中山大学移动信息工程学院本科生实验报告

（2017年秋季学期）

课程名称：人工智能

年级	专业方向	学号	姓名
1501	移动（互联网）	15352005	蔡景韬

一、实验题目

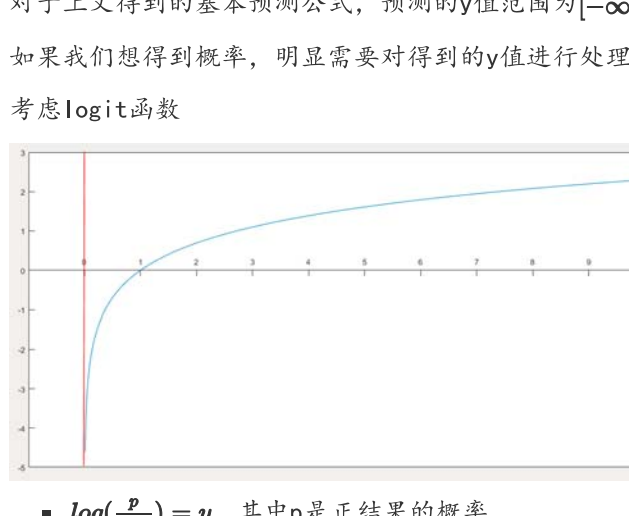
- 逻辑回归模型 Logistic Regression Model

二、实验内容

1. 算法原理

- 软分类、硬分类
  - 软分类
    - 基于**概率模型**的分类模型，输出不同类对应的概率，最后取概率最高的类别作为分类结果
    - **概率模型**：形式为P(X|y)，即在学习过程中，y未知，训练后模型得到的输出是x的一系列值的概率
    - 如，NB分类、逻辑回归分类
  - 硬分类
    - 基于**非概率模型**的分类模型，输出决策函数的决策结果
    - **非概率模型**：形式为决策函数，即输入x到输出y的一个映射，且输出唯一
    - 如，决策树、PLA、KNN算法等
- 逻辑回归（从PLA到LR）
  - 通过训练得到权重矩阵，用以表示各列属性与结果的相关程度
    - 权重为正，说明该属性与“是”的结果正相关，且值越大，正相关程度越大，正影响越大
    - 权重为负，说明该属性与“是”的结果负相关，且绝对值越大，负相关程度越大，负影响越大
  - 基本的预测公式
    - 将权重矩阵与属性值内积（相应的对属性值进行加权），并与某个阈值作差
    - 上述表示可写出表达式：$y=(\sum_{i=1}^{d} W_iX_i)-threshold$
  - 公式的数学处理
    - 为了简化表达式，我们可以把阈值当作W_0，且对X矩阵多添加$X_0=1$（阈值可以当作一个属性且每一个训练文本共用一个阈值，于是可以把阈值属性值都取1）
    - 表达式可以重写为：$y=\sum_{i=0}^{d} W_iX_i=W^TX$
  - PLA算法_预测公式
    - 对于上文得到的公式，PLA使用sign()函数对公式结果进行处理，从而得到PLA的预测公式：$f(x)=sign(W^TX)$
      - 当预测公式得到正结果，则预测为正结果
      - 否则预测为负结果
  - PLA算法_更新公式

    对于权值的更新，使用梯度下降法进行更新（之前写过详细推导，这里就不赘述）
    - 使用向量模型理解有，X矩阵是一个高维的向量，权重矩阵是另一个高维向量，X矩阵与权重矩阵的内积的正负可以表征两个向量之间的夹角
    - 于是可以得到更新公式：$W^{p+1}=W^p+y_iX_i$
  - LR算法_预测公式
    - 对于上文得到的基本预测公式，预测的y值范围为$[-\infty,+\infty]$
    - 如果我们想得到概率，明显需要对得到的y值进行处理
    - 考虑logit函数

