Kronecker-Factored Optimal Curvature

Dominik Schnaus[1,2] **Jongseok Lee**[1,3] **Rudolph Triebel**[1,2]
[1]Institute of Robotics and Mechatronics, German Aerospace Center (DLR)
[2]Chair of Computer Vision and Artificial Intelligence, Technical University of Munich (TUM)
[3]High Performance Humanoid Technologies, Karlsruhe Institute of Technology (KIT)
Correspondence to dominik.schnaus@dlr.de

Abstract

The current scalable Bayesian methods for Deep Neural Networks (DNNs) often rely on the Fisher Information Matrix (FIM). For the tractable computations of the FIM, the Kronecker-Factored Approximate Curvature (K-FAC) method is widely adopted, which approximates the true FIM by a layer-wise block-diagonal matrix, and each diagonal block is then Kronecker-factored. In this paper, we propose an alternative formulation to obtain the Kronecker-factored FIM. The key insight is to cast the given FIM computations into an optimization problem over the sums of Kronecker products. In particular, we prove that this formulation is equivalent to the best rank-one approximation problem, where the well-known power iteration method is guaranteed to converge to an optimal rank-one solution - resulting in our novel algorithm: the Kronecker-Factored Optimal Curvature (K-FOC). In a proof-of-concept experiment, we show that the proposed algorithm can achieve more accurate estimates of the true FIM when compared to the K-FAC method.

1 Introduction

Any information a DNN has learned from training data is encoded in its parameters or weights. The concept of Fisher Information [19] attempts to measure such encoded information and the FIM - a fundamental quantity that characterizes the information content of a model - manifests itself across various domains of deep learning research. Some examples include approximate Bayesian inference [20, 22], optimization [21, 4], continual learning [13, 27], information-theoretic understandings of deep models and beyond [12, 18]. In particular to Bayesian methods, many recent techniques such as Laplace Approximation [28, 17] and Variational Inference with natural gradients [22, 32] have shown both scalability and performance by exploiting the concept of Fisher Information [7].

In this line of research, the Kronecker-Factored Approximate Curvature (K-FAC) method [21] is widely used in practice. Approximating the true FIM by ignoring the mutual information between each DNN layer and further assuming a Kronecker-factored FIM, the K-FAC has an advantage in striking the balance between the computational tractability and the fidelity of approximations [1]. However, the K-FAC also makes significant assumptions, i.e, given two squared matrices $\mathbf{A}$ and $\mathbf{B}$, the K-FAC approximates their Kronecker-factored expectations: $\mathbb{E}[\mathbf{A} \otimes \mathbf{B}] \approx \mathbb{E}[\mathbf{A}] \otimes \mathbb{E}[\mathbf{B}]$. This assumption of statistical independence does not hold in general [31], and there have been steady improvements to K-FAC in terms of efficiency and applicability across different communities [23, 31].

In this work, we relax the given assumption in the K-FAC and develop an alternative algorithm to obtain a more accurate Kronecker-factored FIM. To this end, we make the following contributions:

- We propose K-FOC: an optimal and scalable method for the FIM computations of neural networks (both fully-connected and convolutional). For K-FOC, we prove that an optimization formulation over the sums of Kronecker products is equivalent to the best rank-one approximation problem,

Bayesian Deep Learning workshop, NeurIPS 2021.

and propose the power iteration method as a solution. The proof of convergence is also provided in the paper.

- We empirically show that the proposed method can be more accurate than the K-FAC. The analysis of computational complexities is also detailed, with which, we outline two different versions of K-FOC that exhibit a trade-off between approximation quality and memory consumption.

In the context of continual learning for recurrent neural networks, the recent work of Kao et al. [11] pioneers the K-FAC method as an optimization problem over the sums of Kronecker products. However, Kao et al. [11] rely on solvers that are cubic in cost (for example, QR and singular value decomposition) which can be prohibitively too expensive for DNN parameter space, especially for convolutional layers. In this sense, our method presents for the first time to our knowledge, a scalable algorithm that imposes the K-FAC method as an optimization problem. With this, we hope to augment the FIM-based Bayesian methods for DNNs [5, 29, 14, 25].

