◎热点与综述◎

深度学习优化器进展综述

常禧龙,梁 琨,李文涛
天津科技大学 人工智能学院,天津 300457

摘 要:优化器是提高深度学习模型性能的关键因素,通过最小化损失函数使得模型的参数和真实参数接近从而提高模型的性能。随着GPT等大语言模型成为自然语言处理领域研究焦点,以梯度下降优化器为核心的传统优化器对大模型的优化效果甚微。因此自适应矩估计类优化器应运而生,其在提高模型泛化能力等方面显著优于传统优化器。以梯度下降、自适应梯度和自适应矩估计三类优化器为主线,分析其原理及优劣。将优化器应用到Transformer架构中,选取法-英翻译任务作为评估基准,通过实验深入探讨优化器在特定任务上的效果差异。实验结果表明,自适应矩估计类优化器在机器翻译任务上有效提高模型的性能。同时,展望优化器的发展方向并给出在具体任务上的应用场景。

关键词:优化器;机器翻译;Transformer;深度学习;学习率预热算法
文献标志码:A **中图分类号:**TP183 doi:10.3778/j.issn.1002-8331.2307-0370

Review of Development of Deep Learning Optimizer

CHANG Xilong, LIANG Kun, LI Wentao
College of Artificial Intelligence, Tianjin University of Science and Technology, Tianjin 300457, China

Abstract: Optimization algorithms are the most critical factor in improving the performance of deep learning models, achieved by minimizing the loss function. Large language models (LLMs), such as GPT, have become the research focus in the field of natural language processing, the optimization effect of traditional gradient descent algorithm has been limited. Therefore, adaptive moment estimation algorithms have emerged, which are significantly superior to traditional optimization algorithms in generalization ability. Based on gradient descent, adaptive gradient, and adaptive moment estimation algorithms, and the pros and cons of optimization algorithms are analyzed. This paper applies optimization algorithms to the Transformer architecture and selects the French-English translation task as the evaluation benchmark. Experiments have shown that adaptive moment estimation algorithms can effectively improve the performance of the model in machine translation tasks. Meanwhile, it discusses the development direction and applications of optimization algorithms.

Key words: optimizer; machine translation; Transformer; deep learning; learning rate warm-up algorithm

随着GPU、大数据[1-2]技术的普及,以深度神经网络为研究核心的深度学习[3]在多个任务上刷新了最佳结果而受到关注。例如将针对时序序列任务的循环神经网络(RNN)应用于自然语言处理[4-6](NLP)领域,提升了文本生成[7]、文本分类[8]等任务的效果;在机器翻译[9]任务中,以Transformer[10]为代表的大语言模型,可以对复杂语义和句子结构进行学习,实现高质量的文本翻译[11];基于Transformer解码器的GPT[12-14]模型经过不断迭代成为ChatGPT,这是当今人工智能模型热潮的开端;基于Transformer编码器的BERT[15]模型将预训练拓展到双向语言模型,并在多项下游任务中取得了最好结果。在语音领域,深度学习能够对语音进行识别、分类等。

深度学习通过深层神经网络学习到数据的特征,并以损失函数(loss function)衡量模型预测与真实的差

基金项目:国家自然科学基金(62377036,61807024);天津市教委科研计划项目(2019KJ235);天津市企业科技特派员项目(22YDTPJC00940)。

作者简介:常禧龙(2000—),男,硕士研究生,CCF学生会员,研究方向为自然语言处理;梁琨(1982—),通信作者,女,硕士,副教授,CCF会员,研究方向为知识图谱、自然语言处理,E-mail:liangkun@tust.edu.cn;李文涛(1999—),男,硕士,CCF学生会员,研究方向为知识图谱、自然语言处理。

收稿日期:2023-07-26 **修回日期:**2023-09-21 **文章编号:**1002-8331(2024)07-0001-12

距。优化器(optimizer)用以最小化损失函数使得模型参数与真实参数接近从而选择最优模型。

已经有一些文献对优化器进行了研究,史加荣等人[16]较为全面地分析对比了不同优化器,并且给出了一系列的实验分析;张慧[17]分析了优化器的改进。以上文章对深度学习及优化器有较为全面的介绍,但少有人关注在自然语言处理领域中对不同优化器选择的探讨,并且多数文章聚焦于Adam[18]及之前的优化器,对新出现的许多模型[12-15,19],例如ELMo[20]、BERT、GPT模型及其变种,这些模型的复杂度不断提高,参数量不断增加:BERT模型的参数量在1.1亿;GPT模型的参数量大约为1.2亿,发展到GPT-3模型已经达到了1 750亿,原有的优化器无法适应这些模型,因此新提出的优化器则着重优化这些大型模型。

传统的优化器基于梯度下降优化器,其原理是通过计算目标函数的梯度进行优化,该方法需要在全体数据集上取梯度,随着数据量的增大,其计算成本十分高昂。由此出现了随机梯度下降(SGD),在迭代中选取几个样本进行梯度下降,降低了时间复杂度。随机梯度下降方法中引入动量能有效改善梯度下降的速度即带动量的随机梯度下降(SGDM),其原因在于动量能记录历史梯度,使得无需重新寻找梯度下降的方向。学习率的调整影响着优化效果,自适应学习率优化器通过累计梯度等方式使得学习率的调整摆脱了手动操作,降低了优化成本,这类优化器中,带热重启的随机梯度下降(SGDR)将热重启和随机梯度下降优化器结合,减少了迭代次数,提高了优化速度,还能防止优化器陷入局部最小值。AdaGrad[21]引入了梯度累计进一步加快了迭代速度,并且能够适应稀疏问题。Adadelta[22]引入了自适应学习率(adaptive learning rate)使得迭代速度再进一步加快,降低了选择学习率的工作量并且不再累计全体梯度降低了存储成本。RMSProp也有类似的效果。

自适应矩估计类优化器结合了以上几类自适应优化器,能够获得高内存效率以及应对稀疏问题,Adam[18]在Transformer[10]等大语言模型中获得了广泛的应用。针对Adam的改进也有许多新进展。针对Adam中正则化和权重衰减不一致的问题,AdamW[23]引入了权重衰减提高了模型的泛化性能;针对Adam中初始化不稳定问题,整流项能够使RAdam[24]在不依赖学习率预热(warm-up)的情况下取得和Adam类似的效果;在优化器的优化方式上,Lookahead[25]使用两组参数对模型进行优化,避免了优化器落入局部最小值等问题;组合优化器能够集众家之长,Ranger21[26]结合了八种优化方法获得了更进一步的效果。

本文介绍了优化器的新进展,将优化器分为梯度下降、自适应学习率和自适应矩估计优化器三类,对这些优化器的原理和推导过程进行了研究,为了实地评估其性能,选取了Transformer模型的法-英翻译任务评估不同优化器在机器翻译中的性能,文末展望了优化器的改进方向,总结了全文工作并给出了优化器的选择方法。

1　梯度下降类优化器

梯度下降是深度学习中最经典的优化器之一,几乎所有的优化器都由该优化器演变而来。梯度下降的原理是最小化由模型参数 $\alpha\in\mathrm{R}$ 构成的目标函数 $J(\alpha)$,方法是根据沿该函数梯度 $\nabla_{\alpha}J(\alpha)$ 的反方向更新参数从而使得模型参数逼近真实参数。引入学习率 η 控制每次更新参数的步长,能够决定模型参数与真实参数接近的速度。

梯度下降类优化器包括全梯度下降(full gradient descent,FGD),随机梯度下降以及小批量随机梯度下降(MBGD)这三类。带动量的随机梯度下降通过引入动量可以记录前期动量更新的方向从而避免振荡,加速收敛。

本章对这四类优化器的原理进行了简要介绍,分析了其优劣给出了优化器的应用场景。

1.1　深度学习中的损失函数

本节引入损失函数模型[27]来说明优化器在深度学习中的作用。

给定包含 n 个数据的数据集 $\left\{\left(x_i,y_i\right)\right\}_{i=1}^{n}$,$x_i,y_i\in\mathrm{R}$,$x_i,y_i$ 分别为第 i 个样本的输入和标签。模型根据 $\hat{y}_i=h\left(x_i,a\right)$ 预测输出,其中 $h\left(x_i,a\right)$ 是关于标签 x_i 和可学习参数 $a\in\mathrm{R}$ 的决策函数。

