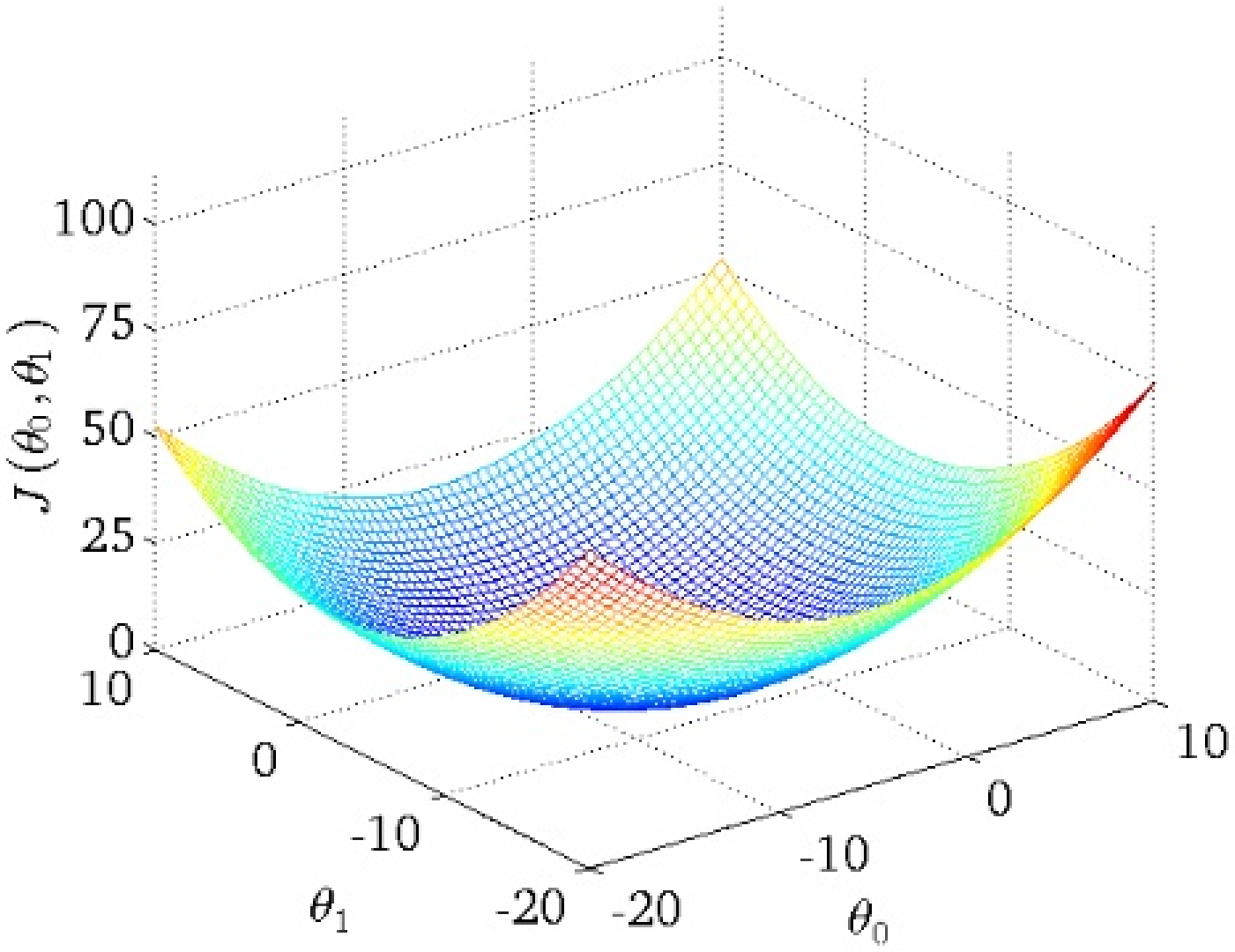


详解梯度下降法的三种形式BGD、SGD以及MBGD

忆臻
哈尔滨工业大学 计算机科学与技术博士在读

在应用机器学习算法时，我们通常采用梯度下降法来对采用的算法进行训练。其实，常用的梯度下降法还具体包含有三种不同的形式，它们也各自有着不同的优缺点。

下面我们以线性回归算法来对三种梯度下降法进行比较。

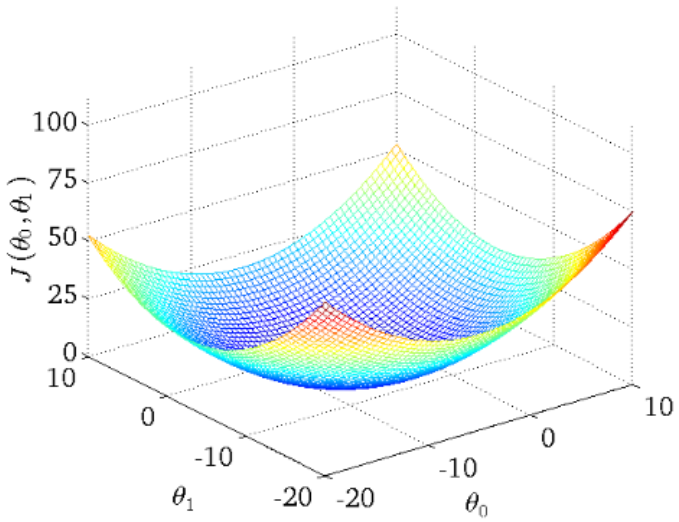
一般线性回归函数的假设函数为：

$$h_\theta = \sum_{j=0}^{n} \theta_j x_j$$

对应的损失函数为：

$$J_{train}(\theta) = 1/(2m)\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

（这里的1/2是为了后面求导计算方便）
下图作为一个二维参数（θ_0，θ_1）组对应能量函数的可视化图：



下面我们来分别讲解三种梯度下降法

批量梯度下降法BGD

我们的目的是要**误差函数尽可能的小**，即求解weights使误差函数尽可能小。首先，我们随机初始化weigths，然后**不断反复的更新weights使得误差函数减小，**直到满足要求时停止。这里更新算法我们选择梯度下降算法，利用初始化的weights并且反复更新weights：

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$$

这里α代表学习率，表示**每次向着J最陡峭的方向迈步的大小**。为了更新weights，**我们需要求出函数J的偏导数。首先当我们只有一个数据点（x,y）的时候，J的偏导数是：**

