

从梯度下降到 Adam！一文看懂各种神经网络优化算法

Datawhale 2022-05-04 21:00

Datawhale干货

编译：王小新，来源：量子位



Datawhale

一个专注于AI领域的开源组织，汇聚了众多优秀学习者，愿景-for the learner，和学习者一起成长。

441篇原创内容

公众号

在调整模型更新权重和偏差参数的方式时，你是否考虑过哪种优化算法能使模型产生更好且更快的效果？应该用梯度下降，随机梯度下降，还是Adam方法？

这篇文章介绍了不同优化算法之间的主要区别，以及如何选择最佳的优化方法。

什么是优化算法？

优化算法的功能，是通过改善训练方式，来最小化(或最大化)损失函数E(x)。

模型内部有些参数，是用来计算测试集中目标值Y的真实值和预测值的偏差程度的，基于这些参数，就形成了损失函数E(x)。

比如说，权重(W)和偏差(b)就是这样的内部参数，一般用于计算输出值，在训练神经网络模型时起到主要作用。

在有效地训练模型并产生准确结果时，模型的内部参数起到了非常重要的作用。这也是为什么我们应该用各种优化策略和算法，来更新和计算影响模型训练和模型输出的网络参数，使其逼近或达到最优值。

优化算法分为两大类：

1. 一阶优化算法

这种算法使用各参数的梯度值来最小化或最大化损失函数E(x)。**最常用的一阶优化算法是梯度下降。**

函数梯度：导数dy/dx的多变量表达式，用来表示y相对于x的瞬时变化率。往往为了计算多变量函数的导数时，会用梯度取代导数，并使用偏导数来计算梯度。梯度和导数之间的一个主要区别是函数的梯度形成了一个向量场。

因此，对单变量函数，使用导数来分析；而梯度是基于多变量函数而产生的。更多理论细节在这里不再进行详细解释。

2. 二阶优化算法

二阶优化算法使用了二阶导数(也叫做**Hessian方法**)来最小化或最大化损失函数。由于二阶导数的计算成本很高，所以这种方法并没有广泛使用。

详解各种神经网络优化算法

梯度下降

在训练和优化智能系统时，梯度下降是一种最重要的技术和基础。梯度下降的功能是：

通过寻找最小值，控制方差，更新模型参数，最终使模型收敛。

网络更新参数的公式为： $\theta = \theta - \eta \times \nabla(\theta) \cdot J(\theta)$ ，其中 η 是学习率， $\nabla(\theta) \cdot J(\theta)$ 是损失函数J(θ)的梯度。

这是在神经网络中最常用的优化算法。

如今，梯度下降主要用于在神经网络模型中进行权重更新，即在一个方向上更新和调整模型的参数，来最小化损失函数。

2006年引入的反向传播技术，使得训练深层神经网络成为可能。反向传播技术是先在前向传播中计算输入信号的乘积及其对应的权重，然后将激活函数作用于这些乘积的总和。这种将输入信号转换为输出信号的方式，是一种对复杂非线性函数进行建模的重要手段，并引入了非线性激活函数，使得模型能够学习到几乎任意形式的函数映射。然后，在网络的反向传播过程中回传相关误差，使用梯度下降更新权重值，通过计算误差函数E相对于权重参数W的梯度，在损失函数梯度的相反方向上更新权重参数。