2 Background and Notation

In this section, we introduce our notations on DNNs and provide background on the K-FAC method.

Preliminaries on Neural Networks. In this work, we adapt the notation of Grosse and Martens [8] as follows. For defining a DNN generally, we consider a layered architecture each consisting of a parameterized function ϕ_l followed by a non-linear activation function σ_l:

$$\mathbf{s}^l = \phi_l(\mathbf{a}^{l-1}) \quad \text{and} \quad \mathbf{a}^l = \sigma_l(\mathbf{s}^l), \tag{1}$$

for layer $l \in [L] := \{1, \dots, L\}$, *pre-activations* $\mathbf{s}^l$ and *activations* $\mathbf{a}^l$. In the following, we consider one specific layer $l \in [L]$. For this, we drop the layer index l and define the corresponding pre-activation as $\mathbf{s} := \mathbf{s}^l$ and the activation from the previous layer as $\mathbf{a} := \mathbf{a}^{l-1}$.

For the parameterized function ϕ, we investigate both the fully-connected and the convolutional layers. Firstly, a fully-connected layer is a matrix multiplication on the previous activation, *i.e.*

$$\phi(\mathbf{a}) = \mathbf{W} \begin{pmatrix} \mathbf{a} \\ 1 \end{pmatrix} =: \mathbf{W}\bar{\mathbf{a}}, \tag{2}$$

where the activations and pre-activations are vectors, $\mathbf{s} \in \mathbb{R}^{d_l}, \mathbf{a} \in \mathbb{R}^{d_{l-1}}$, and the weight matrix contains the bias in the last column $\mathbf{W} \in \mathbb{R}^{d_l \times (d_{l-1}+1)}$. On the other hand, convolutional layers are linear transformations on tensors with weight sharing among the spatial positions. To ease the notation, we use a multi-index notation for the spatial position, *i.e.* $\mathbf{s}_{k,(i,j)} := \mathbf{s}_{k,i,j}$ for the pre-activation $\mathbf{s} \in \mathbb{R}^{c_l \times h_l \times w_l}$. Similarly, the activation is also a three-tensor $\mathbf{a} \in \mathbb{R}^{c_{l-1} \times h_{l-1} \times w_{l-1}}$ and we use $\mathcal{T} = [h_l] \times [w_l]$ as the set of spatial positions. The weight is a four-tensor $\mathbf{W} \in \mathbb{R}^{c_l \times c_{l-1} \times h_l^\Delta \times w_l^\Delta}$ with corresponding spatial positions $\Delta = [h_l^\Delta] \times [w_l^\Delta]$. Then the convolutional layer is defined as:

$$\phi(\mathbf{a})_{k,\mathbf{t}} = \mathbf{b}_k + \sum_{k'=1}^{c_{l-1}} \sum_{\delta \in \Delta} \mathbf{W}_{k,k',\delta} \mathbf{a}_{k',\zeta(\mathbf{t},\delta)} \tag{3}$$

with the index function $\zeta(\mathbf{t}, \delta) = (\mathbf{t} - \mathbb{1}) \odot \mathbf{r} - \mathbf{p} + \delta$ using stride $\mathbf{r} \in \mathbb{N}^2$, padding $\mathbf{p} \in \mathbb{N}^2$ and $\odot$ to denote the Hadamard product. We define $\mathbf{a}_{k',\mathbf{t}'} := 0$ if $\mathbf{t}' \notin [h_{l-1}] \times [w_{l-1}]$. By transforming the activations into a $|\mathcal{T}| \times (c_{l-1}|\Delta| + 1)$-sized matrix $\hat{\mathbf{a}}$ where row $\mathbf{t}$ is defined by the row-vector

