zhiyong_will 专家
码龄10年 《Python机器学习算法》作者
185
原创
1万+
积分
私信 关注

搜博主文章

热门文章

Python中时间与时间戳之间的转换 260390
python读取文件——python读取和保存mat文件 133758
简单易学的机器学习算法——极限学习机(ELM) 118963
数据处理——One-Hot Encoding 116179
简单易学的机器学习算法——神经网络之BP神经网络 111959

分类专栏

梯度下降优化算法综述

翻译 zhiyong_will 2017-04-14 17:28:56 36985 收藏 158
分类专栏： Optimization Algorithm 文章标签： 优化

> 本文翻译自Sebastian Ruder的“An overview of gradient descent optimization algoritms”，作者首先在其博客中发表了这篇文章，其博客地址为：An overview of gradient descent optimization algoritms，之后，作者将其整理完放在了arxiv中，其地址为：An overview of gradient descent optimization algoritms，在翻译的过程中以作者发布在Arxiv的论文为主，参考其在博客中的内容。
>
> 本文的翻译已经获得作者的同意。

摘要

虽然梯度下降优化算法越来越受欢迎，但通常作为黑盒优化器使用，因此很难对其优点和缺点的进行实际的解释。本文旨在让读者对不同的算法有直观的认识，以帮助读者使用这些算法。在本综述中，我们介绍梯度下降的不同变形形式，总结这些算法面临的挑战，介绍最常用的优化算法，回顾并行和分布式架构，以及调研用于优化梯度下降的其他的策略。

1 引言

梯度下降法是最著名的优化算法之一，也是迄今优化神经网络时最常用的方法。同时，在每一个最新的深度学习库中都包含了各种优化的梯度下降法的实现（例如：参见lasagne，caffe和keras的文档）。然而，这些算法通常是作为黑盒优化器使用，因此，很难对其优点和缺点的进行实际的解释。

本文旨在让读者对不同的优化梯度下降的算法有直观的认识，以帮助读者使用这些算法。在第2部分，我们首先介绍梯度下降的不同变形形式。在第3部分，我们将简要总结在训练的过程中所面临的挑战。随后，在第4部分，我们将介绍最常用的优化算法，包括这些算法在解决以上挑战时的动机以及如何得到更新规则的推导形式。在第5部分，我们将简单讨论在并行和分布式环境中优化梯度下降的算法和框架。最后，在第6部分，我们将思考对优化梯度下降有用的一些其他策略。

梯度下降法是最小化目标函数$J(\theta)$的一种方法，其中，$\theta \in \mathbb{R}^d$为模型参数，梯度下降法利用目标函数关于参数的梯度$\nabla_\theta J(\theta)$的反方向更新参数。学习率η决定达到最小值或者局部最小值过程中所采用的步长的大小。即，我们沿着目标函数的斜面下降的方向，直到到达谷底。如果你对梯度下降法不熟悉，你可以从http://cs231n.github.io/optimization-1/找到介绍神经网络优化的材料。

2 梯度下降法的变形形式

梯度下降法有3中变形形式，它们之间的区别为我们在计算目标函数的梯度时使用到多少数据。根据数据量的不同，我们在参数更新的精度和更新过程中所需要的时间两个方面做出权衡。

2.1 批梯度下降法

最新评论

优化算法——粒子群算法(PSO)
Mar_: 为什么每一次运行，最后迭代的结构都不一样

优化算法——粒子群算法(PSO)
Mar_: 我的结果也是，每次的迭代结构有一样，是一条直线

优化算法——拟牛顿法之DFP算法
不喝也中: 这个是真的牛批

优化算法——粒子群算法(PSO)
Whenever@Wherever: 怎么输出最优解啊，只有一个迭代的图像

Python技巧——list与字符串互相转换
Tisfy: 顶！

最新文章

C++中的explicit关键字

文本分类fastText算法

推荐系统中的常用算法——序列深度匹配SDM

2021年 2篇	2020年 9篇
2019年 6篇	2018年 14篇
2017年 23篇	2016年 37篇
2015年 53篇	2014年 43篇

目录

Vanilla梯度下降法，又称为批梯度下降法（batch gradient descent），在整个训练数据集上计算损失函数关于参数θ的梯度：

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

因为在执行每次更新时，我们需要在整个数据集上计算所有的梯度，所以批梯度下降法的速度会很慢，同时，批梯度下降法无法处理超出内存容量限制的数据集。批梯度下降法同样也不能在线更新模型，即在运行的过程中，不能增加新的样本。

批梯度下降法的代码如下所示：

```
for i in range(nb_epochs):
    params_grad = evaluate_gradient(loss_function, data, params)
    params = params - learning_rate * params_grad
```

对于给定的迭代次数，首先，我们利用全部数据集计算损失函数关于参数向量`params`的梯度向量`params_grad`。注意，最新的深度学习库中提供了自动求导的功能，可以有效地计算关于参数梯度。如果你自己求梯度，那么，梯度检查是一个不错的主意（关于如何正确检查梯度的一些技巧可以参见http://cs231n.github.io/neural-networks-3/）。