优化器最小化参数 a 从而使得预测输出接近真实输出,损失函数可以表达为式(1):

$$\min_{a}D(a)=\frac{1}{n}\sum_{i=1}^{n}l_i(a)\tag{1}$$

$l_i(a)$ 为第 i 个样本中的损失函数,$D(a)$ 为平均损失函数。

1.2　全梯度下降

全梯度下降更新一次参数需要在整个数据集上计算所有的梯度,这使得在大规模数据集上的更新成本过高,数据集数量 n 增加,计算代价线性增加。全梯度下降更新公式如式(2):

$$a_t=a_{t-1}-\eta\frac{1}{n}\sum_{i=1}^{n}\nabla l_i\left(a_{t-1}\right)\tag{2}$$

第 t 轮待更新的参数 a_n 在每次更新时向着梯度的负方向更新,学习率 η 调整了每次更新的幅度。

1.3　随机梯度下降

为了提高全梯度下降的更新效率,随机梯度下降每次更新参数都从数据集中采样一个样本进行计算梯度。该梯度为全局梯度估计,随机梯度下降的更新式如式(3):

$$a_{t+1} \leftarrow a_t - \eta \nabla l(a_t) \tag{3}$$

该方法能够有效降低计算成本,但由于梯度是估计值,且样本存在噪声等问题,因此每次更新都需要重新寻找更新方向,从而造成振荡现象,这是其弊端之一。

给定二元函数 $f(x)=0.1x_1^2+2x_2^2$,图1展示了随机梯度下降的优化效果。由于振荡的存在,随机梯度下降需要进行多次迭代才能找到最优解,这增加了优化成本。

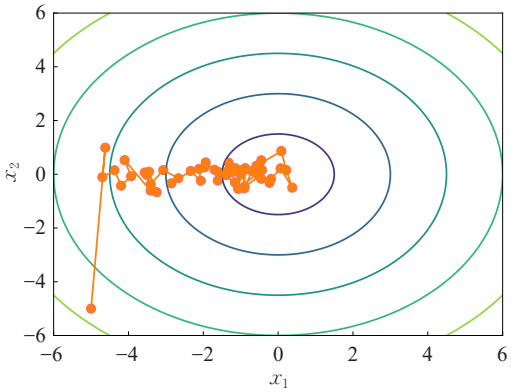

图1 随机梯度下降优化效果

Fig.1 Stochastic gradient descent optimization effect

1.4 小批量随机梯度下降

小批量随机梯度下降选取小批量(mini-batch)[28]数据计算梯度,是介于全梯度下降和随机梯度下降之间的优化器。

小批量随机梯度下降每次更新都采样若干个样本,采样的数量称为批量 B,该方法能够降低全梯度下降选取全部数据进行更新的成本,还能避免随机梯度下降选取单条数据计算梯度后对全局梯度估计的偏离,从而提高了计算效率,小批量随机梯度下降的更新式如式(4)所示:

$$a_{t+1} = a_t - \eta \frac{1}{B} \sum_{i=1}^{B} \nabla l_i(a_t) \tag{4}$$

其中,B 为批次大小,每轮训练根据批次大小重新选取样本。

小批量随机梯度下降需要手动调整超参数批量 B。当批量 B 较小时,更新效率降低,并且可能引起振荡;当批量 B 较大时增加计算成本,可能包含多余信息。

1.5 带动量的随机梯度下降

带动量的随机梯度下降在随机梯度下降中引入动量项,如图2所示,带动量的随机梯度下降的更新原理是以动量替代更新式中的梯度项,更新方向是上一次参数更新的方向信息和本次更新的梯度方向来合成的方向。该方法由于每次参数更新都没有重新寻找新的梯度下降方向,因此能解决振荡问题,减少迭代次数,提升优化速度。

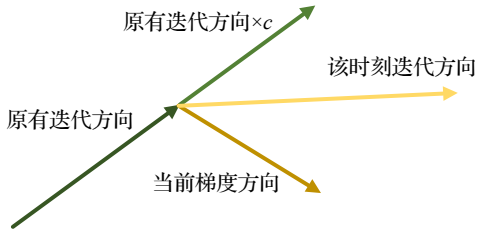

图2 带动量的随机梯度下降的更新方向

Fig.2 Update direction of stochastic gradient descent with momentum method

带动量的随机梯度下降采用动量替代梯度进行参数更新,动量的表达式如式(5):

$$v_t \leftarrow c v_{t-1} + g_{t,t-1} \tag{5}$$

$g_{t,t-1}=\nabla l(a_t)$ 是梯度,动量 v_t 记录 t 时刻更新方向,cv_{t-1} 记录原有更新方向,动量系数 $c<1$,防止每次更新都完全根据上一次更新方向进行更新。以动量替换原有参数更新中的梯度数据,参数更新如式(6):

$$a_{t+1} \leftarrow a_t - v_t \tag{6}$$

带动量的随机梯度下降加快参数更新的速度,效果如图3,由于动量对梯度的累计,使得每次更新不需要重新寻找更新方向,减少了振荡。表1对比梯度下降优化器并给出了应用场景。

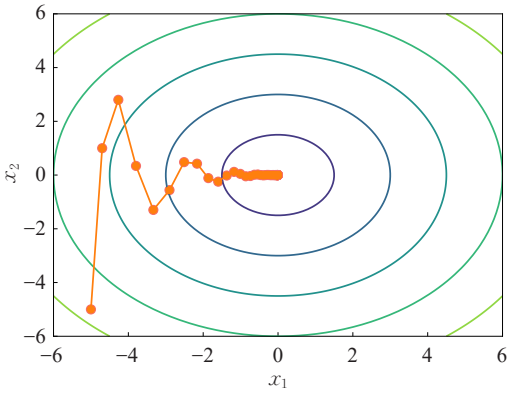

图3 带动量的随机梯度下降优化效果

Fig.3 Stochastic gradient descent with momentum optimization effect

表1 梯度下降更新方式对比

Table 1 Comparison of gradient descent update methods

优化器	优势	缺陷	梯度更新	参数更新	适用场景
FGD	不易出现振荡	复杂度 $O(n)$	$\nabla L(a_t)=\frac{1}{n}\sum_{i=1}^{n}\nabla l_i(a_t)$	$a_{t+1} \leftarrow a_t - \eta \nabla L(a_t)$	小数据量
SGD	运算成本低复杂度 $O(1)$	容易出现振荡	$\nabla l(a_t)$	$a_{t+1} \leftarrow a_t - \eta \nabla l(a_t)$	大数据量
MBGD	运算成本适中	需要手动调整 B	$\nabla l(a_t)$	$a_{t+1}=a_t-\eta\frac{1}{B}\sum_{i=1}^{B}\nabla l_i(a_t)$	中数据量、随机梯度下降训练成本较高
SGDM	避免出现振荡、优化速度快	迭代过程不够平滑	$g_{t,t-1}=\nabla l(a_t)$	$a_{t+1} \leftarrow a_t - v_t$	训练成本较高、随机梯度下降出现振荡

2　自适应学习率类优化器

梯度下降类优化器需要手动调节学习率,其弊端是在靠近最小值时梯度不断减小,使得最开始的学习率显得过大,很可能错过最优解。因此在梯度下降类优化器的基础上提出了可以对学习率进行自动调整的优化器,极大提升了优化效率。本章先介绍了一些在梯度下降类优化器中有益的学习率自动调整方法,再介绍了一些典型的自适应学习率优化器。

2.1　学习率预热

学习率预热是学习率在学习过程中自动调校的方法,先设定学习率预热步数,在预热步数到达之前学习率缓慢增加,到达学习率预热步数后逐渐减小。这种方法的优势在于学习率预热过程中学习率能够自适应调整,避免了初始学习率过大,在预热步数后能够以小学习率逼近防止错过最优解。

Transformer采用了式(7)的学习率预热。

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \tag{7}$$

图4选取了三种学习率预热步数对比了不同预热步数对学习率变化的影响。随着预热步数的增加,学习率最大值不断减小,学习率变化也趋于平缓,这对于学习率的调节有着重要意义。

