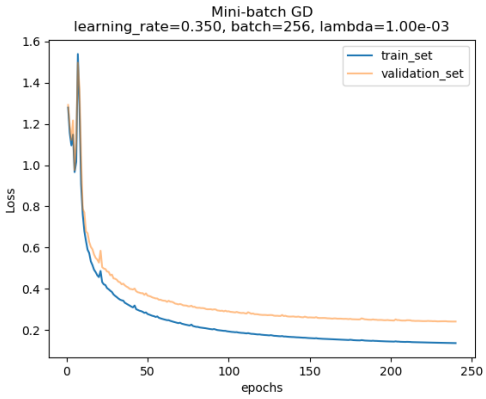
小批量梯度下降（Mini-batch Gradient Descent）：

取不同的批量大小（batch size），得到如下训练过程：



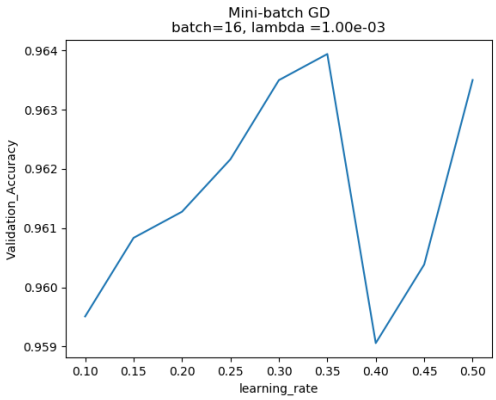


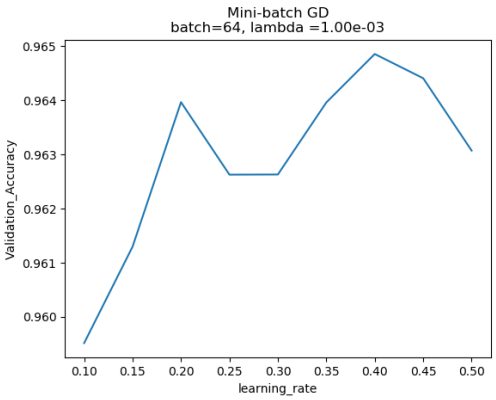


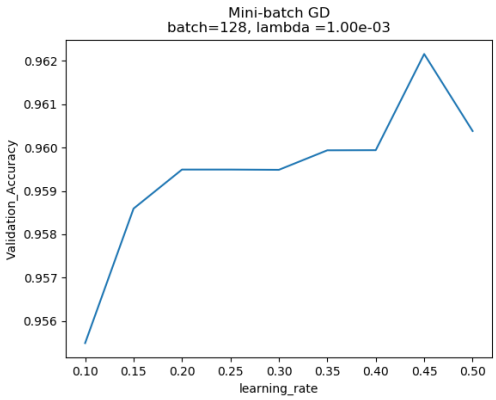


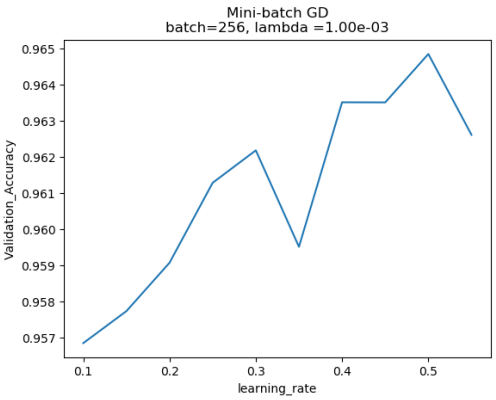
小批量梯度下降的批量大小为 1 个常数，为方便计算机处理，这个值通常取 2 的幂。**相比于全批量梯度下降，小批量梯度下降单次迭代更快；相比于随机梯度下降，小批量梯度下降的收敛过程震荡更小，训练所需的迭代次数更少。因此若选取适中的批量大小，即综合了另外 2 种算法的优点，则小批量梯度下降的训练过程所需时间会比其他 2 种算法更短（按照我们之前设定的训练终止条件）。**

当批量很小时，模型的损失函数值下降过程接近随机梯度下降;当批量很大时，模型的损失函数值下降过程接近全批量梯度下降。**批量大小也被看做一个超参**。因此我们这里需要为小批量梯度下降算法选择 2 个超参的值。取**不同的批量大小，不同的学习率**进行训练，作各个批量大小的验证集准确率-学习率曲线：









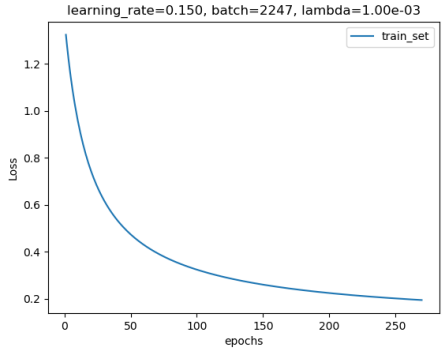
综合准确率，训练迭代次数，训练过程损失值曲线震荡情况等因素，我们**确定批量大小=64，学习率 η =0.40**。