$$\begin{aligned}\frac{\partial}{\partial\theta_j}J(\theta) &= \frac{\partial}{\partial\theta_j}\frac{1}{2}(h_\theta(x)-y)^2 \\ &= 2\cdot\frac{1}{2}(h_\theta(x)-y)\cdot\frac{\partial}{\partial\theta_j}(h_\theta(x)-y) \\ &= (h_\theta(x)-y)\cdot\frac{\partial}{\partial\theta_j}\left(\sum_{i=0}^{n}\theta_i x_i - y\right)\end{aligned}$$

则对**所有数据点，**上述损失函数的偏导（**累和**）为：

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m}\sum_{i=1}^{m}(y^i - h_\theta(x^i))x_j^i$$

再最小化损失函数的过程中，**需要不断反复的更新weights使得误差函数减小**，更新过程如下：

$$\theta_j^{'} = \theta_j + \frac{1}{m}\sum_{i=1}^{m}(y^i - h_0(x^i))x_j^i$$

那么好了，**每次参数更新的伪代码**如下：

repeat{

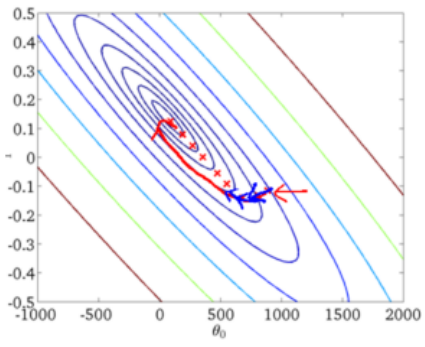
$$\theta_j^{'} = \theta_j + \frac{1}{m}\sum_{i=1}^{m}(y^i - h_0(x^i))x_j^i$$