然后，我们利用梯度的方向和学习率更新参数，学习率决定我们将以多大的步长更新参数。对于凸误差函数，批梯度下降法能够保证收敛到全局最小值，对于非凸函数，则收敛到一个局部最小值。

2.2 随机梯度下降法

相反，随机梯度下降法（stochastic gradient descent, SGD）根据每一条训练样本$x^{(i)}$和标签$y^{(i)}$更新参数：

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

对于大数据集，因为批梯度下降法在每一个参数更新之前，会对相似的样本计算梯度，所以在计算过程中会有冗余。而SGD在每一次更新中只执行一次，从而消除了冗余。因而，通常SGD的运行速度更快，同时，可以用于在线学习。SGD以高方差频繁地更新，导致目标函数出现如图1所示的剧烈波动。

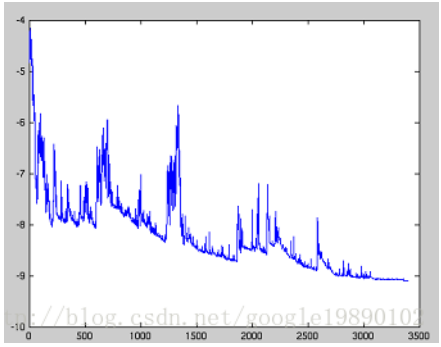


图1：SGD波动（来源： Wikipedia）

与批梯度下降法的收敛会使得损失函数陷入局部最小相比，由于SGD的波动性，一方面，波动性使得SGD可以跳到新的和潜在更好的局部最优。另一方面，这使得最终收敛到特定最小值的过程变得复杂，因为SGD会一直持续波动。然而，已经证明当我们缓慢减小学习率，SGD与批梯度下降法具有相同的收敛行为，对于非凸优化和凸优化，可以分别收敛到局部最小值和全局最小值。与批梯度下降的代码相比，SGD的代码片段仅仅是在对训练样本的遍历和利用每一条样本计算梯度的过程中增加一层循环。注意，如6.1节中的解释，在每一次循环中，我们打乱训练样本。

```
for i in range(nb_epochs):
    np.random.shuffle(data)
    for example in data:
        params_grad = evaluate_gradient(loss_function, example, params)
        params = params - learning_rate * params_grad
```

2.3 小批量梯度下降法

小批量梯度下降法最终结合了上述两种方法的优点，在每次更新时使用n个小批量训练样本：

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

这种方法，a)减少参数更新的方差，这样可以得到更加稳定的收敛结果；b)可以利用最新的深度学习库中高度优化的矩阵优化方法，高效地求解每个小批量数据的梯度。通常，小批量数据的大小在50到256之间，也可以根据不同的应用有所变化。当训练神经网络模型时，小批量梯度下降法是典型的选择算法，当使用小批量梯度下降法时，也将其称为SGD。注意：在下文的改进的SGD中，为了简单，我们省略了参数$x^{(i:i+n)}; y^{(i:i+n)}$。

在代码中，不是在所有样本上做迭代，我们现在只是在大小为50的小批量数据上做迭代：

```
for i in range(nb_epochs):
    np.random.shuffle(data)
    for batch in get_batches(data, batch_size=50):
        params_grad = evaluate_gradient(loss_function, batch, params)
        params = params - learning_rate * params_grad
```

3 挑战

虽然Vanilla小批量梯度下降法并不能保证较好的收敛性，但是需要强调的是，这也给我们留下了如下的一些挑战：

- 选择一个合适的学习率可能是困难的。学习率太小会导致收敛的速度很慢，学习率太大会妨碍收敛，导致损失函数在最小值附近波动甚至偏离最小值。
- 学习率调整[17]试图在训练的过程中通过例如退火的方法调整学习率，即根据预定义的策略或者当相邻两代之间的下降值小于某个阈值时减小学习率。然而，策略和阈值需要预先设定好，因此无法适应数据集的特点[4]。
- 此外，对所有的参数更新使用同样的学习率。如果数据是稀疏的，同时，特征的频率差异很大时，我们也许不想以同样的学习率更新所有的参数，对于出现次数较少的特征，我们对其执行更大的学习率。
- 高度非凸误差函数普遍出现在神经网络中，在优化这类函数时，另一个关键的挑战是使函数避免陷入无数次优的局部最小值。Dauphin等人[5]指出出现这种困难实际上并不是来自局部最小值，而是来自鞍点，即那些在一个维度上是递增的，而在另一个维度上是递减的。这些鞍点通常被具有相同误差的点包围，因为在任意维度上的梯度都近似为0，所以SGD很难从这些鞍点中逃开。