$$\hat{\mathbf{a}}_\mathbf{t} = \mathrm{vec}((\mathbf{a}_{k',\zeta(\mathbf{t},\delta)})_{k' \in [c_{l-1}], \delta \in \Delta}, 1)^T \tag{4}$$

and the weight into an $c_l \times (c_{l-1}|\Delta| + 1)$-sized matrix $\hat{\mathbf{W}}$ by the vectorization of the last three dimensions and appending of the bias vector to the last row, the layer is equivalent to the matrix product:

$$\hat{\mathbf{s}} = \hat{\mathbf{a}}(\hat{\mathbf{W}})^T. \tag{5}$$

In contrast to Martens and Grosse [21], we use row vectorization (consistent with PyTorch [24]).

Kronecker-Factored Approximate Curvature. Having defined a DNN, we now describe the K-FAC [8], which is a popular method to compute the FIM of neural networks. Consider a conditional probability density $p(\mathbf{y}|\mathbf{x}, \theta)$ given by a neural network with the weight-vector $\theta = \operatorname{vec}\left((\mathbf{W}^l)_{l\in[L]}\right)$, where $\mathbf{W}^l$ is the weight matrix corresponding to layer l. Then the FIM is defined as

$$\mathbf{F} = \mathbb{E}_{\mathbf{x}\sim Q_x}\mathbb{E}_{\mathbf{y}\sim p(\cdot|\mathbf{x},\theta)}\left[\frac{d\ln p(\mathbf{y}|\mathbf{x},\theta)}{d\theta}\frac{d\ln p(\mathbf{y}|\mathbf{x},\theta)}{d\theta}^T\right], \tag{6}$$

with the unknown data distribution Q_x [21]. This is usually approximated with the empirical distribution over the training data $\hat{Q}_x$ and it is computed over batches $B_p = ((\mathbf{x}_k^p, \mathbf{y}_k^p))_{k=1}^{K_p}$ with $\mathbf{x} \sim \hat{Q}_x$ and $\mathbf{y} \sim p(\cdot|\mathbf{x}, \theta)$ for $p \in [P]$. For the sake of an easier notation, we write $\mathbb{E}$ instead of $\mathbb{E}_{\mathbf{x}\sim Q_x}\mathbb{E}_{\mathbf{y}\sim p(\cdot|\mathbf{x},\theta)}$ and $\mathcal{D}\cdot = \frac{d\ln p(\mathbf{y}|\mathbf{x},\theta)}{d\cdot}$ for the derivative of the log-likelihood in the following.

The full FIM and even a block-diagonal approximation without correlations between different layers are usually not feasible to store or compute for modern neural networks [21]. Common approximations of the block-diagonal form are by a diagonal or a Kronecker-factored matrix. The second approximation comes from the fact that for a single sample, the FIM is the sum of Kronecker-factored matrices. For fully-connected layers, the derivative after the weight matrix can be computed as $\mathcal{D}\mathbf{W} = \mathcal{D}\mathbf{s}(\bar{\mathbf{a}})^T$. With this, the block of the FIM corresponding to layer l is given by

$$\mathbf{F}_l = \mathbb{E}[\mathcal{D}\mathbf{s}(\mathcal{D}\mathbf{s})^T \otimes \bar{\mathbf{a}}(\bar{\mathbf{a}})^T]. \tag{7}$$

As convolutional layers share the weight tensor among the spatial positions, the derivative is a sum of outer products: $\mathcal{D}\hat{\mathbf{W}}_{i,k} = \sum_{\mathbf{t}\in\mathcal{T}} \mathcal{D}\mathbf{s}_{\mathbf{t},i}\hat{\mathbf{a}}_{\mathbf{t},k}$. Therefore, the FIM block for layer l can be formulated as

$$\mathbf{F}_l = \mathbb{E}[\sum_{\mathbf{t}\in\mathcal{T}}\sum_{\mathbf{t}'\in\mathcal{T}} \mathcal{D}\mathbf{s}_{\mathbf{t}}(\mathcal{D}\mathbf{s})_{\mathbf{t}'}^T \otimes \hat{\mathbf{a}}_{\mathbf{t}}(\hat{\mathbf{a}}_{\mathbf{t}'})^T]. \tag{8}$$