(for every j=0, ... , n)

}

由上图更新公式我们就可以看到，**我们每一次的参数更新都用到了所有的训练数据**（比如有m个，就用到了m个），如果训练数据非常多的话，**是非常耗时的。**

下面给出批梯度下降的收敛图：



从图中，我们可以得到BGD迭代的次数相对较少。

随机梯度下降法SGD

由于批梯度下降每跟新一个参数的时候，要用到所有的样本数，所以训练速度会随着样本数量的增加而变得非常缓慢。随机梯度下降正是为了解决这个办法而提出的，它是利用每个样本的损失函数对θ求偏导得

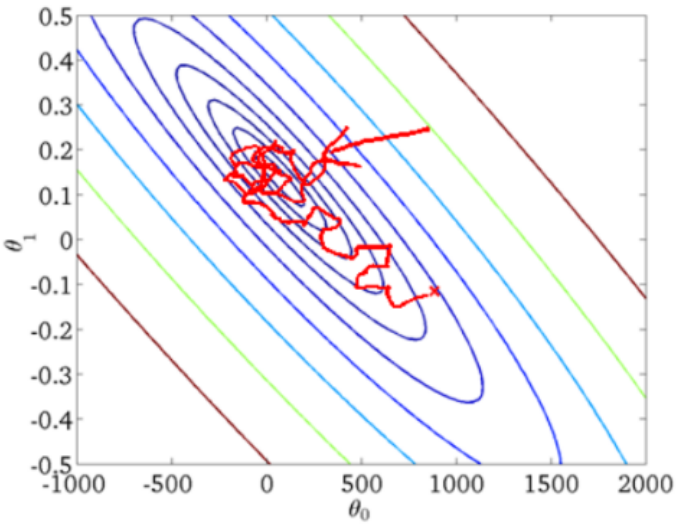
$$\theta_j^{'} = \theta_j + (y^i - h_\theta(x^i))x_j^i$$

更新过程如下：

1. Randomly shuffle dataset ;
2. **repeat**{
   for i=1, ... , m){
   $$\theta_j^{'} = \theta_j + (y^i - h_\theta(x^i))x_j^i$$
   (for j=0, ... , n)
   }
}

随机梯度下降是通过每个样本来迭代更新一次，对比上面的批量梯度下降，迭代一次需要用到所有训练样本（**往往如今真实问题训练数据都是非常巨大**），一次迭代不可能最优，如果迭代10次的话就需要遍历训练样本10次。**但是，SGD伴随的一个问题是噪音较BGD要多，使得SGD并不是每次迭代都向着整体最优化方向。**

随机梯度下降收敛图如下：



我们可以从图中看出SGD迭代的次数较多，在解空间的搜索过程看起来很盲目。**但是大体上是往着最优值方向移动。**

min-batch 小批量梯度下降法MBGD

我们从上面两种梯度下降法可以看出，其各自均有优缺点，那么能不能在两种方法的性能之间取得一个折衷呢？**即，算法的训练过程比较快，而且也要保证最终参数训练的准确率，**而这正是小批量梯度下降法（Mini-batch Gradient Descent，简称MBGD）的初衷。

我们假设每次更新参数的时候用到的样本数为10个（**不同的任务完全不同，这里举一个例子而已**）

更新伪代码如下：

Repeat{

for i=1, 11, 21, 31, … , 991{

$$\theta_j := \theta_j - \alpha\frac{1}{10}\sum_{k=i}^{i+9}(h_\theta(x^{(k)}) - y^{(k)})x_j^{(k)}$$

(for every j=0, … , n)