4 梯度下降优化算法

下面，我们将列举一些算法，这些算法被深度学习社区广泛用来处理前面提到的挑战。我们不会讨论在实际中不适合在高维数据集中计算的算法，例如诸如牛顿法的二阶方法。

4.1 动量法

SGD很难通过陡谷，即在一个维度上的表面弯曲程度远大于其他维度的区域[19]，这种情况通常出现在局部最优点附近。在这种情况下，SGD摇摆地通过陡谷的斜坡，同时，沿着底部到局部最优点的路径上只是缓慢地前进，这个过程如图2a所示。

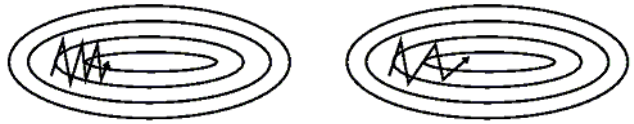


图2：来源：Genevieve B. Orr

如图2b所示，动量法[16]是一种帮助SGD在相关方向上加速并抑制摇摆的一种方法。动量法将历史步长的更新向量的一个分量γ增加到当前的更新向量中（部分实现中交换了公式中的符号）

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$

$$\theta = \theta - v_t$$

动量项γ通常设置为0.9或者类似的值。

从本质上说，动量法，就像我们从山上推下一个球，球在滚下来的过程中累积动量，变得越来越快（直到达到终极速度，如果有空气阻力的存在，则$\gamma < 1$）。同样的事情也发生在参数的更新过程中：对于在梯度点处具有相同的方向的维度，其动量项增大，对于在梯度点处改变方向的维度，其动量项减小。因此，我们可以得到更快的收敛速度，同时可以减少摇摆。

4.2 Nesterov加速梯度下降法

然而，球从山上滚下的时候，盲目地沿着斜率方向，往往并不能令人满意。我们希望有一个智能的球，这个球能够知道它将要去哪，以至于在重新遇到斜率上升时能够知道减速。

Nesterov加速梯度下降法（Nesterov accelerated gradient，NAG）[13]是一种能够给动量项这样的预知能力的方法。我们知道，我们利用动量项γv_{t-1}来更新参数θ。通过计算$\theta - \gamma v_{t-1}$能够告诉我们参数未来位置的一个近似值（梯度并不是完全更新），这也就是告诉我们参数大致将变为多少。通过计算关于参数未来的近似位置的梯度，而不是关于当前的参数θ的梯度，我们可以高效的求解：

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$

$$\theta = \theta - v_t$$

同时，我们设置动量项γ大约为0.9。动量法首先计算当前的梯度值（图3中的小的蓝色向量），然后在更新的累积梯度（大的蓝色向量）方向上前进一大步，Nesterov加速梯度下降法NAG首先在先前累积梯度（棕色的向量）方向上前进一大步，计算梯度值，然后做一个修正（绿色的向量）。这个具有预见性的更新防止我们前进得太快，同时增强了算法的响应能力，这一点在很多的任务中对于RNN的性能提升有着重要的意义[2]。

图3：Nesterov更新（来源： G. Hinton的课程6c）

对于NAG的直观理解的另一种解释可以参见http://cs231n.github.io/neural-networks-3/，同时Ilya Sutskever在其博士论文[18]中给出更详细的综述。

既然我们能够使得我们的更新适应误差函数的斜率以相应地加速SGD，我们同样也想要使得我们的更新能够适应每一个单独参数，以根据每个参数的重要性决定大的或者小的更新。

4.3 Adagrad

Adagrad[7]是这样的一种基于梯度的优化算法：让学习率适应参数，对于出现次数较少的特征，我们对其采用更大的学习率，对于出现次数较多的特征，我们对其采用较小的学习率。因此，Adagrad非常适合处理稀疏数据。Dean等人[6]发现Adagrad能够极大提高了SGD的鲁棒性并将其应用于Google的大规模神经网络的训练，其中包含了YouTube视频中的猫的识别。此外，Pennington等人[15]利用Adagrad训练Glove词向量，因为低频词比高频词需要更大的步长。

前面，我们每次更新所有的参数θ时，每一个参数θ_i都使用的是相同的学习率η。由于Adagrad在t时刻对每一个参数θ_i使用了不同的学习率，我们首先介绍Adagrad对每一个参数的更新，然后我们对其向量化。为了简洁，令$g_{t,i}$为在t时刻目标函数关于参数θ_i的梯度：

$$g_{t,i} = \nabla_\theta J(\theta_i)$$

在t时刻，对每个参数θ_i的更新过程变为：

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

对于上述的更新规则，在t时刻，基于对θ_i计算过的历史梯度，Adagrad修正了对每一个参数θ_i的学习率：

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

其中，$G_t \in \mathbb{R}^{d \times d}$是一个对角矩阵，对角线上的元素$i,i$是直到$t$时刻为止，所有关于$\theta_i$的梯度的平方和（Duchi等人[7]将该矩阵作为包含所有先前梯度的外积的完整矩阵的替代，因为即使是对于中等数量的参数d，矩阵的均方根的计算都是不切实际的。），ϵ是平滑项，用于防止除数为0（通常大约设置为$1e-8$）。比较有意思的是，如果没有平方根的操作，算法的效果会变得很差。

