基于 K-FAC 算法的神经网络优化

Neural Network Optimization Based on K-FAC Algorithm

姓　　名：徐嘉浩

学　　号：1652048

学　　院：汽车学院

专　　业：车辆工程

指导教师：王哲

职　　称：教授

2021 年　05 月

目录

摘要 3

Abstract 4

一、引言 5

二、K-FAC 算法概述 6

（一） 块对角的 K-FAC 算法 7

（二） 卷积层的 K-FAC 算法 8

（三） EKFAC 算法 8

三、K-FAC 算法优化 9

（一）K-FAC 算法优化理论 9

1. EKFACnag 算法 9

2. EKFACadam 算法 9

3. EKFACradam 算法 10

（二）EKFACadam 算法伪代码 10

四、实验设计与结果 12

（一）实验设计 12

（二）网格搜索 13

（三）实验结果 13

五、结论 16

参考文献 17

致谢 ..... 错误!未定义书签。

基于 K-FAC 算法的神经网络优化

摘要

深度学习的广泛应用迫切需要训练神经网络的高效精准的优化算法，近似自然梯度下降的 K-FAC 算法为二阶优化算法的发展提供了新思路。目前，将 K-FAC 算法与大批量训练和分布式计算结合的研究结果颇丰，也有将 K-FAC 算法与强化学习等结合达到理想效果。本文主要利用拓展到卷积层上的 K-FAC 算法以及经过特征值修正后得到的 EKFAC 算法来实现对神经网络的优化。同时，借鉴一阶优化算法的优化历程和方向，将 K-FAC 的优化算法 EKFAC 分别与 Nesterov 动量、自适应算法 Adam 以及最近提出的修正 Adam 算法相结合，得到 EKFACnag, EKFACadam 和 EKFACradam 这三种新型优化算法。利用 CIFAR-10 数据集在 ResNet-50 上对以上三种算法进行实验，并与基准算法 SGD, K-FAC 和 EKFAC 进行对比，结果显示，EKFACadam 在训练集和测试集上均达到最高精度，具有良好的优化效果和泛化能力。

关键词：K-FAC 算法；EKFAC 算法；神经网络优化

Neural Network Optimization Based on K-FAC Algorithm

Abstract

The wide application of deep learning urgently requires efficient and accurate optimization algorithms for training neural networks. K-FAC, which is an efficient method for approximating natural gradient, provides a new perspective for the development of second-order optimization algorithms. At present, there are a lot of research results on the combination of K-FAC with large batch training and distributed computation, and some of them have achieved ideal results by combining K-FAC with reinforcement learning. In this paper, the K-FAC algorithm extended to the convolution layer and the EKFAC algorithm obtained after the correction of eigenvalues are used to realize the optimization of the neural network. At the same time, using the optimization history and direction of the first-order optimization algorithm for reference, the K-FAC optimization algorithm EKFAC is combined with Nesterov momentum, the adaptive algorithm ADAM and the recently proposed rectified ADAM algorithm respectively, and three new optimization algorithms, EKFACnag, EKFACadam and EKFACradam, are obtained. The CIFAR-10 data set was used to experiment the above three algorithms on ResNet-50, and compared with the benchmark algorithms SGD, K-FAC and EKFAC. The results show that EKFACadam achieves the highest accuracy in both the training set and the test set, and has good optimization effect and generalization ability.

Keyword: K-FAC, EKFAC, Neural network optimization

一、引言

深度学习近些年来在图像识别，语音识别，自然语言处理等方面都取得了突破进展并有十分亮眼的表现。但在这份漂亮结果的背后是对环境计算能力的高要求和网络训练时间的大投入。目前训练神经网络的主要手段还是利用 SGD, 尽管精心调整后的 SGD 方法以及 SGD 结合动量或批量归一化的方法会有较好的优化效果和泛化能力，但神经网络损失函数的非凸性及其表面上的不平衡曲率限制了 SGD 的效率。而二阶优化方法，如自然梯度下降法，有可能通过修正损失函数的不平衡曲率来加速神经网络的训练。

Martens(2014)[1]对自然梯度下降方法进行了详细的研究，讨论了利用费雪信息矩阵（简称为 FIM）来计算自然梯度方向以及该方法的收敛速度和参数化不变性质。他与 Grosse(2015)[2]提出了近似自然梯度下降的有效算法 K-FAC，这一算法通过将 FIM 对应于整个层的各种大块近似为两个小得多的矩阵的 Kronecker 积得出，并提供了 FIM 逆矩阵近似为块对角和块三对角的两种选择。由于 K-FAC 是针对最普通的全链接神经网络的算法，主流的卷积和循环神经网络还无法使用，因此 Martens 和 Grosse(2016)[4]在几个假设的基础上将 K-FAC 推广到了卷积层上，Martens, Ba 与 Johnson(2018)[8]将 K-FAC 推广到了 RNNs 上。