}

}

实例以及代码详解

这里参考他人博客，创建了一个数据，如下图所示：

	A	B	C
1	1.1	1.5	2.5
2	1.3	1.9	3.2
3	1.5	2.3	3.9
4	1.7	2.7	4.6
5	1.9	3.1	5.3
6	2.1	3.5	6
7	2.3	3.9	6.7
8	2.5	4.3	7.4
9	2.7	4.7	8.1
10	2.9	5.1	8.8

待训练数据A、B为自变量，C为因变量。

我希望通过这些训练数据给我训练出一个线性模型，用于进行下面数据的预测，test集合如下：

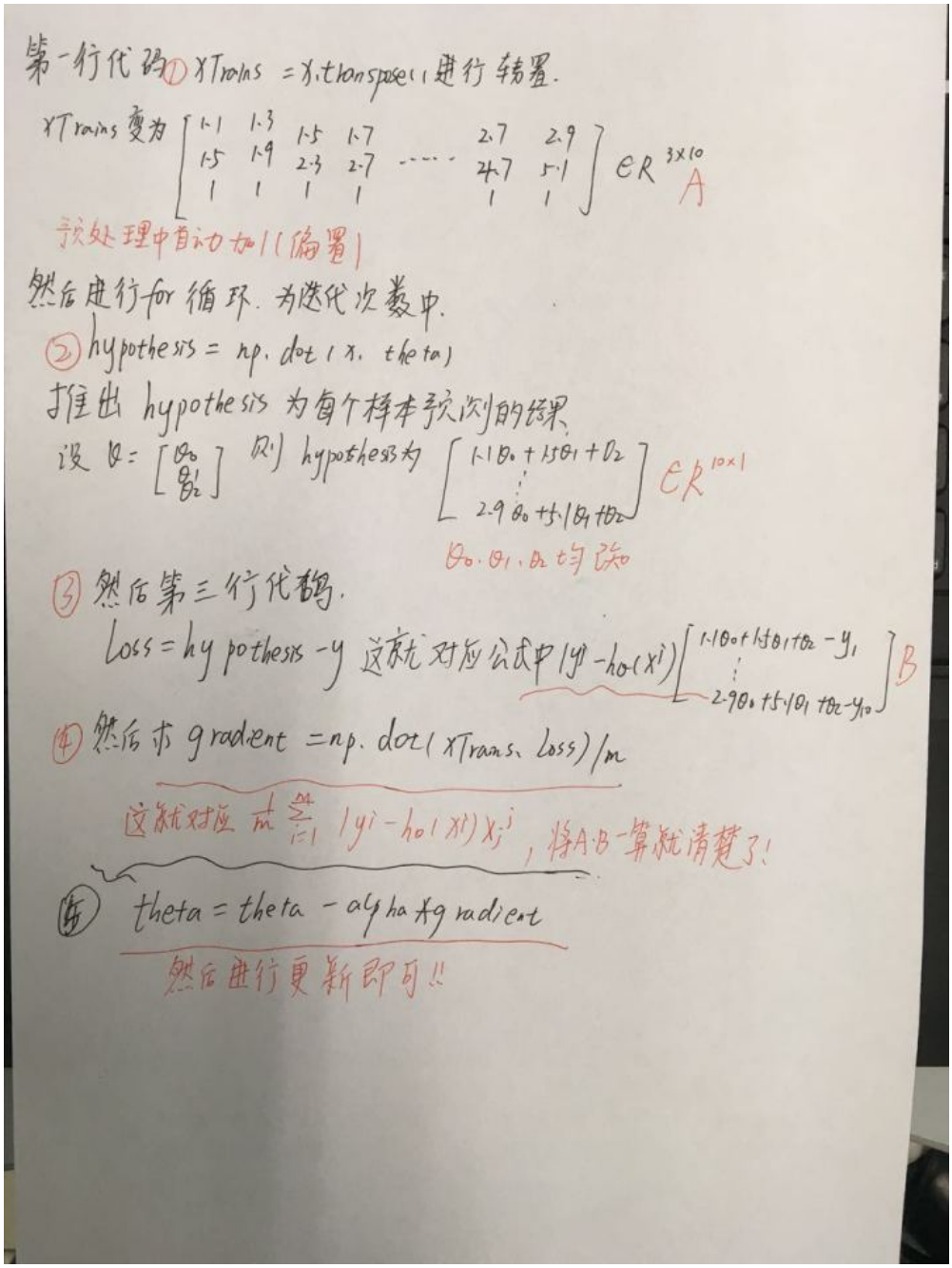
11	3.1	5.5	9.5
12	3.3	5.9	10.2
13	3.5	6.3	10.9
14	3.7	6.7	11.6
15	3.9	7.1	12.3

