机器学习课程

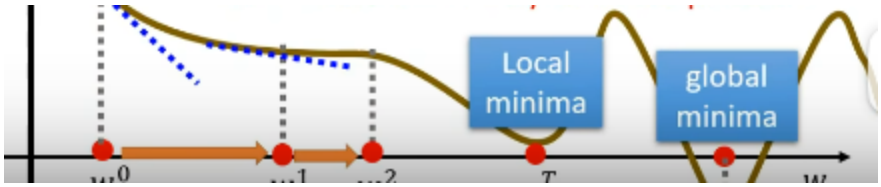
7.4日进度

机器学习初识

1. 机器学习目标？
    i. 说白了就是人类无法描述清楚的公式，让机器来描述
    ii. 一段语音，一张照片，如何识别出文字？如何识别出图片内容？复杂的表达式人无法写出，让机器来构成
2. 任务有什么？
    i. regression(The function outputs a scale)：使用一个函数，输出一个范围，例如给定一些天气参数，预测出明天pm2.5的数值
    ii. classification(The function outputs a choose)：给定一些选项，输出一个确定的值
    iii. structure learning：让机器输出结构化的东西，也就是创造东西
3. 什么是model？
    i. 对于一个任务，例如预测pm2.5数值，先猜测函数，例如是

    ```
    y = b + wx
    ```

    x：今日pm2.5数值 ，y：明日pm2.5数值，w作为拟合系数(weight)，b为修正值(bias)，这个函数就叫做一个model
    ii. 随后定义一个Loss函数，这个函数的参数就来自上一步骤中的w和b，作用是评价这两个值作为最终结果的话，准不准确？能不能有效预测？用真实的值(label)减去预测的结果，得到每个数据之间的差距，再求和除以个数就是一个简单的Loss值，根据不同wb组合可以得到不同的拟合情况，叫做error surface
    iii. Optimization(最佳化)
        a. 利用gradient descent方法，假设把某一个参数掩盖，例如b，然后随即选取w，并计算w对于Loss的微分，微分为正的时候，就减小w，微分为负的时候就增大w
        b. 同时引入N作为learning rate，N和微分结果同时影响w的变化，微分决定方向，N决定变化的速度，N越大，每次w的改变就越大，反之则越小
        c. 什么时候停止更新参数？当参数变化次数达到自设的值，或者微分值为0时停止，但是可能会出现下述情况，当前最优解算作Local minima，全局最优解作为global minima，还没达到最好的情况



iv. 以上这个就叫做训练，将其迁移到多参数上就是考虑多个微分值，然后每个参数进行变化，当一次训练结束后，发现拟合效果并不好，那么根据自己的数据就可以对函数进行更改

$y = b + wx_1$ 2017 - 2020 $L = 0.48k$ 2021 $L' = 0.58k$

$y = b + \sum_{j=1}^{7} w_j x_j$ 2017 - 2020 $L = 0.38k$ 2021 $L' = 0.49k$

b	w_1^*	w_2^*	w_3^*	w_4^*	w_5^*	w_6^*	w_7^*
0.05k	0.79	-0.31	0.12	-0.01	-0.10	0.30	0.18

$y = b + \sum_{j=1}^{28} w_j x_j$ 2017 - 2020 $L = 0.33k$ 2021 $L' = 0.46k$

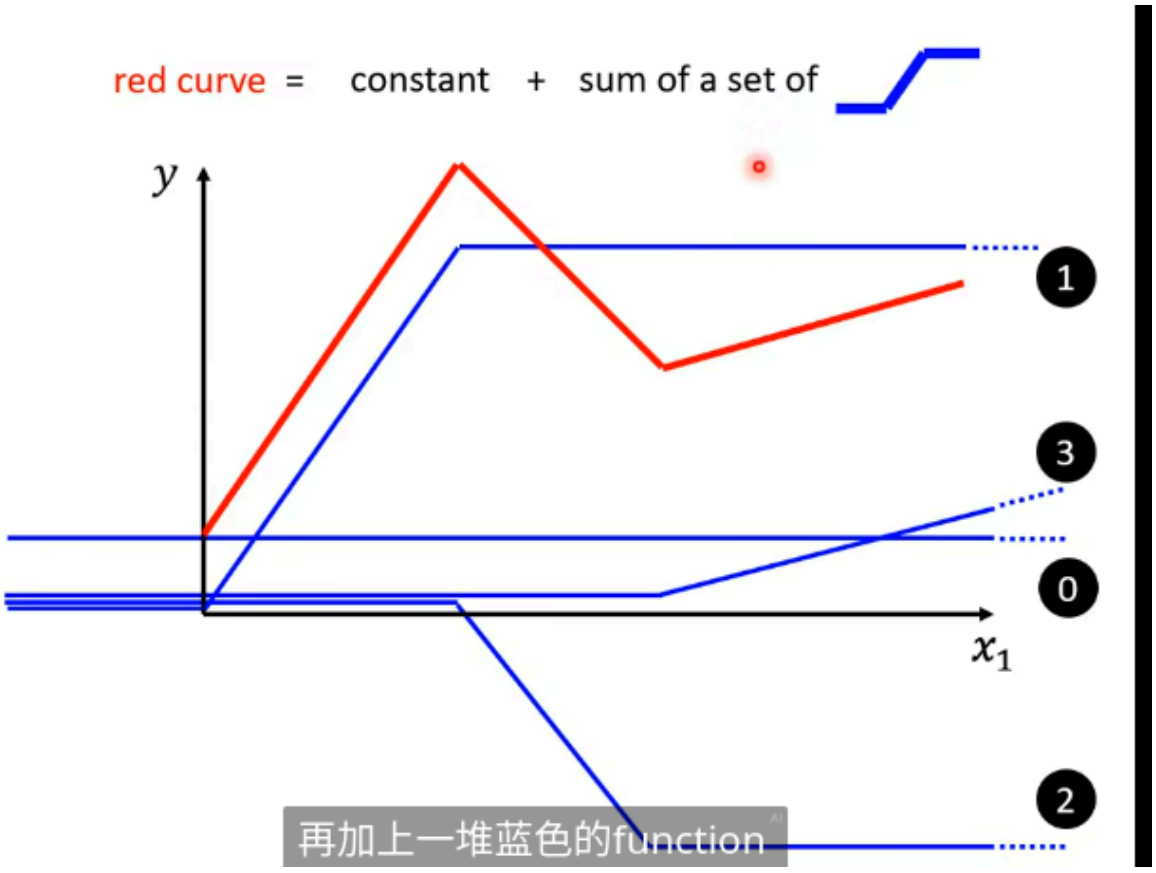
$y = b + \sum_{j=1}^{56} w_j x_j$ 2017 - 2020 $L = 0.32k$ 2021 $L' = 0.46k$