      - $log(\frac{p}{1-p})=y$，其中p是正结果的概率
      - 验证其合理性：
        - 当p趋近于1时（即正结果概率近乎为1），可以看到，y趋近于$+\infty$，根据PLA算法，知道此时是预测为正结果的
        - 当p趋近于0时（即正结果概率近乎为0），可以看到，y趋近于$+\infty$，根据PLA算法，知道此时是预测为负结果的
      - 根据上文的logit函数，于是可以求解正结果概率p

        $log(\frac{p}{1-p})=W^TX \quad \Rightarrow \quad p=\frac{1}{1+e^{-W^TX}}$
    - 可以得到LR算法的概率公式：$f(x)=P(y|x)=p^y(1-p)^{1-y}$，其中y表示x对应的分类标签
      - 当$y=1$，$f(x)=P(1|x)=p$
      - 当$y=0$，$f(x)=P(0|x)1-p$
  - LR算法_更新公式

    对于权值，使用极大似然法进行更新（PLA使用的是最小二乘法）
    - 似然函数：

$$likelihood=\prod_{i=1}^{M}P(label|x_i)=\prod_{i=1}^{M}p^y(1-p)^{1-y}$$

    - 左右取-log有

$$Err(W)=-log(likelihood)=-log\left(\prod_{i=1}^{M}p^y(1-p)^{1-y}\right)=-\sum_{i=1}^{M}\left[y_ilog(p)+(1-y_i)log(1-p)\right]$$

      其中，$p=\frac{1}{1+e^{-W^TX}}$
    - 使用极大平均似然法进行更新，即$\frac{1}{M}Err(W)$取最小时，似然函数达到最大值
    - 根据高数知识，当$Err(W)$梯度为0时，函数取到最小值（关于连续可导性、二阶可微、凸函数的证明）

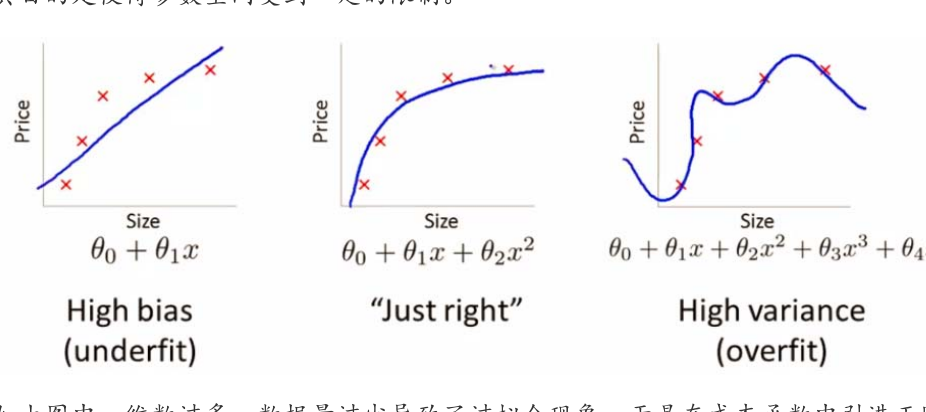
      下列式子中，W为权重矩阵，X_n为第n行的数据集矩阵，y_n为第n行的label：

$$
\begin{aligned}
\nabla Err(W)=\frac{\partial Err(W)}{\partial W}&=-\sum_{n=1}^{N}\left[y_n\frac{\frac{\partial p}{\partial W}}{p}+(1-y_n)\frac{-\frac{\partial p}{\partial W}}{1-p}\right]\\
&\Rightarrow 令\ 1+e^{-W^TX_n}=u,\ 则\ p=\frac{1}{u}\\
&=-\sum_{n=1}^{N}\left[y_n\frac{1}{p}\frac{-\frac{\partial u}{\partial W}}{u^2}+(1-y_n)\frac{-1}{1-p}\frac{-\frac{\partial u}{\partial W}}{u^2}\right]\\
&\Rightarrow 令\ -W^TX_n=v,\ 则\ u=1+e^v\\
&=-\sum_{n=1}^{N}\left[y_n\frac{1}{p}\frac{-1}{u^2}(e^v)\frac{\partial v}{W}+(1-y_n)\frac{-1}{1-p}\frac{-1}{u^2}(e^v)\frac{\partial v}{W}\right]\\
&=-\sum_{n=1}^{N}\left[y_n\frac{1}{p}\frac{-1}{u^2}(e^v)(-X_n)+(1-y_n)\frac{-1}{1-p}\frac{-1}{u^2}(e^v)(-X_n)\right]\\
&=-\sum_{n=1}^{N}\left[y_n\frac{1}{p}(p)(1-p)(X_n)+(1-y_n)\frac{-1}{1-p}(p)(1-p)(X_n)\right]\\
&=-\sum_{n=1}^{N}\left[y_n(1-p)(X_n)+(1-y_n)(-p)(X_n)\right]\\
&=\sum_{n=1}^{N}(p-y_n)X_n
\end{aligned}
$$