由于G_t的对角线上包含了关于所有参数θ的历史梯度的平方和，现在，我们可以通过G_t和g_t之间的元素向量乘法$\odot$向量化上述的操作：

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

Adagrad算法的一个主要优点是无需手动调整学习率。在大多数的应用场景中，通常采用常数0.01。

Adagrad的一个主要缺点是它在分母中累加梯度的平方：由于没增加一个正项，在整个训练过程中，累加的和会持续增长。这会导致学习率变小以至于最终变得无限小，在学习率无限小时，Adagrad算法将无法取得额外的信息。接下来的算法旨在解决这个不足。

4.4 Adadelta

Adadelta[21]是Adagrad的一种扩展算法，以处理Adagrad学习速率单调递减的问题。不是计算所有的梯度平方，Adadelta将计算计算历史梯度的窗口大小限制为一个固定值w。

在Adadelta中，无需存储先前的w个平方梯度，而是将梯度的平方递归地表示成所有历史梯度平方的均值。在t时刻的均值$E[g^2]_t$只取决于先前的均值和当前的梯度（分量γ类似于动量项）：

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

我们将γ设置成与动量项相似的值，即0.9左右。为了简单起见，我们利用参数更新向量$\Delta\theta_t$重新表示SGD的更新过程：

$$\Delta\theta_t = -\eta \cdot g_{t,i}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

我们先前得到的Adagrad参数更新向量变为：

$$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

现在，我们简单将对角矩阵G_t替换成历史梯度的均值$E[g^2]_t$：

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

由于分母仅仅是梯度的均方根（root mean squared，RMS）误差，我们可以简写为：

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t$$

作者指出上述更新公式中的每个部分（与SGD，动量法或者Adagrad）并不一致，即更新规则中必须与参数具有相同的假设单位。为了实现这个要求，作者首次定义了另一个指数衰减均值，这次不是梯度平方，而是参数的平方的更新：

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

因此，参数更新的均方根误差为：

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

由于$RMS[\Delta\theta]_t$是未知的，我们利用参数的均方根误差来近似更新。利用$RMS[\Delta\theta]_{t-1}$替换先前的更新规则中的学习率η，最终得到Adadelta的更新规则：

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

使用Adadelta算法，我们甚至都无需设置默认的学习率，因为更新规则中已经移除了学习率。

4.5 RMSprop

RMSprop是一个未被发表的自适应学习率的算法，该算法由Geoff Hinton在其Coursera课堂的课程6e中提出。

RMSprop和Adadelta在相同的时间里被独立的提出，都起源于对Adagrad的极速递减的学习率问题的求解。实际上，RMSprop是先前我们得到的Adadelta的第一个更新向量的特例：

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

同样，RMSprop将学习率分解成一个平方梯度的指数衰减的平均。Hinton建议将γ设置为0.9，对于学习率η，一个好的固定值为0.001。

4.6 Adam

自适应矩估计（Adaptive Moment Estimation，Adam）[9]是另一种自适应学习率的算法，Adam对每一个参数都计算自适应的学习率。除了像Adadelta和RMSprop一样存储一个指数衰减的历史平方梯度的平均v_t，Adam同时还保存一个历史梯度的指数衰减均值m_t，类似于动量：

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

m_t和v_t分别是对梯度的一阶矩（均值）和二阶矩（非确定的方差）的估计，正如该算法的名称。当m_t和v_t初始化为0向量时，Adam的作者发现它们都偏向于0，尤其是在初始化的步骤和当衰减率很小的时候（例如β_1和β_2趋向于1）。

通过计算偏差校正的一阶矩和二阶矩估计来抵消偏差：

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

正如我们在Adadelta和RMSprop中看到的那样，他们利用上述的公式更新参数，由此生成了Adam的更新规则：

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

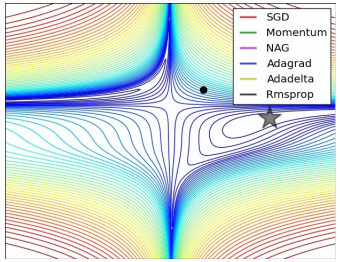
作者建议β_1取默认值为0.9，β_2为0.999，ϵ为10^{-8}。他们从经验上表明Adam在实际中表现很好，同时，与其他的自适应学习算法相比，其更有优势。

4.7 算法可视化

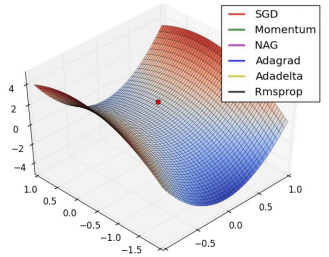
下面两张图给出了上述优化算法的优化行为的直观理解。（还可以看看这里关于Karpathy对相同的图片的描述以及另一个简明关于算法讨论的概述）。