The K-FAC [21, 8] approximates this FIM of a fully-connected layer and a convolutional layer by

$$\mathbf{F}_l \approx \mathbb{E}[\mathcal{D}\mathbf{s}(\mathcal{D}\mathbf{s})^T] \otimes \mathbb{E}[\bar{\mathbf{a}}(\bar{\mathbf{a}})^T] \quad \text{and} \quad \mathbf{F}_l \approx \mathbb{E}[(\mathcal{D}\mathbf{s})^T\mathcal{D}\mathbf{s}] \otimes \frac{1}{|\mathcal{T}|}\mathbb{E}[(\hat{\mathbf{a}})^T\hat{\mathbf{a}}], \tag{9}$$

respectively. This herein estimation of the expectation of the Kronecker product by the Kronecker product of the expectation is used to leverage the benefits of Kronecker factorization. For example, the Kronecker factorization enables the storage of two smaller matrices rather than a prohibitively large matrix [21]. However, these factorizations assume that the activations and the corresponding pre-activations of DNNs are statistically independent, which is usually not met in practice. Furthermore, additional assumptions like the independence of the first and second order statistics of the spatial positions are used for convolutional layers, which might impair the approximation quality of the method.

In the following, we use the superscript k to denote the correspondence of the (pre-) activation and their gradients to the sample $(\mathbf{x}_k^p, \mathbf{y}_k^p) \in B_p$ for $p \in [P]$. Moreover, we assume the FIM to be block-diagonal.

3 The Proposed Method

Now, we present K-FOC, which is the proposed method for computing the FIM of DNNs. In this section, we first formulate the FIM computations as an optimization problem over the sums of Kronecker products, followed by its connection to the best rank-one problem and the power iteration method as its solution (section 3.1). Then, we discuss the derived Algorithm (section 3.2)

3.1 Approximation of Sums of Kronecker Products

The FIM approximated on a batch B_p is a sum of Kronecker-factored matrices for both fully-connected and convolutional layers, namely $\sum_{k=1}^{K_p} \mathcal{D}\mathbf{s}^k (\mathcal{D}\mathbf{s}^k)^T \otimes \frac{1}{K_p} \bar{\mathbf{a}}^k (\bar{\mathbf{a}}^k)^T$ and $\sum_{k=1}^{K_p} \sum_{\mathbf{t} \in \mathcal{T}} \sum_{\mathbf{t}' \in \mathcal{T}} \mathcal{D}\mathbf{s}^k{}_{\mathbf{t}} (\mathcal{D}\mathbf{s}^k)^T_{\mathbf{t}'} \otimes \frac{1}{K_p} \hat{\mathbf{a}}^k_{\mathbf{t}} (\hat{\mathbf{a}}^k_{\mathbf{t}'})^T$. The resulting approximation of the FIM is in general not Kronecker-factored. Nonetheless, a Kronecker-factored approximation brings many advantages like low memory consumption, easy computations of the inverse, and a fast sampling from the corresponding matrix variate distribution [21]. Hence, we aim to find Kronecker-factors that better approximate the FIM. This boils down to approximating a sum of Kronecker-factored matrices (similar to Kao et al. [11]) by one Kronecker-factored matrix, *i.e.* for $M, N, K \in \mathbb{N}$, $\mathbf{L}^k \in \mathbb{R}^{M \times M}$ and $\mathbf{R}^k \in \mathbb{R}^{N \times N}$ for $k \in [K]$ we aim to find:

$$\hat{\mathbf{L}}, \hat{\mathbf{R}} \in \underset{\mathbf{L} \in \mathbb{R}^{M \times M}, \mathbf{R} \in \mathbb{R}^{N \times N}}{\arg\min} \| \sum_{k=1}^{K} \mathbf{L}^k \otimes \mathbf{R}^k - \mathbf{L} \otimes \mathbf{R} \|_F. \tag{10}$$

