EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

# Explaining "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 5, 2019

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

# Explaining "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**

Parallelized approaches for optimization algorithms have validation accuracy drawbacks introduced by the increasing effective mini-batch size over multiple processes. To overcome this issue [OTU+18] introduces a parallelized K-FAC algorithm which is able to achieve highly competitive validation accuracies for ResNet-50 on ImageNet even with large mini-batch sizes. In this paper we want to summarize the main approach and give a little more insight.

## 1 Introduction

With recent advances in machine learning the size of training data and the size of deep neural network models is heavily increasing which raises demand for better performing optimization algorithms. Common approaches are either improving the computational steps of the optimization algorithms or introducing parallel computing to speed up convergence. Using a fixed mini-batch size for each process in parallel computing causes the mini-batch size of the overall system to linearly scale with the number of processes. As the mini-batch size increases past a threshold the validation accuracy decreases [SLA+18]. Other works tried to overcome this by varying the learning rate and batch size over epochs. Now, [OTU+18] tries to tackle this large mini-batch problem with taking a more mathematically rigorous approach where they assume that large mini-batches become more statistically stable which introduces advantages for second-order optimization methods.

## 2 Notation

We use the same notation as described in [Zha19] this alternates the notation from [OTU+18] a little to give more insights. 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 $\Theta$, 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 $\eta$ to determine the step sizes taken for the parameters $\boldsymbol{\theta}$ at each update step $\tau$ 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 $\tau$ with $\tau \geq 1$.

## 3 Related Work

Related Work in the realm of Deep Learning Optimizers can broadly be classified in improvements on First- and Second-Order Optimization Algorithms. Here, we want to give a quick overview over these 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* 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 (see Appendix A).

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

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 $\ell$ 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:

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

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 $\ell$. 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.

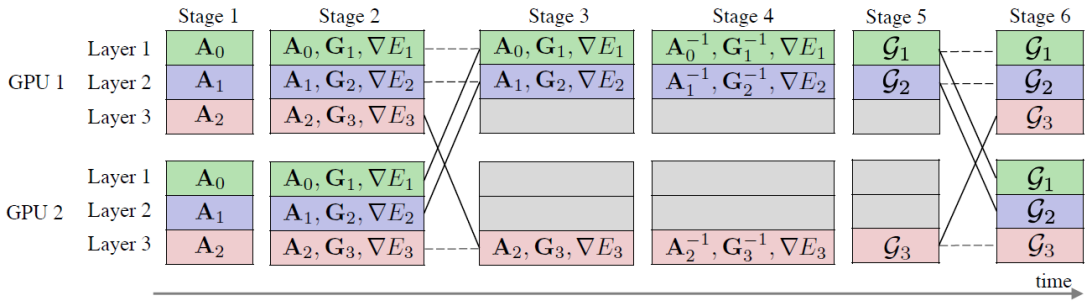


Figure 1: Different stages of distributed K-FAC [OTU+18]

## 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-

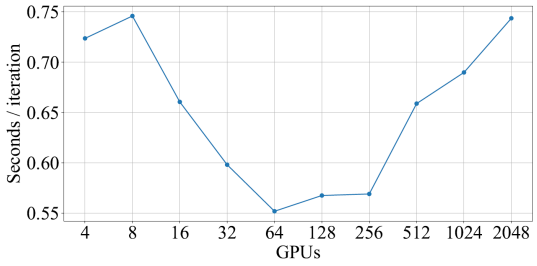


Figure 2: Time per iteration of K-FAC on ResNet-50 using different amount of GPUs [OTU+18]

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 $\leq 32$K [OTU+18].

