NLP方向总结-优化器

SGD：随机梯度下降(stochastic gradient descent)

梯度下降法(BGD、SGD),即普通GD优化器、缺点主要是收敛速度慢，可能在鞍点处震荡

theta_t+1 = theta_t − η *梯度

梯度可以理解为小球滚动的方向，学习率理解为小球滚动的步长

Adagrad调整的是学习率，即步长

$$W_{n e w}=W_{o l d}-\alpha * \frac{\partial(\text { Loss })}{\partial\left(W_{o l d}\right)}$$

任选一个样本的梯度来更新模型参数，

特点：因为每次更新参数的时候都是选择一个样本的梯度来进行更新，当样本为正常样本的时候，更新就比较快，但是当样本为异常样本的时候，可能会出现反方向的更新；正常情况下，数据集中是正常样本居多，所以导致SGD具有收敛快、在收敛的位置可能出现震动的情况。

高方差性

在一次更新中计算整个数据集的梯度，计算速度非常慢。
它需要大量的内存，而且计算成本很高。

优化器

优化器是用于最小化误差函数（损失函数）或最大化生产效率的算法或方法

定义

使神经网络的损失函数f最小化，就像在山上的山坡上行走一样。
你随机地初始化你的网络的权重，这相当于从山上的一个随机点开始。你的目标是尽快达到损失函数的良好最小值（山谷）。
在每一步之前，你要计算梯度∇f（确定山坡在哪个方向倾斜度最大），并向相反的方向走一步。
新的权重w(t)等于旧的权重w(t−1)减去梯度乘以学习率α。

理解

梯度下降进入局部最小值的步骤有多大/有多小是由学习率决定的，学习率决定了我们走向最优权重的速的快慢，

学习率

优化器选择

如果数据是稀疏的，使用自适用方法，即Adagrad、Adadelta、RMSprop、Adam。

各优化器对比图

https://miro.medium.com/max/1240/1*XVFmo9NxLnwDr3Sxz Ky-rA.gif

https://miro.medium.com/max/1240/1*SjtKOauOXFVJWRR7ICt HiA.gif

发展历程

SGD->SGDM->NAG->AdaGrad->AdaDelta->Adam->NAdam

Adam

https://arxiv.org/abs/1412.6980

计算过去梯度的指数加权平均值，将其存储在变量m和mcorrect
计算过去的梯度的平方的指数加权值，存储在变量v和vcorrect中
利用以上计算出来的值，更新模型参数

Adam优化是一种随机梯度下降法，它是基于对一阶和二阶矩的自适应估计。
Adam可以认为是 RMSprop 和 Adadelta的结合。和 RMSprop对二阶动量使用指数移动平均类似，Adam 中对一阶动量也是用指数移动平均计算, gt 和v1都是动态的.

它是一种为每个参数计算自适应学习率的方法。它既存储过去梯度的衰减平均值，类似于动量，也存储过去平方梯度的衰减平均值

缺点是在最后微调的时候容易震荡

动量优化器的优点：
增加了梯度修正方向的稳定性，减小突变。
在梯度方向比较稳定的区域，小球滚动会越来越快（当然，因为alpha<1，其有一个速度上限）这有助于，小球快速冲过平坦区域，加快收敛。
带有惯性的，小球更容易滚过一些狭窄的局部极值。

动量优化器的缺点：
学习率eta以及动量alpha仍需手动设置，这往往需要较多的实验来确定合适的值。

1. 计算目标函数关于参数的梯度
2.根据历史梯度计算一阶和二阶动量
3.更新模型参数

计算效率高。对内存的要求很小。

当梯度变得稀疏时，Adam会比RMSprop表现得更好。

Adaptive Moment Estimation

$$\begin{aligned} &m(t)=\beta 1 \cdot m(t-1)+(1-\beta 1) \cdot g(t) \\ &v(t)=\beta 2 \cdot v(t-1)+(1-\beta 2) \cdot g(t)^{2} \end{aligned}$$

$$\mathrm{w}_{\mathrm{t}} =\mathrm{w}_{\mathrm{t}-1}-\frac{\eta}{\sqrt{\mathrm{S}_{\mathrm{dw}_{\mathrm{t}}}}-\varepsilon} * \mathrm{~V}_{\mathrm{dw}_{\mathrm{t}}}$$

$$b_{\mathrm{t}} =b_{\mathrm{t}-1}-\frac{\eta}{\sqrt{\mathrm{S}_{b_{\mathrm{t}}}}-\varepsilon} * \mathrm{~V}_{b_{\mathrm{t}}}$$

Adam跟踪梯度的（指数移动）平均值（称为第一时刻，从现在起表示为m）和梯度的平方（称为原始第二时刻，从现在起表示为v）。

Adafactor

Adafactor: Adaptive learning rates with sublinear memory cost.

