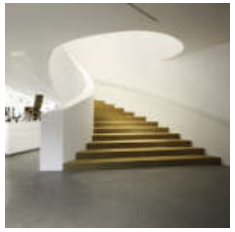
roslei的博客

目录视图 摘要视图 RSS 订阅

个人资料



roslei

关注 发私信

访问： 81624次

积分： 2725

等级： BLOG 5

排名： 第11604名

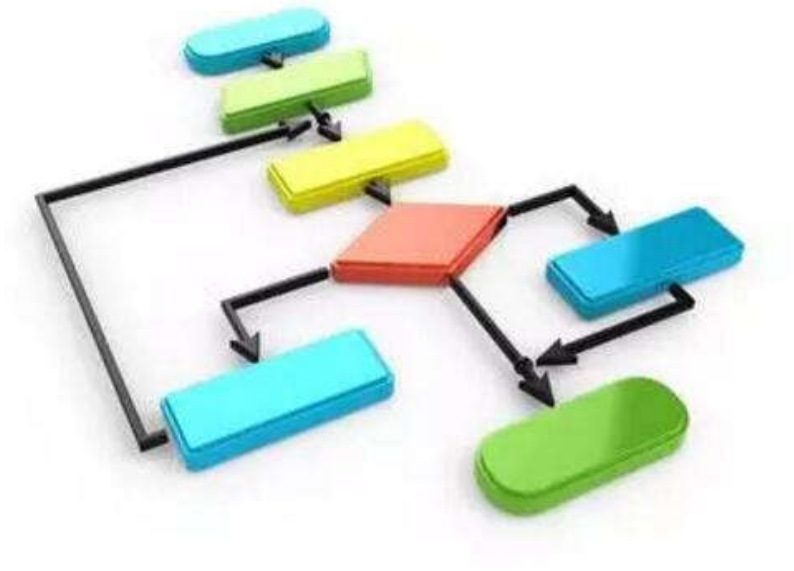
从梯度下降到拟牛顿法：详解训练神经网络的五大学习算法

2017-03-12 21:22 98人阅读 评论(0) 收藏 举报

> 在神经网络中，系统的学习过程一般是由训练算法所主导。而现如今有许多不同的学习算法，它们每一个都有不同的特征和表现。因此本文力图描述清楚五大学习算法的基本概念及优缺点，给读者们阐明最优化在神经网络中的应用。

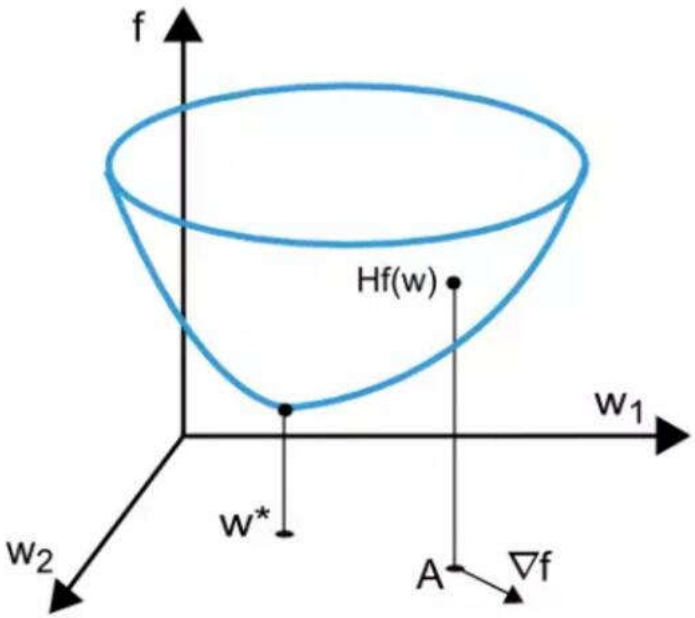
问题形式化

神经网络中的学习过程可以形式化为最小化损失函数问题，该损失函数一般是由训练误差和正则项组成。误差项会衡量神经网络拟合数据集的好坏，也就是拟合数据所产生的误差。正则项主要就是通过给特征权重增加罚项而控制神经网络的有效复杂度，这样可以有效地控制模型过拟合问题。



关闭

训练损失函数取决于神经网络中的自适应参数（偏置项和突触权重）。我们很容易地将神经网络的权重组合成一个 n 维权重向量 w，而训练损失就是以这些权重为变量的函数。下图描述了损失函数 f（w）。