7.5进度

机器学习

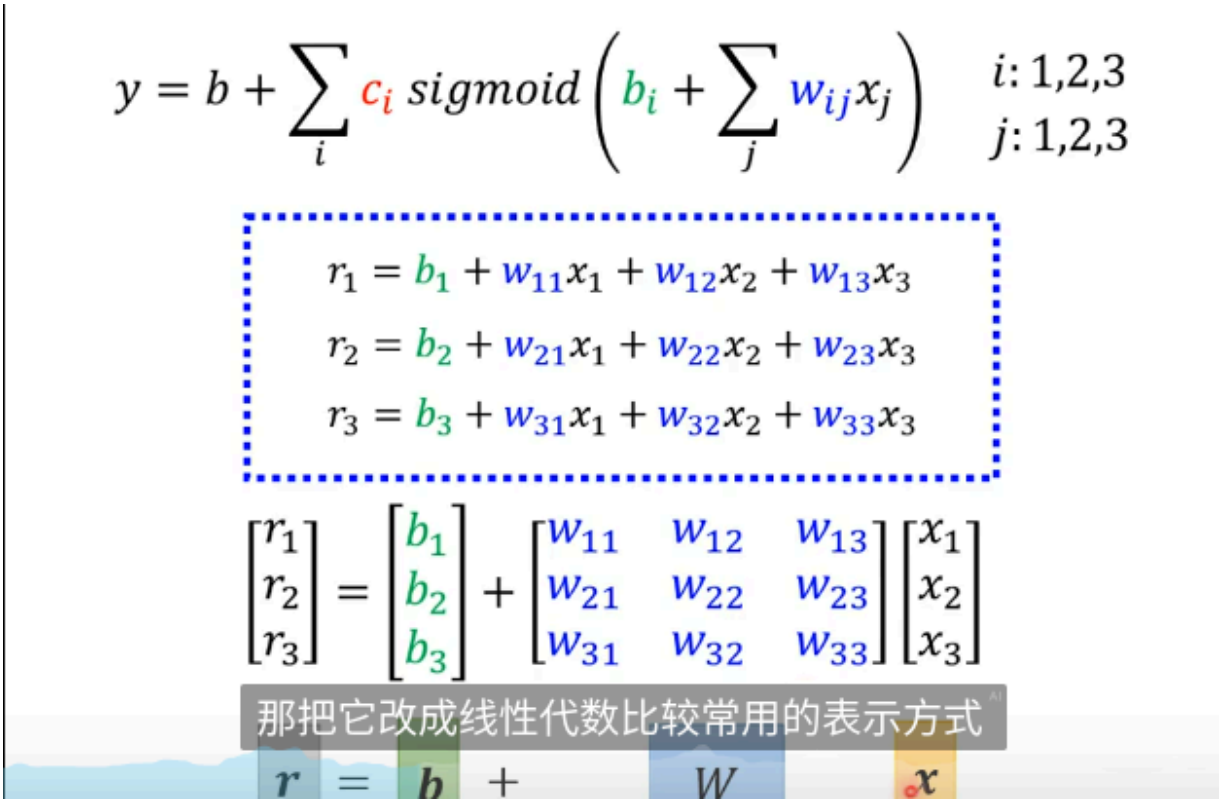
model深入

1. 普通的linear函数根本无法概括真实曲线
   i. 将曲线结合起来，通过一系列函数将其结合起来，通过判断条件，决定何时采用某个函数结合，理论上任何连续曲线只要点选的足够多，都可以转换成piecewise linear curve，也就是被直线表示出来



ii. 利用sigmoid可以进行曲线的模拟，从而得到蓝色曲线的表达式，sigmoid对初始的linear curve取e的负指数，再加上1作为分母，以1为分子并乘系数C得到最后模拟曲线，改变w和b的值可以得到各种各样的曲线

当wx变成wx的和时，拟合曲线的表达式就可以进行线性代数的转变，转为矩阵的乘法



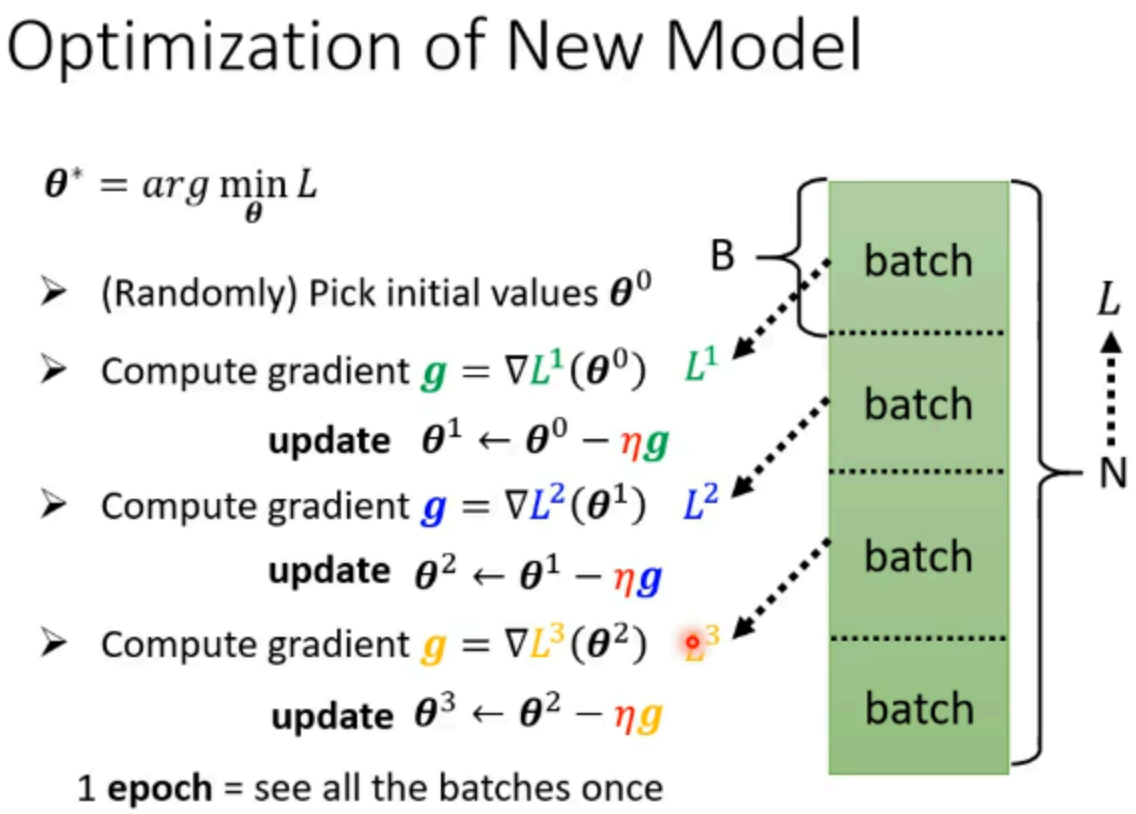
Latex学习

1. latex所有命令以反斜线\开头
2. 一篇文档主要包括：
   i. \documentclass[UTF-8]{文档类型}，文档类型：article|book|ctexart(中英文混合排版)
   ii. \title{} \author{} \date{\today}
   iii. \begin{document} \end{document}
3. 常用标记：
   i. \textbf{加粗内容}
   ii. \textit{斜体内容}
   iii. \underline{下划线内容}
   iv. 两个换行符表示新的一段，一个换行符表示空格
   v. 章节标题用\section{内容}，子章节用\subsection{}\subsubsection{}
   vi. 引用图片\usepackage{graphicx}，使用图片\includegraphics[width=0.5\textwidth]{file}，\capture{图片标题}\centering居中命令
   vii. 列表使用\begin{itemize} \end{itemize}，或者enumerate，每一项应用命令\item 内容