In general, a solution of equation 10 is not unique. This can be seen by scaling the left factor $\hat{\mathbf{L}}$ by $\alpha \neq 0$ and $\hat{\mathbf{R}}$ by $\frac{1}{\alpha}$. Thus, in the following, we additionally assume that $\hat{\mathbf{L}}$ is normalized, $\|\hat{\mathbf{L}}\|_F = 1$. The choice that $\hat{\mathbf{L}}$ is normalized is arbitrary and one could achieve all results presented in this work when instead $\hat{\mathbf{R}}$ would be normalized. As it turns out, the problem is equivalent to a rank-one approximation as proven in the lemma below.

Lemma 3.1. *Let $M, N, K \in \mathbb{N}$, $\mathbf{L}^k \in \mathbb{R}^{M \times M}$ and $\mathbf{R}^k \in \mathbb{R}^{N \times N}$ for $k \in [K]$. Then*

$$\| \sum_{k=1}^{K} \mathbf{L}^k \otimes \mathbf{R}^k - \mathbf{L} \otimes \mathbf{R} \|_F = \| \sum_{k=1}^{K} \operatorname{vec}(\mathbf{L}^k) \operatorname{vec}(\mathbf{R}^k)^T - \operatorname{vec}(\mathbf{L}) \operatorname{vec}(\mathbf{R})^T \|_F. \tag{11}$$

Proof. See Appendix A.1. □

Therefore, one can use the power method to solve this problem. Nonetheless, each step of the plain power method consists of a matrix multiplication with an $M^2 \times N^2$-sized matrix, which is in general not feasible for layers of modern neural networks. The complexity can be reduced by utilizing that the matrix is a sum of few rank-one matrices. This is shown in Algorithm 1. With this, the convergence properties of the power method are achieved with a computational complexity of $\mathcal{O}\left(n^{max} K (N^2 + M^2)\right)$. Also, only $\mathcal{O}\left(K(N^2 + M^2)\right)$ memory is needed. Here, we assume that a matrix multiplication $\mathbf{AB}$ for $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times k}$ has the complexity $\mathcal{O}(mnk)$ while a Hadamard product $\mathbf{A} \odot \mathbf{C}$ for $\mathbf{C} \in \mathbb{R}^{m \times n}$ can be computed in $\mathcal{O}(mn)$. The correctness of Algorithm 1 is shown in Lemma 3.2.

Lemma 3.2. *Let $\mathbf{A} = \sum_{k=1}^{K} \operatorname{vec}(\mathbf{L}^k) \operatorname{vec}(\mathbf{R}^k)^T$ and $\mathbf{A} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$ be its singular value decomposition with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ and $\mathbf{u}_i^T \mathbf{u}_j = \mathbf{v}_i^T \mathbf{v}_j = \mathbb{1}[i = j]$. Then there is a solution of equation 10 with*

$$\operatorname{vec}(\hat{\mathbf{L}}) = \mathbf{u}_1, \operatorname{vec}(\hat{\mathbf{R}}) = \sigma_1 \mathbf{v}_1. \tag{12}$$

If $\sigma_1 > \sigma_2$, the solution is unique up to changing the sign of both factors and Algorithm 1 converges almost surely to this solution.

Proof. See Appendix A.2. □

Remark. Lemma 3.2 shows that algorithm 1 is equivalent to the power method for the corresponding best rank-one problem. Therefore, convergence rates and error bounds are directly inherited from the power method, see *e.g.* [26].