与此同时，对 K-FAC 算法本身的优化和利用也没有停止。虽然 K-FAC 能比 SGD 更快达到更好的优化效果，但它却引入了额外的计算量。Ba, Grosse 与 Martens(2017)[6]将 K-FAC 的梯度和额外附加的计算量分布在多台机器上，利用 K-FAC 对大型迷你批量的适用性从并行计算中获利从而减轻额外的计算量。Osawa(2019)等人[9]也利用 K-FAC 对大规模的分布式深度卷积神经网络进行了研究；J. G. Pauloski(2020)等人[11]通过对分布式的 K-FAC 加入一系列优化技巧实现了比 SGD 更短的时间收敛到 75.9% MLPerf 基线；也有 L. Ma(2020)等[13]考虑超参数调整的时间后提出 K-FAC 在大批量训练中的低效性。

K-FAC 算法除了利用大批量训练与分布式计算的结合，也应用到了强化学习中的演员评论员算法(Y. Wu 等, 2017[5])并取得了不错的效果，还与科学计算中的预处理方法两级域分解方法结合(N. Tselepidis 等, 2020[12])，以高效的计算方式为 K-FAC 丰富了非对角的曲率信息。

在这一系列 K-FAC 算法的优化与应用中，T. George(2018)等[7]提出的利用

Kronecker 因子分解特征基快速近似自然梯度下降的方法吸引了我的注意。这一方法的核心在于追踪对角线方差，不是在普通的参数空间中，而是在 Kronecker 因子特征基上，简称为 EKFAC。更直观的，EKFAC 等价于在 Kronecker 因子特征基中利用梯度的二阶矩来进行正规化，得到一个新的经过预处理后的梯度。

很自然的，我们可以想到对这一预处理的新梯度添加动量或者 Nesterov 动量，也可以考虑利用对角线梯度信息的自适应算法如 Adagrad, RMSProp 和 Adam 等来进行学习率调整模式的探索。

基于以上想法，本文将在 EKFAC 算法中添加 Nesterov 动量（简称为 EKFACnag），并将 Adam 算法和最近提出的 RAdam 算法应用到 EKFAC 算法中(分别简称为 EKFACadam 和 EKFACradam)，以此来探究对 K-FAC 算法的优化。利用 CIFAR-10 数据集在 ResNet-50 上进行实验，将这三种优化算法与普通的 K-FAC 算法、EKFAC 算法和带动量的 SGD 对比，结果表明自适应的 EKFACadam 算法能达到最高的精度。

本文剩余安排如下：第二部分将对 K-FAC 算法进行概述，第三部分将对本文的创新优化算法进行详细的阐述并给出伪代码，第四部分展示实验设计与结果，第五部分给出结论。

二、K-FAC 算法概述

首先，我们定义本文需要使用的一些基本符号表示。

神经网络通过一系列网络层将输入$x = a_0$转化为输出$f(x,\theta) = a_L$，每一层$l \in \{1,\dots,L\}$需要进行如下计算：

$$s_l = W_l a_{l-1}, \tag{1}$$

$$a_l = \varphi_l(s_l). \tag{2}$$

其中，s_l是前一层输出的加权总和，作为本层的输入量，也称作预激活量，a_l是本层的输出量，也称作激活量，W_l是一个权重矩阵，φ_l表示一个非线性激活函数。为了方便，我们将网络的所有参数连接成一个向量$\theta = (vec(W_1)^T,\dots,vec(W_L)^T)^T$.

训练神经网络的目标是找到一组参数θ使得训练集D_{train}上的平均损失$h(\theta) = E_{(x,y)\in D_{train}}[L(y,f(x,\theta))]$最小化，其中$L(y,z)$是损失函数，定义为预测分布$R_{y|z}$上的负对数似然函数$L(y,z) = -logr(y|z)$。此外，我们还定义$D_\theta = \frac{\partial L(y,f(x,\theta))}{\partial \theta}$，$g_l = D_{s_l}$.

