中山大学数据科学与计算机学院

移动信息工程专业-人工智能

本科生实验报告

（2017-2018 学年秋季学期）

课程名称：**Artificial Intelligence**

教学班级	**m2**	专业（方向）	**软件工程（移动工程信息）**
学号	**15352446**	姓名	**钟展辉**

一、 实验题目

Logistic Regression Model 逻辑回归

二、 实验内容

1. 算法原理

logistic/sigmoid 函数作用：把取值范围从负无穷到正无穷的公式计算结果，压缩到 0 和 1 之间，这样的输出值表达为“可能性”更直观。

逻辑回归算法用于估计预测目标的可能性，它属于软分类算法，即最终得到的是一个具体的概率，而不仅仅是“是”或“不是”这样的二分类结果；

逻辑回归能提供一个样本属于正类的可能性是多少，比如假设对于样本 x，回归系数 w，（w^T 是向量 w 的转置），使用 sigmoid 函数可以获得这个样本属于正类的概率为:

$$h(x) = \frac{1}{1+e^{-w^T x}}$$

本次实验标签是二分类模型的（1 或者 0），所以当我们预测得到概率 h(x)时这样判断结果：如果样本 x 属于正类的概率大于 0.5，那么就判定它是正类，否则就是负类。

这条公式很简洁，唯一的未知的变量就是参数 w，我们需要通过用训练集来训练求解得到最优 w，用这个最优 w 代入上述公式中即可对测试集样本进行预测。先初始化一个 w 向量（比如设置其所有元素为 1），然后不断迭代找到最优的 w，那么要怎么判断一个 w 是最优的呢？这就又要用到最大似然，或者说是代价函数了。

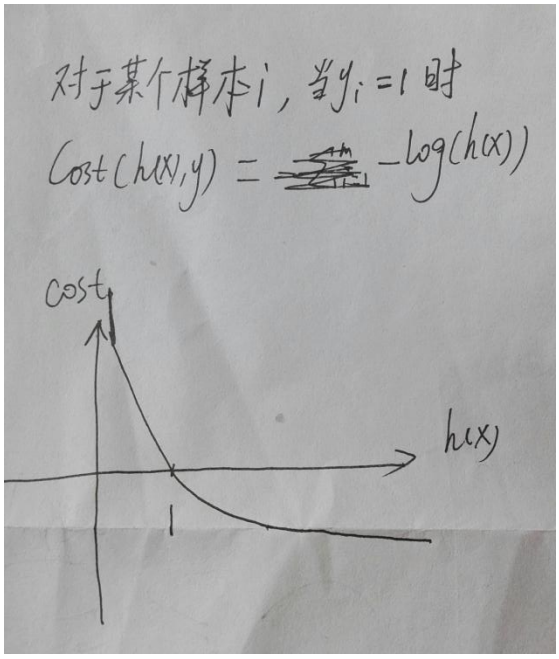
构建逻辑回归模型 f(θ)，最典型的构建方法便是应用最大似然估计。似然函数公式：

$$p(y \mid x, \theta) = (h_\theta(x))^y (1-h_\theta(x))^{1-y}$$
其中 $y = 1$(或0)

当标签 y=1 时 p=h(x),是 x 样本属于标签 1 的概率；当标签 y=2 时 p=1-h(x)，是 x 样本属于标签 0 的概率，因此 p 必是 x 样本属于 y 标签的概率，或者说 p 代表了准确度。在这个基础上，推导出代价函数：

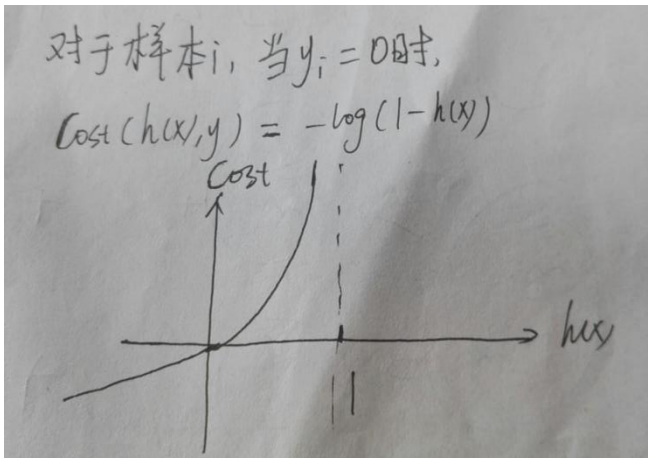
$$-log(likelihood) = -log\prod_{i=1}^{M} P(label|x_i)$$
$$= -\sum_{i=1}^{M} y_i log(h(x_i)) + (1-y_i)log(1-h(x_i))$$