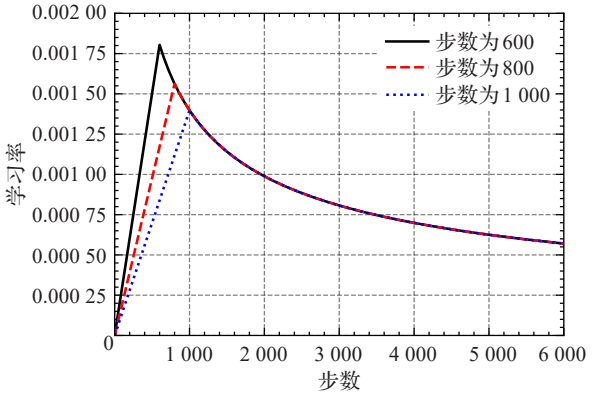

图4　学习率预热

Fig.4　Learning rate warm-up

此外,靠事先指定学习率调整策略的学习率预热不能完全适应学习情况,因此引入了自适应学习率,在学习过程中自适应调节学习率,这类优化器中以AdaGrad最为重要。

2.2　带热重启的随机梯度下降

学习率按余弦进行调整,并结合热重启[29-31]应用到随机梯度下降就得到了带热重启的随机梯度下降[32],避免由于学习率过小而训练速度过慢,提高了神经网络的收敛速度。

学习率按余弦衰减,每隔一定周期后重新初始化为预设值,该方法采用的学习率调整函数如式(8):

$$\eta_t = \eta_{\min}^i + \frac{1}{2}(\eta_{\max}^i - \eta_{\min}^i)\left(1 + \cos\left(\frac{T_{\text{cur}}}{T_i}\pi\right)\right) \tag{8}$$

$\eta_{\max}^i$、$\eta_{\min}^i$ 指定了第 i 次重启的学习率范围,T_{cur} 是自上次重启以来执行的epoch。当 $t=0$,$T_{\text{cur}}=0$ 时,学习率最大;$T_{\text{cur}}=T_i$ 时,学习率最小,$\eta_t=\eta_{\min}^i$。

带热重启的随机梯度下降在epoch较小时使用,按式(9)进行调参,其中mult是调整的倍数。

$$\begin{cases} T_0=1, T_{\text{mult}}=2 \\ T_0=10, T_{\text{mult}}=2 \end{cases} \tag{9}$$

图5展示了两种热重启策略的学习率变化情况。$\eta_{\max}^i=0.05$,$\eta_{\min}^i=0$。在 $T_0=100$,$T_{\text{mult}}=1$ 时,每100步进行重启,每次重启次数相同。在 $T_0=10$,$T_{\text{mult}}=2$ 时,在10步进行重启,每次重启步数为原有的两倍。

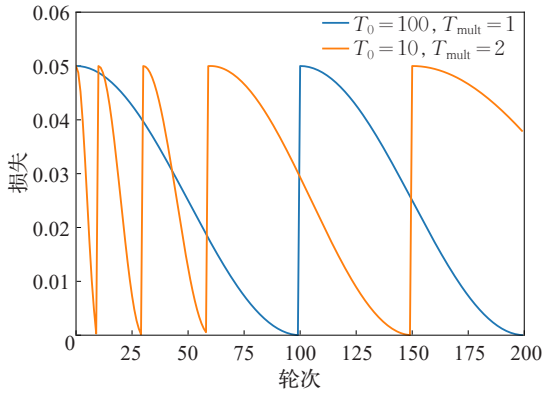

图5　带热重启的随机梯度下降中学习率的变化

Fig.5　Changes in learning rate of stochastic gradient descent with warm restarts

2.3　AdaGrad

数据集中可能存在稀疏特征,其梯度为0,会导致学习率下降过快,而对于稠密特征,学习率下降速度则过慢,称为数据不平衡。一种方法是拆分参数进行学习。

AdaGrad[21]即自适应学习率方法,引入了梯度累计负责记录梯度,如式(10):

$$s_t = s_{t-1} + g_t^2 \tag{10}$$

s_t 记录了累计梯度,累计梯度能够对学习率进行调节,防止学习率的更新影响梯度的计算。累计梯度的计算采用 g_t^2 的形式,更新参数采用累计梯度进行。

$$a_{t+1} \leftarrow a_t - \frac{\eta}{\sqrt{s_t + k}} g_t \tag{11}$$

AdaGrad考虑梯度和学习率的关系,学习率根据梯度进行调整,梯度过大时学习率减小,梯度减小,学习率增加。k 防止累计梯度为0。这说明,对于稀疏特征,AdaGrad能够累计较小梯度,学习率较大,反之对于稠密特征,能够累计较大梯度,学习率较小从而适应不同的情况。学习过程中的学习率调节更加自然,调节学习率到5的效果如图6。表明AdaGrad相较随机梯度下降收敛速度更快,此外自适应调节学习率减少了手动操作的局限性。但AdaGrad会一直累计梯度,增大存储成本。

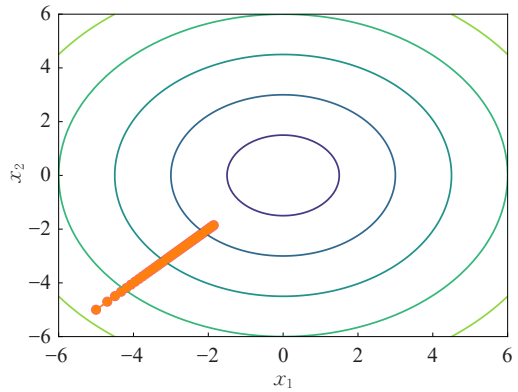

图6 AdaGrad优化效果

Fig.6 AdaGrad optimization effect

2.4 Adadelta

Adadelta[22]引入了泄露平均值代替存储梯度的平方和从而改进了AdaGrad,使得存储成本降低。泄露平均值如式(12)将梯度累计的大小限制在固定大小为 w 的窗口内。梯度之和的累计被定义为过去梯度累计的衰减平均值。

$$E[g^2]_t = bE[g^2]_{t-1} + (1-b)g_t^2 \tag{12}$$

衰减因数 $b \in (0,1)$,一般取0.9,$E[g^2]_t$ 为 t 轮的衰减期望,引入衰减平均,AdaGrad参数更新式可更改为式(13):

$$a_{t+1} \leftarrow a_t - \frac{\eta}{\sqrt{E[g^2]_t + k}} g_t \tag{13}$$

分子与分母单位不匹配,将分子修改为与分母类似但梯度改为参数更新的式(14):

$$a_{t+1} \leftarrow a_t - \frac{\sqrt{E[a^2]_t + k}}{\sqrt{E[g^2]_t + k}} g_t \tag{14}$$

Adadelta中不仅可以自适应学习率,并且无需设定初始学习率,并且解决了AdaGrad存储成本过高的缺陷。

2.5 RMSProp

RMSProp[16]是AdaGrad的拓展,与Adadelta的区别在于参数更新的分子采用的是学习率,RMSProp的参数更新如式(15):

$$a_{t+1} \leftarrow a_t - \frac{\eta}{\sqrt{E[g^2]_t + k}} g_t \tag{15}$$

3 自适应矩估计类优化器

Adam结合了AdaGrad、RMSProp,能够考虑到稀疏特征和优化非平稳目标等情况,在Transformer及其后来的架构上取得了更好的效果,针对该优化器已有AMSGrad、AdamW、RAdam等改进。