7.6进度

机器学习

1. 将所有的未知参数均使用线性代数中的向量运算替代，$f = b + w * x$转化为$y = b + c^T * \sigma\left(\overrightarrow{b} + \overrightarrow{w} * \overrightarrow{x}\right)$，将该式子转化为一个参数进行代替$\theta$
2. 上述将model从一个单一参数迁移到了两个参数，随后进行Loss函数的计算，同单一参数一致，将预测的结果同真实结果$\hat{y}$计算差值，得到Loss表达式$L = \frac{1}{N}\sum_{n=1}^{N} e_n$，分别用$L$对每个参数进行偏微分，求得其变化率，最后得到的是gradient，调整到较好的一个状态
3. 三步走：
   i. 选择合适的linear模型，可能是随机的
   ii. 设计一个较为贴近的Loss函数
   iii. 进行最优化
4. 参数？
   i. 当计算量非常大的时候，如果直接用全部的数据去更新Loss函数的下降，会导致计算量激增，所以将数据分为一份一份的，每份叫做一个$batch$，每个$batch$的大小叫做$batch_size$，Loss函数的下降就是分开用每一个batch的数据进行更新，当前$batch$更新结束得到g^1，下一个$batch$更新得到g^2，每一次的更新叫做一个$update$，所有$batch$更新一次过后，叫做一个$epoch$
   hypeparamter指由用户定义的参数，上述参数以及之前的learning rate和sigmoid数量也属于hypeparamter
   这就是模型的部分超参数含义



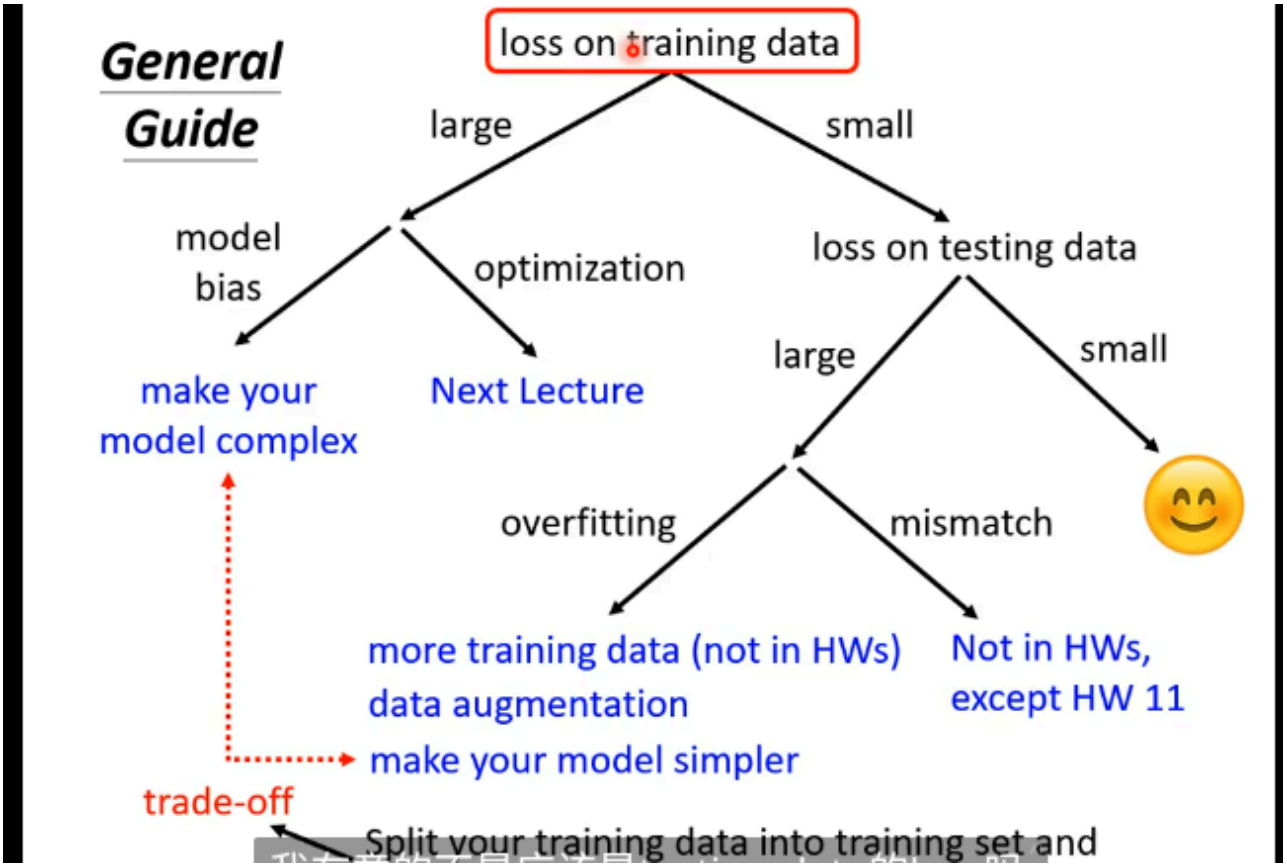
ii. $ReLU$与$Sigmoid$属于同一类参数，称为$Activation\ function$，前者的函数重叠表达式是$y = c * max(0, b + w * x)$，选取比0大的那一段函数，并且一个$Sigmoid$就需要两个$ReLU$来进行替换，函数如下$y = b + \sum_t c_i max(0, b + \overrightarrow{w} * \overrightarrow{x})$，对所有参数进行第一次处理之后，产生了$a$，为了学习更多特征，就把计算结果当作新的参数，再次进行更新，产生a'，这样可以不停的重复学习，迭代的次数就叫做$Layer$，整个迭代过程就叫做$deep\ learning$(7.13日补充)