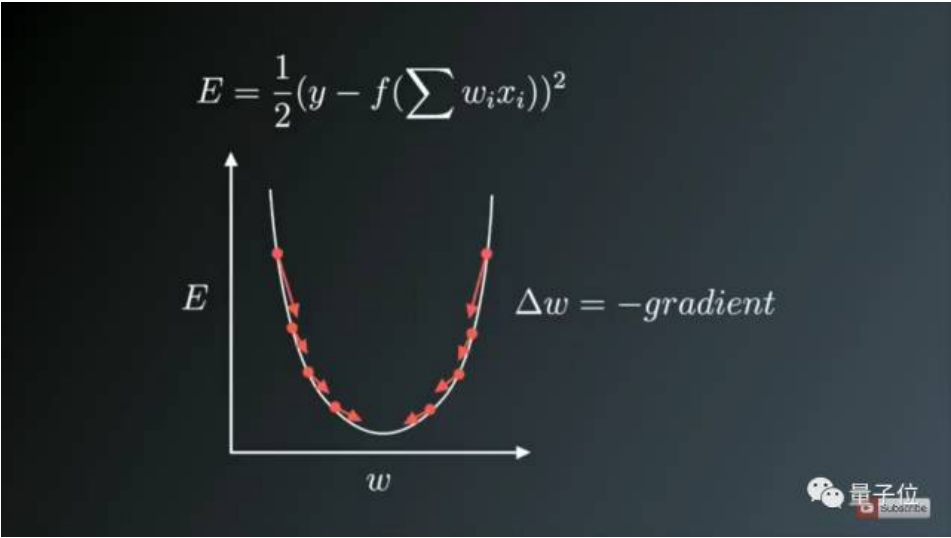


图1：权重更新方向与梯度方向相反

图1显示了权重更新过程与梯度矢量误差的方向相反，其中U形曲线为梯度。要注意到，当权重值W太小或太大时，会存在较大的误差，需要更新和优化权重，使其转化为合适值，所以我们试图在与梯度相反的方向找到一个局部最优值。

梯度下降的变体

传统的批量梯度下降将计算整个数据集梯度，但只会进行一次更新，因此在处理大型数据集时速度很慢且难以控制，甚至导致内存溢出。

权重更新的快慢是由学习率η决定的，并且可以在凸面误差曲面中收敛到全局最优值，在非凸曲面中可能趋于局部最优值。

使用标准形式的批量梯度下降还有一个问题，就是在训练大型数据集时存在冗余的权重更新。

标准梯度下降的上述问题在随机梯度下降方法中得到了解决。

1. 随机梯度下降(SDG)

随机梯度下降（Stochastic gradient descent，SGD）对每个训练样本进行参数更新，每次执行都进行一次更新，且执行速度更快。

$\theta = \theta - \eta \cdot \nabla(\theta) \times J(\theta;x(i);y(i))$ ，其中x(i)和y(i)为训练样本。

频繁的更新使得参数间具有高方差，损失函数会以不同的强度波动。这实际上是一件好事，因为它有助于我们发现新的和可能更优的局部最小值，而标准梯度下降将只会收敛到某个局部最优值。