3.1 Adam

Adam[18]使用梯度的一阶矩和二阶矩来动态调节自适应学习率。结合了AdaGrad能够处理稀疏特征以及RMSProp能够对非平稳目标进行优化的特点,从而适合大数据集。其计算效率高,并且节省内存。参数通常不需要调整。由于以上优势,这种方法成为了现今最受欢迎的优化器之一。Adam需要计算 g_t 的一阶矩 m_t 和二阶矩 v_t 如式(16)、(17):

$$m_t \leftarrow cm_{t-1} + (1-c)g_t \tag{16}$$

$$v_t \leftarrow hv_{t-1} + (1-h)g_t^2 \tag{17}$$

其中,$c, h \in [0,1)$,由于该式可能由于两者的变化不同而产生偏差即 $c < h$,因此对两者进行调整如式(18)、(19):

$$\hat{m}_t = \frac{m_t}{1-c^n} \tag{18}$$

$$\hat{v}_t = \frac{v_t}{1-h^n} \tag{19}$$

梯度进行缩放如式(20),k 放在外面简化计算。

$$g_t' = \frac{\eta}{\sqrt{\hat{v}_t + k}} \hat{m}_t \tag{20}$$

但Adam需要结合自适应学习率才能保证在初始方差不至于过大从而得到较好的效果,这是Adam的缺陷之一。

3.2 AMSGrad

3.2.1 不收敛的Adam

Adam优化器可能不收敛,式(21)表示了自适应的学习率倒数随时间的变化,记为兴趣数量。

$$\Gamma_{t+1} = \frac{\sqrt{V_{t+1}}}{\eta_{t+1}} - \frac{\sqrt{V_t}}{\eta \alpha_t} \tag{21}$$

V_t 为二阶矩,对于随机梯度下降和AdaGrad,$\Gamma_t \geqslant 0$,$t \in T$,对于AdaGrad,η 不会改变。在Adam中,Γ_t 的值则不确定,以线性函数(22)为例。该函数在 $C > 2$,收敛于 $x = -1$。此时,若使用Adam,当 $c = 0, h = 1/(1+C)$,优化器则会收敛在 $x = 1$。

由于该优化器每三步进行梯度和的计算,其中两步得到 $x = -1$,而一次得到了 C,那么指数平均值会使得优化器偏离正确收敛方向,无法抵消大梯度 C 带来的影响。

$$f(x) = \begin{cases} Cx, t \bmod 3 = 1 \\ -x, \text{其他} \end{cases} \tag{22}$$

3.2.2 AMSGrad的策略

AMSGrad[24]改动超参数 c 和 h,使得随着 t 变化,兴趣数量大于0,$t \in T$。该优化器和Adam的区别在于AMSGrad使用 v_t 的最大值进行学习率更新,Adam使用的是平均值。

3.3 AdamW

Adam在语言建模领域,选区解析等任务上,泛化能力和带动量的随机梯度下降仍有差距。文献[33]和[34]指出存在尖锐最小值问题。

在Adam上采用 L_2 正则化和权重衰减是提高泛化能力的重要因素,对Adam加入 L_2 正则化能够提高Adam的泛化能力,这种方法称为AdamW[23]。

L_2正则化也被称为权重衰减,但在Adam中,这两者并不等同。由于Adam引入了多种动量,因此参数更新会变成式(23):

$$a_t \rightarrow a_{t-1} - \eta \frac{b_1 m_{t-1} + (1-b_1)(\nabla f_t + \lambda a_{i-1})}{\sqrt{\hat{V}_t}} \tag{23}$$

在Adam和正则化共同作用时,$\lambda\theta$ 部分的元素被 $\sqrt{\hat{V}_t}$ 调整,导致在梯度快速变化的方向上 $\sqrt{\hat{V}_t}$ 过大,这引起了调整后的 $(\lambda\theta_t - 1)/\sqrt{\hat{V}_t}$ 的减小,因此参数 θ 被更少正则化。AMSGrad引入了单独的权重衰减。

3.4　RAdam

RAdam在原有Adam上引入了整流项从而摆脱学习率预热,减少了调参数量。文献[24]指出,Adam自身有陷入局部最优解的问题,学习率预热能够有效改善这种情况,如图7所示,使用学习率预热的Adam效果要好于没有使用学习率的,后者在一轮即陷入局部最优解,损失无法下降。

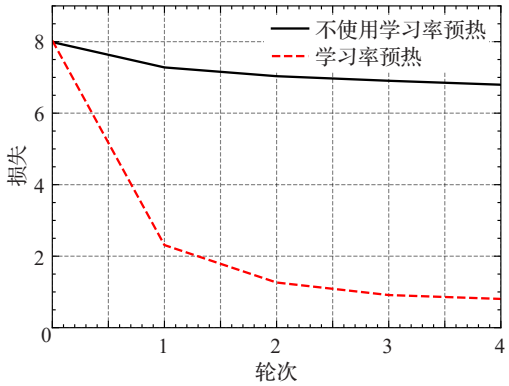

图7　学习率预热对Adam的影响

Fig.7　Impact of learning rate preheating on Adam

由于数据样本有限,未使用学习率预热会使得方差在初始时刻过大,容易陷入局部最优解,因此需要借助学习率预热,在初始时刻采用小的学习率进行学习。

为了避免局部最优解的问题,RAdam引入了整流项来矫正较大方差,并且取得了与加入学习率预热的Adam类似的效果。为了获得整流项,先计算出优化器中自适应学习率的方差如式(24):

$$\mathrm{Var}\left[\psi\psi_t(g_1,g_2,\cdots,g_t)\right] = r^2\left(\frac{\rho}{\rho-2}\frac{\rho 2^{2\rho-5}}{\pi}B\left(\frac{\rho-1}{2},\frac{\rho-1}{2}\right)^2\right) \tag{24}$$

ψ_t 为 t 轮的自适应学习率的方差,B 是beta函数,ρ 为自由度。

该函数为自由度的单调递减函数,且自由度与样本数量相关,与 t 成正比,因此在训练初期的方差要大,有500倍的差异。式(25)是该函数的最小值。

$$\min_{\rho_t} \mathrm{Var}\left[\psi_t\right] = C_{\mathrm{var}} \tag{25}$$

为了矫正方差,需要将保证方差始终为最小值,经式(26)可以矫正方差:

$$\mathrm{Var}\left[r_t\psi(g_1,g_2,\cdots,g_t)\right] = C_{\mathrm{var}} \tag{26}$$

$r_t = \sqrt{C_{\mathrm{var}}/\mathrm{Var}\left[\psi(g_1,g_2,\cdots,g_t)\right]}$ 为方差修正因子,$\psi(g_1,g_2,\cdots,g_t)$ 为自适应学习率。在原有优化器的基础上加入了修正因子,即式(27)的整流项。

$$r_t = \sqrt{\frac{(\rho_t-4)(\rho_t-2)\rho}{(\rho-4)(\rho-2)\rho_t}} \tag{27}$$

该式中 $\rho > 3$,否则自适应学习率失效,优化器退化为随机梯度下降。结合上式原优化器可以修改为式(28):

$$a_t \leftarrow a_{t-1} - \alpha_t r_t \hat{m}_t l_t \tag{28}$$

式中,l_t 为自适应学习率,α_t 为步长参数,是一个事先指定的序列 $\{\alpha_t\}_{t=1}^{T}$。

3.5　Lookahead

对随机梯度下降的优化可以分为两种方法:(1)自适应学习率,例如AdaGrad。(2)加快优化速度,例如带动量的随机梯度下降等。而这两类方法均需要进行多次调参才能得到较好的结果。

在一个优化器中引入两组参数,以不同速度进行学习,按参数更新快慢称为fast weight以及slow weight,经 k 轮训练slow weight根据fast weight进行更新。这种优化方法被称为LookAhead[25],参数更新可以参考图8。测试了在一个函数 $f(x_1,x_2)=x_1+2x_2^3$ 中使用随机梯度下降和Lookahead进行 $k=30$ 次学习的结果,该图是一个等高线图,从四周向中心表示函数值越来越小,右上的图为随机梯度下降的测试结果,可随机梯度下降需要多次更新才能达到最小值。而右下角为Lookahead的结果,结合左图可以观察到,Lookahead的slow weight是根据fast weight的结果进行梯度方向的选择,从而更快接近最小值。

该方法好处在于:(1)对参数调整不敏感。(2)加快收敛速度。(3)计算成本低廉。

Lookahead需要结合其他优化器进行优化,其中,标准优化器负责更新fast weight。slow weight则是Lookahead引入的,参考fast weight的结果进行更新。slow weight的更新如式(29):

$$\phi_{t+1} = \phi_t + \alpha(\theta_{t,k} - \phi_t) = \alpha\left[\theta_{t,k} + (1-\alpha)\theta_{t-1,k} + \cdots + (1-\alpha)^{t-1}\theta_{0,k}\right] + (1-\alpha)^t\theta_0 \tag{29}$$