比如我们给出(3.1,5.5)希望模型预测出来的值与我们给定的9.5的差别是多少？这不是重点，重点是我们训练模型过程中的参数更新方法（**这是我们这篇文章的重点**）批梯度下降以及随机梯度下降代码如何实现。下面分别来讲：

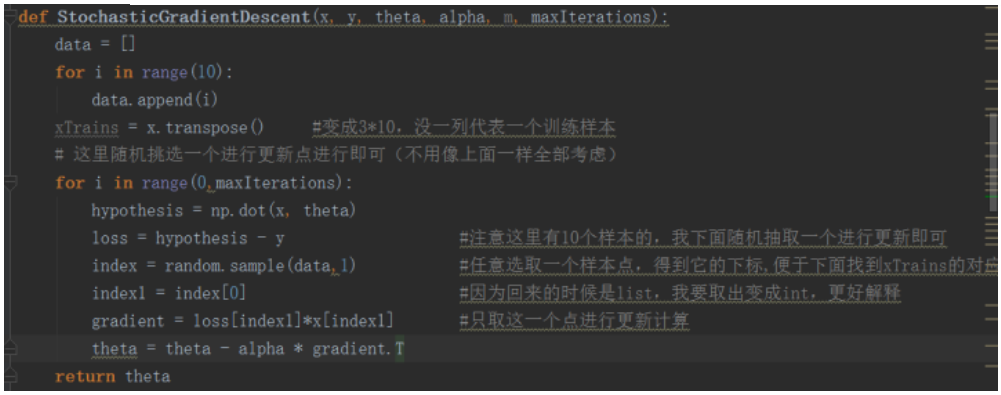
首先我们看批梯度下降法的代码如下：

```python
#下面实现的是批量梯度下降法
def batchGradientDescent(x, y, theta, alpha, m, maxIterations):
    xTrains = x.transpose()                          #得到它的转置
    for i in range(0, maxIterations):
        hypothesis = np.dot(x, theta)
        loss = hypothesis - y
        # print loss
        gradient = np.dot(xTrains, loss) / m          #对所有的样本进行求和，然后除以样本数
        theta = theta - alpha * gradient
    return theta
```

这里有可能还是比如抽象，为了让大家更好的弄懂理解这俩个重要的方法，**我下面结合例子，一行一行代码解释：**



我们看随机梯度下降法的代码如下：



```
def StochasticGradientDescent(x, y, theta, alpha, m, maxIterations):
    data = []
    for i in range(10):
        data.append(i)
    xTrains = x.transpose()     #变成3*10，没一列代表一个训练样本
    # 这里随机挑选一个进行更新点进行即可（不用像上面一样全部考虑）
    for i in range(0,maxIterations):
        hypothesis = np.dot(x, theta)
        loss = hypothesis - y                   #注意这里有10个样本的，我下面随机抽取一个进行更新即可
        index = random.sample(data,1)           #任意选取一个样本点，得到它的下标,便于下面找到xTrains的对应
        index1 = index[0]                       #因为回来的时候是list，我要取出变成int，更好解释
        gradient = loss[index1]*x[index1]       #只取这一个点进行更新计算
        theta = theta - alpha * gradient.T
    return theta
```