Algorithm 1: Power method for sums of Kronecker products

input :
- $(\mathbf{L}^k)_{k\in[K]}$ left matrices
- $(\mathbf{R}^k)_{k\in[K]}$ right matrices
- $n^{max} = 100$ maximal number of steps
- $\delta = 10^{-5}$ stopping precision

output : $\hat{\mathbf{L}}, \hat{\mathbf{R}} \in \arg\min_{\mathbf{L},\mathbf{R}} \| \sum_{k=1}^K \mathbf{L}^k \otimes \mathbf{R}^k - \mathbf{L} \otimes \mathbf{R}\|_F$

1 **function** `kronecker_power_method`$((\mathbf{L}^k)_{k\in[K]}, (\mathbf{R}^k)_{k\in[K]})$
2 $\quad \mathrm{vec}(\bar{\mathbf{L}}^{(0)}) \leftarrow \mathcal{N}(\mathbf{0}, \mathbf{I})$ ▷ standard normal initialization of $\bar{\mathbf{L}}^{(0)}$
3 $\quad \mathbf{L}^{(0)} \leftarrow \frac{\bar{\mathbf{L}}^{(0)}}{\|\bar{\mathbf{L}}^{(0)}\|_F}$ ▷ normalize $\bar{\mathbf{L}}^{(0)}$
4 $\quad$ **for** $n \in n^{max}$ **do**
5 $\quad\quad \bar{\mathbf{R}}^{(n)} \leftarrow \sum_{k=1}^K \langle \mathbf{L}^k, \mathbf{L}^{(n-1)} \rangle_F \mathbf{R}^k$ ▷ first power iteration step
6 $\quad\quad \mathbf{R}^{(n)} \leftarrow \frac{\bar{\mathbf{R}}^{(n)}}{\|\bar{\mathbf{R}}^{(n)}\|_F}$ ▷ normalize $\bar{\mathbf{R}}^{(n)}$
7 $\quad\quad \bar{\mathbf{L}}^{(n)} \leftarrow \sum_{k=1}^K \langle \mathbf{R}^k, \mathbf{R}^{(n)} \rangle_F \mathbf{L}^k$ ▷ second power iteration step
8 $\quad\quad \mathbf{L}^{(n)} \leftarrow \frac{\bar{\mathbf{L}}^{(n)}}{\|\bar{\mathbf{L}}^{(n)}\|_F}$ ▷ normalize $\bar{\mathbf{L}}^{(n)}$
9 $\quad\quad$ **if** $\|\mathbf{L}^{(n)} - \mathbf{L}^{(n-1)}\|_F < \delta$ **then**
10 $\quad\quad\quad$ **break**
11 $\quad \hat{\mathbf{L}} \leftarrow \mathbf{L}^{(n)}$
12 $\quad \hat{\mathbf{R}} \leftarrow \sum_{k=1}^K \langle \mathbf{L}^k, \mathbf{L}^{(n)} \rangle_F \mathbf{R}^k$ ▷ first power iteration step