第 $t+1$ 轮的slow weight为 ϕ_t,根据fast weight为 ϕ_t 进行更新,每次更新 ϕ_t 的时根据两套参数的差值 $\theta_{t,k}-\phi$ 进行更新。更新式中还能保留原有fast weight,避免优化进入局部最小值,从而加快收敛。

3.6　Ranger21

2021年,Wright等人[26]提出了Ranger21。结合了八

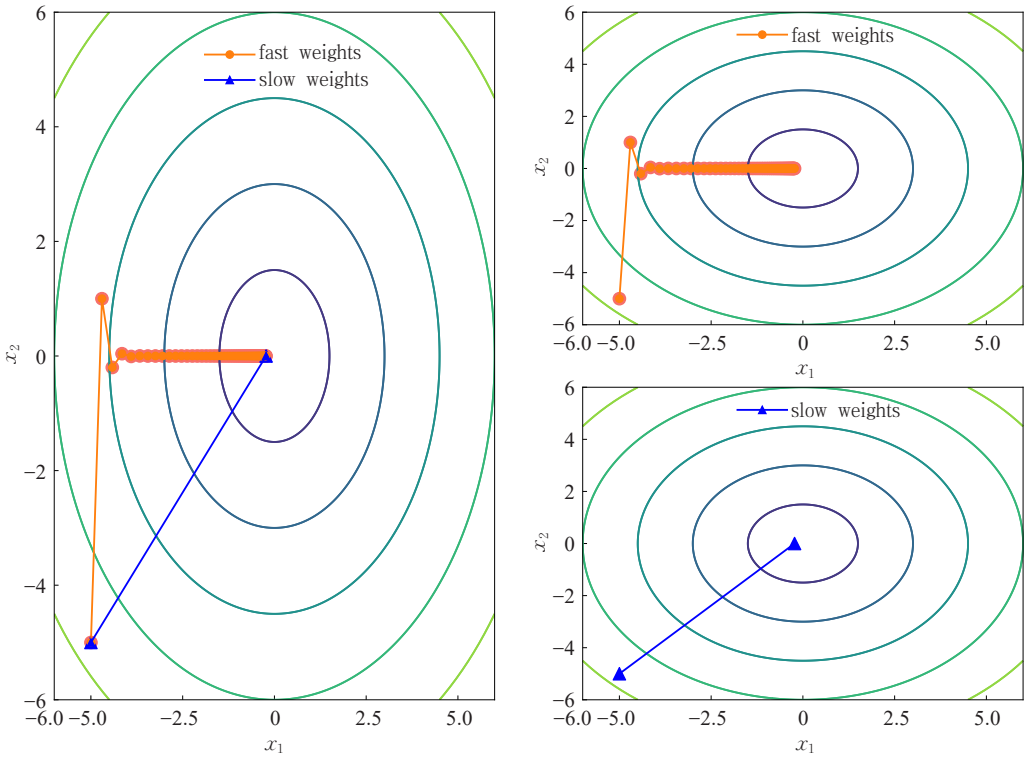

图8 Lookahead优化效果

Fig.8 Optimization effect of Lookahead

种优化思想能够显著提高优化的速度和准确性,并且在训练中拥有更平滑的曲线。

(1)Ranger

将Lookahead和RAdam结合形成了Ranger。和AdamW相比训练更加流畅,准确性也得到了提高。

(2)自适应梯度裁剪

在Ranger21中引入式(30)进行梯度裁剪可以保证梯度在给定值之下,防止梯度爆炸,加速训练过程[35]。

$$g_t=\begin{cases}\theta\dfrac{g_t}{\|g_t\|},\|g_t\|>\theta\\ g_t,\text{其他}\end{cases} \tag{30}$$

给定梯度 g_t,若超过阈值 θ 则会将梯度裁剪到小于 θ 的值。θ 的手动选择带来了训练的不稳定性问题,因此Ranger21采用式(31)的动态阈值更新。

$$g_t=\begin{cases}\theta\dfrac{\max\left(\|\theta_t^r\|\right)}{\|g_t^r\|}g_t,\dfrac{\max\left(\|\theta_t^r\|\right)}{\|g_t^r\|}>\theta\\ g_t^r,\text{其他}\end{cases} \tag{31}$$

此处,r 默认为 10^{-2},这表明该式处理的不是完整的一层,而是某个维度。

(3)梯度中心化

Ranger21引入梯度中心化[36]对梯度进行标准化,有助于提高模型的泛化能力和训练速度。Ranger21使用的梯度中心化如式(32):

$$g_{gc}=\nabla f_t(\theta_{t-1})-mean\left(\nabla f_t(\theta_{t-1})\right) \tag{32}$$

中心梯度 g_{gc} 将 θ_{t-1} 对应的梯度减去梯度的均值从而达到中心化的效果,这对损失函数进行约束,训练过程更平滑。

(4)正负动量

Ranger21结合了正负动量[37],为了让优化过程能进入更平坦的最小值提高性能,需要模拟一组噪声加入到梯度中。正负动量保存了两组一阶矩估计,取平均值分别用于奇数和偶数迭代。在优化过程中,两组分别分配正负权重模拟出一组噪声来。

(5)权重软正则化

AdamW引入的权重衰减公式如式(33):

$$d=-\eta\alpha\theta \tag{33}$$

Ranger21则采用了权重矩阵的范数损失进行软正则化[38],如式(34):

$$d=-\eta\lambda\left(1-\frac{1}{\|\theta_{c_0}\|}\right)\theta \tag{34}$$

式(34)考虑了权重矩阵 c_0 的范数,权重矩阵倾向于单位范数,否则权重倾向0。

(6)稳定的权重衰减

AdamW的权重衰减是使用学习率来进行衰减,但实际中的学习率也需要考虑梯度大小。使得其不能应对梯度为0的情况。为了获取更稳定的权重衰减[39],可以考虑让梯度除以其均值的平方,即式(35):

$$d=-\frac{\eta}{\sqrt{mean(\hat{v}_t)}}\theta \tag{35}$$

$\hat{v}_t$ 为梯度的大小,与软正则化可以共同使用,即式(36):

$$d=-\frac{\eta}{\sqrt{mean(\hat{v}_t)}}\lambda\left(1-\frac{1}{\left\|\theta_{c_0}\right\|}\right)\theta \tag{36}$$

(7)线性学习率预热

RAdam能够克服Adam不依赖学习率预热的不稳定问题,但随后Ma等人[40]指出如式(37)的线性学习率预热[40]能够让优化器仅依赖预热即可达到良好的效果。图9展示了该学习率策略随轮数的增加对学习率的影响。

$$\eta_t=\min\left(1,\max\left(\frac{1-a_2}{2}t,\frac{t}{t_{\text{warmup}}}\right)\right)\eta \tag{37}$$

依靠二阶动量参数 a_2,t_{warmup} 重置 t 防止可能出现的过大学习率情况。该方法的效果与RAdam类似。

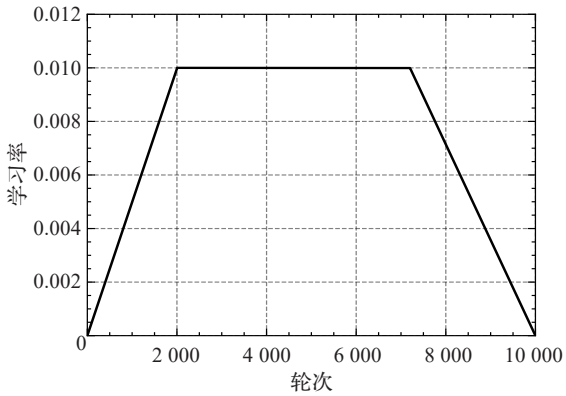

图9　学习率策略

Fig.9　Learning rate strategy

4　实验分析

本章先对学习率的选取进行实验,分析了其对优化过程的影响,在后面对比实验中采用了同一学习率排除了其影响。Lookahead需要结合其他优化器共同使用,因此单独分析了其对其他优化器的优化效果。为了得到优化器在自然语言处理领域的实际应用情况,选择Transformer模型进行法-英转换任务。通过不同优化器在同一数据集上的优化情况进行比较,得到各优化器的性能。