但SGD的问题是，由于频繁的更新和波动，最终将收敛到最小限度，并会因波动频繁存在超调量。

虽然已经表明，当缓慢降低学习率η时，标准梯度下降的收敛模式与SGD的模式相同。

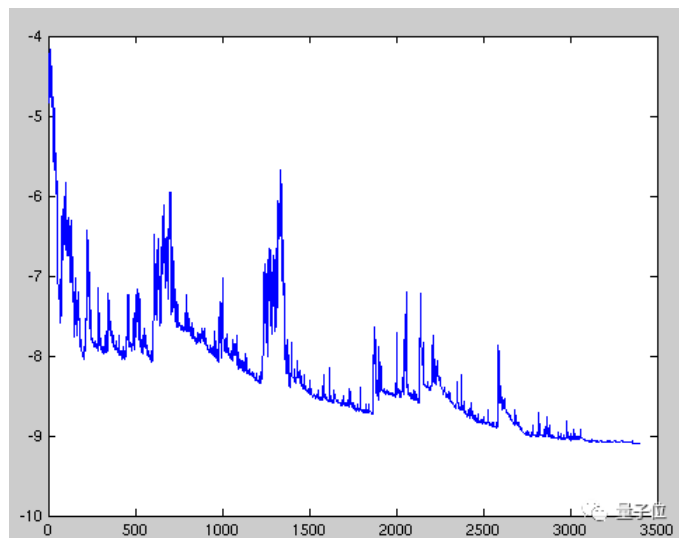


图2：每个训练样本中高方差的参数更新会导致损失函数大幅波动，因此我们可能无法获得给出损失函数的最小值。

另一种称为“小批量梯度下降”的变体，则可以解决高方差的参数更新和不稳定收敛的问题。

2. 小批量梯度下降

为了避免SGD和标准梯度下降中存在的问题，一个改进方法为小批量梯度下降（Mini Batch Gradient Descent），因为对每个批次中的n个训练样本，这种方法只执行一次更新。

使用小批量梯度下降的优点是：

- 1) 可以减少参数更新的波动，最终得到效果更好和更稳定的收敛。
- 2) 还可以使用最新的深度学习库中通用的矩阵优化方法，使计算小批量数据的梯度更加高效。
- 3) 通常来说，小批量样本的大小范围是从50到256，可以根据实际问题而有所不同。
- 4) 在训练神经网络时，通常都会选择小批量梯度下降算法。

这种方法有时候还是被成为SGD。

使用梯度下降及其变体时面临的挑战

1. 很难选择出合适的学习率。太小的学习率会导致网络收敛过于缓慢，而学习率太大可能会影响收敛，并导致损失函数在最小值上波动，甚至出现梯度发散。
2. 此外，相同的学习率并不适用于所有的参数更新。如果训练集数据很稀疏，且特征频率非常不同，则不应该将其全部更新到相同的程度，但是对于很少出现的特征，应使用更大的更新率。
3. 在神经网络中，最小化非凸误差函数的另一个关键挑战是避免陷于多个其他局部最小值中。实际上，问题并非源于局部极小值，而是来自鞍点，即一个维度向上倾斜且另一维度向下倾斜的点。这些鞍点通常被相同误差值的平面所包围，这使得SGD算法很难脱离出来，因为梯度在所有维度上接近于零。

进一步优化梯度下降

现在我们要讨论用于进一步优化梯度下降的各种算法。

1. 动量

SGD方法中的高方差振荡使得网络很难稳定收敛，所以有研究者提出了一种称为动量（Momentum）的技术，**通过优化相关方向的训练和弱化无关方向的振荡，来加速SGD训练**。换句话说，这种新方法将上个步骤中更新向量的分量 γ 添加到当前更新向量。

$$V(t) = \gamma V(t-1) + \eta \nabla(\theta) \cdot J(\theta)$$

最后通过 $\theta = \theta - V(t)$ 来更新参数。

动量项 γ 通常设定为0.9，或相近的某个值。

这里的动量与经典物理学中的动量是一致的，就像从山上投出一个球，在下落过程中收集动量，小球的速度不断增加。

在参数更新过程中，其原理类似：

- 1) 使网络能更优和更稳定的收敛；
- 2) 减少振荡过程。

当其梯度指向实际移动方向时，动量项 γ 增大；当梯度与实际移动方向相反时， γ 减小。这种方式意味着动量项只对相关样本进行参数更新，减少了不必要的参数更新，从而得到更快且稳定的收敛，也减少了振荡过程。

2. Nesterov梯度加速法

一位名叫Yurii Nesterov研究员，认为动量方法存在一个问题：

如果一个滚下山坡的球，盲目沿着斜坡下滑，这是非常不合适的。一个更聪明的球应该要注意到它将要去哪，因此在上坡再次向上倾斜时小球应该进行减速。

实际上，当小球达到曲线上的最低点时，动量相当高。由于高动量可能会导致其完全地错过最小值，因此小球不知道何时进行减速，故继续向上移动。

Yurii Nesterov在1983年发表了一篇关于解决动量问题的论文，因此，我们把这种方法叫做Nestrov梯度加速法。

在该方法中，他提出先根据之前的动量进行大步跳跃，然后计算梯度进行校正，从而实现参数更新。这种预更新方法能防止大幅振荡，不会错过最小值，并对参数更新更加敏感。

Nesterov梯度加速法（NAG）是一种赋予了动量项预知能力的方法，通过使用动量项 $\gamma V(t-1)$ 来更改参数 θ 。通过计算 $\theta - \gamma V(t-1)$ ，得到下一位置的参数近似值，这里的参数是一个粗略的概念。因此，我们不是通过计算当前参数 θ 的梯度值，而是通过相关参数的大致未来位置，来有效地预知未来：