先解释一下这个公式，当标签 y=1 时，此时的代价可表示为：



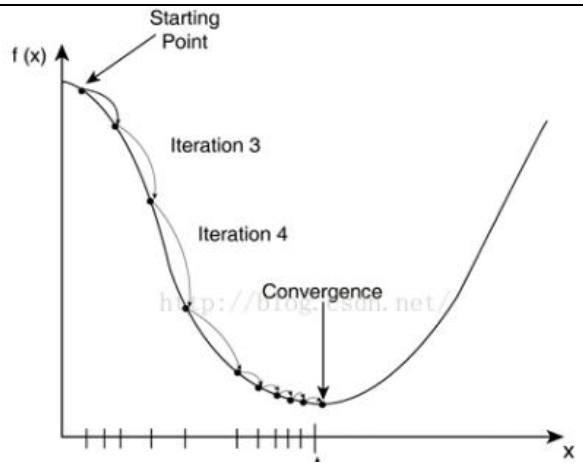
如果判定的 h(x)=1,即预测正确，则代价 cost=0，h(x)越小，代价越大，如果判定的 h(x)=0，即预测错误，则代价 cost 趋于无穷大。

同理当标签 y=0 时，cost 可表示为：



如果判定 h(x)=0，即预测正确则代价 cost=0，h(x)越大，代价越大，如果判定 h(x)=1 即预测错误，则代价 cost 趋于无穷大。

这个代价函数表示在坐标系上是一个凹函数，这样我们需要一步步靠近局部最底部，最终就能找出最优值



利用梯度下降法，通过对 w 的不断迭代，直至逼近最优解且收敛，有公式：

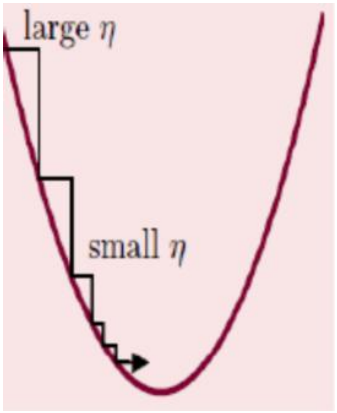
$$\tilde{\mathbf{W}}_{new}^{(j)} = \tilde{\mathbf{W}}^{(j)} - \eta \frac{\partial C(\tilde{\mathbf{W}})}{\partial \tilde{\mathbf{W}}^{(j)}}$$

$$= \tilde{\mathbf{W}}^{(j)} - \eta \sum_{i=1}^{n} \left[\left(\frac{e^{\tilde{\mathbf{W}}^{\mathrm{T}} \tilde{\mathbf{X}}_i}}{1 + e^{\tilde{\mathbf{W}}^{\mathrm{T}} \tilde{\mathbf{X}}_i}} - y_i \right) \tilde{\mathbf{X}}_i^{(j)} \right]$$

公式首先计算整个数据集的梯度（代价），然后使用 alpha（学习率） x gradient（梯度）来更新回归系数。

这里面 η 表示学习率，可以理解为逼近最优解过程中每一步的速度，如果把它设置为一个常量，则迭代将会匀速靠近最优解，实际上我一开始就是这样做的，这样做的问题是，如果 η 设置的比较大，则代价会极不稳定，达不到收敛效果，因为每次更新 w 时幅度太大，很有可能越过最优解，达不到优化效果；而如果 η 设置的比较小，虽然最后能达到收敛效果且收敛到最优解附近，但是耗时太长，如果数据集很大的话，这种将要花费极大的时间。

比较理想的方法是梯度下降中使用动态学习率，即需要随着迭代的进行，同时调整学习率，初始学习率较大，而随后逐渐减小，这样既能省时又能最终达到收敛效果。在实验报告的优化部分有展示动态学习率相比于恒定学习率的优点。