4.1　数据集介绍

tatoeba[41]数据库收集了大量面向外语学习者的例句,内含庞大的句子和翻译数据库,本次实验采用该数据库的法-英数据集,表2为该数据集的具体参数。

表2　数据集参数

Table 2　Dataset parameters

参数	数据集	大小/MB	数据集容量/句子对
取值	法译英	7.1(压缩后)	217 975

4.2　实验环境

4.2.1　实验配置

表3给出了本次实验的配置。

表3　实验配置

Table 3　Experimental configuration

环境	CPU	GPU
参数	12 vCPU Xeon® Platinum 8255C	RTX 3090(24 GB)×1

4.2.2　模型参数设定

本次实验的模型参数参考了文献[10]并根据实验进行了调整,参数设置如表4。

表4　模型参数

Table 4　Model parameter

参数	批次	训练次数	最大填充	隐藏层大小	注意力头数	层数	预热步数
取值	128	15	32	32	2	2	6 000

4.3　实验结果分析

4.3.1　学习率实验结果

学习率的调整会影响收敛速度和收敛情况,自适应学习率能够根据公式调节学习率从而防止其对学习过程的影响。为了得到较为一致的实验结果,在优化器实验之前先通过实验对比学习率对优化器的影响。

(1)学习率的取值影响

采用Adam并固定学习率,使用三个学习率进行测试,具体参数如表5。

表5　学习率参数

Table 5　Learning rate parameter

参数	批次大小	训练次数	最大填充	学习率预热步数
取值	128	5	64	6 000

图10展示了三种学习率对训练的影响。选取了三种学习率取值随轮次增加的损失变化曲线,适中的学习率使得损失较快下降,而过大的学习率则让损失下降后增加,学习率过小则计算成本增加。

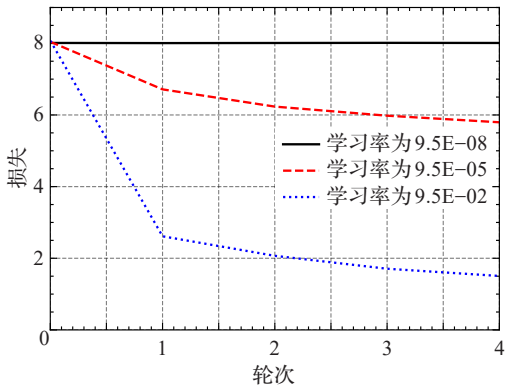

图10　学习率取值的影响

Fig.10　Influence of learning rate value

这说明学习率的选取对训练过程有重要作用,为了降低计算成本和提高学习效果,应该尽量摆脱手动调参,在不影响优化效果的情况下可以选取自适应矩估计类优化器并结合学习率预热进行训练。

(2)学习率预热步数的影响

不同的学习率调整策略也会影响实验过程,采用Adadelta作为优化器,图11选取预热步数为60和6 000进行学习。预热步数为60,经过15次学习明显有损失的下降,调整预热步数为6 000,15次学习的效果明显要弱于60步预热。表6展示了两种学习率预热步数的效果。

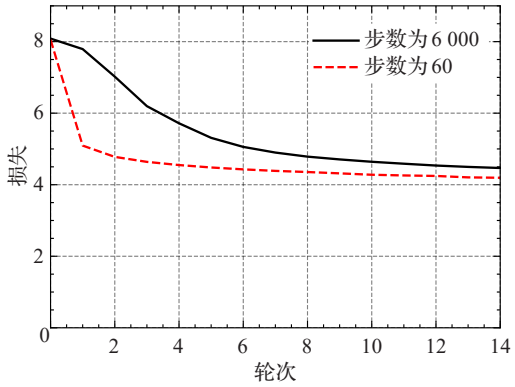

图11 学习率预热步数的影响

Fig.11 Influence of warm-up steps of learning rate

表6 不同学习率预热步数的效果

Table 6 Effect of warm-up steps with different learning rate

步数	学习时间/s	验证损失	测试损失	准确率/%	BLEU
6 000	1 412.63	4.40	4.43	6.37	0.06
60	1 470.82	4.14	4.17	9.94	0.38

学习率预热的优势在于减少了手动调参成本,但同时学习率预热的步数也成为了新的成本,学习率预热步数影响着损失、准确率和BLEU值,在进行训练的时候若选择梯度下降类和自适应学习率类优化器应该要结合学习率预热算法。而对于选取自适应矩估计类优化器的训练过程来说,可以采用不依赖学习率预热的优化器Radam、Ranger,降低学习率预热步数选取的成本。

4.3.2 Lookahead实验

选取随机梯度下降、带动量的随机梯度下降和Adadelta,分别测试了是否使用Lookahead的优化效果,结果如图12。Lookahead能够加速随机梯度下降和带动量的随机梯度下降,对于Adadelta优化性能较差。由实验结果可知,Lookahead能够避免梯度下降中影响优化速度的问题,达到加速优化的效果。因此在使用其他优化器时,应该尽量加入Lookahead,避免优化器落入局部最小值、鞍点等问题。但搭配Adadelta优化器的时候应选择其他更有效的加速优化方法。

4.3.3 优化器对比实验结果

采取了学习率预热和不同学习率结合来调整学习率的策略,并选取了上面介绍的多种优化器对同一数据集进行测试。测试的内容包括:法-英数据集的学习,比较多种优化器的学习损失下降情况。

图13展示了优化器学习损失随轮次变化的情况。动量能够有效加快随机梯度下降学习的速度,但在较短的轮次内学习速度依然无法达到自适应类优化器的水平。

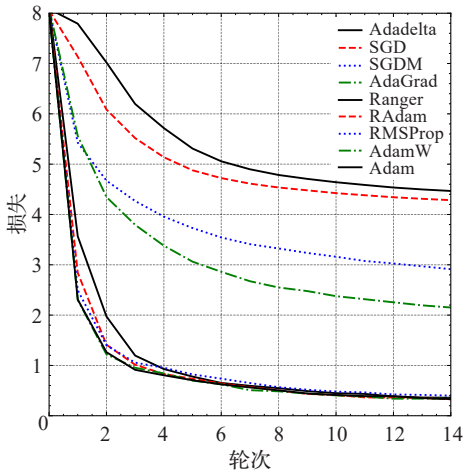

图13 优化器优化速度对比

Fig.13 Comparison of optimization algorithm optimization speed

AdaGrad的效果要好于带动量的随机梯度下降,自适应学习率免去了手动调整的,但学习率预热的选取也对这类方法有一定影响,本例中Adadelta的效果没有达到最优,这是学习率预热步数选取的问题,而采用上一实验中60的参数,就可以在5个轮次内达到最合适的效果。

Adam、AdamW、RAdam、RMSProp、Ranger能够在15个轮次内达到较低损失。在验证集上,上述五种的优化性能也能在较少轮次内能达到较好效果,如图14所示是优化器的学习和验证过程损失变化。