在图4a中，我们看到不同算法在损失曲面的等高线上走的不同路线。所有的算法都是从同一个点出发并选择不同路径到达最优点。注意：Adagrad，Adadelta和RMSprop能够立即转移到正确的移动方向上并以类似的速度收敛，而动量法和NAG会导致偏离，想像一下球从山上滚下的画面。然而，NAG能够在偏离之后快速修正其路线，因为NAG通过对最优点的预见增强其响应能力。

图4b中展示了不同算法在鞍点出的行为，鞍点即为一个点在一个维度上的斜率为正，而在其他维度上的斜率为负，正如我们前面提及的，鞍点对SGD的训练造成很大困难。这里注意，SGD，动量法和NAG在鞍点处很难打破对称性，尽管后面两个算法最终设法逃离了鞍点。而Adagrad，RMSprop和Adadelta能够快速想着梯度为负的方向移动，其中Adadelta走在最前面。



(a)损失去面的等高线上SGD优化



(b)在鞍点处的SGD优化

图4：来源和全部动画： Alec Radford

正如我们所看到的，自适应学习速率的方法，即 Adagrad、 Adadelta、 RMSprop 和Adam，最适合这些场景下最合适，并在这些场景下得到最好的收敛性。

4.8 选择使用哪种优化算法?

那么，我们应该选择使用哪种优化算法呢？如果输入数据是稀疏的，选择任一自适应学习率算法可能会得到最好的结果。选用这类算法的另一个好处是无需调整学习率，选用默认值就可能达到最好的结果。

总的来说，RMSprop是Adagrad的扩展形式，用于处理在Adagrad中急速递减的学习率。RMSprop与Adadelta相同，所不同的是Adadelta在更新规则中使用参数的均方根进行更新。最后，Adam是将偏差校正和动量加入到RMSprop中。在这样的情况下，RMSprop、Adadelta和Adam是很相似的算法并且在相似的环境中性能都不错。Kingma等人[9]指出在优化后期由于梯度变得越来越稀疏，偏差校正能够帮助Adam微弱地胜过RMSprop。综合看来，Adam可能是最佳的选择。

有趣的是，最近许多论文中采用不带动量的SGD和一种简单的学习率的退火策略。已表明，通常SGD能够找到最小值点，但是比其他优化的SGD花费更多的时间，与其他算法相比，SGD更加依赖鲁棒的初始化和退火策略，同时，SGD可能会陷入鞍点，而不是局部极小值点。因此，如果你关心的是快速收敛和训练一个深层的或者复杂的神经网络，你应该选择一个自适应学习率的方法。

5 并行和分布式SGD

当存在大量的大规模数据和廉价的集群时，利用分布式SGD来加速是一个显然的选择。SGD本身有固有的顺序：一步一步，我们进一步进展到最小。SGD提供了良好的收敛性，但SGD的运行缓慢，特别是对于大型数据集。相反，SGD异步运行速度更快，但客户端之间非最理想的通信会导致差的收敛。此外，我们也可以在一台机器上并行SGD，这样就无需大的计算集群。以下是已经提出的优化的并行和分布式的SGD的算法和框架。

5.1 Hogwild!

Niu等人[14]提出称为Hogwild!的更新机制，Hogwild!允许在多个CPU上并行执行SGD更新。在无需对参数加锁的情况下，处理器可以访问共享的内存。这种方法只适用于稀疏的输入数据，因为每一次更新只会修改一部分参数。在这种情况下，该更新策略几乎可以达到一个最优的收敛速率，因为CPU之间不可能重写有用的信息。

5.2 Downpour SGD

Downpour SGD是SGD的一种异步的变形形式，在Google，Dean等人[6]在他们的DistBelief框架（TensorFlow的前身）中使用了该方法。Downpour SGD在训练集的子集上并行运行多个模型的副本。这些模型将各自的更新发送给一个参数服务器，参数服务器跨越了多台机器。每一台机器负责存储和更新模型的一部分参数。然而，因为副本之间是彼此不互相通信的，即通过共享权重或者更新，因此可能会导致参数发散而不利于收敛。

5.3 延迟容忍SGD

通过容忍延迟算法的开发，McMahan和Streeter[11]将AdaGraad扩展成并行的模式，该方法不仅适应于历史梯度，同时适应于更新延迟。该方法已经在实践中被证实是有效的。

5.4 TensorFlow

TensorFlow[1]是Google近期开源的框架，该框架用于实现和部署大规模机器学习模型。TensorFlow是基于DistBelief开发，同时TensorFlow已经在内部用来在大量移动设备和大规模分布式系统的执行计算。在2016年4月发布的分布式版本依赖于图计算，图计算即是对每一个设备将图划分成多个子图，同时，通过发送、接收节点对完成节点之间的通信。

5.5 弹性平均SGD

Zhang等人[22]提出的弹性平均SGD（Elastic Averaging SGD，EASGD）连接了异步SGD的参数客户端和一个弹性力，即参数服务器存储的一个中心变量。EASGD使得局部变量能够从中心变量震荡得更远，这在理论上使得在参数空间中能够得到更多的探索。经验表明这种增强的探索能力通过发现新的局部最优点，能够提高整体的性能。