随机梯度下降：梯度下降算法一般指批量梯度下降，在每次更新回归系数的时候都需要遍历整个数据集（计算整个数据集的回归误差），计算出下降最快的方向，然后踏出一步，这属于一次迭代，更新一次 w 值，该方法对小数据集效果很好，但应用于大数据集，则它的计算复杂度太高。改进的方法是一次仅用一个样本点（的回归误差）来更新回归系数,这个方法叫随机梯度下降算法。它克服了批量梯度下降复杂度高的缺点，但是它也有缺点：就

是它每次迭代只选取一个样本进行分析，因此随机梯度下降的方向，会因为所选取的样本的具体不同而不同，从而在整体方向上会显得左右摇摆。这一点在优化部分也会有展示。

最后迭代到一定次数后，即可用这个训练好的 w 向量，根据最前面那条公式对测试集进行预测了。

2. 伪代码

读取文件、写入文件以及计算准确率等之前实验做过的或者过于简单的函数功能不列出。

```
int main()
{
    读取训练集和测试集，其中训练集每三个样本取前两个作为训练集，第三个作为验证集。
    初始化 w: for(int i=0;i<Length;i++) w[i]=1;
    for(int k=0:7)
        for(int i=0:traincnt)//遍历训练集样本
        {
            CalWeight(i);//计算样本 i 的权重分数
            CalCost(i);//每一维的梯度（代价）计算
            Updatew();  //更新 w
            if(i%20==0)  //每更新 w20 次计算一次准确率
            {
                Predict();//预测验证集样本
                Cal_acc();//计算准确率
                ac[cnt]=accuracy;
                cnt++;
            }
        }
    output_result();//输出验证集准确率以供调试
    output_test_result();//输出测试集预测结果
}
void CalWeight(int index){
    weight=当前向量 w 的转置*样本 i 向量;
}
void CalCost(int index){
    计算每一维的梯度，存储在向量数组 Cost[]中;
}
void Updatew(){
    使用 w= w - alpha x gradient 来更新回归系数（w）
}
void Predict(){
    P=1/(1+exp(-1* w^T *样本 i 向量);
    if(P>0.5) p_label=1;
    else p_label=0;
}
```

3. 关键代码截图（带注释）

main 函数

```
int main()
{
    Readtrain();//读取训练集，且其中训练集每三个样本取前两个作为训练集，第三个作为验证集
    Readtest();//读取测试集
    cout<<"Length="<<Length<<"   traincnt="<<traincnt<<"  valicnt"<<valicnt<<endl;
    for(int i=0;i<Length;i++) w[i]=1;//初始化w
    cnt=0;//记录迭代次数/20的数值
    for(int k=0;k<7;k++)
        for(int i=0;i<traincnt;i++)//遍历训练集样本
        {
            CalWeight(i);//计算样本i的权重分数
            CalCost(i);//每一维的梯度（代价）计算
            Updatew();  //更新w
            if(i%20==0) //每更新w20次计算一次准确率
            {
                Predict();//预测验证集样本
                Cal_acc();//计算准确率
                ac[cnt]=accuracy;
                cout<<"Accuracy:"<<accuracy<<endl;
                cnt++;
            }
        }
    output_result();//输出验证集准确率以供调试
    Predict_test();
    output_test_result();//输出测试集预测结果
    cout<<"pos"<<test_pos<<" neg"<<test_neg<<" testcnt"<<testcnt<<endl;
}
```