    - 又$\nabla Err(W)$是关于矩阵的N元函数，难以求得$\nabla Err(W)=0$，所以使用数值分析中的迭代方法求解零点，则可以得出**权值的更新公式**：

$$
\begin{aligned}
W_{t+1}&=W_t-\frac{\eta}{M}\nabla Err(W_t)\\
&=W_t-\frac{\eta}{M}\sum_{n=1}^{N}(p-y_n)X_n\\
&=W_t-\frac{\eta}{M}\sum_{n=1}^{N}\left[(\frac{1}{1+e^{-W^TX_n}}-y_n)X_n\right]
\end{aligned}
$$

    - **注意事项**
      - 如果e的指数太大或太小，如e^{18}或e^{-18}，算出来的$\frac{1}{1+e^{-W^TX_n}}$要么是$\frac{1}{\infty}=0$，或者是$\frac{1}{1+0}=1$
      - 要控制点乘运算W^TX_n的大小
        - 初始化W不能太大
        - 学习率η不能太大
  - LR算法_正则化
    - 正则化的概念
      - 当模型参数很多，而我们可用的数据非常少时，极易出现过拟合的问题。为此，就引入了正则化。
      - 其目的是使得参数空间受到一定的限制。

      - 如上图中，维数过多、数据量过少导致了过拟合现象，于是在成本函数中引进正则项（惩罚项）限制其参数空间
      - 正则化一般分为L1正则化与L2正则化

        $J(w)\cong J(w)+\lambda\|w\|\begin{cases}L2\ \rightarrow \frac{1}{2}\lambda\|w\|_2{}^2\\ L1\ \rightarrow \lambda\|w\|_1\end{cases}$ 其中，$\|x\|_p:=\left(\sum_{i=1}^{n}|x_i|^p\right)^{\frac{1}{p}}$
      - 正则项的加入改变了参数的学习规则，它在参数的每一步更新中都新增了一个系数参量。
      - L1正则和L2正则的差别
        - L2对参数的正则更加的平滑，即它限制了参数空间，但对参数的影响是平滑的
        - 而L1就比较突兀，可能会直接使得某些参数的取值为0.。
        - 总结起来就是：L1会引入稀疏性，而L2会充分利用更多的特征。
    - 本次实验使用L2正则化
    - 进行L2正则化
      - 在误差函数$Err(W)$中引入正则项，重写误差函数有

        $Err(W)=-\frac{1}{M}\sum_{i=1}^{M}\left[y_ilog(p)+(1-y_i)log(1-p)\right]+\frac{1}{2}\lambda\sum_{j=1}W_j^2$
      - 重新推导误差函数的梯度

        下式中的w是W矩阵的一个维度

$$
\begin{aligned}
\nabla \tilde{Err}(w)&=\nabla\left[Err(w)+\frac{1}{2}\lambda w^2\right]\\
&=\nabla Err(w)+\lambda w
\end{aligned}
$$