6 优化SGD的其他策略

最后，我们介绍可以与前面提及到的任一算法配合使用的其他的一些策略，以进一步提高SGD的性能。对于其他的一些常用技巧的概述可以参见[10]。

6.1 数据集的洗牌和课程学习

总的来说，我们希望避免向我们的模型中以一定意义的顺序提供训练数据，因为这样会使得优化算法产生偏差。因此，在每一轮迭代后对训练数据洗牌是一个不错的主意。

另一方面，在很多情况下，我们是逐步解决问题的，而将训练集按照某个有意义的顺序排列会提高模型的性能和SGD的收敛性，如何将训练集建立一个有意义的排列被称为课程学习[3]。

Zaremba and Sutskever[20]只能使用课程学习训练LSTM来评估简单程序，并表明组合或混合策略比单一的策略更好，通过增加难度来排列示例。

6.2 批量归一化

为了便于学习，我们通常用0均值和单位方差初始化我们的参数的初始值来归一化。 随着不断训练，参数得到不同的程度的更新，我们失去了这种归一化，随着网络变得越来越深，这种现象会降低训练速度，且放大参数变化。

批量归一化[8]在每次小批量数据反向传播之后重新对参数进行0均值单位方差标准化。通过将模型架构的一部分归一化，我们能够使用更高的学习率，更少关注初始化参数。批量归一化还充当正则化的作用，减少（有时甚至消除）Dropout的必要性。

6.3 Early stopping

如Geoff Hinton所说：“Early Stopping是美丽好免费午餐”（NIPS 2015 Tutorial slides）。你因此必须在训练的过程中时常在验证集上监测误差，在验证集上如果损失函数不再显著地降低，那么应该提前结束训练。

6.4 梯度噪音

Neelakantan等人[12]在每个梯度更新中增加满足高斯分布$N(0,\sigma_t^2)$的噪音：

$$g_{t,i} = g_{t,i} + N(0,\sigma_t^2)$$

高斯分布的方差需要根据如下的策略退火：

$$\sigma_t^2 = \frac{\eta}{(1+t)^\gamma}$$

他们指出增加了噪音，使得网络对不好的初始化更加鲁棒，同时对深层的和复杂的网络的训练特别有益。他们猜测增加的噪音使得模型更优机会逃离当前的局部最优点，以发现新的局部最优点，这在更深层的模型中更加常见。

7 总结

在这篇博客文章中，我们初步研究了梯度下降的三个变形形式，其中，小批量梯度下降是最受欢迎的。 然后我们研究了最常用于优化SGD的算法：动量法，Nesterov加速梯度，Adagrad，Adadelta，RMSprop，Adam以及不同的优化异步SGD的算法。 最后，我们已经考虑其他一些改善SGD的策略，如洗牌和课程学习，批量归一化和early stopping。

参考文献

- [1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., … Zheng, X. (2015). TensorFlow : Large-Scale Machine Learning on Heterogeneous Distributed Systems.
- [2] Bengio, Y., Boulanger-Lewandowski, N., & Pascanu, R. (2012). Advances in Optimizing Recurrent Networks. Retrieved from http://arxiv.org/abs/1212.0901

- [3] Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. Proceedings of the 26th Annual International Conference on Machine Learning, 41–48. http://doi.org/10.1145/1553374.1553380
- [4] Darken, C., Chang, J., & Moody, J. (1992). Learning rate schedules for faster stochastic gradient search. Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop, (September), 1–11. http://doi.org/10.1109/NNSP.1992.253713
- [5] Dauphin, Y., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., & Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. arXiv, 1–14. Retrieved from http://arxiv.org/abs/1406.2572
- [6] Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V, … Ng, A. Y. (2012). Large Scale Distributed Deep Networks. NIPS 2012: Neural Information Processing Systems, 1–11. http://doi.org/10.1109/ICDAR.2011.95
- [7] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12, 2121–2159. Retrieved from http://jmlr.org/papers/v12/duchi11a.html
- [8] Ioffe, S., & Szegedy, C. (2015). Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv Preprint arXiv:1502.03167v3
- [9] Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, 1–13.
- [10] LeCun, Y., Bottou, L., Orr, G. B., & Müller, K. R. (1998). Efficient BackProp. Neural Networks: Tricks of the Trade, 1524, 9–50. http://doi.org/10.1007/3-540-49430-8_2
- [11] Mcmahan, H. B., & Streeter, M. (2014). Delay-Tolerant Algorithms for Asynchronous Distributed Online Learning. Advances in Neural Information Processing Systems (Proceedings of NIPS), 1–9. Retrieved from http://papers.nips.cc/paper/5242-delay-tolerant-algorithms-for-asynchronous-distributed-online-learning.pdf
- [12] Neelakantan, A., Vilnis, L., Le, Q. V., Sutskever, I., Kaiser, L., Kurach, K., & Martens, J. (2015). Adding Gradient Noise Improves Learning for Very Deep Networks, 1–11. Retrieved from http://arxiv.org/abs/1511.06807
- [13] Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence o(1/k2). Doklady ANSSSR (translated as Soviet.Math.Docl.), vol. 269, pp. 543–547.
- [14] Niu, F., Recht, B., Christopher, R., & Wright, S. J. (2011). Hogwild! : A Lock-Free Approach to Parallelizing Stochastic Gradient Descent, 1–22.
- [15] Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, 1532–1543. http://doi.org/10.3115/v1/D14-1162
- [16] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. Neural Networks : The Official Journal of the International Neural Network Society, 12(1), 145–151. http://doi.org/10.1016/S0893-6080(98)00116-6
- [17] H. Robinds and S. Monro, “A stochastic approximation method,” Annals of Mathematical Statistics, vol. 22, pp. 400–407, 1951.