迭代和梯度下降过程用到的三个函数：

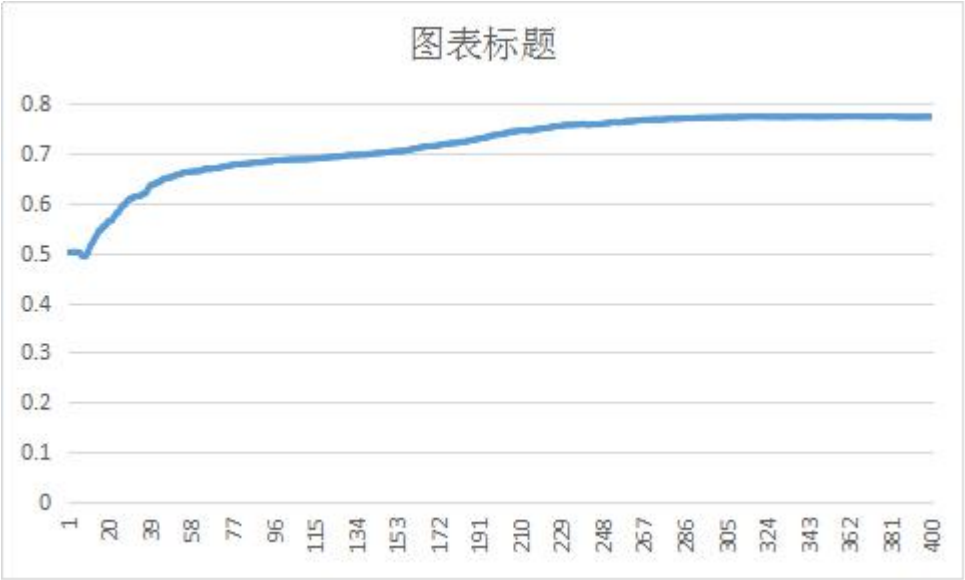
```
void CalWeight(int index)//计算样本i的权重分数
{
    weight=0;
    for(int j=0;j<Length;j++)
        weight+=w[j]*train[index][j];
}
void CalCost(int index)//每一维的梯度（代价）计算
{
    for(int i=0;i<Length;i++)
        Cost[i]=((1/(1+exp(-1*weight)))-label[index])*train[index][i];
}
void Updatew()//更新w
{
    double alpha=0.001;
//  alpha=0.001+1/(1+(cnt+1));//cnt=500 时0.0005最好1/2000，即乘以一个 4
    for(int i=0;i<Length;i++)
        w[i]=w[i]-alpha*Cost[i];
}
```

预测验证集样本的函数，预测测试集同理：

```
void Predict()//预测验证集样本
{
    double P;
    for(int i=0;i<valicnt;i++)
    {
        double sum=0;
        for(int j=0;j<Length;j++){
            sum+=vali[i][j]*w[j];
        }
        P=1/(1+exp(-1*sum));
        if(P>=0.5&&P<=1) p_label[i]=1;
        else if(P<0.5&&P>=0) p_label[i]=0;
    }
}
```

4. 创新点&优化（如果有）

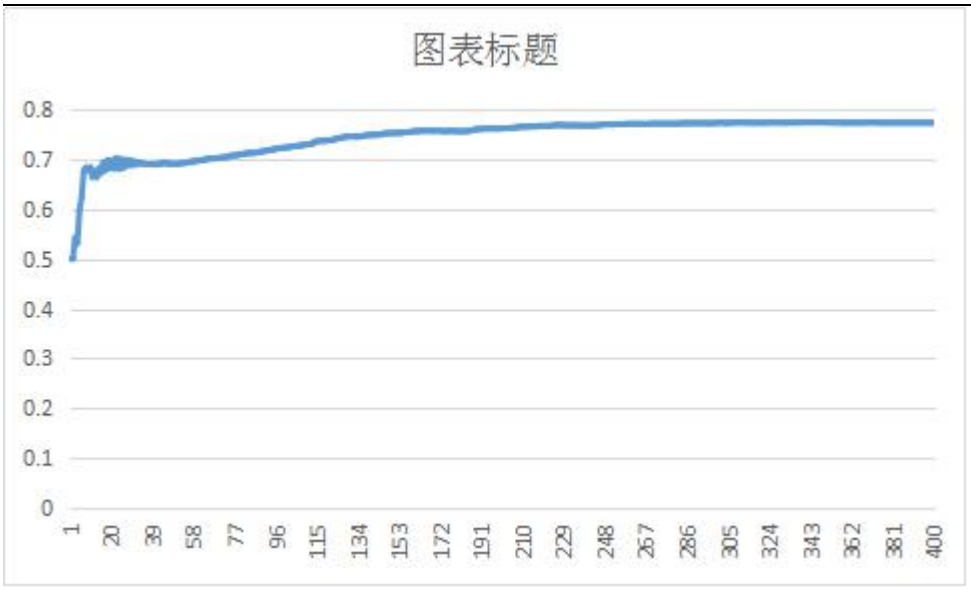
①使用批梯度下降和固定的学习率所得到的迭代过程中准确率的变化曲线，可见在迭代次数约 cnt=300 处收敛到 0.78：



