EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Summary of "Large-Scale Distributed Second-Order Optimization Using Kronecker-Factored Approximate Curvature for Deep Convolutional Neural Networks"

Robin M. Schmidt
Department of Computer Science
Eberhard-Karls-University Tübingen
Tübingen, Germany
ROB.SCHMIDT@STUDENT.UNI-TUEBINGEN.DE

November 2, 2019

Seminar Winter 2019/2020
Advanced Topics in Computer Graphics and Computer Vision

Summary of "Large-Scale Distributed Second-Order Optimization Using Kronecker-Factored Approximate Curvature for Deep Convolutional Neural Networks"

Robin M. Schmidt
Department of Computer Science
Eberhard-Karls-University Tübingen
Tübingen, Germany
rob.schmidt@student.uni-tuebingen.de

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1 Introduction

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2 Notation

We use the same notation as described in [Zha19] this alternates the notation from [OTU+18] a little. In our notation the training data $\mathcal{T} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \cdots, (\mathbf{x}_n, \mathbf{y}_n)\}$ consists of n feature-label pairs. Here, each $\mathbf{x}_i \in \mathbb{R}^{d_x}$ is the feature vector and $\mathbf{y}_i \in \mathbb{R}^{d_y}$ is the label vector with their respective sizes d_x and d_y. The deep learning model is described as a mapping $F(\cdot;\boldsymbol{\theta}) : \mathcal{X} \to \mathcal{Y}$ from the feature space $\mathcal{X}$ to the label space $\mathcal{Y}$ where $\boldsymbol{\theta}$ are the parameters of the model. This leaves us with a model notation which when presented with an input instance $\mathbf{x}_i \in \mathcal{X}$ yields a predicted output denoted as $\hat{\mathbf{y}}_i = F(\mathbf{x}_i;\boldsymbol{\theta})$. The difference of the true label $\mathbf{y}_i \in \mathcal{Y}$ to this predicted label $\hat{\mathbf{y}}_i$ is then described as the loss term $\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i)$ which can have different definitions based on the specific problem (e.g. Mean Squared Error, Hinge Loss, Cross Entropy Loss, etc.). By summing up over all data points in the training data we get the total loss term defined as:

$$\mathcal{L}(\boldsymbol{\theta};\mathcal{T}) = \sum_{(\mathbf{x}_i,\mathbf{y}_i)\in\mathcal{T}} \ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) \tag{1}$$

During training we try to optimize the model parameters $\boldsymbol{\theta} \in \mathbb{R}^{d_\theta}$, which are part of the variable domain Θ, by minimizing the total loss on the training set. This can be described as:

$$\min_{\boldsymbol{\theta}\in\Theta} \mathcal{L}(\boldsymbol{\theta};\mathcal{T}) \tag{2}$$

For this process of finding a global or local minimum for convex and non-convex loss surfaces, a variety of different optimization algorithms are available. Most of these algorithms use the learning rate η to determine the step sizes taken for the parameters $\boldsymbol{\theta}$ at each update step τ in the opposite direction of the gradient of the loss function $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}^{(\tau-1)};\cdot)$. Here, $\boldsymbol{\theta}^{(\tau)}$ are the parameters of the model at the update step τ with $\tau \geq 1$. The initialized hyperparameter values are denoted as $\boldsymbol{\theta}^{(0)}$.

3 Related Work

Related Work in the realm of Deep Learning Optimizers can broudly be classified in First- and Second-Order Optimization Algorithms. Here we want to give a quick overview over those two areas.

3.1 First-Order Optimization Algorithms

There are various First-Order Optimization Algorithms which are widely used in Deep Learning. One of the most popular choices due to its simplicity is still *Stochastic Gradient Descent* (SGD) [RM51] with its update rule shown in Equation 3:

$$\boldsymbol{\theta}^{(\tau)} = \boldsymbol{\theta}^{(\tau-1)} - \eta \cdot \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\boldsymbol{\theta}^{(\tau-1)}; (\mathbf{x}_i, \mathbf{y}_i)\right) \quad (3)$$

However, there have been recent advances yielding new and improved First-Order Optimizers such as *Adam* [KB14], *AdamW* [LH17], *AMSGrad* [RKK19], *AdaBound* [LXLS19], *AMSBound* [LXLS19], *RAdam* [LJH+19], *LookAhead* [ZLHB19], *Ranger*[1] and many more which offer time-convergence improvements based on Adaptive Gradient methods and Momentum Terms. For a more detailed description please see [Rud16, Zha19].