iii. 当不停的添加layer之后，发现随着损失函数的下降，准确率反而降低了，这个现象就叫做模型的过拟合$over\ fitting$

7.14进度

理论学习

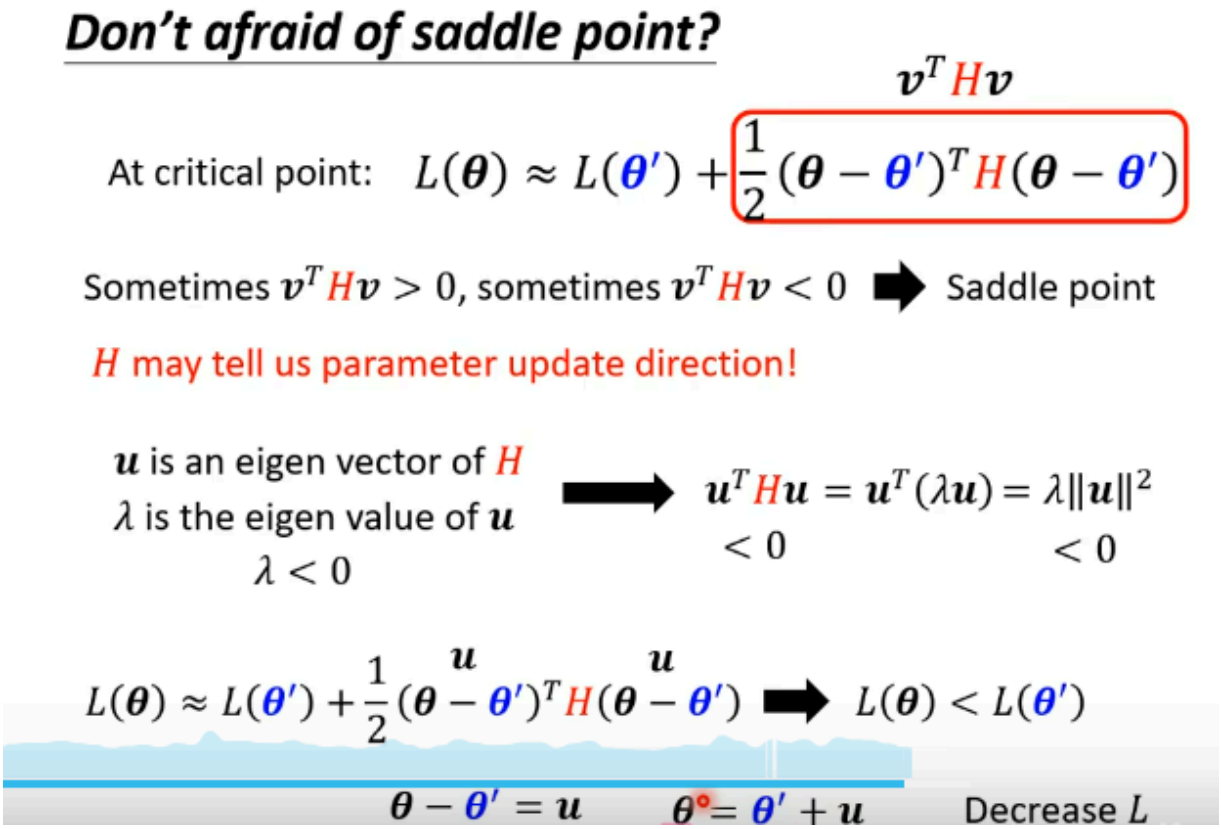
1. 模型表现结果不如人意的时候，如何进行调整？



2. 如何确定某个函数是不是自己需要的model？先从一些简单的函数实验，也许就会有表现较好的情况，Loss会比较低，然后再去用更深层的deep learning模型，如果发现复杂模型比简单模型的Loss反而更高，就说明最优化方法有问题，没有找到最佳参数，如果发现Loss确实降低了，就说明model bias不够，那继续增加模型的参数，还会带来更好的表现
3. 当训练数据上的Loss确实降低到一个很小的数值之后，再去看测试数据上的Loss表现如何，如果表现的比训练数据大很多，有可能是过拟合或者，如果同样也很小，就说明这个模型比较符合了
4. 过拟合出现的原因是数据对于模型的约束太小，导致模型虽然在某些数据上表现不错，但是一但模型遇到新的未知数据，就会偏离很多，所以最简单的解决过拟合的方法就是增加训练数据
    i. 训练数据有限时，可以采用$Data\ agumentation$，在图像领域就可以把图片进行左右反转，或者截取一部分图片，作为新的数据
    ii. 约束模型的参数，假定他的损失函数一定是个二次函数，这样就会限制很大的选择，所以模型就不会胡乱表达
    iii. 如何避免使用了测试数据进行训练从而导致某些测试数据非常准确，但是使用到位置的数据上就会表现很差？答案是把训练数据切分成训练集和验证集($validation\ set$)，在训练集上训练完成后，先用验证集验证一下，再去测试集上进行测试，这样最终结果可能会更符合一点

a. 切分方法：N-fold Cross validation，将数据切分成n等份，第一次先用最后一份作为validation set，第二次用倒数第二份作为validation，如此重复即可

5. gradient descent方法中，导致gradient为0的点称为$critical\ point$，包括局部最优点和马鞍点，如何判断是最优点还是说$saddle\ point$呢？在该点进行泰勒展开，因为gradient也就是一阶偏导为0，所以泰勒展开就抹去了第二项，这样继续计算，如果后面的一项永远都大于0，那么说明这个点附近的其他点都比他大，这就算最优点，如果都比他小，说明这不是最优点，如果有大有小，那么就不属于任何的点
   i. 以函数$y = w_1 w_2 x_i$为例，选取$L = (1 - w_1 w_2)^2$作为损失函数，现在有一个点是critical point，需要判断到底是local min还是saddle point
   ii. 先计算一阶偏导，$\frac{\partial L}{\partial w_1} = 2\ (1 - w_1 w_2)\ \ (-w_2)$，$\frac{\partial L}{\partial w_2} = 2\ (1 - w_1 w_2)\ \ (-w_1)$，因为critical point一阶导都为0，所以就需要继续求二阶偏导，$\frac{\partial L^2}{\partial w_1^2} = 2\ (-w_2)\ \ (-w_2)$，$\frac{\partial L^2}{\partial w_1 \partial w_2} = -2 + 4 w_1 w_2$，$\frac{\partial L^2}{\partial w_2 \partial w_1} = -2 + 4 w_1 w_2$，$\frac{\partial L^2}{\partial w_2^2} = 2(-w_1)(-w_1)$
   iii. 取参数都为0，得最后的值为0,-2,0,-2，该矩阵特征值为2，-2



训练过程

1. Batch，每次更新参数时的计算单位，每个batch计算一个gradient，所有batch轮过一次成为一个epoch，当batch_size=N（数据总量）时，参数的迭代比较慢但是会比较准确，当

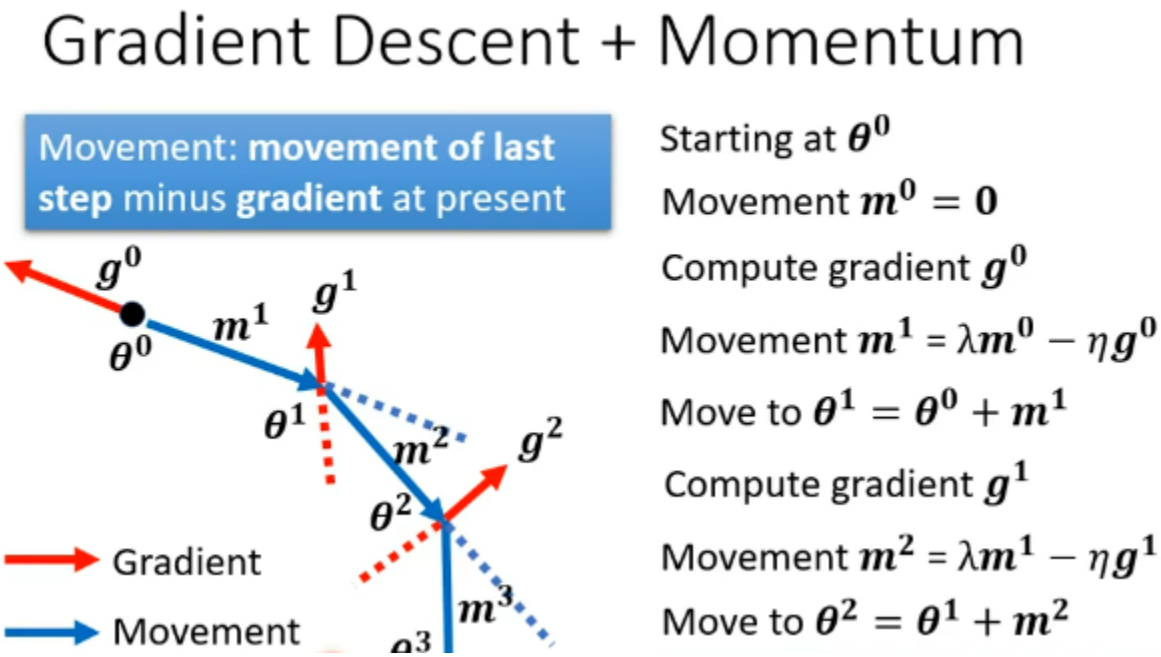
batch_size=1时，参数迭代的会非常快，但是每次迭代的变化比较小，但现在GPU训练都会进行平行计算，就导致大的batch_size和小的计算速度相差极小，而大的batch由于只经过少次计算年年，最后的critical point有可能并不是最好的，并且也许处于一个临界值，就是说左右的损失函数都比该点大，而且大很多，导致模型停留在一个"峡谷"里，这样模型的泛化能力可能表现的会很差