$V(t) = \gamma V(t-1) + \eta \nabla J(\theta - \gamma V(t-1))$ ，然后使用 $\theta = \theta - V(t)$ 来更新参数。

现在，我们通过使网络更新与误差函数的斜率相适应，并依次加速SGD，也可根据每个参数的重要性来调整和更新对应参数，以执行更大或更小的更新幅度。

3. Adagrad方法

Adagrad方法是通过参数来调整合适的学习率 η ，对稀疏参数进行大幅更新和对频繁参数进行小幅更新。因此，Adagrad方法非常适合处理稀疏数据。

在时间步长中，Adagrad方法基于每个参数计算的过往梯度，为不同参数 θ 设置不同的学习率。

先前，每个参数 $\theta(i)$ 使用相同的学习率，每次会对所有参数 θ 进行更新。在每个时间步 t 中，Adagrad方法为每个参数 θ 选取不同的学习率，更新对应参数，然后进行向量化。为了简单起见，我们把在 t 时刻参数 $\theta(i)$ 的损失函数梯度设为 $g(t,i)$ 。

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


量子位

图3：参数更新公式

Adagrad方法是在每个时间步中，根据过往已计算的参数梯度，来为每个参数 $\theta(i)$ 修改对应的学习率 η 。

Adagrad方法的主要好处是，不需要手工来调整学习率。大多数参数使用了默认值0.01，且保持不变。

Adagrad方法的主要缺点是，学习率 η 总是在降低和衰减。

因为每个附加项都是正的，在分母中累积了多个平方梯度值，故累积的总和在训练期间保持增长。这反过来又导致学习率下降，变为很小数量级的数字，该模型完全停止学习，停止获取新的额外知识。

因为随着学习速度的越来越小，模型的学习能力迅速降低，而且收敛速度非常慢，需要很长的训练和学习，即学习速度降低。

另一个叫做Adadelta的算法改善了这个学习率不断衰减的问题。

4. AdaDelta方法

这是一个AdaGrad的延伸方法，它倾向于解决其学习率衰减的问题。Adadelta不是累积所有之前的平方梯度，而是将累积之前梯度的窗口限制到某个固定大小w。

与之前无效地存储w先前的平方梯度不同，梯度的和被递归地定义为所有先前平方梯度的衰减平均值。作为与动量项相似的分数量γ，在t时刻的滑动平均值Eg²仅仅取决于先前的平均值和当前梯度值。

Eg²=γ.Eg²+(1-γ).g²(t)，其中γ设置为与动量项相近的值，约为0.9。

Δθ(t)=-η.g(t,i).

θ(t+1)=θ(t)+Δθ(t)