关闭

如上图所示，点 w* 是训练损失函数的极小值点。在任意点 A，损失函数能分别对权重求一阶偏导数和二阶偏导数。损失函数的一阶偏导可以使用梯度算符来表示，其中每一个权重的损失函数梯度表示如下：

$$\nabla_i f(w) = df/dw_i \ (i = 1,\ldots,n)$$

同样，损失函数的二阶偏导可以使用海塞矩阵（Hessian matrix）来表示，以下就是损失函数对权重向量每个元素的二阶偏导数：

$$H_{i,j} f(w) = d^2 f / dw_i \cdot dw_j \ (i,j = 1,\ldots,n)$$

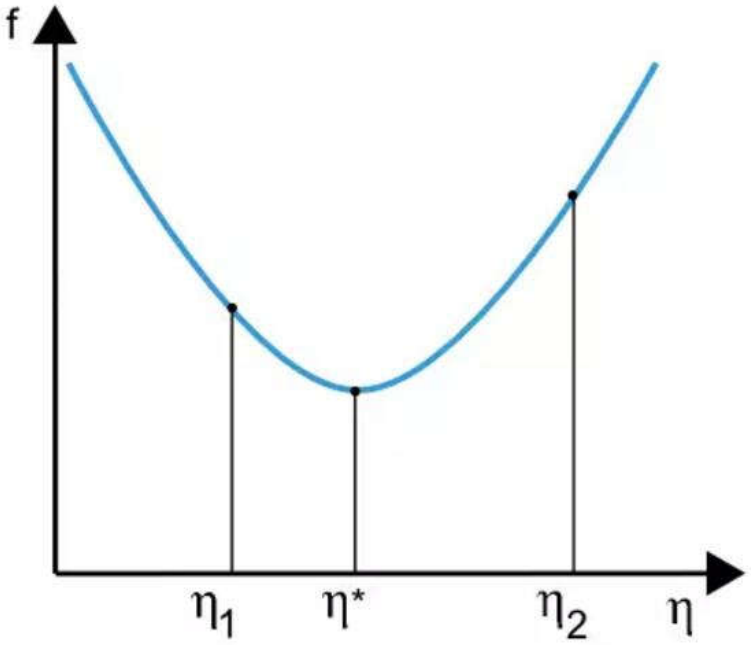
最小化多变量连续可导函数的方法广泛应用于学习过程中，许多常规方法都将这种最优化方法直接应用于神经网络的训练中。

单变量函数优化（**One-dimensional optimization**）

虽然损失函数是由多变量决定的（权重的数量通常十分巨大），但首先理解单变量函数的优化方法是十分重要的。并且实际上单变量优化方法经常应用到神经网络的训练过程中，超参数的调整就可以使用单变量优化法。

在实际模型中，许多训练算法都是首先计算出训练方向 d，然后确定在此训练方向上最小化训练损失 f（η）的学习速率 η。下图就展示了单变量函数 f（η）的优化过程，该优化可求得最优学习速率η*。

关闭



点η1 和点η2 定义了包含单变量函数 f（η）最优点η*的子区间
在该案例中，单变量优化法在给定单变量函数的情况下搜寻函数极小值。其中广泛应用的搜寻算法有黄金分割法（golden section）和 Brent 法。两者都是在减少极小值所在的子区间，直到子区间中两个端点间的距离小于定义的可容忍误差。
多变量函数优化（**Multidimensional optimization**）
神经网络的学习过程可以形式化为求将训练损失函数 f 最小化的参数向量 w*。数学或实际都证明如果神经网络的损失函数达到了极小值，那么梯度也必定为 0 向量。
通常情况下，损失函数为参数的非线性函数，所以找到一个封闭的训练算法（closed training algorithms）求最优解是不可能的。相反，我们考虑通过一系列迭代步在参数空间内搜寻最优解。在每一步迭代中，我们可以通过调整神经网络的参数降低损失函数的值。
通过这种方式，一般我们会由初始参数向量开始（通常为随机初始化）训练神经网络。然后，算法会更新生成一组新参数，训练损失函数也会在每一次算法迭代中使用更新的参数进行函数值的降低。两步迭代之间的的训练损失减少又称之为训练损失衰减率（loss decrement）。最后，当训练过程满足特定的条件或停止标准时，训练算法就会停止迭代，而这个时候的参数也就是最优参数（神经网络中可能是局部最优解），神经网络的性能也由它们所决定。
下面，本文将描述在神经网络中最重要的学习算法。