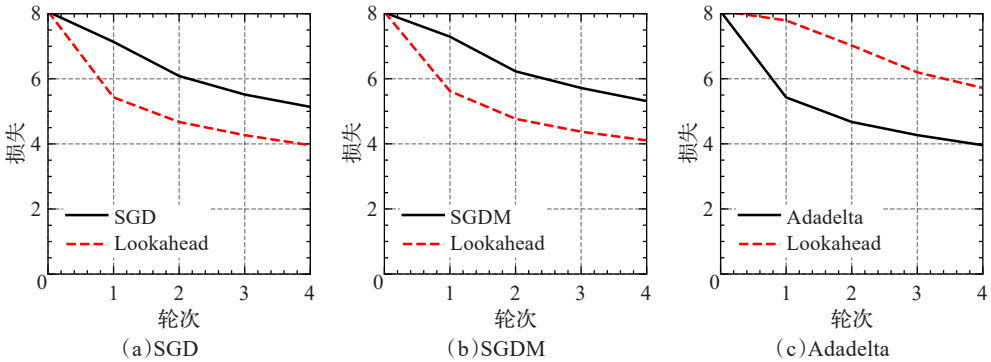

图12 Lookahead优化器的优化效果

Fig.12 Optimization effect of Lookahead optimizer

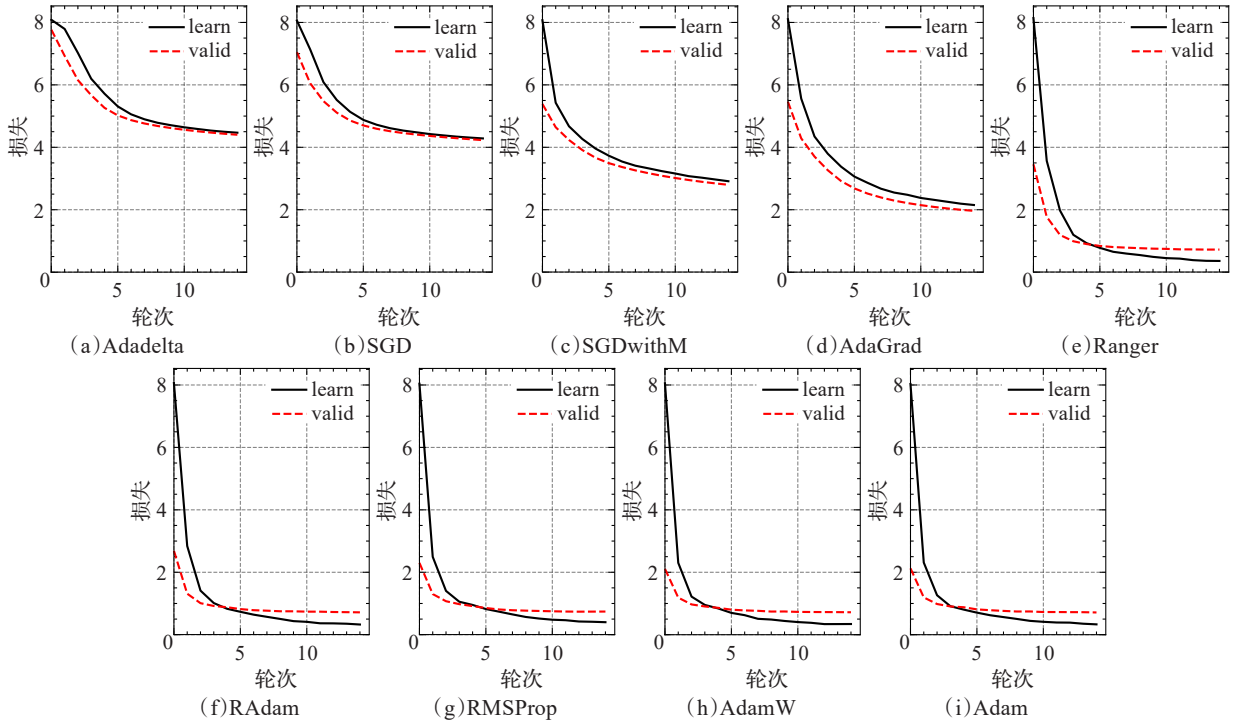

图14　优化器的优化效果

Fig.14　Optimization effect of optimizer

在固定epoch内,学习的时间是相近的,从损失来看,自适应优化器改进了传统的梯度下降的速度。此外,学习率预热的步数则影响学习的效果,以Adadelta为代表,改变预热步数能够改善该优化器的学习效果。表7展示了优化器的效果。

表7　优化器效果对比

Table 7　Comparison of optimizer effects

优化器	学习时间/s	验证损失	测试损失	准确率/%	BLEU
SGD	1 312.4	4.23	4.26	7.51	0.37
SGDM	1 320.0	2.80	2.78	23.15	8.45
AdaGrad	1 516.4	1.96	1.93	35.05	22.73
RMSProp	1 363.5	0.74	0.72	65.14	62.53
Adadelta	1 412.6	4.40	4.43	5.81	22.72
Adam	1 379.0	**0.71**	0.70	66.51	64.32
AdamW	1 466.9	0.72	**0.69**	**66.89**	64.04
RAdam	1 385.8	0.72	0.70	66.01	**64.82**
Ranger	1 516.4	0.72	0.70	65.56	64.42

自适应矩估计类优化器在15个轮次内就能取得最快优化速度,若只考虑优化速度则应该选取该类优化器。其中,RAdam和Ranger类优化器降低摆脱学习率预热调参成本,并且翻译质量较高,应该优先考虑这两类优化器。在模型的准确率上则是Adam和AdamW优化器占优势,考虑学习效果时优先选择这两类优化器。梯度下降优化器虽然在速度上有劣势,但其更新公式较为简单,因此模型稳定性上占有优势,对于深度学习的大数据量数据集,应该优先考虑随机梯度下降,若数据集较小,可以选择全梯度下降。为了平衡两者的优劣,可以直接选择小批量随机梯度下降。不仅如此,如果还需要考虑优化速度,应该选择带动量的随机梯度下降。

在梯度下降之上,自适应学习率优化器在学习率选择上占据优势,对于数据集中包含过多稀疏数据要考虑AdaGrad。为了降低调参成本,AdaGrad、Adadelta和RMSProp能够自适应学习率,应该优先选择。若需要考虑优化速度,RMSProp是较好的选择。

5　总结与建议

本文剖析了优化器的研究进展,并就不同优化器的特点进行了综述,从优化器的演化来看,梯度下降类优化器直接使用了梯度作为优化的策略,但学习率需要手动调参,这影响了该类优化器的学习过程。

自适应学习率优化器中,学习率预热能够解决一定的学习率选择问题,并且能根据预先设定好的学习策略进行学习率调节。为了解决尖锐最小值问题,带热重启的随机梯度下降采用热重启优化器。AdaGrad[42-43]引入了泄露平均值让优化速度进一步加快[44]并能够适应稀疏,Adadelta在此基础上则优化了学习率的选取,并进一步降低了内存成本。

自适应矩估计类优化器中,Adam结合了上述优化器的优点,实现了快速学习。AdamW提高了泛化能力,AMSGrad提高了收敛性。RAdam调整了初始化方案,减少了学习率预热的调参成本。Lookahead引入了两套更新速度不同的参数从而避免一些优化问题。Ranger和Ranger21结合了众多优化器的优势。

实验部分,本文结合了自然语言处理任务,将Transformer架构和几种优化器结合,全面对比了其性能,从实验结果来看自适应类优化器能够扬长避短,对梯度下降类优化器实现有效改进结果,其中RMSProp[45]、Adam、AdamW、RAdam、Ranger是针对Transformer速度快、性能较好的优化器。

优化器多种多样且评价标准也各不相同,各种优化器的优化水平参差不齐[46-47]。因此给出以下优化器选择建议:

(1)梯度下降优化器

随机梯度下降的优势在于其泛化性,因此以泛化性为主的任务上可以选择随机梯度下降,若数据集较大,可以采用小批量随机梯度下降。如果还需要考虑优化速度,可以采用带动量的随机梯度下降。

(2)自适应学习率优化器

对于存在尖锐最小值的函数,可以使用带热重启的随机梯度下降。存在稀疏问题应该使用AdaGrad,Adadelta解决了梯度累计问题,在对内存有需求的任务上可以使用,或者考虑使用RMSProp。

(3)自适应矩估计优化器

Adam的通用性强,可以直接使用。若Adam不收敛,可以采用AMSGrad,泛化能力不够强可以采用AdamW。若学习率预热的选择成本较高,可以采用RAdam。Lookahead对大多数优化器都有适应性,可以考虑同时结合这些优化器使用。Ranger结合了多种优点,在上述优化器应用性差的时候考虑使用。

近年来,优化器仍然在不断适应新出现的大型模型进行调整、优化。优化器的发展方向可以分为结合预训练模型和基于传统的梯度下降类优化器的改进。

预训练是自然语言处理的热点,针对预训练模型的优化,LARS[48]引入了局部学习率的概念,根据具体层的学习训练调节学习率,并且在8 000批次量上得到了512批次量的效果。

针对随机梯度下降的改进中,SWATS[49]结合了Adam和随机梯度下降的优势,可以从前者切换到后者。SWA[50]观察随机梯度下降的轨迹取平均值也能得到较好的效果,能够得到损失更低的效果。

参考文献:

[1] DIEBOLD F X. What's the big idea?"Big Data" and its origins[J]. MBGD Significance, 2021, 18(1): 36-37.