PaLM, PICARD模型使用

带有 "参数缩放 "的Adam,参数矩阵的均方根来缩放学习率

参数缩放的好处是，在不同规模层上的参数矩阵（嵌入和层范式规模），其学习率不会以同样的速度缩减。

AdamW

伪代码： https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html

论文：https://arxiv.org/abs/1711.05101

改进的Adam版本，称为AdamW，产生的模型泛化效果更好，因此能够与SGD竞争，同时训练速度更快

其中权重衰减只在控制了参数的步长后进行。权重衰减或正则化项不会在移动平均数中结束，因此只与权重本身成正比

Adam with weight decay

SparseAdam

执行适用于稀疏张量的Adam算法的懒惰版本。

在这个实体中，只有显示在梯度中的时刻被更新，并且只有梯度的那些部分被应用到参数中。

https://pytorch.org/docs/stable/generated/torch.optim.SparseAdam.html

Adamax

它是Adam的一个变种，基于无穷大范式。Adamax有时优于adam，特别是在有嵌入的模型中。

https://arxiv.org/abs/1412.6980

NAdam

就像Adam本质上是具有动量的RMSprop，Nadam是具有 Nesterov动量的Adam。

超参数： 学习率，一阶动量指数衰减率，权重正则指数衰减率，epsilon

https://openreview.net/forum?id=OM0jvwB8jIp57ZJjtNEZ

RAdam

https://arxiv.org/abs/1908.03265

BGD

特点：因为将所有样本的梯度和来更新一次模型参数，所以异常样本对于模型参数更新影响比较小

用样本的梯度和来更新模型参数

MBGD

任选K个样本的梯度和来更新模型参数，当K=1的时候其实就是SGD，当K等于所有样本的时候，其实就是BGD

如果学习率太小，收敛速度会很慢。如果它过大，损失函数就会振荡，甚至在最小值上出现偏差。

小批次梯度下降并不能保证良好的收敛性。

坐标轴下降法CD

对于函数函数是没有连续可导的要求，固定n−1的模型参数，求剩下的另一个模型参数进行最小值的更新。

和GD的区别：
更新方向不一样，在GD中使用梯度负方向来进行参数更新，而在CD中，使用坐标轴方向来进行梯度更新，
GD要求函数连续可导
CD不要求连续可导

SGDM

SGD with Momentum: 一阶动量的随机梯度下降

提高收敛速度，在常规的随机梯度下降中加入了动量项。动量模拟了目标运动时的惯性，也就是说，在更新过程中一定程度上保留了之前的更新方向，而用当前的更新梯度来做调最终的更新方向。通过这种方式，可以在一定程度上提高稳定性，从而可以更快地学习，同时也有能力摆脱局部优化。

动量有助于减少噪声扰。
指数加权平均数被用来平滑曲线。

$$v_{n e w}=\eta * v_{o l d}-\alpha * \frac{\partial(\text { Loss })}{\partial\left(W_{o l d}\right)}$$

SGD 在遇到沟壑时容易陷入震荡。为此，可以为其引入动量 Momentum，加速

SGD 在正确方向的下降并抑制震荡。引入动量有效的加速了梯度下降收敛过程。

在原步长之上，增加了与上一时刻步长相关的γmt−1，γ通常取 0.9 左右。这意味着 参数更新方向不仅由当前的梯度决定，也与此前累积的下降方向有关。这使得参数中那些梯度方向变化不大的维度可以加速更新，并减少梯度方向变化较大的维度上的更新幅度。由此产生了加速收敛和减小震荡的效果。

牛顿法(Newton Methods)

引入二阶导函数来作为动量，可以让迭代速度比SGD的速度快。
二阶导是对梯度求导，考虑梯度的变化速度
gt是一阶导数，vt是二阶导数

牛顿法是二阶收敛，梯度下降是一阶收敛，所以牛顿法就更快。如果更通俗地说的话，比如你想找一条最 短的路径走到一个盆地的最底部，梯度下降法每次只 从你当前所处位置选一个坡度最大的方向走一步，牛 顿法在选择方向时，不仅会考虑坡度是否够大，还会 考虑你走了一步之后，坡度是否会变得更大。所以， 可以说牛顿法比梯度下降法看得更远一点，能更快地 走到最底部