关闭

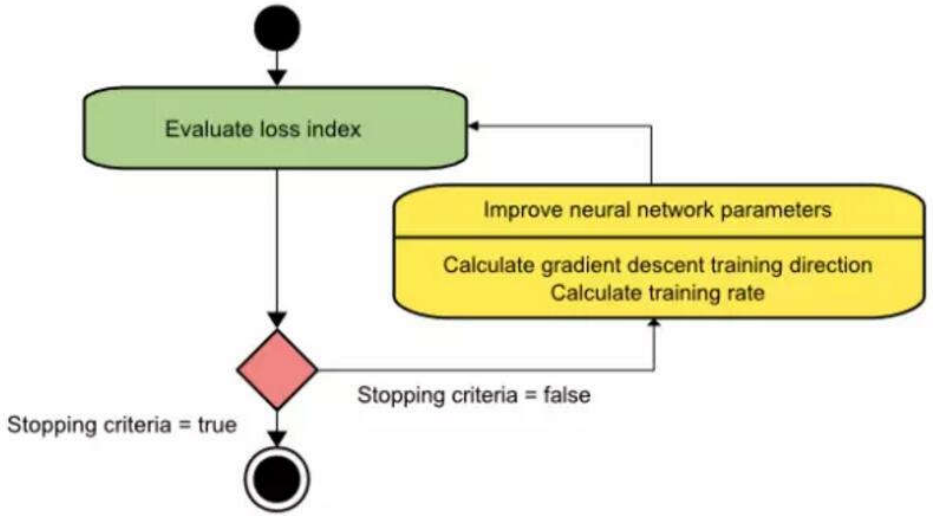
梯度下降

梯度下降，又称为最速下降法是一种非常简单和直观的训练算法。该算法从梯度向量中获取优化信息，因此其为一阶算法（通过一阶偏导求最优权重）。
如果我们指定 f（wi）= fi、∇f（wi）= gi，那么该优化方法由点 w0 开始迭代，满足终止条件之前，就在训练方向 di=-gi 上将 wi 移向 wi+1。因此，梯度下降法就是如下方程式进行迭代。

$$w_{i+1} = w_i - d_i \cdot \eta_i, \quad i=0,1,...$$

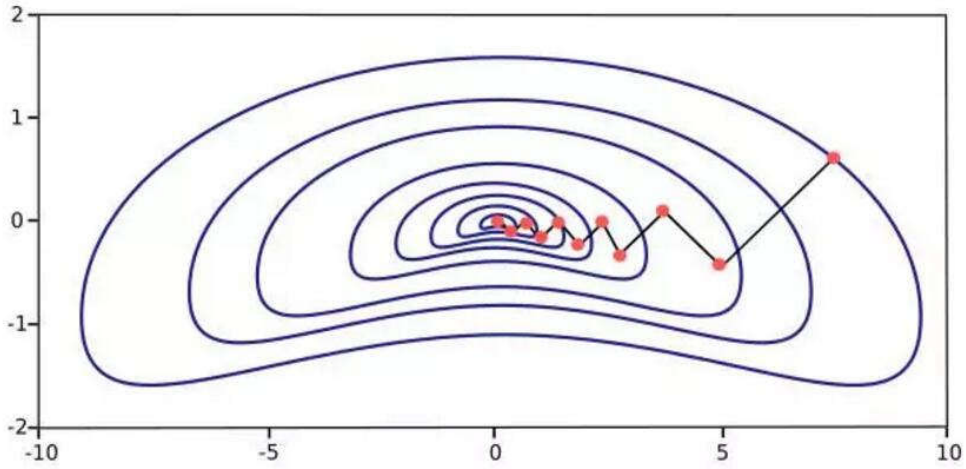
其中参数 η 是学习速率。该学习速率的值可以设定为一个常量也可以沿着训练方向使用单变量优化法求得。通常学习速率的最优值可以在连续迭代步（successive step）上通过线最小化（line minimization）获得。然而，现在还是有很多机器学习模型仅仅只使用固定的学习速率。