与批梯度下降最大的区别就在于，我们这里更新参数的时候，**并没有将所有训练样本考虑进去，**然后求和除以总数，而是我自己编程实现**任取一个样本点（代码中random函数就能清楚看到）**，然后利用这个样本点进行更新！这就是最大的区别！

那么到这个时候，我们也非常容易知道**小批量随机梯度下降法**的实现就是在这个的基础上，**随机取batch个样本，**而不是1个样本即可，掌握了本质就非常容易实现！

下面给出这个线性模型所有代码，训练，预测以及结果供参考：

```
#coding=utf-8
import numpy as np
import random

#下面实现的是批量梯度下降法
def batchGradientDescent(x, y, theta, alpha, m, maxIterations):
    xTrains = x.transpose()                                 #得到它的转置
    for i in range(0, maxIterations):
        hypothesis = np.dot(x, theta)
        loss = hypothesis - y
        # print loss
        gradient = np.dot(xTrains, loss) / m                #对所有的样本进行求和，然
        theta = theta - alpha * gradient
    return theta

#下面实现的是随机梯度下降法
def StochasticGradientDescent(x, y, theta, alpha, m, maxIterations):
    data = []
    for i in range(10):
        data.append(i)
    xTrains = x.transpose()     #变成3*10，没一列代表一个训练样本
    # 这里随机挑选一个进行更新点进行即可（不用像上面一样全部考虑）
    for i in range(0,maxIterations):
        hypothesis = np.dot(x, theta)
        loss = hypothesis - y                   #注意这里有10个样本的，我下面随机抽取
        index = random.sample(data,1)           #任意选取一个样本点，得到它的下标,便
        index1 = index[0]                       #因为回来的时候是list，我要取出变成
        gradient = loss[index1]*x[index1]       #只取这一个点进行更新计算
        theta = theta - alpha * gradient.T
    return theta

def predict(x, theta):
    m, n = np.shape(x)
    xTest = np.ones((m, n+1))                   #在这个例子中，是第三列放1
    xTest[:, :-1] = x                           #前俩列与x相同
    res =
```

```
    return res

trainData = np.array([[1.1,1.5,1],[1.3,1.9,1],[1.5,2.3,1],[1.7,2.7,1],[1.9,3.1,
trainLabel = np.array([2.5,3.2,3.9,4.6,5.3,6,6.7,7.4,8.1,8.8])
m, n = np.shape(trainData)
theta = np.ones(n)
alpha = 0.1
maxIteration = 5000
#下面返回的theta就是学到的theta
theta = batchGradientDescent(trainData, trainLabel, theta, alpha, m, maxIterat
print "theta = ",theta
x = np.array([[3.1, 5.5], [3.3, 5.9], [3.5, 6.3], [3.7, 6.7], [3.9, 7.1]])
print predict(x, theta)
theta = StochasticGradientDescent(trainData, trainLabel, theta, alpha, m, maxI
print "theta = ",theta
x = np.array([[3.1, 5.5], [3.3, 5.9], [3.5, 6.3], [3.7, 6.7], [3.9, 7.1]])
print predict(x, theta)
#yes,is the code
```

最后运行结果为：

```
theta =  [ 0.71493625  1.39253188 -0.37522769]
[  9.5  10.2  10.9  11.6  12.3]
theta =  [ 0.71493625  1.39253188 -0.37522769]
[  9.5  10.2  10.9  11.6  12.3]
```

说明与我们给定的真实值是完全对应的。

三种梯度下降方法的总结

1.批梯度下降每次更新使用了所有的训练数据，最小化损失函数，**如果只有一个极小值，那么批梯度下降是考虑了训练集所有数据，是朝着最小值迭代运动的，**但是缺点是如果样本值很大的话，更新速度会很慢。