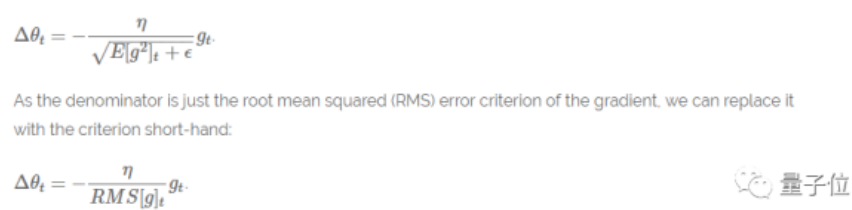


图4：参数更新的最终公式

AdaDelta方法的另一个优点是，已经不需要设置一个默认的学习率。

目前已完成的改进

- 1) 为每个参数计算出不同学习率；
- 2) 也计算了动量项momentum；
- 3) 防止学习率衰减或梯度消失等问题的出现。

还可以做什么改进？

在之前的方法中计算了每个参数的对应学习率，但是为什么不计算每个参数的对应动量变化并独立存储呢？这就是Adam算法提出的改良点。

Adam算法

Adam算法即自适应时刻估计方法（Adaptive Moment Estimation），能计算每个参数的自适应学习率。这个方法不仅存储了AdaDelta先前平方梯度的指数衰减平均值，而且保持了先前梯度M(t)的指数衰减平均值，这一点与动量类似：

M(t)为梯度的第一时刻平均值，V(t)为梯度的第二时刻非中心方差值。

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}.$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}.$$

量子位

图5：两个公式分别为梯度的第一个时刻平均值和第二个时刻方差

则参数更新的最终公式为：

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

量子位

图6：参数更新的最终公式

其中， β_1 设为0.9， β_2 设为0.9999， ϵ 设为 10^{-8} 。

在实际应用中，Adam方法效果良好。与其他自适应学习率算法相比，其收敛速度更快，学习效果更为有效，而且可以纠正其他优化技术中存在的问题，如学习率消失、收敛过慢或是高方差的参数更新导致损失函数波动较大等问题。

对优化算法进行可视化

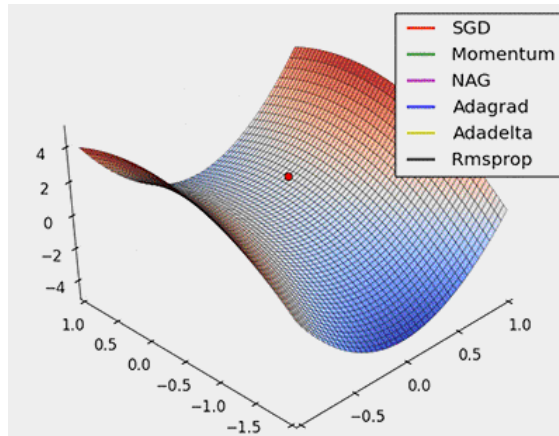


图7：对鞍点进行SGD优化

从上面的动画可以看出，自适应算法能很快收敛，并快速找到参数更新中正确的目标方向；而标准的SGD、NAG和动量项等方法收敛缓慢，且很难找到正确的方向。

结论

我们应该使用哪种优化器？

在构建神经网络模型时，选择出最佳的优化器，以便快速收敛并正确学习，同时调整内部参数，最大程度地最小化损失函数。

Adam在实际应用中效果良好，超过了其他的自适应技术。

如果输入数据集比较稀疏，SGD、NAG和动量项等方法可能效果不好。因此对于稀疏数据集，应该使用某种自适应学习率的方法，且另一好处为不需要人为调整学习率，使用默认参数就可能获得最优值。

如果想使训练深层网络模型快速收敛或所构建的神经网络较为复杂，则应该使用Adam或其他自适应学习速率的方法，因为这些方法的实际效果更优。

希望你能通过这篇文章，很好地理解不同优化算法间的特性差异。

相关链接：

二阶优化算法：

<https://web.stanford.edu/class/msande311/lecture13.pdf>

Nesterov梯度加速法：<http://cs231n.github.io/neural-networks-3/>

Datawhale
和学习者一起成长

一个专注于AI的开源组织，让学习不再孤独



长按扫码关注

干货学习，点赞三连↓

发表于浙江

喜欢此内容的人还喜欢

有人声称「解决了」MNIST与CIFAR 10，实现了100%准确率
机器之心

复旦博士写了130行代码，2分钟解决繁琐核酸报告核查
量子位

我是吴恩达：人在美国，刚上知乎，先答个「如何系统学习机器学习」
量子位