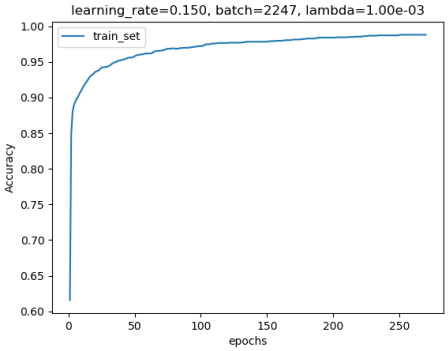
（2）3 种梯度下降方法的对比评价

	优点	缺点
FBGD	1. 训练所需迭代次数少。 2. 每次参数更新使用全部训练集样本，使得每次更新都朝着极值点方向进行，最后能保证收敛于极值点。（在凸函数上能收敛于全局最优点，在非凸函数上收敛于局部最优点） 3. 1 次迭代即对整个训练集的数据进行计算，利用矩阵运算，可以充分发挥并行计算的优势。	1. 单次迭代时间很长。 2. 不能进行在线模型参数更新。当训练集很大且不能完全存入内存中时，FBGD 一般不可行。
SBGD	1. 单次迭代时间很短。 2. 在训练集很大时，SBGD 的效率比 FBGD 高，甚至可能在处理完整个训练集之前就收敛。 3. 在非凸函数上，训练过程中函数值的震荡可能会使参数从一个局部最优到另一个更好的局部最优，甚至到全局最优。	1. 模型收敛过程中损失函数值波动很大，不稳定。 2. 训练所需迭代次数很多，收敛速度慢。
MBGD	1. 克服了 FBGD 的缺点：可以保证单次迭代使用的样本能装入内存中且单次迭代速度较快。 2. 克服了 SBGD 的缺点：收敛过程波动性降低，训练所需迭代次数较少。 3. 综合了 FBGD 和 SBGD 的优点：选取合适的批量大小时，模型收敛速度比 FBGD 和 SBGD 都快。且也可以利用矩阵运算进行高效的并行计算。	1.相比于其他 2 种方法没有显著的缺点，但需要额外确定一个超参——批量大小（batch size）。

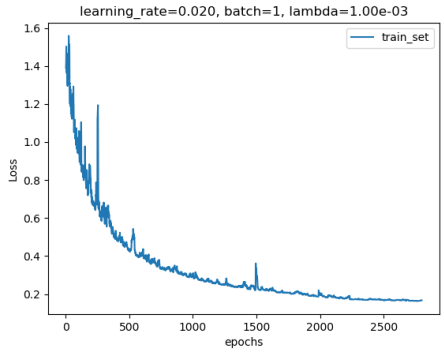
5.模型测试结果

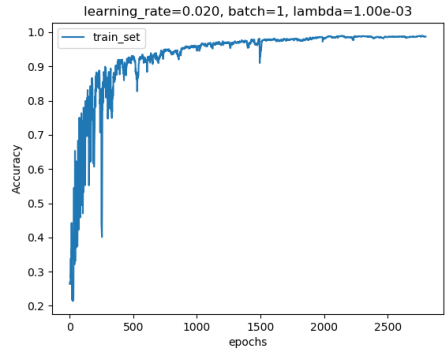
	学习率 η	正则化系数 λ	迭代次数	测试准确率
FBGD	0.150	1×10^{-3}	270	92.647%
SBGD	0.020	1×10^{-3}	2700	92.981%
MBGD (batch-size=64)	0.400	1×10^{-3}	240	93.115%



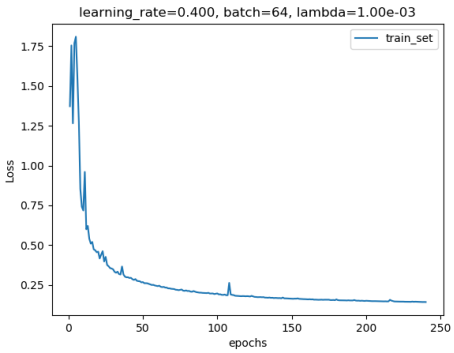


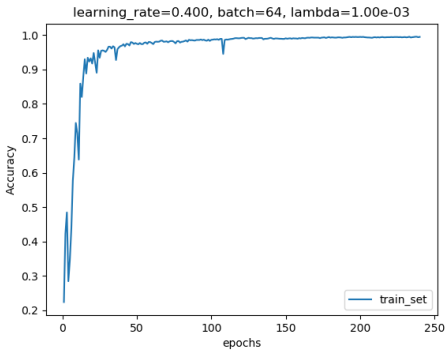
FBGD





SBGD





MBGD

6. 备注：程序使用方法

usage: source.py [-h] [--method {part1,grad_check,fb,sb,mb}] [--kcv]
[--iter ITER] [--learning_rate LEARNING_RATE] [--lamda LAMDA]
[--batch BATCH] [--lmin LMIN] [--lmax LMAX] [--lstep LSTEP]

示例：

①Decision Region of Part1：

python source.py --method part1

②Gradient Check:
python source.py --method grad_check

③Mini-batch Gradient Descent, train & test :
(learning rate=0.40, lambda=1e-3, batch size=64, max epochs = 1000)
python source.py --method mb -l 0.40 --lamda 1e-3 -b 64 --iter 1000

④Full Batch Gradient Descent, k-cross-validation, the selection of learning rate:
(learning rate in [0.10, 0.40], step of experiment=0.05, lambda=1e-3, max epochs = 500)
python source.py --method fb --kcv --iter 500 --lmin 0.10 --lmax 0.40 --lstep 0.05 --lamda 1e-3

参考文献

1. 斯坦福大学 CS229 课程讲义，吴恩达
2. Python Machine Learning (2nd edition), Sebastian Raschka, Vahid Mirjalili.
3. 神经网络与深度学习，邱锡鹏
4. 梯度下降算法分类及调参优化技巧：
http://xudongyang.coding.me/gradient-descent/