下面是一张使用梯度下降算法进行学习的流程图。我们可以看到，参数向量通过两步进行优化：首先，计算梯度下降的训练方向。其次，寻找合适的学习速率。



梯度下降算法也有一些缺点，首先就是其迭代方向会呈现一种锯齿现象，其并不能朝着极小值点径直优化，所以迭代的次数也就多，收敛的速度也就慢。当它的函数梯度图又窄又长时（变量没有归一化，值处于不同的量级），迭代所需要的步数就会更多了。最速下降法确实沿着最陡的梯度下降，损失函数减少得最迅速，但这并不代表梯度下降法或最速下降法会最快收敛（因为锯齿现象）。下图就可以直观地了解到这种锯齿现象，因为非线性函数局部的梯度方向并不一定就是朝着最优点。并且该图还表明，如果横轴量级与纵轴量级有差别时，损失函数梯度图会呈现为一种椭圆形，而如果从椭圆长半轴端点开始下降，那么迭代步数就会很多。

关闭



在训练大规模神经网络时，因为有上万的参数，所以梯度下降法是比较有效的。因为梯度下降算法储存的梯度算符向量规模为 n，而海塞矩阵储存的规模就为 n^2 了，同时梯度和海塞矩阵的计算量也是天差地别。

关闭

牛顿法

牛顿法是二阶算法，因为该算法使用了海塞矩阵（Hessian matrix）求权重的二阶偏导数。牛顿法的目标就是采用损失函数的二阶偏导数寻找更好的训练方向。现在我们将采用如下表示： f(wi) = fi、∇f(wi) = gi 和 Hf(wi) = Hi。在 w0 点使用泰勒级数展开式二次逼近函数 f。

$$f = f_0 + g_0 \cdot (w - w_0) + 0.5 \cdot (w - w_0)^2 \cdot H_0$$

H0 为函数 f 在点 w0 的海塞矩阵。通过将 g 设定为 0，我们就可以找到 f(w) 的极小值，也就得到了以下方程式。

$$g = g_0 + H_0 \cdot (w - w_0) = 0$$

因此，从参数向量 w0 开始，牛顿法服从以下方式进行迭代：

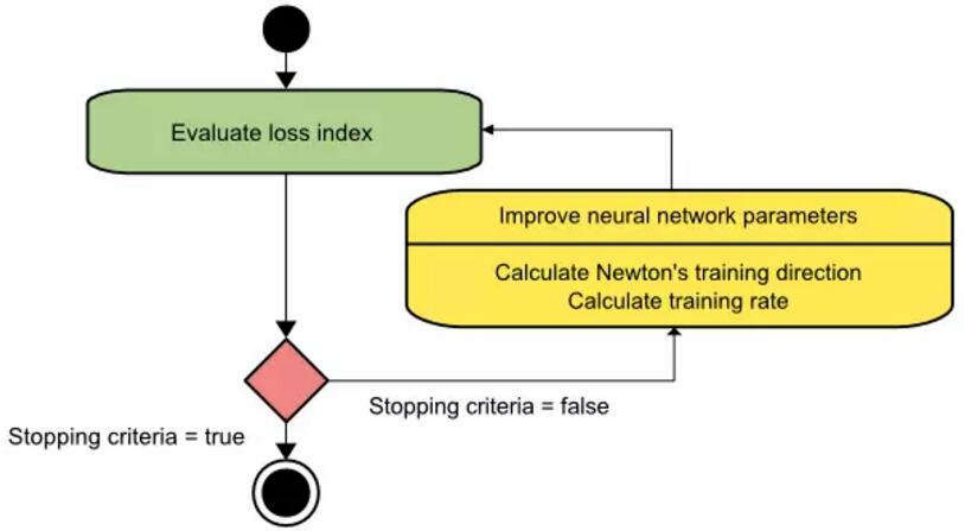
$$w_{i+1} = w_i - H_i^{-1} \cdot g_i, \quad i=0,1,\ldots$$

关闭

向量 Hi-1·gi（参考上式）也就是所说的牛顿下降步（Newton's step）。注意，参数的这些变化将朝着极大值而不是极小值逼近，出现这样的情况是因为海塞矩阵非正定。因此在不能保证矩阵正定的情况下，损失函数并不能保证在每一次迭代中都是减少的。为了防止上述问题，牛顿法的方程式通常可以修改为：