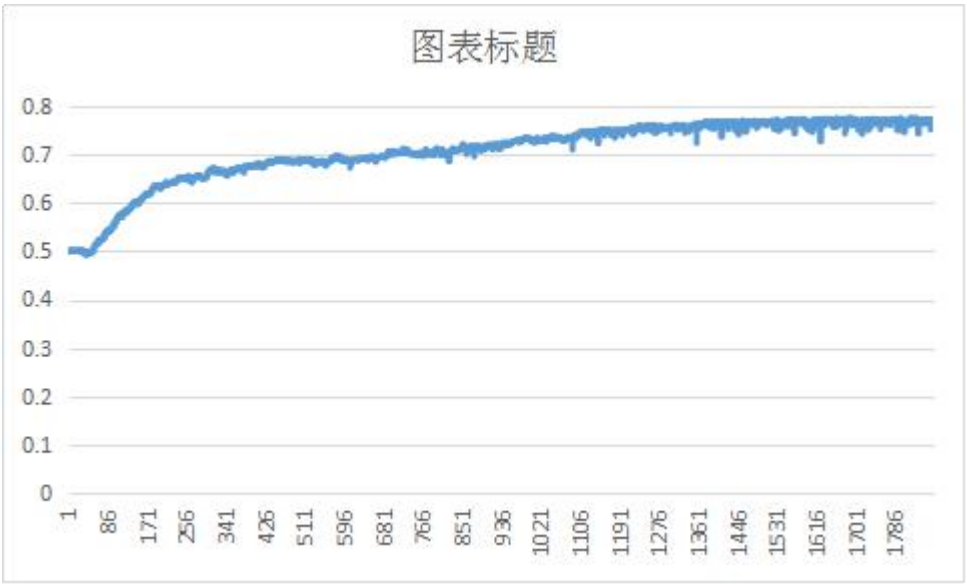
②使用批梯度下降和动态学习率：

```
void Updatew(){
    double alpha=0.1;
    alpha=0.01+1/(1+0.1*(cnt+1));
    for(int i=0;i<Length;i++)
        w[i]=w[i]-alpha*Cost[i]/5000;
}
```

得到的迭代过程中准确率的变化曲线如下，迭代过程前面梯度下降幅度大，然后学习率慢慢减小，下降幅度也变小，由图可见在迭代次数约 cnt=220 处收敛到 0.78，证明动态学习率能在保持最终收敛的前提下加快收敛速度。



③使用随机梯度下降和固定学习率，所得到的迭代过程中准确率的变化曲线如下，可见约在 cnt=1300 处收敛于 0.78.



随机梯度下降对比于批梯度的优点：

在批梯度下降中，每迭代一次，cnt++，因此 cnt 为迭代次数，在随机梯度下降中，由于迭代次数过多且为了使效果更太明显，我每迭代 20 次才 cnt++，因此两个算法各自到达收敛过程中的运算次数：

批梯度：300*5000=1500000=150 万

随机梯度：1400*20=28000=2 万 8 千

可见随机梯度比批梯度快得多。

但相应的，随机梯度也有它的缺点，由图可见它并不稳定，迭代过程中一直在震荡。

三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

训练集：

$$\mathbf{x}_B = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad , \quad y_B = 0$$

$$\mathbf{x}_C = \begin{pmatrix} 3 \\ 3 \end{pmatrix} \quad , \quad y_C = 1$$

$$\mathbf{x}_D = \begin{pmatrix} 4 \\ 3 \end{pmatrix} \quad , \quad y_D = 1$$

```
1  3,3,1
2  4,3,1
3  1,1,0
```

①只迭代一次，初始 w：{-3，1,1}，步长为,0.1：

```
weight0:3
weight1:4
weight2:-1
w:-3.02035 0.994528 0.992729
weight0:2.94142
weight1:3.93595
weight2:-1.0331
```