      - 则$\tilde{Err}(W)$的梯度重写为$\nabla\tilde{Err}(W)=\nabla Err(W)+\lambda W$
      - 更新公式重写为

$$
\begin{aligned}
W_{t+1}&=W_t-\eta\left(\frac{1}{N}\sum_{n=1}^{N}\left[(\frac{1}{1+e^{-W^TX_n}}-y_n)X_n\right]+\lambda W_t\right)\\
&=(1-\eta\lambda)W_t-\frac{\eta}{N}\sum_{n=1}^{N}\left[(\frac{1}{1+e^{-W^TX_n}}-y_n)X_n\right]
\end{aligned}
$$

      - 与上文相同，需要保证W^TX_n足够小，所以λ也要较小
- 动态学习率
  - 在迭代过程前半部分学习率应该设置较大，因为初始化的W是随机给的，可能离最优解很远
  - 在迭代过程的后半部分学习率应该设置较小，因为已经接近最优解附近，此时应该减小学习率（步长），让其不至于错过最优解，可以一步一步的靠近最优解
  - **算法实现：每次迭代都将** η **乘以0.998**
- 逻辑回归的优缺点
  - LR有很多方法来对模型正则化，如L1正则化、L2正则化，从而避免了欠拟合和过拟合的问题。
  - LR有很好的概率解释，且很容易利用新的训练数据来更新模型。
  - 比起NB的条件独立性假设，LR不需要考虑样本是否是相关的。
  - LR与DT/RF（决策树/随机森林）的比较：
    - 逻辑回归对数据整体结构的分析优于决策树，而决策树对局部结构的分析优于逻辑回归。
    - 逻辑回归擅长辨识线性关系，而决策树对线性关系的把握较差。

      线性联系在实践中有很多优点：简洁，易理解，可以在一定程度上防止对数据的过度拟合。
    - 逻辑回归对极值比较敏感，**容易受极端值的影响**，而*决策树对极值有很好的抗干扰性。
    - 两者的算法差别
      - 决策树由于采用分割的方法，所以能够深入数据细部，但同时就失去了对全局的把握。
        - 一个分层一旦形成，它和别的层面或节点的关系就被切断了，以后的挖掘只能在局部中进行。
        - 同时由于切分，样本数量不断萎缩，所以无法支持对多变量的同时检验。
      - 逻辑回归，始终着眼整个数据的拟合，所以对全局把握较好。但无法兼顾局部数据，或者说缺乏探查局部结构的内在机制。
    - 总结：
      - 两种方法都很快且可扩展。
      - **在正确率方面，RF比LR更优。**
      - 但是LR可以在线更新且提供有用的概率信息。

2. 伪代码

- 划分训练集与验证集

```
训练集 = 数据集的前75%
验证集 = 数据集的后25%
```

- LR算法
  - 训练W矩阵

```
while iterations --
    动态学习率: eta=eta*0.995
    更新W矩阵:  update(eta,lambda)
```

  - 更新W矩阵

```
for i:属性列数Cols
    根据矩阵更新公式更新W矩阵,lambda是正则项系数:
        W_new[i] = (1-eta*lambda)*W[i] - 1/TrainSize*eta*文本梯度和(i)
```

  - 计算某一列属性的文本梯度和

```
第j列属性
for i:文本数Rows
    根据梯度公式求解每一行文本的梯度，并求和:
    sum += ( 1/(1+exp(- W矩阵·第i个文本矩阵) ) - 第i个文本第j列属性值 ) * 第i个文本的label
```

3. 关键代码截图

- 划分训练集与验证集

```
void getTrainVal(string str="IS NOT TEST"){
    if ( str=="IS NOT TEST" ){
        int Size = Data.size(),TrainSize=Size*0.75 ;
        Train.assign(Data.begin(),Data.begin()+TrainSize) ;
        Val.assign(Data.begin()+TrainSize,Data.end()) ;
    }
    // 小数据测试
    else
        Train.assign(Data.begin(),Data.end()) ;

}
```