自然梯度下降可以被解释为一种二阶优化方法，参数通过$\theta \leftarrow \theta - \mu F^{-1}\nabla_\theta h$来更新，其中$\mu$是学习率，$F$是费雪信息矩阵，定义为

$$F = E_{\substack{x\in D_{train} \\ y\sim R_{y|f(x,\theta)}}}[D_\theta (D_\theta)^T]. \tag{3}$$

假设神经网络含有n个参数，那么F便是一个$n \times n$矩阵。对于现代神经网络数千万级的参数量，F难以显式表达与计算求逆，因此需要一种能简易计算F或F^{-1}的同时又保留F重要曲率信息的近似结构。

（一）块对角的 K-FAC 算法

块对角的 K-FAC 算法（Martens & Grosse, 2015[2]）基于对F的两次近似估计，将F处理为了可操作求逆的矩阵。首先，假设神经网络的每一层相互独立即每一层的参数导数互不相关，这可以将F近似为块对角矩阵，每一块可表示为

$$F^{(l)} = E[vec(DW_l)vec(DW_l)^T], \tag{4}$$

但这样的块矩阵依然非常巨大，难以计算，因此加入了第二种 Kronecker 因子分解的近似方法。

注意到$DW_l = g_l a_{l-1}^T$，$vec(DW_l) = vec(g_l a_{l-1}^T) = a_{l-1} \otimes g_l$, 同时假设激活量与预激活量的导数相互独立，那么有

$$\begin{aligned} F^{(l)} = E[vec(DW_l)vec(DW_l)^T] &= E[(a_{l-1} \otimes g_l)(a_{l-1} \otimes g_l)^T] \\ &= E[(a_{l-1} \otimes g_l)({a_{l-1}}^T \otimes {g_l}^T)] \\ &= E[a_{l-1}{a_{l-1}}^T \otimes g_l {g_l}^T] \\ &\approx E[a_{l-1}{a_{l-1}}^T] \otimes E[g_l {g_l}^T]. \end{aligned} \tag{5}$$

定义$A_{l-1} = E[a_{l-1}{a_{l-1}}^T], G_l = E[g_l {g_l}^T]$, 得到$F^{(l)} \approx A_{l-1} \otimes G_l \triangleq \tilde{F}^{(l)}$.

我们已知，求块对角矩阵$\tilde{F}$的逆等价于求其中每一个块矩阵$\tilde{F}^{(l)}$的逆，而结合张量积的良好性质，我们可以快速求出$\tilde{F}^{(l)}$的逆

$${\tilde{F}^{(l)}}^{-1} = (A_{l-1} \otimes G_l)^{-1} = {A_{l-1}}^{-1} \otimes {G_l}^{-1}, \tag{6}$$

及其对应的部分近似自然梯度

$${\tilde{F}^{(l)}}^{-1}\nabla_{W_l} h = \left({A_{l-1}}^{-1} \otimes {G_l}^{-1}\right)\nabla_{W_l} h = vec\left({G_l}^{-1}\nabla_{W_l} h {A_{l-1}}^{-1}\right), \tag{7}$$

由此解决了F求解逆矩阵及计算自然梯度的问题。

（二）卷积层的 K-FAC 算法

与上一小节同样的原理可以应用到卷积层的参数梯度协方差的 Kronecker 因子分解表达式上(Grosse & Martens, 2016[4])。通常，我们会假设激活量与预激活量导数相互独立(IAD)；空间同质性(SH)；预激活量导数的空间不相关性(SUD)或者激活量的空间不相关性(SUA)。结合以上三条假设，我们可以得到

$$F^{(l)} \approx A_{l-1} \otimes G_l, \tag{8}$$

其中，$A_{l-1} = E\left[\sum_{t,\, t' \in T} a_{l-1,t} {a_{l-1,t'}}^T\right]$, $G_l = E[\sum_{t,\, t' \in T} g_{l,t} {g_{l,t'}}^T]$. $t,\ t' \in T$表示由卷积核遍历的空间位置，$a_{l-1,t}$表示在位置t的子激活量，$g_{l,t}$表示在位置t的预激活量的导数。

（三）EKFAC 算法

EKFAC 算法(George 等，2018[7])的核心是在 Kronecker 因子分解特征基中对梯度向量$\nabla_\theta h$除以相应二阶矩来进行预处理。这一近似操作是单独在每一个块$F^{(l)}$上进行的，为了简洁，本节将忽略这一上标。