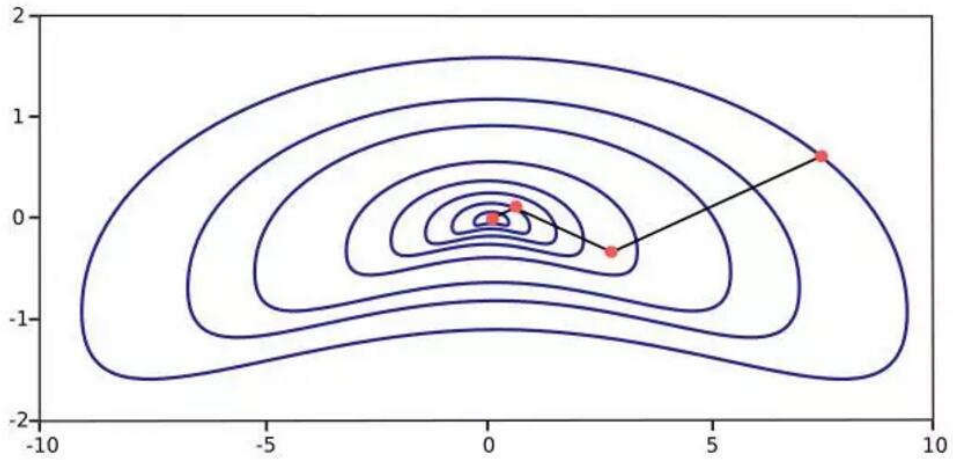
$$w_{i+1} = w_i - (H_i^{-1} \cdot g_i) \cdot \eta_i, \quad i=0,1,\ldots$$

学习速率η同样可是设定为固定常数或通过单变量优化取值。向量 d=Hi-1·gi（参考上式）现在就称为牛顿训练方向（Newton's training direction）。
使用牛顿法的训练过程状态图就如下图所示。从此图可以看出来，系统首先通过获得牛顿训练方向，然后获得合适的学习速率来进行参数的更新优化。



下面的梯度图展示了牛顿法的性能。因为牛顿法是采用其损失函数的二阶偏导数寻找更好的训练下降方向，所以它相比梯度下降只要更少的迭代次数就能下降到损失函数的极小值，因此函数收敛速度也会大幅度地加快。

关闭



然而，牛顿法的困难之处在于其计算量，因为对海塞矩阵及其逆的精确求值在计算量方面是十分巨大的。

共轭梯度法（**Conjugate gradient**）

共轭梯度法可认为是梯度下降法和牛顿法的中间物。该算法希望能加速梯度下降的收敛速度，同时避免使用海塞矩阵进行求值、储存和求逆获得必要的优化信息。
在共轭梯度训练算法中，因为是沿着共轭方向（conjugate directions）执行搜索的，所以通常该算法要比沿着梯度下降方向优化收敛得更迅速。共轭梯度法的训练方向是与海塞矩阵共轭的。
我们用 d 表示训练方向向量，然后从初始参数向量 w0 和初始训练方向向量 d0=-g0 开始，共轭梯度法所构建的训练方向序列为：

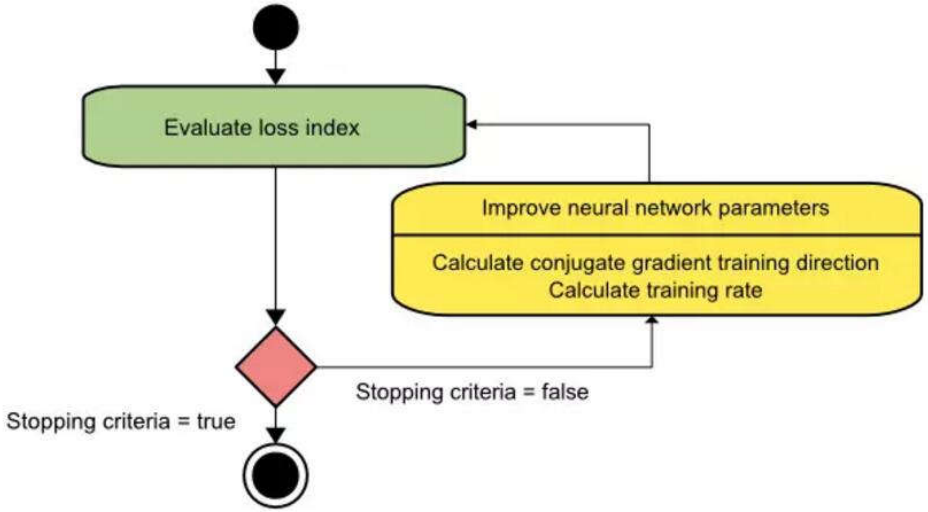
$$d_{i+1} = g_{i+1} + d_i \cdot \gamma_i, \quad i=0,1,\ldots$$