Statistics, Vol. 22, pp. 400–407, 1951.

- [18] Sutskever, I. (2013). Training Recurrent neural Networks. PhD Thesis.
- [19] Sutton, R. S. (1986). Two problems with backpropagation and other steepest-descent learning procedures for networks. Proc. 8th Annual Conf. Cognitive Science Society.
- [20] Zaremba, W., & Sutskever, I. (2014). Learning to Execute, 1–25. Retrieved from http://arxiv.org/abs/1410.4615
- [21] Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. Retrieved from http://arxiv.org/abs/1212.5701
- [22] Zhang, S., Choromanska, A., & LeCun, Y. (2015). Deep learning with Elastic Averaging SGD. Neural Information Processing Systems Conference (NIPS 2015), 1–24. Retrieved from http://arxiv.org/abs/1412.6651

不同的优化梯度下降法的算法有一个直观认识，以帮助你使用这些算法。我们首先会考察梯度下降法的各种变体，然后会简要地总结在训练（神经网络或是机器学习算法）的过程中可能遇到的挑战。

Matlab 神经网络， newff， train trainrp叫弹性梯度下降法，的原理解释 kebu12345678的博客 2068

多层神经网络的隐层大多采用sigmoid型传递函数，这类函数又称为"挤压"函数。因为它们将没有边界限制的输入信号压缩到有限的输出范围内，当输入量很大或很小时，输出函数的斜率接近于0。那么当应用梯度下降法训练多层网络时，其梯度数量级会很小，从而使得权值和阈值的调整范围很小，也就是说即使没有达到最优值，也会形成训练停止的结果。 弹性梯度下降法就能够消除这种影响。应用弹性梯度下降法训练BP网络时，权值修...

机器学习入门——梯度下降算法详解 日积月累，天道酬勤 2597

在机器学习领域，熟练的掌握梯度法来求解目标函数的最优解是一个非常重要的事情。跟我一起来详细了解梯度下降法的原理吧。

梯度下降优化算法综述 我和我追逐的梦~~~ 6万+

梯度下降优化算法综述 该文翻译自An overview of gradient descent optimization algorithms。 总所周知，梯度下降算法是机器学习中使用非常广泛的优化算法，也是众多机器学习算法中最常用的优化方法。几乎当前每一个先进的(state-of-the-art)机器学习库或者深度学习库都会包括梯度下降算法的不同变种实现。但是，它们就像一个黑盒优化器，很难

随机梯度下降(SGD)与经典的梯度下降法的区别 米兰小子SHC 2万+

随机梯度下降(SGD)与经典的梯度下降法的区别 经典的优化方法，例如梯度下降法，在每次迭代过程中需要使用所有的训练数据，这就给求解大规模数据优化问题带来挑战。 知识点：随机梯度下降法(SGD)、小批量梯度下降法。 在机器学习中，目标函数通常可以表示成为如下形式: 而经典的梯度下降法采用所有的训练数据的平均损失来近似目标函数。其中M是训练样本的个数。 模型参数的更新公式为： 因此，经典的梯度下...

机器学习中常见问题_几种梯度下降法 深度学习思考者 1万+

随机梯度下降 批梯度下降 minibatch

YOLOv3目标检测：原理与源码解析 06-07

   Linux创始人Linus Torvalds有一句名言：Talk is cheap, Show me the code.（冗谈不够，放码过来！）。   代码阅读是从入门到提高的必由之路。尤其对深度学习，许多框架隐藏了神经网络底层的实现，只能在上层调包使用，对其内部原理很难认识清晰，不利于进一步优化和创新。   YOLOv3是一种基于深度学习的端到端实时目标检测方法，以速度快见长。   YOLOv3的实现Darknet是使用C语言开发的轻型开源深度学习框架，依赖少，可移植性好，可以作为很好的代码阅读案例，让我们深入探究其实现原理。   本课程将解析YOLOv3的实现原理和源码，具体内容包括：
    YOLO目标检测原理   神经网络及Darknet的C语言实现，尤其是反向传播的梯度求解和误差计算   代码阅读工具及方法   深度学习计算的利器：BLAS和GEMM   GPU的CUDA编程方法及在Darknet的应用   YOLOv3的程序流程及各层的源码解析   <!--[if !supportLists]-->
     本课程将提供注释后的Darknet的源码程序文件。   除本课程《YOLOv3目标检测：原理与源码解析》外，本人推出了有关YOLOv3目标检测的系列课程，包括：
    《YOLOv3目标检测实战：训练自己的数据集》   《YOLOv3目标检测实战：交通标志识别》   《YOLOv3目标检测：原理与源码解析》   《YOLOv3目标检测：网络模型改进方法》
     建议先学习课程《YOLOv3目标检测实战：训练自己的数据集》或课程《YOLOv3目标检测实战：交通标志识别》，对YOLOv3的使用方法了解以后再学习本课程。