由 K-FAC 算法，我们已经得知$F \approx A \otimes G$. 考虑A的特征分解$A = U_A S_A U_A^T$和G的特征分解$G = U_G S_G U_G^T$, 这推出$A \otimes G$的如下特征分解

$$\begin{aligned} A \otimes G &= (U_A S_A U_A^T) \otimes (U_G S_G U_G^T) \\ &= (U_A \otimes U_G)(S_A \otimes S_G)(U_A \otimes U_G)^T, \end{aligned} \tag{9}$$

其中，$U_A \otimes U_G$称作 Kronecker 因子分解特征基(KFE)，$S_A \otimes S_G$是含有矩阵A和G相关特征值的对角矩阵。

由于尺度变换因子$(S_A \otimes S_G)_{ii}$并不一定保证与沿相关特征向量的二阶矩$\mathrm{E}[((U_A \otimes U_G)^T \nabla_\theta h)_i^2]$匹配，因此，我们用上述二阶矩对这一对角矩阵$S_A \otimes S_G$ 进行修正，得到新的对角阵S^*, 其对角元素定义为

$$S_{ii}^* = s_i^* = \mathrm{E}[((U_A \otimes U_G)^T \nabla_\theta h)_i^2]. \tag{10}$$

由此，我们得到了经过特征值修正过后的 K-FAC 算法，简称为 EKFAC 算法。

利用 EKFAC 算法计算自然梯度等价于进行如下操作

$$\begin{aligned} \mathrm{F}_{EKFAC}^{-1} \nabla_\theta h &= [(U_A \otimes U_G) S^* (U_A \otimes U_G)^T]^{-1} \nabla_\theta h \\ &= (U_A \otimes U_G) {S^*}^{-1} (U_A \otimes U_G)^T \nabla_\theta h, \end{aligned} \tag{11}$$

其中，$(U_A \otimes U_G)^T \nabla_\theta h$相当于将梯度映射到 KFE 中得到$\tilde{\nabla}$，${S^*}^{-1}\tilde{\nabla}$表示在 KFE 中进行

尺度调节得到新的$\tilde{\nabla}$，而$(U_A \otimes U_G)\tilde{\nabla}$则表示将梯度拉回参数空间得到预处理梯度$\tilde{\nabla}_{pre}$.

三、K-FAC 算法优化

（一）K-FAC 算法优化理论

对二阶算法的优化可以从一阶算法优化的进程中找到灵感，本文探究的 K-FAC 优化算法也不例外。

在机器学习领域，研究者对 SGD 算法的优化没有停止过脚步。从平衡计算量与准确率的角度，小批量的随机梯度下降法横空出世，代替了传统的随机利用一个训练数据的 SGD 和利用整个训练数据集的批量 SGD，成为现在的主流算法，本文训练的 K-FAC 优化算法也使用了这一随机技巧，添加了小批量进行梯度下降。为了避免神经网络悬崖结构带来的梯度爆炸问题，梯度截断通常在传统的梯度下降法提议更新很大一步时使用以减小步长，本文的 K-FAC 优化算法也将利用这一技巧。

1. EKFACnag 算法

除此之外，为了解决 SGD 在遇到平坦或高曲率区域学习过程缓慢、振幅较大的问题，引入了动量概念，提高了算法的收敛速度和稳定性。动量算法是由之前下降方向的一个累积和当前梯度方向组合而成，而 Nesterov 通过先按照历史梯度向前走一小步，再在这一位置修正梯度方向得到了一个加速梯度，这种预更新方法简称为 NAG 算法。本文将在前面提到的 EKFAC 算法中使用这一 Nesterov 加速梯度(简称 EKFACnag 算法)，进行 K-FAC 算法的优化，探究其在神经网络中的效用。假设学习率为μ, 动量参数为α, 初始速度为v, 初始参数为θ, $\tilde{\nabla}_{pre}$为在超前点$\tilde{\theta} \leftarrow \theta + \alpha v$经过 EKFAC 算法预处理的梯度，那么 EKFACnag 算法的核心操作如下：

$$v \leftarrow \alpha v - \mu\tilde{\nabla}_{pre},$$

$$\theta \leftarrow \theta + v. \tag{12}$$

2. EKFACadam 算法

学习率是影响优化算法效率最重要的超参数之一，对其调整模式的探究在神经网

络的训练中意义非凡。Adam 本质上是带有动量的 RMSProp 算法，它利用梯度的一阶矩和二阶矩估计动态调整每个参数的学习率，做到了对梯度方向和学习率的同时优化，并且是一种自适应算法，在神经网络的训练中取得了非常好的效果。本文将探究二阶优化算法的自适应模式，利用前文 EKFAC 算法获得的预处理梯度$\widetilde{\nabla}_{pre}$，计算其一阶和二阶矩估计，并进行移动指数平均和相应修正，最后实现对参数的更新。沿用上一小节假设，并补充假设一阶矩与二阶矩指数衰减速率为ρ_1和ρ_2，取值在$[0,1)$之间；以及一个为了数值稳定设置的小参数δ；初始一阶矩和二阶矩分别为$\mathrm{s}=0,\mathrm{r}=0$. 在每一个时间步$t$内，我们进行操作：