7.15进度

机器学习

应对critical point:

1. momentum:利用物理的思想，想象参数的下降和小球沿着斜坡滚落是一致的，这样子当小球滚落到一个critical point后，会因为惯性的原因继续往下滚动，能够跳出该点
2. 具体方法：在普通的gradient descent移动上添加新的向量，该向量代表上一次的移动方向



学习率调整:

1. 自动调整学习率:当固定使用一个学习率η时，在不同的位置变化都一致，导致当下降坡度较大时，参数更新也极大，使得直接越过最优点，当坡度极小时，有可能导致移动步幅过小，从而导致损失函数下降几乎不变，所以就需要让学习率自动调整，在坡度较大的地方采用较小的学习率，在坡度平缓的地方采用较大的学习率，这样可以保证损失函数的理想下降趋势

Root Mean Square $\theta_i^{t+1} \leftarrow \theta_i^t - \frac{\eta}{\sigma_i^t} g_i^t$

$$\theta_i^1 \leftarrow \theta_i^0 - \frac{\eta}{\sigma_i^0} g_i^0 \qquad \sigma_i^0 = \sqrt{\left(g_i^0\right)^2} = \left|g_i^0\right|$$

$$\theta_i^2 \leftarrow \theta_i^1 - \frac{\eta}{\sigma_i^1} g_i^1 \qquad \sigma_i^1 = \sqrt{\frac{1}{2}\left[\left(g_i^0\right)^2 + \left(g_i^1\right)^2\right]}$$

$$\theta_i^3 \leftarrow \theta_i^2 - \frac{\eta}{\sigma_i^2} g_i^2 \qquad \sigma_i^2 = \sqrt{\frac{1}{3}\left[\left(g_i^0\right)^2 + \left(g_i^1\right)^2 + \left(g_i^2\right)^2\right]}$$

$$\vdots$$

$$\theta_i^{t+1} \leftarrow \theta_i^t - \frac{\eta}{\sigma_i^t} g_i^t \qquad \sigma_i^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t}\left(g_i^t\right)^2}$$

6

2. RMSProp:与上述方法不同的地方在于，σ_i^t的计算中采用了加权平均，即$\sigma_i^t = \sqrt{\alpha(\sigma_i^t - 1)^2 + (1-\alpha)(g_i^t)^2}$，可以动态调整$\alpha$，从而能够在坡度不同时调整决定因素是$g$还是$\sigma$
3. 仅此调整还不够，还需要对η关联时间，让其随着时间变化，有两种调整策略，第一种是decay，也就是衰减，随着训练时间的增加逐步减小，第二种是warm up，最初先慢慢提高，再进行衰减，第二种方法可以让模型先进行一些特征的简单学习

7.16进度

分类classification

1. 任务是输入的值通过模型处理后，输出一个给定的标签列表中的某一个，关于标签分类问题，由于多数标签并不是完全独立的，可能标签之间互相影响，于是使用$one-hot\ vector$来进行标签的定义，几个标签就有几个向量，每个向量代表一个标签，输出时也需要多次进行计算，向量长度为n，模型就要输出n个y，这样才能组成向量$\overrightarrow{y}$
2. 获得向量结果后，还需要通过$Softmax$进行归一化，输入称为$logit$，输出产生结果与每个标签的相似度e，在计算相似度的时候，有两种方法
   i. 第一种是直接使用向量相减的平方和$Mean\ square\ error$，$e = \sum_i (\hat{y}_i - y_i^{"})^2$

ii. 第二种是$Cross-entropy$，计算为$e=-\sum_i \hat{y}_i lny_i^"$

吴恩达课程

机器学习算法分类

1. 监督学习（supervised learning）：提供x，输出y的映射算法，经典特征是用户提供正确答案，机器从中学习，从而能够给定一个输入，机器自己输出一个预测值
   i. $Regression\ algorithm$:预测结果来自于无穷无尽的集合中，例如房价的预测
   ii. $Classification\ algorithm$:预测结果在给定的选项中产生，结果是一个有限集合
2. 非监督学习(Unsupervised learning)：输入的数据不进行区分，而是由模型自己觉得如何分组，以及分组的类别有多少个。
   i. $clustring\ algorithm$：例如在一组肿瘤大小和良性情况数据集中，监督学习就必须要有标签类别，肿瘤的良性或恶性，无监督学习则不会区分恶性还是良性，他看到的数据都是年龄以及肿瘤大小，再让模型自己去区分标签，模型也许会将这一圈数据分为一组，那一圈分为另一组，这个是特殊的无监督学习
3. 训练集：training set包括feature和traget，通过把训练集提供给模型，模型的功能是根据x输出对应的预测值$\hat{y}$，以线性回归为例，函数被表示成$f(x)=wx+b$，通过决定w和b的值来为x提供预测功能，也成为$univariate\ linear\ regression$（单变量回归函数）

损失函数

1. 损失函数可以用来评估模型表现情况，对于最简单的单变量线性回归函数，如何判断某一对参数就是最适合当前情况的？答案是通过计算结果和预测结果之间的距离，也就是损失函数的来源$J=\frac{1}{2m}\sum(\hat{y}^i-y^i)^2$，对每次选择的参数计算损失后绘制损失函数曲线，能够得到其变化过程，一般损失最小的点所使用的参数是一个比较适合的选择。模型训练的过程其实就是最小化损失函数的过程
2. 梯度下降($Gradient\ descent$):最普遍的最小化损失函数的方法，在最小化过程中会很容易遇到局部最优点，再者参数更新的过程中，应该是更新一个在更新另一个还是同时更新？应该是同时更新，也就是说将更新后的值先赋值给一个中间变量，因为如果直接更新，那么其他参数在进行求导的时候，用到的数不再是旧值了，会影响最优化

7.17进度

吴恩达机器学习

回顾：

1. 对于简单线性回归函数，剃度下降的公式：$w = w - \alpha \frac{\partial J}{\partial w}$，其中$\alpha$是自己定义的学习率，且面对多个参数时，都需要先分别计算结果后再进行值的更新
2. 机器学习中的监督学习和非监督学习区别在于数据集的定义，监督学习给出了$result$，也就是问题的答案，模型学习如何预测结果，非监督学习并没有给出$result$，而是由模型自己区分数据间的差异性