梯度下降优化算法综述.pdf 05-20

梯度下降优化算法虽然越来越流行，但通常都是如此用作黑盒优化器，作为其优势和实践的实际解释弱点很难得到…。 本文旨在为读者提供关于允许她的不同算法的行为的直觉把它们用到。 在本概述过程中，我们会看到不同的变体度下降，总结挑战，介绍最常见的优化算法，在并行和分布式设置中查看体系结构，并进行调查优化梯度下降的其他策略

深度学习原理详解及Python代码实现 12-07

<h3>【为什么要学习这门课程】</h3> 深度学…框架如TensorFlow和Pytorch掩盖了深度学习底层实现方法，那能否能用Python代码从零实现来学习深度学习原理呢？ 本课程就为大家提供了这个可能，有助于深刻理解深度学习原理。 左手原理、右手代码，双管齐下！ 本课程详细讲解深度学习原理并进行Python代码实现深度学习网络。课程内容涵盖感知机、多层感知机、卷积神经网络、循环神经网络，并使用Python 3及Numpy、Matplotlib从零实现上述神经网络。本课程还讲述了神经网络的训练方法与实践技巧，且开展了代码实践演示。课程对于核心内容讲解深入细致，如基于计算图理解反向传播算法，并用数学公式推导反向传播算法；另外还讲述了卷积加速方法im2col。 【课程收获】 本课程力求使学员通过深度学习原理、算法公式及Python代码的对照学习，摆脱框架而掌握深度学习底层实现原理与方法。 本课程将给学员分享深度学习的Python实现代码。课程代码通过Jupyter Notebook演示，可在Windows、ubuntu等系统上运行，且不需GPU支持。 <h3 class="MsoNormal" align="left">【优惠说明】</h3> 课程正在优惠中！   备注：购课后可加入白勇老师课程学习交流QQ群：957519975 <h3>【相关课程】</h3> 学习本课程的前提是会使用Python语言以及Numpy和Matplotlib库。 相关课程链接如下： 《Python编程的术与道：Python语言入门》https://edu.csdn.net/course/detail/27845 《玩转Numpy计算库》https://edu.csdn.net/lecturer/board/28656 《玩转Matplotlib数据绘图库》https://edu.csdn.net/lecturer/board/28720 <h3>【课程内容导图及特色】</h3>

深入浅出--梯度下降法及其实现 Mr.CHEN专栏 3703

转自：https://www.jianshu.com/p/c7e642877b0e深入浅出--梯度下降法及其实现 六尺帐篷 关注2018.01.17 21:06 …数 3001 阅读 1210评论 2喜欢 23赞赏 1梯度下降的场景假设梯度梯度下降算法的数学解释梯度下降算法的实例梯度下降算法的实现Further reading本文将从一个下山的场景开始，先提出梯度下降算法的基本思想，进而从数学上解...

CSDN开发者助手，常用网站自动整合，多种工具一键调用

CSDN开发者助手由CSDN官方开发，集成一键呼出搜索、万能快捷工具、个性化新标签页和官方免广告四大功能。.帮助您提升10倍开发效率！

An overview of gradient descent optimization algorithms 03-11

梯度下降法，是当今最流行的优化（optimization）算法，亦是至今最常用的优化神经网络的方法。本文旨在让你对..不同的优化梯度下降法的算法有一个直观认识，以帮助你使用这些算法。我们首先会考察梯度下降法的各种变体，然后会简要地总结在训练（神经网络或是机器学习算法）的过程中可能遇到的挑战。

深度学习2--梯度下降算法 ddddddddrose的博客 127

import numpy as np def sigmoid(x): """ Calculate sigmoid """ return 1/(1+np.exp(-x)) def sigmoid_prime(x): return …gmoid(x)*(1-sigmoid(x)) learnrate = 0.8 x = np.array([1, 2])...

线性回归 weixin_45728943的博客 47

线性回归常用的地方 贷款预测 销售预测 房价的预测 线性回归就跟函数差不多 原理是这样的 但是细节确实按照情况.来走 是利用回归方程(函数)对一个或多个自变量(特征值)和因变量(目标值)之间关系进行建模的一种分析方式。 房子价格 = 0.02×中心区域的距离 + 0.04×城市一氧化氮浓度 + (-0.12×自住房平均房价) + 0.254×城镇犯罪率 这就算是一

个关系 ...