3.2 Kronecker-Factored Optimal Curvature

Now, we present K-FOC for both fully-connected and convolutional layers. For fully-connected layers, one can apply Algorithm 1 with $\mathbf{L}^k = \mathcal{D}\mathbf{s}^k(\mathcal{D}\mathbf{s}^k)^T$ and $\mathbf{R}^k = \bar{\mathbf{a}}^k(\bar{\mathbf{a}}^k)^T$ for $k \in [K_p]$ for batch $p \in [P]$ to obtain the optimal Kronecker factors for this batch. The resulting computational complexity is $\mathcal{O}\left(n^{max} K_p(d_{l-1}^2 + d_l^2)\right)$ with a memory consumption of $\mathcal{O}\left(K_p(d_{l-1}^2 + d_l^2)\right)$. Compared to fully-connected layers, convolutional layers involve a sum over $|\mathcal{T}|^2 = h_l^2 w_l^2$ Kronecker factors in equation 8. Hence, the complexity of directly applying Algorithm 1 with $\mathbf{L}^{k,\mathbf{t},\mathbf{t}'} = \mathcal{D}\mathbf{s}^k{}_{\mathbf{t}}(\mathcal{D}\mathbf{s}^k)^T_{\mathbf{t}'}$ and $\mathbf{R}^{k,\mathbf{t},\mathbf{t}'} = \hat{\mathbf{a}}^k_{\mathbf{t}}(\hat{\mathbf{a}}^k_{\mathbf{t}'})^T$ would be $\mathcal{O}\left(n^{max} K_p |\mathcal{T}|^2((c_{l-1}|\Delta| + 1)^2 + c_l^2)\right)$ and would use $\mathcal{O}\left(K_p|\mathcal{T}|^2((c_{l-1}|\Delta| + 1)^2 + c_l^2)\right)$ memory, which is usually not feasible especially for high-resolution images. To reduce the complexity, one can incorporate that each Kronecker factor is an outer product of two vectors and that the summation is over all combinations of $\mathbf{t} \in \mathcal{T}$ and $\mathbf{t}' \in \mathcal{T}$. With this, line 7 of Algorithm 1 can be computed as

$$\sum_{k=1}^{K_p} \sum_{\mathbf{t},\mathbf{t}' \in \mathcal{T}} \langle \mathcal{D}\mathbf{s}^k{}_{\mathbf{t}}(\mathcal{D}\mathbf{s}^k)^T_{\mathbf{t}'}, \mathbf{L}^{(n-1)} \rangle_F \hat{\mathbf{a}}^k_{\mathbf{t}}(\hat{\mathbf{a}}^k_{\mathbf{t}'})^T = \sum_{k=1}^{K_p} (\mathcal{D}\mathbf{s}^k)^T \hat{\mathbf{a}}^k \mathbf{L}^{(n-1)} (\hat{\mathbf{a}}^k)^T \mathcal{D}\mathbf{s}^k. \tag{13}$$

Line 5 can be computed similarly. Altogether, one can pre-compute the matrix

$$\mathbf{X}^k \leftarrow (\mathcal{D}\mathbf{s}^k)^T \hat{\mathbf{a}}^k \text{ for } k \in [K] \tag{14}$$

in advance and in each iteration, lines 5 and 7 can be replaced by

$$\bar{\mathbf{R}}^{(n)} \leftarrow \sum_{k=1}^{K_p} \mathbf{X}^k \mathbf{L}^{(n-1)} (\mathbf{X}^k)^T \quad \text{and} \quad \bar{\mathbf{L}}^{(n)} \leftarrow \sum_{k=1}^{K_p} (\mathbf{X}^k)^T \mathbf{R}^{(n)} \mathbf{X}^k, \tag{15}$$

respectively. The full adaption of Algorithm 1 for convolutions is shown in Algorithm 2.

Algorithm 2: Power method for convolutions

input : $(\mathcal{D}\mathbf{s}^k)_{k\in[K]}$ pre-activation derivatives; $(\hat{\mathbf{a}}^k)_{k\in[K]}$ activations; $n^{max}=100$ maximal number of steps; $\delta=10^{-5}$ stopping precision

output : $\hat{\mathbf{L}},\hat{\mathbf{R}} \in \arg\min_{\mathbf{L},\mathbf{R}} \|\frac{1}{k}\sum_{k=1}^{K}\sum_{\mathbf{t}\in\mathcal{T}}\sum_{\mathbf{t}'\in\mathcal{T}}\mathcal{D}\mathbf{s}^k{}_{\mathbf{t}}(\mathcal{D}\mathbf{s}^k)^T_{\mathbf{t}'}\otimes\hat{\mathbf{a}}^k_{\mathbf{t}}(\hat{\mathbf{a}}^k_{\mathbf{t}'})^T-\mathbf{L}\otimes\mathbf{R}\|_F$

function `kronecker_power_method_convolutions`$((\mathcal{D}\mathbf{s}^k)_{k\in[K]},\hat{\mathbf{a}}^k)_{k\in[K]})$