- LR算法

```
void doTrain(int iterations, double eta=1, double lambda=0){
    while ( iterations-- ){
        eta *= 0.995 ;
        update(eta,lambda) ;
//      cout << "AccuracyRate: " << AccuracyRate() << endl ;
//      cout << " " << eta << endl ;
    }
}
```

```
// W矩阵的更新函数
void update(double eta, double lambda){
    vector<double> W_new(W.size()) ;
    int TrainSize = Train.size() ;
    for ( int i=0 ; i<Col ; i++ )
        W_new[i] = (1-eta*lambda)*W[i] - eta*gradSum(i)/TrainSize ;
    W.swap(W_new) ;
}
```

```
// 求解各个文本的梯度和
double gradSum(int col){
    double ans=0,TrainSize=Train.size() ;
    for ( int i=0 ; i<TrainSize ; i++ ){
        double dot_product = dotProduct(Train[i]) ;
        ans += ( (double)1/(double)(1+exp(-dot_product)) - Train[i][Col] ) * Train[i][col] ;
    }
    return ans ;
}
```

```
// 矩阵点乘
double dotProduct(const vector<double>& data){
    double sum=0 ;
    for ( int j=0 ; j<Col ; j++ )
        sum += data[j]*W[j] ;
    return sum ;
}
```

4. 创新点&优化

- 时间优化
  - 使用0-3代码优化

    执行1000次的情况下
    - 没有使用优化的情况下

```
D:\GT2\As student\AI_Lab\Lab5\15352005_caijingtao.exe
cost time: 393191ms
```

    - 使用优化的情况下

```
D:\GT2\As student\AI_Lab\Lab5\15352005_caijingtao.exe
cost time: 56974ms
```

  - 用二维数组代替vector

    执行1000次迭代时能减少2s，但后来加了正则化等优化之后就被vector赶上来了，这里就不贴了
- 准确率优化

  使用正则化与动态学习率

三、实验结果及分析

1. 实验结果展示示例（使用小数据集）

- 训练集

```
0,1,0
1,1,0
3,3,1
4,3,1
```

- 初始化的W

```
// 小数据集
    W.push_back(-5) ;
    W.push_back(2) ;
    W.push_back(1) ;
```

- 步长：$\eta=0.1$
- 迭代一次计算

$$\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}_A=\begin{pmatrix}-5\\2\\1\end{pmatrix}^T\begin{pmatrix}1\\0\\1\end{pmatrix}=w_0\times1+w_1\times x_A^1+w_2\times x_A^2=-5\times1+2\times0+1\times1=-4$$

$$\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}_B=-5\times1+2\times1+1\times1=-2$$

$$\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}_C=-5\times1+2\times3+1\times3=4$$

$$\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}_D=-5\times1+2\times4+1\times3=6$$

$$
\begin{aligned}
w_0^{new}&=w_0^{old}-\eta\sum\left[\left(\frac{e^{\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}}}{1+e^{\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}}}-y\right)x_0\right]\\
&=-5-\eta\left[\left(\frac{e^{-4}}{1+e^{-4}}-0\right)\times1+\left(\frac{e^{-2}}{1+e^{-2}}-0\right)\times1+\left(\frac{e^{4}}{1+e^{4}}-1\right)\times1+\left(\frac{e^{6}}{1+e^{6}}-1\right)\times1\right]\\
&=-5-\eta\times0.1167\\
&=-5.0117
\end{aligned}
$$

$$
\begin{aligned}
w_1^{new}&=w_1^{old}-\eta\sum\left[\left(\frac{e^{\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}}}{1+e^{\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}}}-y\right)x_1\right]\\
&=2-\eta\left[\left(\frac{e^{-4}}{1+e^{-4}}-0\right)\times0+\left(\frac{e^{-2}}{1+e^{-2}}-0\right)\times1+\left(\frac{e^{4}}{1+e^{4}}-1\right)\times3+\left(\frac{e^{6}}{1+e^{6}}-1\right)\times4\right]\\
&=2-\eta\times0.0554\\
&=1.9945
\end{aligned}
$$

$$
\begin{aligned}
w_2^{new}&=w_2^{old}-\eta\sum\left[\left(\frac{e^{\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}}}{1+e^{\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}}}-y\right)x_2\right]\\
&=1-\eta\left[\left(\frac{e^{-4}}{1+e^{-4}}-0\right)\times1+\left(\frac{e^{-2}}{1+e^{-2}}-0\right)\times1+\left(\frac{e^{4}}{1+e^{4}}-1\right)\times3+\left(\frac{e^{6}}{1+e^{6}}-1\right)\times3\right]\\
&=1-\eta\times0.0758\\
&=0.9924
\end{aligned}
$$