学习：

1. 批量梯度下降：每次进行参数的更新时，需要计算所有的实例来考虑结果，相当于batch_size=N
2. 学习率α的选择：
   i. 过大，每次前进步幅越过收敛点，甚至会导致模型发散
   ii. 过小，模型长时间得不到收敛，或者收敛需要很长时间
   iii. 最好是能够根据模型当前所处情况进行自动调整，在较为陡峭的地方缩小，较为平缓的地方扩大
3. 多元线性回归：
   i. 参数多，能够共同影响模型结果，如果写成乘积的形式，随着参数变多，函数长度会逐渐增加，所以使用$\overrightarrow{w} = [w_1, w_2...]$来代表所有的参数，使用$\overrightarrow{x} = [x_1, x_2...]$来代表所有自变量，这样模型函数就可以写成向量的点积形式，$f_{w,b}(x) = \overrightarrow{w} \cdot \overrightarrow{x} + b$来表示，提高阅读效率
4. 向量化：
   i. 即使不使用向量，也能进行乘积的计算，对于手动写出每个乘积，当参数个数增加，代码会很冗长，不利于阅读，对于for循环而言，计算效率比较低

```
n = 4
w = np.array([1, 2, 3])
x = np.array([1, 2, 3])
b = 4
f = np.dot(w, x) + b
"手动列出"
f = w_1 * x_1 + w_2 * x_2 + b
"循环列出"
for j in range(n):
result = w[j] * x[j]
result += b
```

ii. np.dot的点积运算能够利用设备硬件进行并行计算，效率远比for循环高，所以向量化一方面能够简化代码，另一方面也能够提高运算速度

iii. 对于梯度下降$Gradient\ descent$，向量化也能够提高迭代速度，相比与每个参数求导更新，向量化后，能够同时处理所有参数的求导计算，随后对所有参数同时更改，大大提高了迭代速度

7.18进度

机器学习

一元线性回归函数：

1. 数据生成采用numpy，通过自己设定w,b的值生成对应的x和y，在计算y的时候添加噪声，使生成的数据不完全符合方程，模拟数据集

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(20)
def generate_data(w_weight, b_weight, n, noise, scale):
  x = np.random.uniform(-10, 10, n)
  y = w_weight * x + b_weight + np.random.normal(noise, scale, n)
  plt.scatter(x, y, label = "data")
  return x, y
```

2. 生成数据后，下一步应该就是定义模型的损失函数，以及初始化参数，np.mean()函数是一

个能够对numpy数组进行计算算术平均值的函数，也能够运用到高阶矩阵中

```
def loss_function(y_true, y_label, n):
  sum_loss = 0
  for i in range(n):
    sum_loss += np.pow((y_true[i] - y_label[i]), 2)
  return sum_loss / n
  """return np.mean(y_true - y_label) ** 2"""
w = np.random.randn()
b = np.random.randn()
n = 100
```

3. 接下来就是要进行梯度下降的迭代过程，要利用偏导对参数w，b进行更新，理论上可以利用torch的自动求导，但是在这里首先使用手写的方式，对w的偏导结果$\frac{\partial L}{\partial w} = \frac{1}{m}\sum(w * x + b - \hat{y}) * x$，对b的偏导结果$\frac{\partial L}{\partial b} = \frac{1}{m}\sum(w * x + b - \hat{y})$，然后分别更新w和b，直到损失函数的下降幅度小于给定的范围

```
def partial(w, b, x, y):
  w_sum = 0
  b_sum = 0
  for i in range(len(x)):
    w_sum += (w * x[i] + b - y[i]) * x[i]
    b_sum += (w * x[i] + b - y[i])
  return w_sum / len(x), b_sum / len(x)

def linear_model(epochs, x, y, n):
    w = np.random.randn()
    b = np.random.randn()
    a = 0.01
    loss_list = []
    pre_loss = float("inf")
    stop_scale = 1e-5
    for i in range(epochs):
        y_true = w * x + b
        loss = loss_function(y_true, y, n)
        if(abs(pre_loss - loss) < stop_scale):
            return w, b
        pre_loss = loss
        w_partial, b_partial= partial(w, b, x, y)
        w -= a * w_partial
        b -= a * b_partial
```

4. 训练完毕后，可以进行损失函数和拟合情况的可视化，能够更好的理解模型训练过程

```
def plot_loss(loss_list):
    plt.plot(loss_list, label = 'loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Loss over Epochs')
    plt.legend()
    plt.show()

def plot_model(x, y, w, b):
    plt.scatter(x, y, label = 'data')
    plt.plot(x, w * x + b, color = 'red', label = 'model')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.grid(True)
    plt.show()
```

matplotlib.pyplot as plt:

1. 常用函数：
    i. plt.scatter(value，label='example')绘制散点图
    ii. plt.plot(x, y, 'string',label='example')绘制直线
        a. x和y是横纵坐标
        b. string是格式化字符串，结构是color-marker-line
            a. color包括blue，red等等
            b. marker是指标记字符：
                a. `'.'`: point marker 点标记
                b. `','`: pixel marker 像素标记
                c. `'o'`: circle marker 圆形标记
                d. `'v'`: triangle_down marker 向下三角形标记
                e. `'^'`: triangle_up marker 向上三角形标记
                f. `'<'`: triangle_left marker 向左三角形标记
                g. `'>'`: triangle_right marker 向右三角形标记
                h. `'s'`: square marker 正方形标记
                i. `'p'`: pentagon marker 五边形标记
                j. `'*'`: star marker 星号标记
                k. `'h'`: hexagon1 marker 六边形标记1

l. `'H'`: hexagon2 marker 六边形标记2
m. `'+'`: plus marker 加号标记
n. `'x'`: x marker X 标记
o. `'D'`: diamond marker 菱形标记
p. `'d'`: thin_diamond marker 细菱形标记
q. `'|'`: vline marker 竖直线标记
r. `'_'`: hline marker 水平线标记

c. line是指线的形状:
a. `'-'`: solid line style 实线
b. `'--'`: dashed line style 虚线
c. `'-.'`: dash-dot line style 点划线
d. `':'`: dotted line style 点线

iii. plt.xlabel/ylabel()绘制横纵坐标
iv. plt.legend()显示图例
v. plt.grid(True)显示网格
vi. plt.title()设置标题

多元线性回归函数：

1. 针对多元问题，要搞清楚多元中的数据集组成是什么样的，特征权重的数量同自变量的数量保持一致，所以在数据生成上，需要完成多组数据的生成，绘制3D图形利用mpl_toolkits.mplot3d中的Axes3D，numpy.random的uniform函数，最后一个参数可以传入矩阵的行列数，甚至是更高维度的参数

```
def generate_data(w_weight, b_weight, n):
    """
    w:[1, 2, 3]
    x[1]:[x1, x2, x3]
    """
    x = np.random.uniform(-5, 5, (n, 3))
    y = np.dot(x, w_weight) + b_weight   + np.random.normal(0, 0.5, n)
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    sc = ax.scatter(x[:, 0],x[:, 1],x[:, 2],c = y, cmap = 'viridis')
    plt.colorbar(sc)
    plt.show()
    return x, y
```

2. 损失函数与一元回归保持相同
3. 模型迭代函数中，先设立初始的w参数和b参数，随后进行y的预测，再根据损失函数计算差值，随后对$\overrightarrow{w}$中每个参数进行偏导计算，较为复杂

```
def model_train(x, y, epochs, n):
  w = np.random.uniform(-1, 1, 3)
  b = np.random.randn()
  y_predict = np.dot(x, w) + b
  loss_list = []
  pre_loss = float(int)
  for i in range(epochs):
    loss = loss_function(y_predict, y)
    w_partial, b_partial = partial(w, b, x, y, n)
    w -= w_partial
    b -= b_partial
    if(abs(pre_loss - loss) < scale):
      break;
    loss_list.append(loss)
    pre_loss = loss
  return w, b, loss_list
```

4. 偏导的计算，首先要弄清楚每个参数的表达式，损失函数$L=\frac{1}{2n}\sum_{i=1}^{n}(w_1x_1^i+w_2x_2^i+w_3x_3^i+b-\hat{y})^2$，对于$w_1$，求得偏导结果为$\frac{\partial L}{\partial w_1}=\frac{1}{n}\sum_{i=1}^{n}(w_1x_1^i+w_2x_2^i+w_3x_3^i+b-\hat{y})x_1^i$，$w_2,w_3$同理