$$\mathrm{s} \leftarrow \rho_1 s + (1-\rho_1)\widetilde{\nabla}_{pre},$$

$$r \leftarrow \rho_2 r + (1-\rho_2)\widetilde{\nabla}_{pre} \odot \widetilde{\nabla}_{pre},$$

$$\hat{s} = \frac{s}{1-{\rho_1}^t},$$

$$\hat{r} = \frac{r}{1-{\rho_2}^t},$$

$$\Delta\theta = -\mu\frac{\hat{s}}{\sqrt{\hat{r}}+\delta}. \tag{13}$$

最后得到参数更新$\theta \leftarrow \theta + \Delta\theta$.

3. EKFACradam 算法

最近，一种修正的 Adam 算法(Liu 等，2020[10])的提出加快了一阶优化算法的收敛效率。RAdam 通过对早期模型训练时，各参数自适应学习率方差很大的问题进行修正，使得各参数学习率的方差在训练过程中保持恒定。本文也将探究这一思想在二阶优化算法的应用，与前文 EKFAC 算法进行结合，我们得到 EKFACradam 算法。

（二）EKFACadam 算法伪代码

本小节以 EKFACadam 为例给出伪代码。

算法 1 EKFACadam 算法

Require: 神经网络初始参数θ, 全局学习率μ, 权重衰减惩罚系数σ, 阻尼系数τ, 一阶矩与二阶矩指数衰减速率$\rho_1 = 0.9$和$\rho_2 = 0.999$, 数值稳定常数$\delta = 10^{-8}$, 梯度截断参数$C = 10^{-2}$, Kronecker 因子A, G衰减指数$\epsilon = 0.95$, 卷积层 Kronecker 因子A, G更新频率$T_c = 10$, 尺度调节因子s^*更新频率$T_s = 10$, 进行特征分解并计算逆矩阵的频率$T_{inv} = 100$.

procedure EKFACadam(D_{train})

while 没有达到停止准则，每个迭代i **do**

从D_{train}中选取一个迷你批量D, 通过前向和后向传播得到神经网络每一层l的a_l和g_l.

for all 神经网络层l **do**

if i%T_{inv}= 0 **then**

$UpdateInv(D, l)$

end if

if i%T_s= 0 **then**

$UpdateScale(D, l)$

end if

$\nabla_{mini} \leftarrow E_{(x,y)\in D}[\frac{\partial L(y, f(x,\theta))}{\partial \theta^{(l)}}]$

$\widetilde{\nabla}_{pre} \leftarrow GetNaturalGradient(\nabla_{mini}, l)$

end for

$KLClipandUpdateGrad(\widetilde{\nabla}_{pre})$

$AdamUpdateParameters(closure)$

end while

end procedure

procedure $UpdateInv(D, l)$

${U_A}^{(l)} {S_A}^{(l)} {U_A}^{(l)^T} \leftarrow A_{l-1}$,

${U_G}^{(l)} {S_G}^{(l)} {U_G}^{(l)^T} \leftarrow G_l$

end procedure

procedure $UpdateScale(D, l)$

$s^{*(l)} \leftarrow E_D\left[\left(\left({U_A}^{(l)} \otimes {U_G}^{(l)}\right)^T \frac{\partial L(y, f(x,\theta))}{\partial \theta^{(l)}}\right)^2\right]$

end procedure

procedure $GetNaturalGradient(\nabla_{mini}, l)$

$\tilde{\nabla} \leftarrow \left({U_A}^{(l)} \otimes {U_G}^{(l)}\right)^T \nabla_{mini}$,

$\tilde{\nabla} \leftarrow \frac{\tilde{\nabla}}{s^{*(l)} + \tau}$,

$\tilde{\nabla}_{pre} \leftarrow \left({U_A}^{(l)} \otimes {U_G}^{(l)}\right) \tilde{\nabla}$.