3.2 Second-Order Optimization Algorithms

The generalized *Gauss-Newton-Method* [Sch02] and *Natural Gradient Descent* (NGD) [Ama98] set the groundwork for improvements on Second-Order Optimization Algorithms [KBH19, Mar14, DHH19, BRB17, PB13]. Such work yielded the *Kronecker-factored Approximate Curvature* (K-FAC) [MG15] which effeciently approximates the empirical Fisher information matrix (FIM) $\mathbf{F}_{\boldsymbol{\theta}}$ given by Equation 4 through block-diagonalization and Kronecker factorization of these blocks.

$$\mathbf{F}_{\boldsymbol{\theta}} = \mathop{\mathbb{E}}_{p(\mathbf{x},\mathbf{y})} \left[\nabla \log p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) \nabla \log p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})^T\right] \quad (4)$$

For a neural network with L Layers K-FAC approximates $\mathbf{F}_{\boldsymbol{\theta}}$ as displayed in Equation 5 with $\mathbf{F}_\ell$ being the block matrix for the FIM of the ℓ th layer:

$$\mathbf{F}_{\boldsymbol{\theta}} \approx \operatorname{diag}\left(\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_\ell, \ldots \mathbf{F}_L\right) \quad (5)$$

Each block is then approximated using the Kronecker-factorization:

$$\mathbf{F}_\ell \approx \mathbf{G}_\ell \otimes \mathbf{A}_{\ell-1} \quad (6)$$

[1] `https://github.com/lessw2020/Ranger-Deep-Learning-Optimizer`

With the properties of the Kronecker-factorization we can write the blocks as:

$$\mathcal{G}_\ell^{(\tau-1)} = \left(\mathbf{G}_\ell^{(\tau-1)^{-1}} \otimes \mathbf{A}_{\ell-1}^{(\tau-1)^{-1}}\right) \quad (7)$$

Now using the NGD update rule we get the update rule for the parameters $\boldsymbol{\theta}_\ell^{(\tau)}$:

$$\boldsymbol{\theta}_\ell^{(\tau)} = \boldsymbol{\theta}_\ell^{(\tau-1)} - \eta \cdot \mathcal{G}_\ell^{(\tau-1)} \cdot \nabla \mathcal{L}_\ell\left(\boldsymbol{\theta}_\ell^{(\tau-1)}; \cdot\right) \quad (8)$$

Besides the problem of inverting infeasible large matrices such as the FIM or the Hessian, which K-FAC tries to solve, a common drawback for Second-order optimizers is the complexity to optimize them for distributed computing. This is where [OTU+18] tries to contribute a method which will improve the state-of-the-art.

4 Parallelized K-FAC

The design which gets proposed in [OTU+18] is visualised in figure 1. Each stage corresponds to a needed step of computation, here representative with 2 GPUs and a 3 layer neural network. In the first two stages $\mathbf{A}_{\ell-1}$ and $\mathbf{G}_\ell$ get computed by forward and backward passing the input through the network. For that, each process uses different mini-batches to calculate the Kronecker factors. After that, the values of these factors get summed up to calculate the global factors and the results get distributed to the different processes (ReduceScatterV) to keep model-parallelism. The purpose of distributing the results to each process is so that every GPU can compute the preconditioned gradient $\mathcal{G}_\ell$ for a different layer ℓ. If there are more layers than processes then one process computes multiple preconditioned gradients as shown in Stage 3 of figure 1. Stage 4 and Stage 5 are respectively the inverse computation stage and the matrix multiplication stage needed for equation 7. After stage 5 we distribute each $\mathcal{G}_\ell$ to each process (AllGatherV) to reach stage 6 where each process can now update the parameters $\boldsymbol{\theta}$ by using the preconditioned gradients. In [OTU+18] they also use some methods to speed up communication, use damping [MG15] for the FIM to make training more stable as well as learning rate schedules and momentum for K-FAC to speed up convergence. These methods are not explicitly explained here since they are not the main contribution of this work and have been applied in other settings as well.

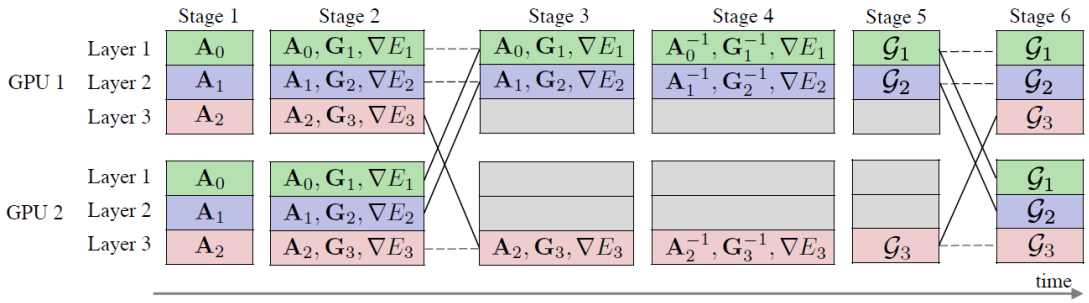

Figure 1: Different stages of distributed K-FAC [OTU+18]