2.随机梯度下降在每次更新的时候，只考虑了一个样本点，这样会大大加快训练数据，也恰好是批梯度下降的缺点，但是有可能由于训练数据的噪声点较多，**那么每一次利用噪声点进行更新的过程中，就不一定是朝着极小值方向更新，但是由于更新多轮，整体方向还是大致朝着极小值方向更新，又提高了速度。**

3.小批量梯度下降法是**为了解决批梯度下降法的训练速度慢，以及随机梯度下降法的准确性综合而来，但是这里注意，不同问题的batch是不一样的，**听师兄跟我说，我们nlp的parser训练部分batch一般就设置为10000，那么为什么是10000呢，我觉得这就和每一个问题中神经网络需要设置多少层，没有一个人能够准确答出，只能通过实验结果来进行超参数的调整。

好了，本篇文章要讲的已经讲完了，真心希望对大家理解有帮助，欢迎大家指错交流！

参考：

梯度下降算法以及其Python实现

[Machine Learning] 梯度下降法的三种形式BGD、SGD以及MBGD

致谢：郭江师兄，晓明师兄，德川

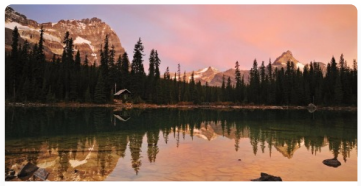
编辑于 2017-03-22

文章被以下专栏收录

机器学习算法与自然语言处理
公众号[自然语言处理与机器学习] 微信号yizhennotes

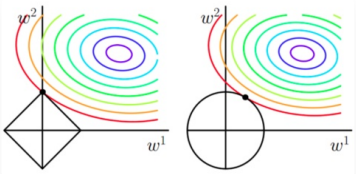
进入专栏

推荐阅读



深入浅出理解决策树算法（二）-ID3算法与C4.5算法

忆臻　　发表于机器学习算...



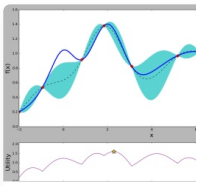
L1正则化与L2正则化

bingo酱

样本贡献不均：Focal Loss和Gradient Harmonizing...

本文主要介绍两个在目标检测中解决正负样本不平衡问题的方法，分别是发表在ICCV 2017上的Focal Loss for Dense Object Detection和AAAI 2019上的Gradient Harmonized Single-stage...

Beyon...　　发表于Beyon...



贝叶斯优化: 一种调优方式

tobe

44 条评论

切换为时间排序

写下你的评论...

彭鹏　2 年前

请问收敛图中的那个圈表示什么呀?

赞

忆臻 (作者) 回复 **彭鹏**　2 年前

表示，你迭代一次，在当前的参数下，位于损失函数的点位置

赞

知乎用户　2 年前

batch如果设置更大会有啥后果，以及学习率如何调整

赞

忆臻 (作者) 回复 知乎用户　2 年前

这个是调参的活，根据实验效果来

赞

知乎用户　2 年前

开始的图是cousera的好像...

赞

这图网上很多😀

赞

呵哈 2 年前

求问 sgd不是也是把m个样本迭代了一遍么?

赞

忆臻 (作者) 回复 **呵哈** 2 年前

你好，是的，但是关心的是一次迭代用了多少样本

1

雪猴儿 2 年前

你好！你SGD初始化的theta是BGD训练后的theta，不妥吧?

赞

忆臻 (作者) 回复 **雪猴儿** 2 年前

什么意思?