[2] SZE V, CHEN Y H, EINER J, et al. Hardware for machine learning: challenges and opportunities[C]//2017 IEEE Custom Integrated Circuits Conference (CICC), 2017.

[3] SCHMIDHUBER J. Deep learning in neural networks: an overview[J]. Neural Networks, 2015, 61: 85-117.

[4] TORFI A, SHIRVANI R A, KENESHLOO Y, et al. Natural language processing advancements by deep learning: a survey[J]. arXiv:2003.01200, 2020.

[5] KOROTEEV M. BERT: a review of applications in natural language processing and understanding[J]. arXiv:2103.11943, 2021.

[6] SHEN Y C, HSIA T C, HSU C H. Analysis of electronic health records based on deep learning with natural language processing[J]. Arabian Journal for Science and Engineering, 2021: 1-11.

[7] SUTSKEVER I, MARTENS J, HINTON G E. Generating text with recurrent neural networks[C]//International Conference on Machine Learning, 2016.

[8] YANG Z, YANG D, DYER C, et al. Hierarchical attention networks for document classification[C]//Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2016.

[9] TAN Z X, SU J S, WANG B L, et al. Lattice-to-sequence attentional neural machine translation models[J]. Neurocomputing, 2018, 284: 138-147.

[10] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[C]//Advances in Neural Information Processing Systems, 2017.

[11] HAO S, LEE D H, ZHAO D. Sequence to sequence learning with attention mechanism for short-term passenger flow prediction in large-scale metro system[J]. Transportation Research Part C Emerging Technologies, 2019, 107: 287-300.

[12] BROWN T, MANN B, RYDER N, et al. Language models are few-shot learners[C]//Advances in Neural Information Processing Systems, 2020: 1877-1901.

[13] RADFORD A, NARASIMHAN K, SALIMANS T, et al. Improving language understanding by generative pre-training [Z]. 2018.

[14] RADFORD A, WU J, CHILD R, et al. Language models are unsupervised multitask learners[J]. OpenAI Blog, 2019, 1(8): 9.

[15] DEVLIN J, CHANG M W, LEE K, et al. BERT: pre-training of deep bidirectional transformers for language understanding[J]. arXiv:1810.04805, 2018.

[16] 史加荣, 王丹, 尚凡华, 等. 随机梯度下降算法研究进展[J]. 自动化学报, 2021, 47(9): 2103-2119.
SHI J R, WANG D, SHANG F H, et al. Research advances on stochastic gradient descent algorithms[J]. Acta Automatica Sinica, 2021, 47(9): 2103-2119.

[17] 张慧. 深度学习中优化算法的研究与改进[D]. 北京: 北京邮电大学, 2018.
ZHANG H. Research and improvement of optimization algorithms in deep learning[D]. Beijing: Beijing University of Posts and Telecommunications, 2018.

[18] KINGMA D P, BA J. Adam: a method for stochastic optimi-

zation[J]. arXiv:1412.6980, 2014.
[19] SUTSKEVER I, VINYALS O, LE Q V. Sequence to sequence learning with neural networks[C]//Advances in Neural Information Processing Systems, 2014.
[20] PETERS M, NEUMANN M, ZETTLEMOYER L, et al. Dissecting contextual word embeddings: architecture and representation[J]. arXiv:1808.08949, 2018.
[21] DUCHI J, HAZAN E, SINGER Y. Adaptive subgradient methods for online learning and stochastic optimization[J]. Journal of Machine Learning Research, 2011, 12(7): 2121-2159.
[22] ZEILER M D. ADADELTA: an adaptive learning rate method [J]. arXiv:1212.5701, 2012.
[23] LOSHCHILOV I, HUTTER F. Decou pled weight decay regularization[J]. arXiv:1711.05101, 2017.
[24] LIU L, JIANG H, HE P, et al. On the variance of the adaptive learning rate and beyond[J]. arXiv:1908.03265, 2019.
[25] ZHANG M, LUCAS J, BA J, et al. Lookahead optimizer: k steps forward, 1 step back[C]//Advances in Neural Information Processing Systems, 2019.
[26] WRIGHT L, DEMEURE N. Ranger21: a synergistic deep learning optimizer[J]. arXiv:2106.13731, 2021.
[27] BOTTOU L, CURTIS F E, NOCEDAL J. Optimization methods for large-scale machine learning[J]. Society for Industrial and Applied Mathematics, 2018(2): 223-311.
[28] DEKEL O, GILAD-BACHRACH R, SHAMIR O, et al. Optimal distributed online prediction using mini-batches[J]. arXiv: 1012.1367, 2010.
[29] SMITH L N. No more pesky learning rate guessing games [J]. arXiv:1506.01186, 2015.
[30] SMITH L N. Cyclical learning rates for training neural networks[J]. arXiv:1506.01186, 2015.
[31] O'DONOGHUE B, CANDES E. Adaptive restart for accelerated gradient schemes[J]. Foundations of Computational Mathematics, 2015, 15: 715-732.
[32] LOSHCHILOV I, HUTTER F. SGDR: stochastic gradient descent with restarts[J]. arXiv:1608.03983, 2016.
[33] DINH L , PASCANU R , BENGIO S , et al. Sharp minima can generalize for deep nets[J]. arXiv:1703.04933, 2017.
[34] KESKAR N S, MUDIGERE D, NOCEDAL J, et al. On large-batch training for deep learning: generalization gap and sharp minima[J]. arXiv:1609.04836, 2016.
[35] ZHANG J, HE T, SRA S, et al. Why gradient clipping accelerates training: a theoretical justification for adaptivity[C]// International Conference on Learning Representations, 2020.
[36] IOFFE S, SZEGEDY C. Batch normalization: accelerating deep network training by reducing internal covariate shift [C]//International Conference on Machine Learning, 2015: 448-456.
[37] XIE Z, YUAN L, ZHU Z, et al. Positive-negative momentum: manipulating stochastic gradient noise to improve generalization[C]//International Conference on Machine Learning, 2021: 11448-11458.
[38] GEORGIOU T, SCHMITT S, BÄCK T, et al. Norm loss: an efficient yet effective regularization method for deep neural networks[C]//2020 25th International Conference on Pattern Recognition, 2021: 8812-8818.
[39] XIE Z, SATO I, SUGIYAMA M. Stable weight decay regularization[J]. arXiv:2011.11152, 2020.
[40] MA J, YARATS D. On the adequacy of untuned warmup for adaptive optimization[C]//Proceedings of the AAAI Conference on Artificial Intelligence, 2021: 8828-8836.
[41] ARTETXE M, SCHWENK H. Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond[J]. Transactions of the Association for Computational Linguistics, 2019, 7: 597-610.
[42] IYER N, THEJAS V, KWATRA N, et al. Wide-minima density hypothesis and the explore-exploit learning rate schedule[J]. Journal of Machine Learning Research, 2023, 24(65): 1-37.
[43] PENNINGTON J, SOCHER R, MANNING C. Glove: global vectors for word representation[C]//Proceedings of the Conference on Empirical Methods in Natural Language Processing, 2014.
[44] DEAN J, CORRADO G, MONGA R, et al. Large scale distributed deep networks[C]//Advances in Neural Information Processing Systems, 2012.
[45] TIELEMAN T, HINTON G. Lecture 6.5-Rmsprop: divide the gradient by a running average of its recent magnitude[Z]. Coursera: Neural Networks for Machine Learning, 2021.
[46] ANDRYCHOWICZ M, DENIL M, GOMEZ S, et al. Learning to learn by gradient descent by gradient descent[C]// Advances in Neural Information Processing Systems, 2016.
[47] BABICHEV D, BACH F. Constant step size stochastic gradient descent for probabilistic modeling[J]. arXiv:1804.05567, 2018.
[48] YOU Y, GITMAN I, GINSBURG B. Large batch training of convolutional networks[J]. arXiv:1708.03888, 2017.
[49] KESKAR N S, SOCHER R. Improving generalization performance by switching from Adam to SGD[J]. arXiv:1712.07628, 2017.
[50] IZMAILOV P, PODOPRIKHIN D, GARIPOV T, et al. Averaging weights leads to wider optima and better generalization[J]. arXiv:1803.054071, 2018.