关闭

在上式中，γ 称之为共轭参数，并且有一些方法计算这个参数。两种最常用的方法是源自 Fletcher、Reeves 和 Polak 、Ribiere。对于所有的共轭梯度算法，训练方向周期性地重置为负梯度向。
参数通过下面的表达式得以更新和优化。通常学习速率η可使用单变量函数优化方法求得。

$$w_{i+1} = w_i + d_i \cdot \eta_i, \quad i=0,1,\ldots$$

共轭梯度法的训练过程流程图就如下所示。从图中我们可以看出来模型是通过第一次计算共轭梯度训练方向而优化参数的，然后再寻找适当的学习速率。



共轭梯度法已经证实其在神经网络中要比梯度下降法有效得多。并且由于共轭梯度法并没有要求使用海塞矩阵，所以在大规模神经网络中其还是可以做到很好的性能。

关闭

拟牛顿法（**Quasi-Newton method**）

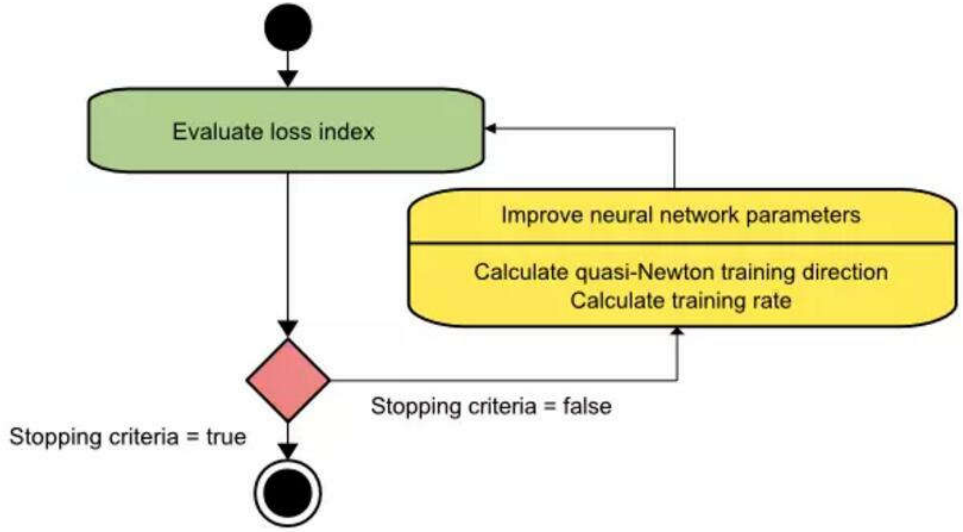
因为需要很多的操作求解海塞矩阵的值还有计算矩阵的逆，应用牛顿法所产生的计算量是十分巨大的。因此有一种称之为拟牛顿法（quasi-Newton）或变量矩阵法来解决这样的缺点。这些方法并不是直接计算海塞矩阵然后求其矩阵的逆，拟牛顿法是在每次迭代的时候计算一个矩阵，其逼近海塞矩阵的逆。最重要的是，该逼近值只是使用损失函数的一阶偏导来计算。
海塞矩阵由损失函数的二阶偏导组成，拟牛顿法背后的思想主要是仅使用损失函数的一阶偏导数，通过另一矩阵 G 逼近海塞矩阵的逆。拟牛顿法的公式可以表示为：

$$w_{i+1} = w_i - (G_i \cdot g_i) \cdot \eta_i, \quad i=0,1,\ldots$$

学习速率 η可以设定为固定常数，也可以通过单变量函数优化得到。其中矩阵 G 逼近海塞矩阵的逆，且有[illegible]进行逼近。通常，最常用的两种方式是 Davidon–Fletcher–Powell formula (DFP) 和 the Broyden–Fletcher–Goldfarb–Shanno formula (BFGS)。