- 得到的W矩阵

$$W=\begin{pmatrix}-5.0117\\1.9945\\0.9924\end{pmatrix}$$

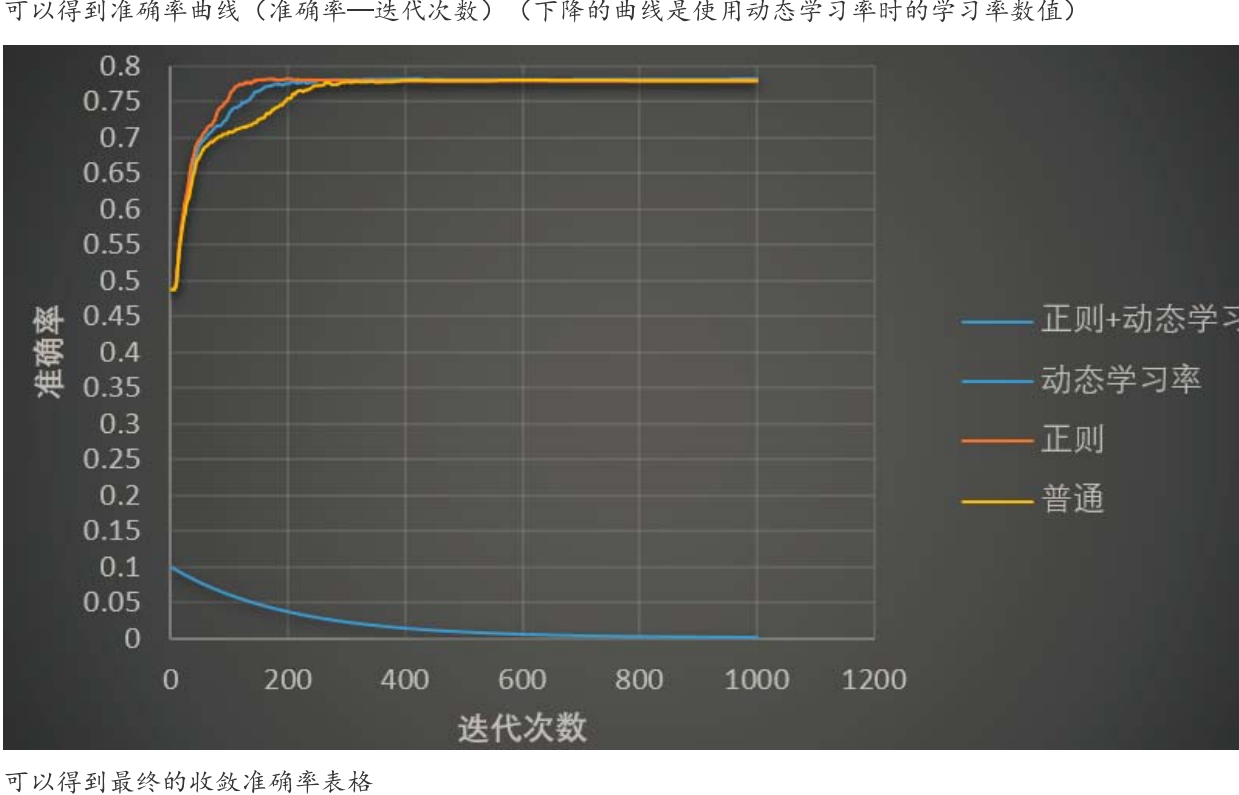
- 程序执行结果（可以看到是正确的答案）

```
D:\GT2\As student\AI_
cost time: 0ms
-5.01167
1.99446
0.992419
```