## 6 Conclusion & Outlook

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

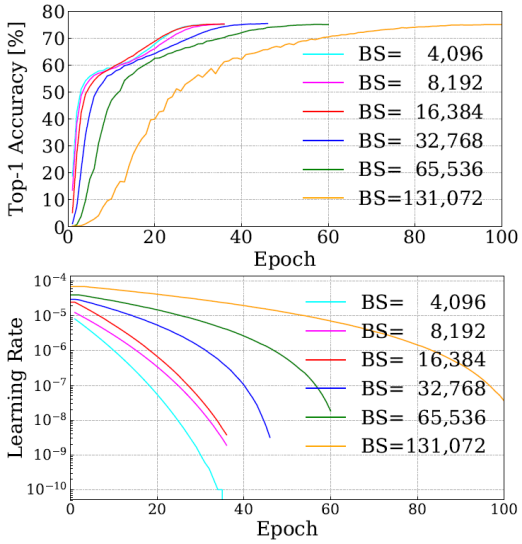


Figure 3: Validation accuracy and learning rate schedules on ResNet-50 [OTU+18]

similar to *SGD* approaches even for large mini-batch sizes. This is a result which is new in its entirety since first-order methods are thought of as having a large edge over second-order alternatives. However, their approach can still be improved by improving the communication complexity as well as approximating the Kronecker-factors without loss of accuracy. For further improvements they mention that each speed-up method for *SGD* which they applied to their approach improved convergence similar to the effect it has on *SGD*. This property opens up the field for further improvements on second-order optimization algorithms by the possibility of further applying already known speed-up techniques for first-order methods.

## 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.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[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.

[Osa18] Kazuki Osawa. Introducing k-fac: A second-order optimization method for large-scale deep learning, 2018.

[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.

[SLA+18] Christopher J. Shallue, Jaehoon Lee, Joseph M. Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. Measuring the effects of data parallelism on neural network training. *CoRR*, abs/1811.03600, 2018.

[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.

# Appendices

## A FIM Approximation and Kronecker-product

This chapter fulfills the purpose to give a more intuitive and visual approach to the FIM approximation as well as the Kronecker-product.

Figure 4 shows the Approximation of the FIM used in *K-FAC*. This shows that the FIM gets block-diagonalized with $\mathbf{F}_\ell$ being the block matrix for the FIM of layer $\ell$. In particular, the FIM of each layer consists of the weights for that specific layer. Each block gets then further approximated using the Kronecker-factorization.

The Kronecker-factorization is visually shown in figure 5. When we assume a white space corresponds to 0, a black space corresponds to 1 and a grey space corresponds to any other value between 0 and 1 the resulting matrix has the shape of multiplying the dimenstions of the two Kronecker-factors. This can basically be thought of as arranging the product of matrix $\mathbf{B}$ with each cell of matrix $\mathbf{A}$ in a new matrix. Because matrix $\mathbf{A}$ has white diagonal elements the resulting matrix also has white block-diagonal elements. Black elements in matrix $\mathbf{A}$ result in matrix $\mathbf{B}$ being put in that position while grey elements yield new alternated entries.

If we now take a look at the approximation process for AlexNet [KSH12] in figure 6 which is a common architecture for image classification we can see the degree of computational improvement this approach offers. AlexNet has $60,000,000$ parameters which yields a Fisher information matrix with shape $\mathbf{F}_{\boldsymbol{\theta}} \in \mathbb{R}^{60,000,000 \times 60,000,000}$ which needs to be inverted in order to make one iteration in updating the parameters when using the *NGD* update rule. If we now take a look at the last block-approximated diagonal block which corresponds to the last AlexNet layer, we observe that it has the shape $\mathbf{F}_\ell \in \mathbb{R}^{4,096,000 \times 4,096,000}$ which can further be approximated by two smaller matrices where $\mathbf{A}_{\ell-1} \in \mathbb{R}^{4,096 \times 4,096}$ is the input to layer $\ell$ and $\mathbf{G}_\ell \in \mathbb{R}^{1,000 \times 1,000}$ is the output of layer $\ell$. The fact that we don't need to invert those enormous matrices but these rather small matrices show how much computational improvement this approach actually offers. Using this approximation for the FIM is the main property which distinguishes *K-FAC* from *NGD* and without it *K-FAC* wouldn't perform much different from regular *NGD*.