拟牛顿法的训练过程流程图就如下所示。从图中我们可以看出来模型是通过第一次计算拟牛顿训练方向而优化参数的，然后再寻找适当的学习速率。

拟牛顿法适用于绝大多数案例中：它比梯度下降和共轭梯度法收敛更快，并且也不需要确切地计算海塞矩阵及其逆矩阵。



关闭

Levenberg-Marquardt 算法

Levenberg-Marquardt 算法，也称之为衰减最小二乘法（damped least-squares method），该算法的损失函数采用平方误差和的形式。该算法的执行也不需要计算具体的海塞矩阵，它仅仅只是使用梯度向量和雅可比矩阵（Jacobian matrix）。

该算法的损失函数如下方程式所示为平方误差和的形式：

$$f = \sum e_i^2, \quad i=0,\ldots,m$$

在上式中，m 是数据集样本的数量。
我们可以定义损失函数的雅可比矩阵以误差对参数的偏导数为元素，如下方程式所示：

$$J_{i,j} f(w) = de_i/dw_j \ (i = 1,\ldots,m \ \& \ j = 1,\ldots,n)$$

关闭

其中 m 是数据集样本的数量，n 是神经网络的参数数量。那么雅可比矩阵就是 m×n 阶矩阵。
损失函数的梯度向量就可以按如下计算出来：

$$\nabla f = 2 J^T \cdot e$$

e 在这里是所有误差项的向量。
最终，我们可以用以下表达式逼近海塞矩阵：

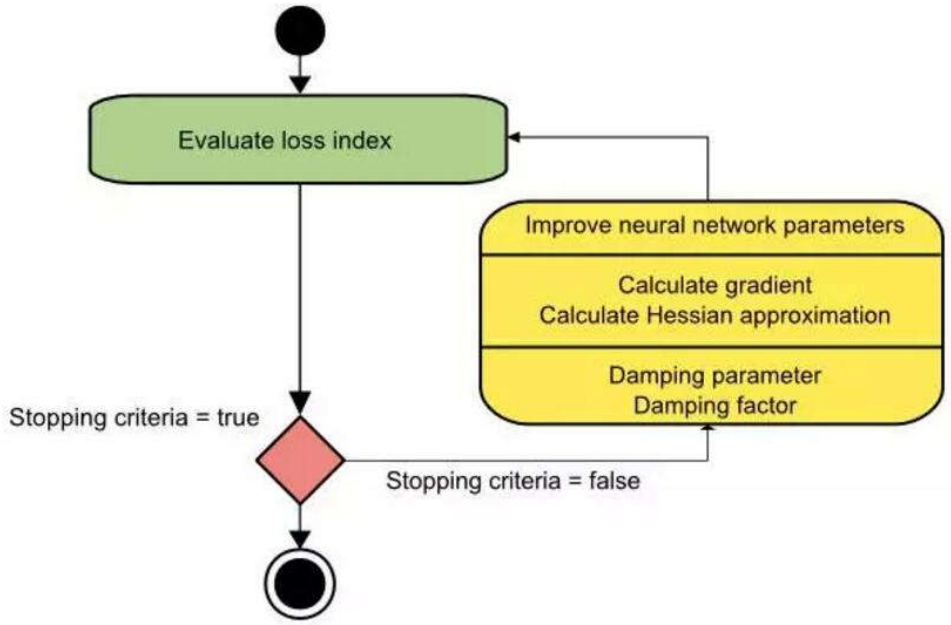
$$Hf \approx 2\,J^T \cdot J + \lambda I$$

其中λ为衰减因子，它确保了海塞矩阵的正定性（positiveness），I 是单位矩阵。
下面的表达式定义了 Levenberg-Marquardt 算法中参数的更新和优化过程：

$$w_{i+1} = w_i - (J_i^T \cdot J_i + \lambda_i I)^{-1} \cdot (2\,J_i^T \cdot e_i), \quad i=0,1,\ldots$$