```
def partial(w, b, x, y, n):
  w_1 = 0
  w_2 = 0
  w_3 = 0
  b_partial = 0
  for i in range(n):
    error = np.dot(x[i],w) + b - y[i]
    w_1 += error * x[i][0]
    w_2 += error * x[i][1]
    w_3 += error * x[i][2]
    b_partial += error
  return [w_1 / n, w_2 / n, w_3 / n], b_partial / n
```

7.19日进度

机器学习课程

1. 特征缩放：对于房价预测，倘若房屋尺寸分布在80-100平米，卧室数量分布在1-4个，该如何确定参数的大小？如果房屋尺寸的参数过大，那么最终结果一定是偏大的数值，且卧室数量起不到影响作用，假如有参数为0.5和10，应当把0.5分配给房屋尺寸，10分配给卧室，这叫做特征缩放
2. 步骤：确定好每个数据的取值范围，例如$80 < x_1 < 100, 1 < x_2 < 4$，接下来就是对数据进行缩放，
   i. 第一种缩放操作要根据数据的上限，使用数据/最大值求得分布范围，$0.8 < x_{1,scale} < 1, 0.25 < x_2 < 1$
   ii. 第二种是均值归一化，使得数据分布在四个象限，首先计算出每个数据的均值η，随后采用公式$x_i = \frac{x_i - \eta}{max - min}$，对所有数据作处理
   iii. Z-score归一化，计算出均值和标准差η, σ，套入公式$x_i = \frac{x_i - \eta}{\sigma}$，最后就可以进行缩放了

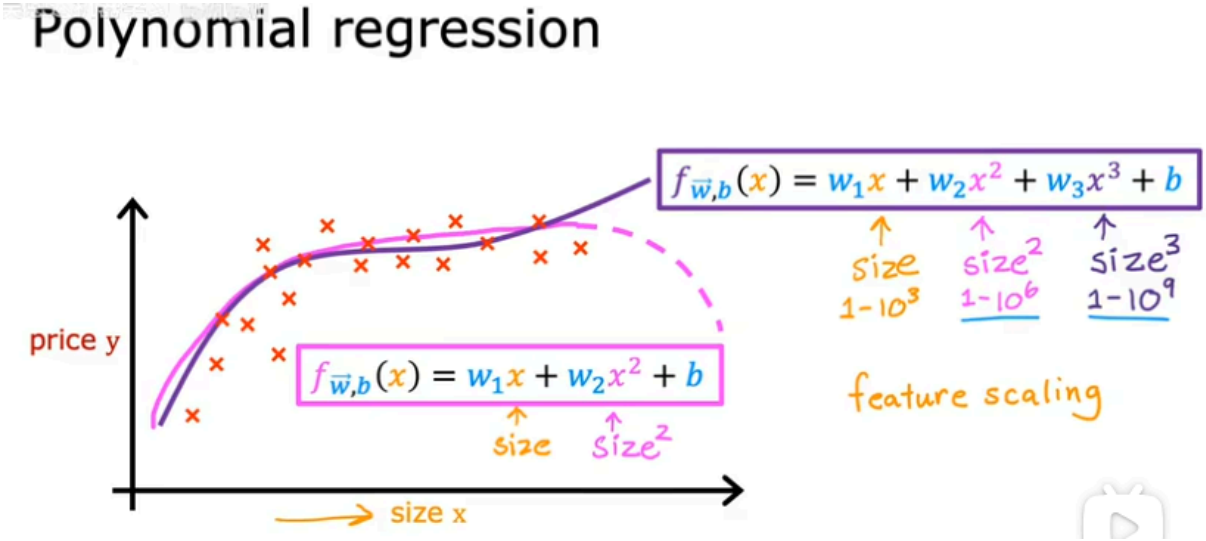
7.21日进度

机器学习课程

1. 学习率的选择：
   i. 过大，在收敛点附近来回振荡，甚至可能越来越远离收敛点；过小，又会收敛的速度非常慢。当遇到收敛函数不下降甚至增大的情况，这个时候就考虑是不是选择了过大的学习率，将学习率缩小到一个极小值，查看收敛函数是否下降，如果随着每次的迭代就减小，说明模型的代码没有问题，如果发现很小的学习率下，收敛函数还上升，就要考虑代码编写问题了
   ii. 如何找到比较适合的学习率：从0.001->0.01->0.1逐渐尝试，增大的过程中可以放大三倍再次尝试，0.001->0.003->0.01->0.03->0.1->0.3，基本就可以找到一个比较合适的学习率
2. 特征工程：
   i. 对于房屋预测，若有长和宽两个因素，第一种是直接作为两个特征处理，$f = w_1x_1 + w_2x_2 + b$，但是这样处理可能两个因素的联系不够紧密，或许面积更能表达房屋的大小，那么就设置出第三个特征x_1x_2，最终的f就变为$f = w_1x_1 + w_2x_2 +$

$w_3x_1x_2+b$，通过转换或者结合原始特征，让问题处理更加精准

3. 多项式回归：
   i. 结合多元线性回归和特征工程，提出了多项式回归方法，可以拟合曲线和非线性函数，但是这个对于数据的缩放要求也更高，不然会出现决定因素基本没有影响的情况(scikit-learn开源机器学习库)



4. 逻辑回归
   i. 不同于线性回归，逻辑回归预测的结果是类别，而不是无限的取值，无论多大的值，都可以转换为0-1的概率分布，依赖的是$Sigmoid$函数，对于值z，$g(z)=\frac{1}{1+e^{-z}}$，当z很小时，分母较大，输出的概率也就比较小，当z很大时，分母趋近于1，输出的概率也比较大，一种表示结果的描述是$f_{\vec{w},b}(\vec{x})=P(y=1|x;w,b)$，其中x代表输入的参数，分号后代表参数parameter，逻辑回归的拟合需要用到高阶多项式，尤其是图形极不规则时，或者内圆是一个类别，外圆是一个类别，就比较适合平方的和。
   ii. 决策边界是判定最终结果为哪个类别的边界值，例如当概率达到百分之70时，就是类别1，小于百分之70时，就是类别2，那么这个百分之70就算决策边界
   iii. 损失函数：如果仍然使用标准差来进行计算，损失函数的曲线会是跌宕的下降，如果使用梯度下降，很容易陷入局部最优点，导致模型表现不佳，于是采用以下的损失函数：L作为单个样本的损失函数，以下选取的函数来自于统计学中最大似然估计
      a. $L(f_{\vec{w},b}(x^i),y^i)=\begin{cases}-log(f_{\vec{w},b}(x_i)) & y^i=1\\ -log(1-f_{\vec{w},b}(x^i)) & y^i=0\end{cases}$，为什么这个L更适合作为逻辑回归的损失函数？
      b. 当$y^i=1$时，随着f的输出值越大，那么整体的L就越接近0，这就符合了类别为1时，预测的概率也应该更大，而且是越接近于1越好，当f很小时，说明只有很小的概率预测结果是1，而现在已知结果就是1，那么损失就很大
      c. 当$y^i=0$时，随着f的输出越小，那么预测为1的可能性就越小，就意味着大概率是0，所以损失比较小，反之损失就比较大。

d. 逻辑回归的损失函数，也要把所有样本的同步加起来，最后除以个数，就得到了最终的J

e. 简化版的损失函数：$L(f_{\overrightarrow{w},b}(x^i),y^i) = -log(1-f_{\overrightarrow{w},b}(x^i))\,y^i - log(f_{\overrightarrow{w},b}(x_i))(1-y^i)$

iv. 逻辑回归的梯度下降和线性回归表达一致，同样也可以进行特征缩放，向量化等等操作

5. 过拟合：

i. 泛化$generalization$：训练后的模型在新的数据集上表现也比较良好，甚至是运用到完全未接触的数据上来进行预测。

ii. 高偏差$high\ bias$：模型不能拟合数据集，或者说不能较好的拟合数据集

iii. 高方差$high\ variance$：模型虽然在数据集上的所有样本都表现很好，但是他表现的太好了，以至于尽管损失函数很低，但是模型可能非常杂乱，只要用一个新的数据，模型可能会预测的相差十万八千里。

7.22日进度

机器学习

1. 过拟合问题的解决：

i. 最简单的就是添加更多的训练数据，涵盖多方面

ii. 减少使用的特征数量，称为特征选择，选取最有用的特征子集来进行训练

iii. 正则化(Dropout)：相比于直接删除参数，正则化则是会减小参数的大小，使得其影响较小，同时又不损失特征的作用，并且一般只会缩小参数$\overrightarrow{w}$，不会对b进行更改

2. 手写逻辑回归:

i. 首先定义好生成数据的函数，包括数量，参数大小