2. 评测指标展示即分析（如果实验题目有特殊要求，否则使用准确率）

- W矩阵初始化为1，执行1000次，学习率$\eta=0.1$，正则参数$\lambda=0.1$，动态学习率每次乘以0.995

  可以得到准确率曲线（准确率—迭代次数）（下降的曲线是使用动态学习率时的学习率数值）

  可以得到最终的收敛准确率表格

正则+动态	正则	普通
0.7815	0.7785	0.779

- 从图像可以看到，收敛速度：正则 ＞ 正则+动态学习 ＞ 普通

  原因：动态学习率到中期学习率较小，所以收敛速度也慢
- 从收敛准确率表格可以看到，收敛准确率：正则+动态学习 ＞ 普通 ＞ 正则

  原因：动态学习率到后面可以一步一步的逼近最优解，而正则能够避免过拟合

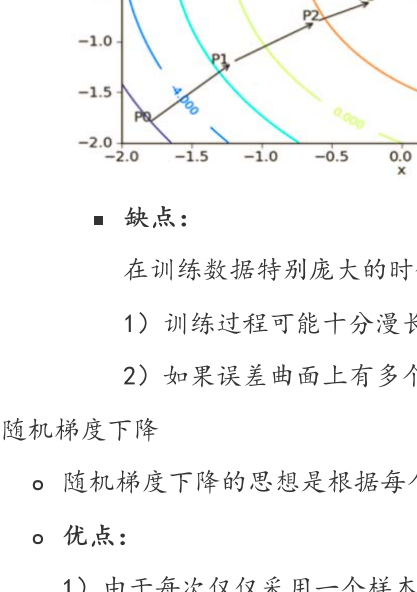
四、思考题

1. 如果把梯度为0作为算法停止的条件，可能存在怎样的弊端？

- 迭代很难得到精确值，如果能够得到满足精度的近似值，迭代就可以停止
- 如果把算法停止的条件设置为梯度为0，则
  - 算法可能无法停止，因为数据集的梯度可能永远无法收敛到0
  - 数据集的梯度可以收敛到0，但所用的时间太长，没有意义

2. η 的大小会怎么影响梯度下降的结果？给出具体的解释，可视化的解释最好，比如图形展示等

- 如果 η 值太大，则每次迭代就有可能出现超调的现象，会在极值点两侧不断发散，最终损失函数的值是越变越大，而不是越来越小，且还可能出现该曲线不断震荡的情形。
  - 如下图图示，由于学习率太大，导致错过了最优解，且由于下次的逼近又会跳回左侧，所以会出现不断振荡的情形
- 如果 η 值太小，这该曲线下降得很慢，甚至在很多次迭代处曲线值保持不变，最终导致在规定的迭代次数中梯度无法下降到一个较好的结果



3. 思考批梯度下降和随机梯度下降这两种优化方法的优缺点

- 批梯度下降
  - 批梯度下降算法在更新权值矩阵的时候要遍历整个数据集
  - 批梯度下降每次移动的方向都是确定的，一直更新后在合适的学习率情况下可以收敛到一个局部最小点的，因为梯度值会越来越小，它和固定的学习率相乘后的积也会越来越小

    - **缺点：**

      在训练数据特别庞大的时候，可能会出现

      1）训练过程可能十分漫长；

      2）如果误差曲面上有多个局极小值，那么不能保证这个过程会找到全局最小值。
- 随机梯度下降
  - 随机梯度下降的思想是根据每个单独的训练样本来更新权值
  - **优点：**

    1）由于每次仅仅采用一个样本来迭代，训练速度很快

    2）由于每次迭代方向不同，可能会跳出局部最优，而收敛到全局最优
  - **缺点：**

    随机梯度下降法用于仅仅用一个样本决定梯度方向，导致解很有可能不是最优。

    对于收敛速度来说，由于随机梯度下降法一次迭代一个样本，导致迭代方向变化很大，不能很快的收敛到最优解。