end procedure

其中 Adam 更新参数函数可参照上一小节中 EKFACadam 算法部分进行，梯度截断函数就是常规的截断操作。而 EKFACnag 和 EKFACradam 算法的伪代码可以通过将$AdamUpdateParameters(closure)$这一 Adam 更新参数函数更改为相应的 Nag 和 RAdam 更新参数函数即可。

四、实验设计与结果

（一）实验设计

本小节的实验在 4 块 MSI GeForce GTX 1080ti aero 11G 的环境下进行。利用 CIFAR-10 数据集在 ResNet-50 上对本文所提出的 EKFACnag, EKFACadam 和 EKFACradam 算法以及带动量的 SGD, K-FAC 与 EKFAC 这三种基准算法进行实验对比。

实验中所需要的全局学习率μ, 权重衰减惩罚系数σ和阻尼系数τ这三个超参数，我们通过网格搜索来确定每一个算法的具体参数。其余参数采用如前文 EKFACadam 算法伪代码中的默认设置，特别的，我们设定训练模型中迷你小批量大小为 64，测试模型中的迷你小批量大小为 256。

（二）网格搜索

对实验所需的二阶优化算法，假设学习率的取值集合为{0.03, 0.01, 0.003}, 权重衰减惩罚系数的取值集合为{0.01, 0.003, 0.001,0.0003,0.0001}, 阻尼系数的取值集合为{0.03, 0.001, 0.003}, 对每一组系数组合训练 5 个 epoch, 以训练目标函数即训练损失值为判断标准，选择使得每个算法训练损失最小化的一组参数组合，作为之后训练选定的超参数。

对实验所需的一阶优化算法，则假设学习率取值集合为{0.3, 0.1, 0.03}, 权重衰减惩罚系数的取值集合依然为{0.01, 0.003, 0.001,0.0003,0.0001}, 不需要阻尼系数，对每一组系数组合训练 10 个 epoch, 选出使得训练损失最小的一组参数。

各算法最终选择的超参数值如下表所示：

表 1 各算法的超参数选择

Table 1 Hyperparameter selection of each algorithm

算法	学习率	权重衰减系数	阻尼系数
SGD	0.03	0.0001	-
K-FAC	0.01	0.0003	0.03
EKFAC	0.01	0.0003	0.03
EKFACnag	0.01	0.0001	0.03
EKFACadam	0.003	0.001	0.001
EKFACradam	0.01	0.0003	0.001