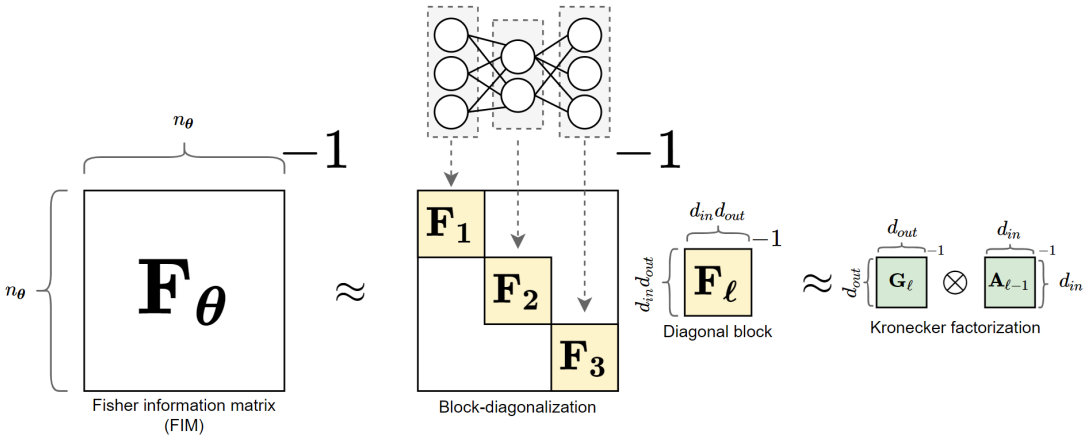


Figure 4: Approximation of the FIM in K-FAC alternated from: [Osa18]

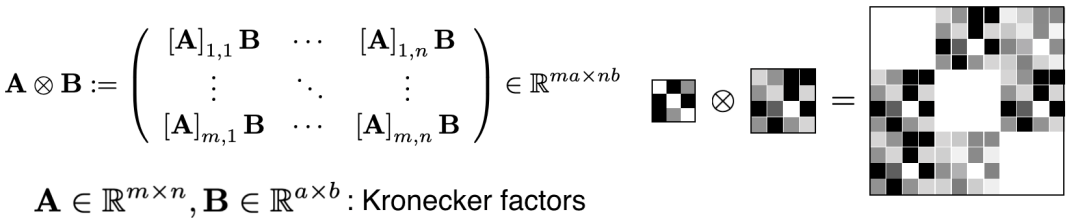


Figure 5: Kronecker Factorization visually explained: [Osa18]

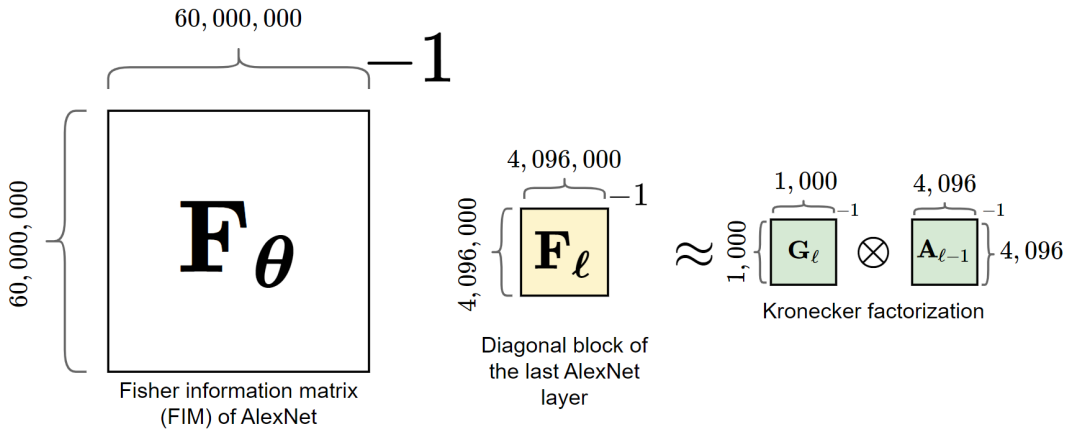


Figure 6: Approximation Example of the FIM for AlexNet in K-FAC alternated from: [Osa18]