```
def generate_data(w, b, n):
  x = np.array([np.random.uniform(50, 100, n), np.random.uniform(1, 3, n), np.rand
  x = np.array([x[0], np.array[[i*i for i in x[1]]], np.array[[x[1][i]*x[2][i] for
  y = np.dot(x.T, w) + b + np.random.normal(0, 5 ,n)
  y = 1 / (1 + np.exp(-y))
  for i in range(len(y)):
    if y[i] > 0.7:
        y[i] = 1
    else:
        y[i] = 0
  plt.scatter(x[0], y, label = 'x_1_y')
  plt.legend()
  plt.xlabel('x_1')
  plt.grid(True)
  plt.show()
  return x, y
```

ii. 其次编写逻辑回归的损失函数，$L_{\vec{w},b}\left(\vec{x}\right) = -y_i log(f_{\vec{w},b}(\vec{x})) - (1 - y_i)log(1 - f_{\vec{w},b}(\vec{x}))$，根据此方程进行计算

```
def loss_function(y_true, y_label):
    loss_sum = 0
    for i in range(len(y_true)):
        Loss = -y_label[i] * np.log(y_true[i]) - (1 - y_label[i]) * np.log(1 - y_t
        loss_sum += Loss
    return loss_sum / len(y_true)
```

iii. 随后编写梯度下降的训练过程

```
def train_model(x, y, epochs, n):
  w = np.random.uniform(-100, 100, 3)
  b = np.random.randn()
  loss_list = []
  pre_loss = float('inf')
  step = 1e-5
  for i in range(epochs):
      y_prob = np.dot(x.T, w) + b
      loss = loss_function(y_prob, y)
      if(abs(loss - pre_loss) < step):
        break
      loss_list.append(loss)
      w, b = update_para(w, b, x, y, y_prob)
      pre_loss = loss
  return loss_list
```

7.23日进度

机器学习

1. 接着继续完成逻辑回归的偏导计算$\frac{\partial L}{\partial w_1}=(f-y)x_1, f=\frac{1}{1+e^{-z}}, z=w_1x_1+w_2x_2^2+w_3x_2x_3+b$

```
def partial(x, y, y_prob):
    sum_1 = 0
    sum_2 = 0
    sum_3 = 0
    sum_b = 0
    y_exp = 1 / (1 + np.exp(-y_prob))
    for i in range(len(y)):
        sum_1 += (y_exp[i]-y[i]) * x[0][i]
        sum_2 += (y_exp[i]-y[i]) * x[1][i]
        sum_3 += (y_exp[i]-y[i]) * x[2][i]
        sum_b += (y_exp[i]-y[i])
    return np.array([sum_1 / len(x[0]), sum_2 / len(x[0]), sum_3 / len(x[0])]), sum_b
```

7.24进度

机器学习

1. generate_data()输出的y值基本全部都是1，这和sigmoid有原因，当$-5 < z < 5$时，可以比较好的处理概率问题，当z的绝对值超过5时，整个sigmoid函数基本都会趋近于1，所以使得结果没区别故需要进行特征缩放，采用z-score归一化

```
def generate_data(w, b, n):
  x = np.array([np.random.uniform(50, 100, n), np.random.uniform(1, 3, n), np.random.u
  x = np.array([x[0], [i*i for i in x[1]], [x[1][i]*x[2][i] for i in range(len(x[2]))]
  x_1_E = averge(x[0])
  x_2_E = averge(x[1])
  x_3_E = averge(x[2])
  x_1_d = de(x[0], x_1_E)
  x_2_d = de(x[1], x_2_E)
  x_3_d = de(x[2], x_3_E)
  for i in range(len(x)):
    for z in range(len(x[i])):
        if i == 0:
          x[i][z] = (x[i][z] - x_1_E) / x_1_d
        elif i == 1:
          x[i][z] = (x[i][z] - x_2_E) / x_2_d
        else:
          x[i][z] = (x[i][z] - x_3_E) / x_3_d
  y = np.dot(x.T, w) + b + np.random.normal(0, 5 ,n)
  y = 1 / (1 + np.exp(-y))
  for i in range(len(y)):
    if y[i] > 0.7:
        y[i] = 1
    else:
        y[i] = 0
  plt.scatter(x[0], y, label = 'x_1_y')
  plt.legend()
  plt.xlabel('x_1')
  plt.grid(True)
  plt.show()
  return x, y
```

2. 接着定义均值和方差函数

```
def averge(x):
    sum = 0
    for i in range(len(x)):
        sum += x[i]
    return sum / len(x)
def de(x, x_averge):
    sum = 0
    for i in range(len(x)):
        sum += (x[i] - x_averge) ** 2
    return np.sqrt(sum / len(x))
```

3. 正则化
    i. 当特征过多时，容易发生过拟合，比较大的参数会主导曲线，为了让模型拟合一个更加平滑的曲线，也更为了缩小部分特征对训练的影响，使用正则化，将参数添加到损失函数中$\frac{\lambda}{2m}\sum w_j^2$，$\lambda$代表了该项对于损失函数的影响程度，由自己决定，这样既可以追求损失的最小化，也可以追求参数的最小化
    ii. 线性回归的损失函数如下$w_j = w_j - \alpha[\frac{1}{m}\sum(f_{\vec{w},b}(x_i) - y_i) + \frac{\lambda}{m}w_j] = (1 - \alpha\frac{\lambda}{m})w_j - \frac{\alpha}{m}\sum(f_{\vec{w},b}(x_i) - y_i)$，可以看到每次的正则化部分其实就是让w乘一个稍微小一点的数，这样就能够起到缩小的作用
    iii. 逻辑回归同上