（三）实验结果

按上述实验设计和参数设定对各算法进行 100 个 epoch 的训练，得到下图所示的训练结果，其中，纵轴数值以对数尺度表达。

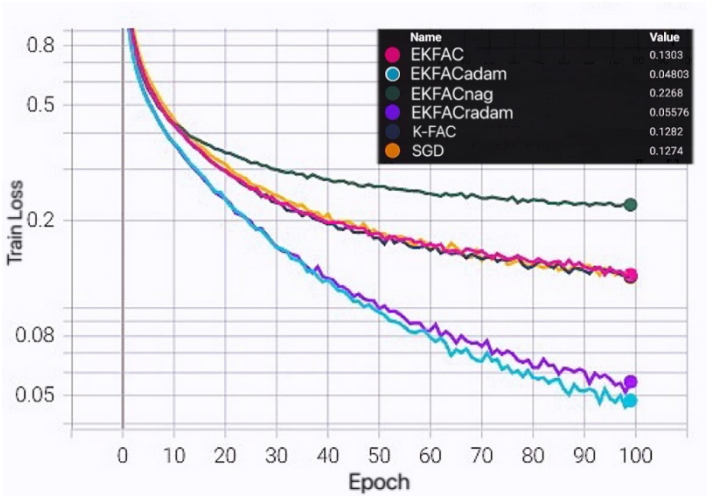

图 1 各算法在 100 个 epoch 内的训练损失

Fig. 1 Training loss of each algorithm within 100 epochs

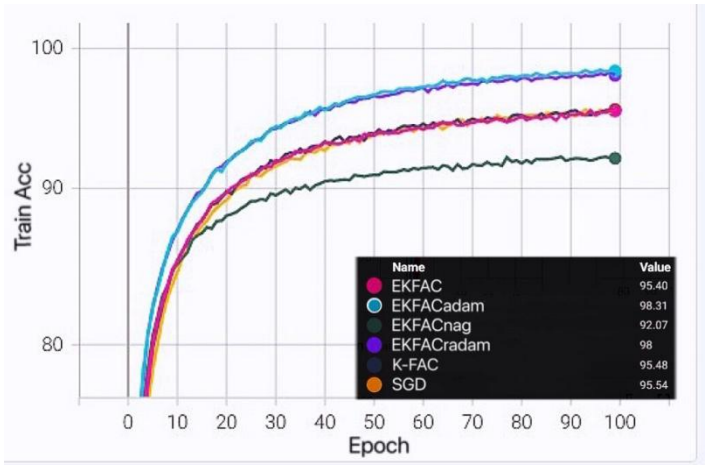

图 2 各算法在 100 个 epoch 内的训练精度

Fig. 2 Training accuracy of each algorithm within 100 epochs

由图 1 可以看出，在训练初期的 10 个 epoch 内，本文所提出的三种算法 EKFACnag, EKFACadam 和 EKFACradam 在每个 epoch 里均能比三种基准算法更快的最小化训练损失。并且，EKFACadam 和 EKFACradam 将这一优势保持了整个训练过程，而 EKFACnag 在训练中后期优化速度有所减慢，被三种基准算法超越。此外，EKFACadam 在训练后期最小化损失的速度逐渐与 EKFACradam 拉开差距，成为这一实验中优化效果最好的算法。

图 2 展示了训练过程每 epoch 的图像分类准确率，可以得出，EKFACadam 和 EKFACradam 达到了这一实验里的最高精度，分别为 98.31%与 98%，三种基准算法均达到 95%左右的精度，由于 EKFACnag 中后期优化速度的减慢，最后仅达到 92.07%

的精度。

上述结论说明本文提出的最优优化算法 EKFACadam 能在给定 epoch 的条件下最快的最小化训练损失并将分类精度比基准算法提高 3 个百分点，证明了这一优化算法的可靠性。

各算法在测试集上的结果由下图所示。从图 3 可以看出 EKFACadam 在验证过程前半段依然能够最快的最小化目标损失，并且在图 4 中仍然达到了测试集验证的最高精度 90.52%，这表明 EKFACadam 在这一实验里还具有良好的泛化能力。

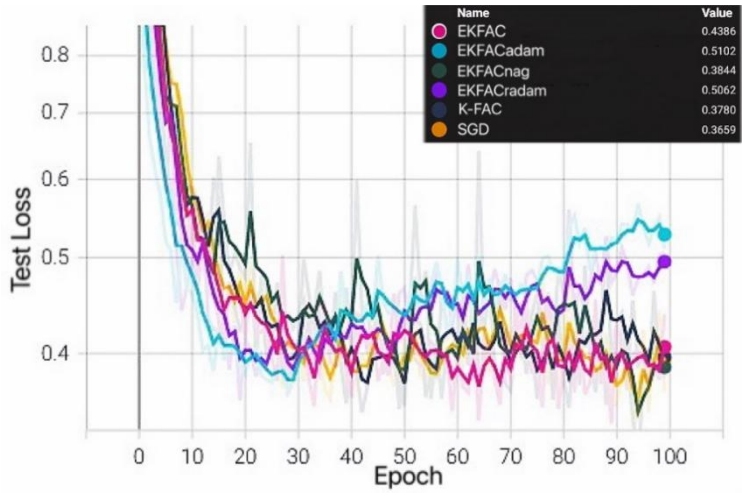

图 3 各算法在 100 个 epoch 内的测试损失

Fig. 3 Test loss of each algorithm within 100 epochs

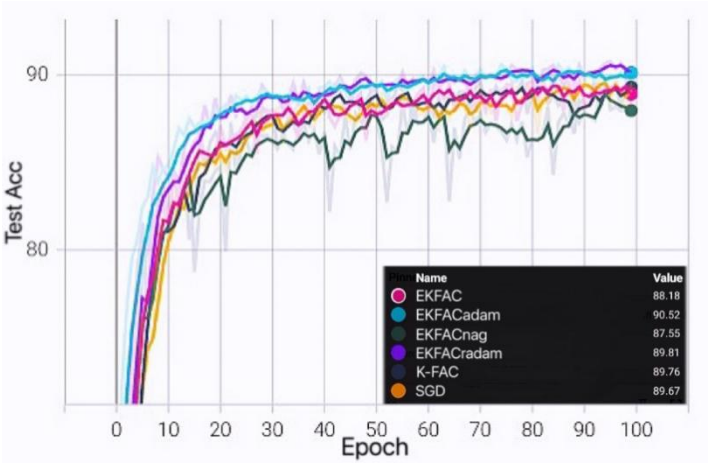

图 4 各算法在 100 个 epoch 内的测试精度

Fig. 4 Test accuracy of each algorithm within 100 epochs

五、结论

本文基于 K-FAC 算法及其相关理论如在卷积层上应用 K-FAC 和利用特征值修正 K-FAC 得到的 EKFAC 算法来实现对神经网络的优化。同时，借鉴一阶优化算法的优化历程与优化方向，本文首先在 K-FAC 的优化算法 EKFAC 的基础上添加了 Nesterov 动量得到了本文第一个优化算法 EKFACnag, 然后考虑到学习率的自适应模式与动量相结合的优化效果，本文提出了第二个优化算法 EKFACadam, 最后将新提出的修正的 Adam 算法应用到二阶优化中，提出 EKFACradam, 探究其优化效果。