赞

雪猴儿 回复 **忆臻** (作者) 2 年前

程序最后，调用stochasticGradientDescent()时传入的theta, 不是应该再次初始化吗?
你直接传入的是调用batchGradientDescent()后，学习到的theta。

1

展开其他 2 条回复

葛溪驿 2 年前

图文并茂，讲的很好 谢谢了/抱拳

赞

忆臻 (作者) 回复 **葛溪驿** 2 年前

谢谢支持，自然语言处理和机器学习会记录我的笔记，欢迎关注交流，谢谢

赞

Mc铭橙 2 年前

好文章

赞

忆臻 (作者) 回复 **Mc铭橙** 2 年前

谢谢，公众号自然语言处理与机器学习 记录我的笔记，欢迎关注交交流。

赞

知乎用户 2 年前

刚看完Andrew Ng的梯度下降不久，学习了😊

赞

忆臻 (作者) 回复 知乎用户 2 年前

谢谢，自然语言处理与机器学习公众号会记录我的笔记，欢迎关注交流～

赞

知乎用户 2 年前

我记

赞

JeffG 2 年前

求教：梯度下降法的收敛速度的下限怎么计算呢？在高维情况下怎么保证sgd收敛呢？

赞

忆臻 (作者) 回复 JeffG 2 年前

抱歉，这快我也不大清楚

赞

hekko hello 2 年前

求教：现在是不是很多地方都把MBGD叫作SGD呀，感觉每次他们说用了SGD,一看，batch_size又不为1.。。。。

2

忆臻 (作者) 回复 hekko hello 2 年前

不是太严谨了，知道会用即可

赞

短发 1 年前

你好，请问你在随机梯度下降中写的伪代码和实际用python写的代码是不是有矛盾？就是，在每次迭代中：伪代码确实是对单独一个样本进行更新，但遍历了m个样本；而python代码中我看好像是随机选了一个i（i属于1:m），并没有遍历m个样本...

赞

忆臻 (作者) 回复 短发 1 年前

这个没有关系的，随机的重点是只取一个结点更新

赞

短发 回复 忆臻 (作者) 1 年前

因为想了下如果随机梯度下降像伪代码那样遍历m个样本的话，其耗费时间就等同于批量梯度下降了。实际在每次迭代中，如果像伪代码那样遍历全部样本的话，预测精度会比每次随机选取一个样本高些。不过，还是谢谢你提的重点，了解了随机的精髓了。

赞

袋鼠育儿 1 年前

讲得还是很详细啊，一定要收藏

赞

忆臻 (作者) 回复 袋鼠育儿 1 年前

谢谢~

赞

王大船 1 年前

求问损失函数为什么要乘以1/2呢？有什么好处？

赞

忆臻 (作者) 回复 王大船 1 年前

求导方便

赞

赞

jjsmz 1 年前

请教下大大，为什么更新参数都是一个个更新的呢，而不是参数整体做一次更新

1

 忆臻 (作者) 回复 jjsmz 1 年前

内存

赞

jjsmz 回复 忆臻 (作者) 1 年前

假设不考虑内存因素，如果参数之间是有影响得，那这两者更新得方式会有所区别么？

赞

展开其他 1 条回复

腾原 1 年前

学习了，题主这个批梯度下降法对某些数据集的loss，迭代反而变大了，还没查清原因，

赞

腾原 1 年前

啊哈，原来我的学习率过大了，导致没有收敛

赞

知乎用户 10 个月前

你好！有个疑问，你说mini-batch 梯度下降是随机选取batch个样本，进行更新参数，而伪代码却不是这样的意思。那么，应该是随机选取？还是按顺序选取？比如对于按顺序选取：样本量为64的数据集，采用mini-batch=8时，依次从数据集中选取样本，那么共8次更新参数，每次更新参数所用的mini-batch数据集是不一样的。。

赞

 忆臻 (作者) 回复 知乎用户 10 个月前

但是没考虑这么细，确实没有完全对应上

赞

 花花 10 个月前

其实一直有个问题，就是 在Deep Learning 领域，通常用得SGD优化器，其实准确地说应该是用得MBGD，因为通常我们都会去类似于32，64，128...这类得mini-batch。好奇当时为什么就笼统叫做SGD了，可能是DL领域得特例吧，我觉得严格意义上要叫MBGD的。

赞

 本体匹配调谐 9 个月前

Tencent MIG 面试被问到SGD的缺点和改进办法，怎么回答？缺点是不是有时候可能收敛不到最优值这个?

赞