Nesterov Momentum(牛顿动量法)

算法能够在目标函数有增高趋势之前，减缓更新速率。和 Momentum 动量法的唯一区别就是更改了梯度的计算方式(引入一个临时的theta变 量)，计算梯度的方式可以使算法更好的"预测未来"，提前调整更新速率

Adagrad(Adaptive Gradient)

自适应梯度下降： AdaGrad背后的直觉是，我们是否可以根据不同的迭代，为每一个隐藏层的每一个神经元使用不同的学习率，可以提高系数梯度问题上的性能

SGD、Momentum 和 Nesterov Momentum均是以相同的学习率去更新新的各个分量。但是在深度学习模型中，存在着大量的模型参数，对于不同模型参数最好使用不一样的更新学习率。对于更新不频繁的参数，希望单次更新的步长更大，能够多学习一些知识;对于更新频繁的参数，则希望单词更新的步长较小，使得学习到的参数更稳定，不至于被单个样本影响太多。Adagrad引入二阶动量Vt，从而让之前频繁更新的参数，其二阶动量最好比较大，对于更新不频繁的模型参数，其二阶动量最好比较小。

引入梯度的平方和作为衰减项，训练过程中自动降低学习率

$$W_{n e w}=W_{\text {old }}+ \frac{\alpha}{\sqrt{\text { cache } e_{n e w}}+\epsilon} * \frac{\partial(\text { Loss })}{\partial\left(W_{\text {old }}\right)}$$

AdagradDAO

加了正则的Adagrad

RMSprop

也维护每个参数的学习率，根据权重梯度的平均值来调整学习率，对噪声数据应对较好

在Adagrad优化算法中，二阶动量vt是单调递增的，所以会导致学习率逐渐减至0，可能会导致训练提前结束。为了改进这个缺点，可以在计算二阶动量的时候不进行全部动量的累加。使用gamma衰减系数来控制对 于历史动量的引入大小，一般建议给定值为γ为0.9

RMS-Prop (Root Mean Square Propagation)：均方根传播

RMS-Prop是Adagrad的一个特殊版本，其中的学习率是梯度的指数平均值，而不是梯度平方的累积和。RMS-Prop基本上结合了动量和AdaGrad。

$$\text { cache }_{n e w}=\gamma * \text { cache }_{\text {old }}+(1-\gamma) *\left(\frac{\partial(\text { Loss })}{\partial\left(W_{\text {old }}\right)}\right)^{2}$$

在RMS-Prop中，学习率被自动调整，它为每个参数选择不同的学习率。

https://arxiv.org/pdf/1308.0850v5.pdf

Adadelta

和RMSprop一样，也属于对于Adagrad的优化，其优化目的是为了解决两个问题：

- 1. 二阶动量累加的问题
- 2. 需要全局学习率的问题

Adadelta是Adagrad的扩展，它也试图减少Adagrad的激进性，单调地降低学习率，并消除学习率衰减的问题。在Adadelta中，我们不需要设置默认的学习率，因为我们采取以前的时间步的运行平均值与当前梯度的比率。

AdaDelta的主要优点是，我们不需要设置默认的学习率。但加大了计算量

Ftrl

FTRL(Follow The Regularized Leader)

FTRL是一种适用于处理超大规模数据的,含大量稀疏特征的在线学习的常见优化算法

FTRL在处理带非光滑正则项（如L1正则）的凸优化问题上表现非常出色，不仅可以通过L1正则控制模型的稀疏度，而且收敛速度快

FTRL的基本思想是将接近于0的梯度直接置零，计算时直接跳过以减少计算量。

ASGD

https://dl.acm.org/doi/10.1137/0330046

平均随机梯度下降： Averaged Stochastic Gradient Descent

LBFGS

https://www.cs.ubc.ca/~schmidtm/Software/minFunc.html

minFunc的默认参数调用了一个准牛顿策略，其中在计算步长方向时使用了具有Shanno-Phua插放功能的有限内存BFGS更新。
而在计算步长方向时使用了满足强Wolfe条件的括弧线搜索。在直线搜索中，（有保障的）三次插值被用来产生试验值。在目标函数进入参数不产生实数输出的区域（即复数，NaN或Inf）的迭代中，该方法切换到Armijo反向跟踪直线搜索。

Rprop

执行弹性反向传播算法

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.1417

LARS

论文：Large batch training of convolutional networks