利用 CIFAR-10 数据集在 ResNet-50 上对上述提出的三种优化算法进行实验，并与基准算法对比，实验结果显示，EKFACnag 仅能在训练前期快速最小化目标损失，而 EKFACadam 和 EKFACradam 能在整个训练过程保持比基准算法更快的速度最小化损失。并且，就图像分类准确率来说，EKFACadam 在训练集和测试集上均达到了最高精度，证明了 EKFACadam 的良好优化效果和泛化能力。

参考文献

[1] J. Martens. New insights and perspectives on the natural gradient method[J]. 2014, arXiv: 1412.1193.

[2] J. Martens and R. Grosse. Optimizing Neural Networks with Kronecker-factored Approximate Curvature[C]. *Proceedings of the 32nd International Conference on Machine Learning*, Lille, France, 2015.

[3] K. He, X. Zhang, S. Ren and J. Sun. Deep Residual Learning for Image Recognition[J]. 2015, arXiv: 1512.03385.

[4] R. Grosse and J. Martens. A Kronecker-factored approximate Fisher matrix for convolution layers[J]. *Proceedings of the 33rd International Conference on Machine Learning*. 2016, 48: 573-582.

[5] Y. Wu, E. Mansimov, S. Liao, R. Grosse and J. Ba. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation[J]. 2017, arXiv: 1708.05144.

[6] J. Ba, R. Grosse and J. Martens. Distributed Second-Order Optimization using Kronecker-Factored Approximations[J]. *ICLR*. 2017.

[7] T. George, C. Laurent, X. Bouthillier, N. Ballas and P. Vincent. Fast Approximate Natural Gradient Descent in a Kronecker-factored Eigenbasis[J]. 2018, arXiv: 1806.03884.

[8] J. Martens, J. Ba and M. Johnson. Kronecker-factored Curvature Approximations for Recurrent Neural Networks[J]. *ICLR*. 2018.

[9] K. Osawa, Y. Tsuji, Y. Ueno, A. Naruse, R. Yokota and S. Matsuoka. Large-scale distributed second-order optimization using kronecker-factored approximate curvature for deep convolutional neural networks[J]. 2019, arXiv: 1811.12019.

[10] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao and J. Han. On the Variance of the Adaptive Learning Rate and Beyond[J]. *ICLR*. 2020.

[11] J. G. Pauloski, Z. Zhang, L. Huang, W. Xu and Ian T. Foster. Convolutional Neural Network Training with Distributed K-FAC[J]. 2020, arXiv: 2007.00784.

[12] N. Tselepidis, J. Kohler and A. Orvieto. Two-Level K-FAC Preconditioning for Deep Learning[J]. 2020, arXiv: 2011.00573.

[13] L. Ma, G. Montague, J. Ye, Z. Yao, A. Gholami, K. Keutzer and Michael W. Mahoney. Inefficiency of K-FAC for Large Batch Size Training[J]. *AAAI-20.* 2020.

[14] Y. Ueno, K. Osawa, Y. Tsuji, A. Naruse, R. Yokota and S. Matsuoka. Rich Information is Affordable: A Systematic Performance Analysis of Second-order Optimization Using K-FAC[J]. *The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining.* 2020.

[15] Y. Chen. An iterative K-FAC algorithm for Deep Learning[J]. 2021, arXiv: 2101.00218.

[16] 邱锡鹏. 神经网络与深度学习[M]. 2020, 77-188.

[17] 伊恩.古德费洛等. 深度学习[M]. 人民邮电出版社, 2017, 196-316.

[18] 吴茂贵, 郁明敏, 杨本法, 李涛, 张粤磊. Python 深度学习基于 PyTorch[M]. 北京: 机械工业出版社, 2020, 21-137.

[19] 赵英俊. TensorFlow2.0 深度学习应用编程快速入门[M]. 北京: 电子工业出版社, 2019, 21-52.