与结果文件数据一致：

增广特征向量	增广权向量的转置乘以增广特征向量
1	3
3	
3	
1	4
4	
3	
1	-1
1	
1	

步长为0.1更新后的增广权向量：

-3.02
0.99
0.99

增广特征向量	增广权向量的转置乘以增广特征向量
1	2.92
3	
3	
1	3.91
4	
3	
1	-1.04
1	
1	

②初始 w：{-3，1,1}，步长为 1：

```
weight0:3
weight1:4
weight2:-1
w:-3.20353 0.945281 0.927295
weight0:2.4142
weight1:3.35948
weight2:-1.33095
```

结果一致

步长为1更新后的增广权向量：		增广特征向量	增广权向量的转置乘以增广特征向量
-3.2	A	1	2.44
0.95		3	
0.93		3	
	B	1	3.39
		4	
		3	
	C	1	-1.32
		1	
		1	

③初始 w：{-3，1,1}，步长为 10：

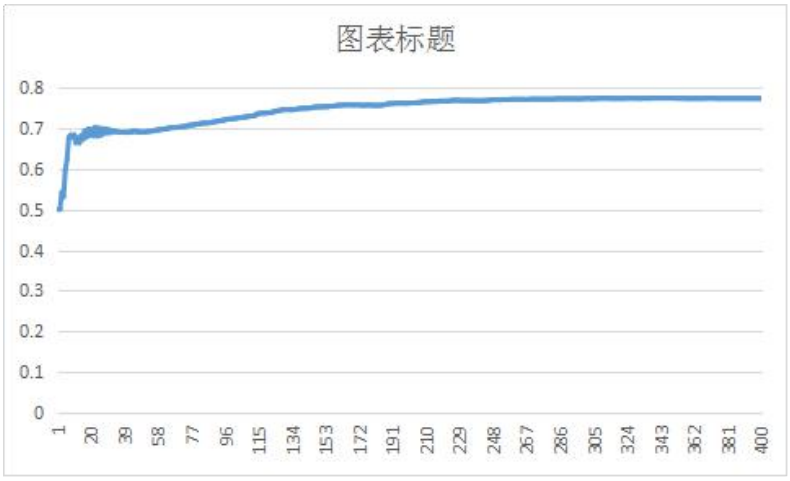
```
weight0:3
weight1:4
weight2:-1
w:-5.03529 0.45281 0.272948
weight0:-2.85802
weight1:-2.40521
weight2:-4.30953
```

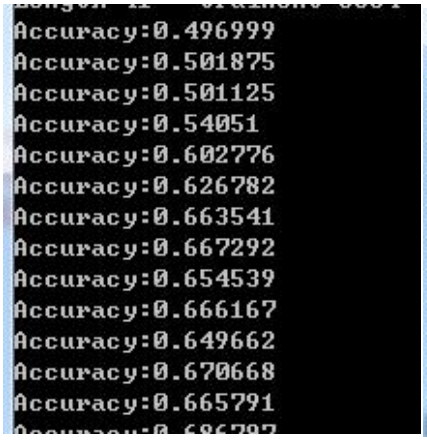
结果一致

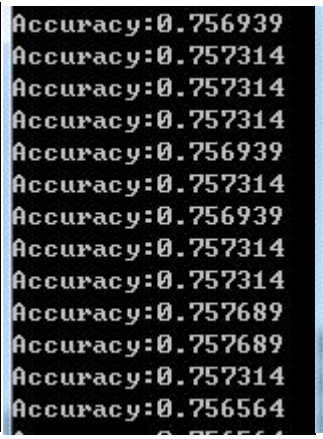
```
步长为10更新后的增广权向量：          增广特征向量    增广权向量的转置乘以增广特征向量
   -5.04                    A                  1    -2.86
    0.45                                       3
    0.27                                       3

                            B                  1    -2.41
                                               4
                                               3

                            C                  1    -4.31
                                               1
                                               1
```