5 Results

All of the presented results are taken from [OTU+18] which obtained them by training ResNet-50 [HZRS15] for ImageNet [DDS+09]. According to figure 2 their results show that the optimal amount of GPUs to use for their experimental setup is 64. After that, the overhead for communication becomes too large which causes a sharp increase in iteration cost. They are able to achieve a really competi-

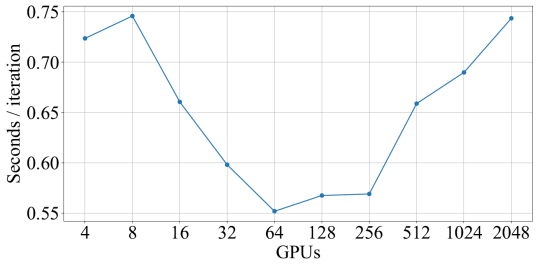

Figure 2: Time per iteration of K-FAC on ResNet-50 using different amount of GPUs [OTU+18]

tive validation accuracy of $\geq 75\%$ using really large batch sizes (BS) which none other first-order optimization method is able to sustain. The respective training curves with their learning rates and batch sizes are shown in figure 3. If we compare the batch sizes for other first-order based methods on the same problem set we can see that the high validation accuracies ($\sim 76\%$) achieved by those methods commonly use batch sizes ≤ 32K [OTU+18].

6 Conclusion & Outlook

Generally, with the obtained results [OTU+18] was able to show that second-order optimization algorithms do in fact generalize relatively similar to

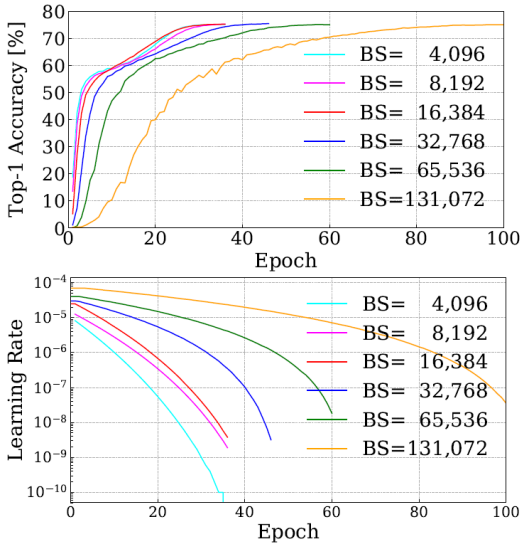

Figure 3: Validation accuracy and learning rate schedules on ResNet-50 [OTU+18]

SGD approaches even for large mini-batch sizes.

References

[Ama98] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, February 1998.

[BRB17] Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical gauss-newton optimisation for deep learning, 2017.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei Fei Li. Imagenet: a large-scale hierarchical image database. pages 248–255, 06 2009.

[DHH19] Felix Dangel, Philipp Hennig, and Stefan Harmeling. Modular block-diagonal curvature approximations for feedforward architectures, 2019.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[KB14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[KBH19] Frederik Kunstner, Lukas Balles, and Philipp Hennig. Limitations of the empirical fisher approximation. 2019.

[LH17] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam, 2017.

[LJH+19] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond, 2019.

[LXLS19] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate, 2019.

[Mar14] James Martens. New insights and perspectives on the natural gradient method, 2014.

[MG15] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature, 2015.

[OTU+18] Kazuki Osawa, Yohei Tsuji, Yuichiro Ueno, Akira Naruse, Rio Yokota, and Satoshi Matsuoka. Large-scale distributed second-order optimization using kronecker-factored approximate curvature for deep convolutional neural networks, 2018.

[PB13] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks, 2013.

[RKK19] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond, 2019.

[RM51] Herbert Robbins and Sutton Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407, 09 1951.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

[Sch02] Nicol Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14:1723–38, 08 2002.

[Zha19] Jiawei Zhang. Gradient descent based optimization algorithms for deep learning models training, 2019.

[ZLHB19] Michael R. Zhang, James Lucas, Geoffrey Hinton, and Jimmy Ba. Lookahead optimizer: k steps forward, 1 step back, 2019.