关闭

当衰减参数λ为 0 时，Levenberg-Marquardt 算法就是使用海塞矩阵逼近值的牛顿法。而当 λ很大时，该算法就近似于采用很小学习速率的梯度下降法。如果进行迭代导致了损失函数上升，衰减因子λ就会增加。如果损失函数下降，那么λ就会下降，从而 Levenberg-Marquardt 算法更接近于牛顿法。该过程经常用于加速收敛到极小值点。
使用 Levenberg-Marquardt 法的神经网络训练过程状态图就如下图所示。第一步就是计算损失函数、梯度和海塞矩阵逼近值，随后再在每次迭代降低损失中调整衰减参数。

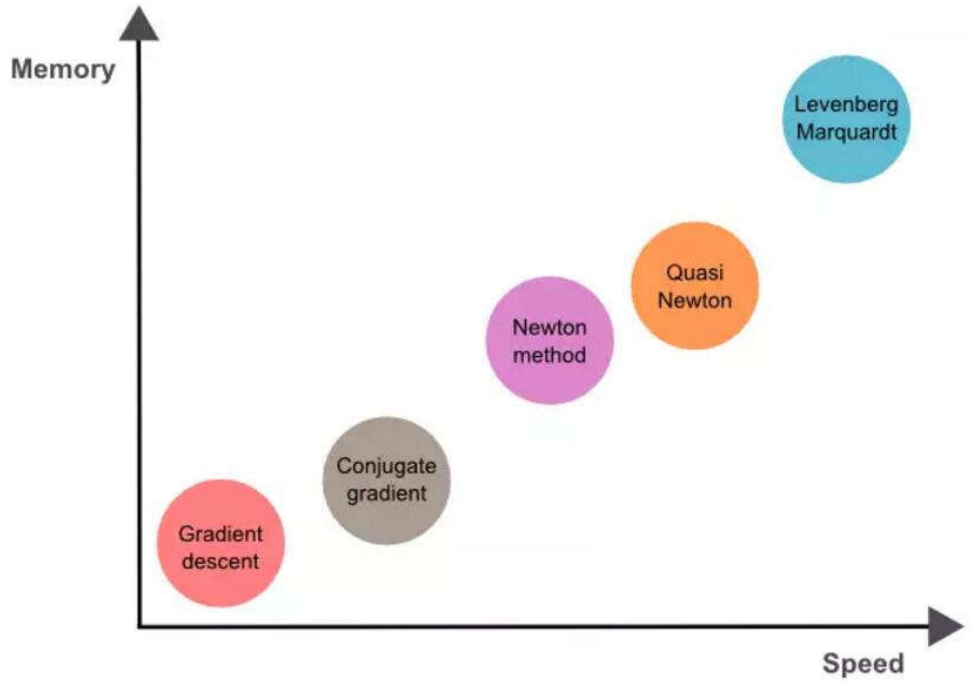


关闭

正如我们所了解到的，Levenberg-Marquardt 算法是为平方误差和函数所定制的。这就让使用这种误差度量的神经网络训练地十分迅速。然而 Levenberg-Marquardt 算法还有一些缺点，第一就是其不能用于平方根误差或交叉熵误差（cross entropy error）等函数，此外该算法还和正则项不兼容。最后，对于大型数据集或神经网络，雅可比矩阵会变得十分巨大，因此也需要大量的内存。所以我们在大型数据集或神经网络中并不推荐采用 Levenberg-Marquardt 算法。

内存与收敛速度的比较

下图展示了所有上文所讨论的算法，及其收敛速度和内存需求。其中收敛速度最慢的是梯度下降算法，但该算法同时也只要求最少的内存。相反，Levenberg-Marquardt 算法可能是收敛速度最快的，但其同时也要求最多的内存。比较折衷方法是拟牛顿法。



关闭

总而言之，如果我们的神经网络有数万参数，为了节约内存，我们可以使用梯度下降或共轭梯度法。如果我们需要训练多个神经网络，并且每个神经网络都只有数百参数、数千样本，那么我们可以考虑 Levenberg-Marquardt 算法。而其余的情况，拟牛顿法都能很好地应对。

原文地址：http://www.jiqizhixin.com/article/2448

- 上一篇 逻辑回归
- 下一篇 易用的深度学习框架Keras简介

参考知识库

猜你在找

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack VPN Spark ERP
IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery BI HTML5 Spring Apache .NET
API HTML SDK IIS Fedora XML LBS Unity Splashtop UML components Windows Mobile Rails
QEMU KDE Cassandra CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App
SpringSide Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
Angular Cloud Foundry Redis Scala Django Bootstrap

关闭