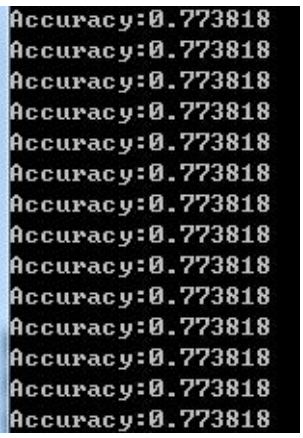
2. 评测指标展示即分析（如果实验题目有特殊要求，否则使用准确率）

读取 train 文件时每三个样本，前两个读入训练集，第三个读入验证集；使用批梯度下降方法和动态学习率，下图是迭代过程中用模型预测验证集时准确率的曲线变化：









上面三个图分别是迭代开始（左）、迭代过程中（中）、迭代结束（右）时的准确率，可见前两个图中，准确率一直在提高，说明梯度一直在下降，且迭代刚开始时准确率提升幅度很大而迭代中途提升幅度已经减慢了很多，说明实现了动态学习率方法，而在最后一个图中，准确率已经不再变化，稳定在 0.7738，说明已经到达收敛，梯度下降已经到达最优解。

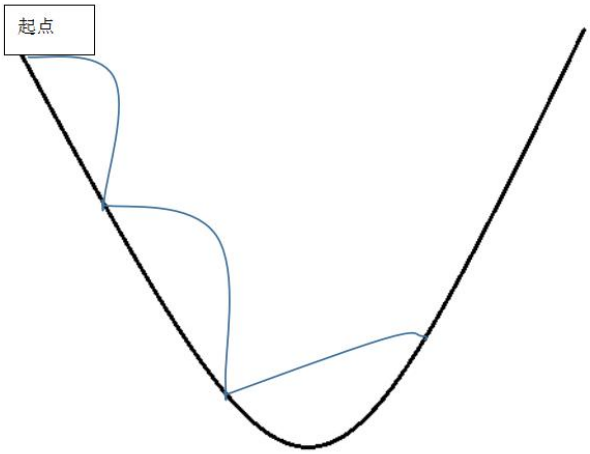
四、 思考题

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？

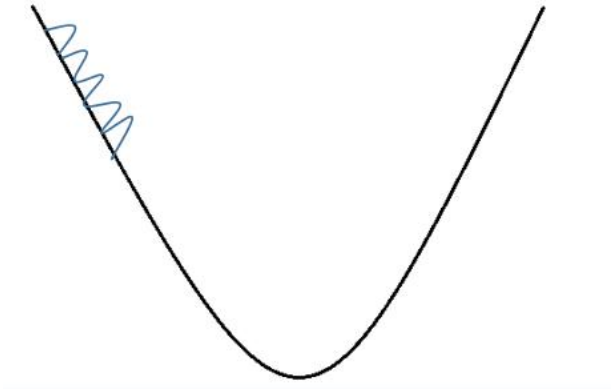
梯度为 0 即代价为 0，此时所有验证集样本预测都正确，这几乎是不可能的，算法将无法停止，即使达到这个正确率百分百的结果，也会出现过拟合的问题，模型与训练集验证集拟合程度过高，无法应用于实际样本。

2、 η 的大小会怎样影响梯度下降的结果？给出具体可视化解释。

如果 η 设置的比较大，则代价会极不稳定，达不到收敛效果，因为每次更新 w 时幅度太大，很有可能越过最优解，达不到优化效果：



而如果 η 设置的比较小，虽然最后能达到收敛效果且收敛到最优解附近，但是耗时太长，如果数据集很大的话，这种将要花费极大的时间：



3、思考批梯度下降和随机梯度下降两种优化方法的优缺点

批梯度下降算法，每次迭代考虑全部样本，选出最优的下降方向更新 w，这样能保证朝着最优解迈进，但是只适合用于较小训练集，如果数据集太大，由于每次迭代都要遍历整个数据

集找最优，因此该算法的复杂度非常高。

随机梯度下降算法(stochastic gradient descent)。随机梯度下降算法，每次迭代只是考虑让该样本点下降速度最快，而不管其他的样本点，这样算法复杂度不高，但是收敛的过程会有震荡，整体效果上，大多数时候它只能接近局部最优解，而无法真正达到局部最优解。所以适合用于较大训练集的 case。

|---------- 如有优化，重复 1，2 步的展示，分析优化后结果 ----------|

PS：可以自己设计报告模板，但是内容必须包括上述的几个部分，不需要写实验感想