- $\mathrm{vec}(\bar{\mathbf{L}}^{(0)}) \leftarrow \mathcal{N}(\mathbf{0},\mathbf{I})$ ▷ standard normal initialization of $\bar{\mathbf{L}}^{(0)}$
- $\mathbf{L}^{(0)} \leftarrow \frac{\bar{\mathbf{L}}^{(0)}}{\|\bar{\mathbf{L}}^{(0)}\|_F}$ ▷ normalize $\bar{\mathbf{L}}^{(0)}$
- $\mathbf{X}^k \leftarrow (\mathcal{D}\mathbf{s}^k)^T\hat{\mathbf{a}}^k$ for $k\in[K]$ ▷ pre-compute $\mathbf{X}^k$
- **for** $n \in n^{max}$ **do**
  - $\bar{\mathbf{R}}^{(n)} \leftarrow \sum_{k=1}^{K}\mathbf{X}^k\mathbf{L}^{(n-1)}(\mathbf{X}^k)^T$ ▷ first power iteration step
  - $\mathbf{R}^{(n)} \leftarrow \frac{\bar{\mathbf{R}}^{(n)}}{\|\bar{\mathbf{R}}^{(n)}\|_F}$ ▷ normalize $\bar{\mathbf{R}}^{(n)}$
  - $\bar{\mathbf{L}}^{(n)} \leftarrow \sum_{k=1}^{K}(\mathbf{X}^k)^T\mathbf{R}^{(n)}\mathbf{X}^k$ ▷ second power iteration step
  - $\mathbf{L}^{(n)} \leftarrow \frac{\bar{\mathbf{L}}^{(n)}}{\|\bar{\mathbf{L}}^{(n)}\|_F}$ ▷ normalize $\bar{\mathbf{L}}^{(n)}$
  - **if** $\|\mathbf{L}^{(n)}-\mathbf{L}^{(n-1)}\|_F<\delta$ **then**
    - **break**
- $\hat{\mathbf{L}} \leftarrow \mathbf{L}^{(n)}$
- $\hat{\mathbf{R}} \leftarrow \frac{1}{k}\sum_{k=1}^{K_p}\mathbf{X}^k\mathbf{L}^{(n)}(\mathbf{X}^k)^T$ ▷ first power iteration step

The computational complexity hence reduces to $\mathcal{O}(K_p(c_{l-1}|\Delta|+1)|\mathcal{T}|c_l + n^{max}K_p(c_{l-1}|\Delta|+1)c_l((c_{l-1}|\Delta|+1)+c_l))$. Moreover, only $\mathcal{O}\left((c_{l-1}|\Delta|+1)^2+c_l^2+K_p|\mathcal{T}|((c_{l-1}|\Delta|+1)+c_l)\right)$ memory is needed. This is of the same order as K-FAC if $|\mathcal{T}| > n^{max}\max\{c_{l-1}|\Delta|+1, c_l\}$. Furthermore, we note that the adaption of the power method for convolutions can also be applied to fully-connected layers by viewing them as convolutional layers with $\mathcal{T}=\{(1,1)\}$. While the complexity usually increases to $\mathcal{O}\left(K_p d_{l-1}d_l + n^{max}K_p d_{l-1}d_l(d_{l-1}+d_l)\right)$ with this approach, the memory needed is reduced for large batch sizes to $\mathcal{O}\left(d_{l-1}^2+d_l^2+K_p(d_{l-1}+d_l)\right)$.

So far, with this method, optimal factors for a batch of samples can be found with a practical complexity and memory consumption. However, the optimal factors $(\mathbf{L}^p,\mathbf{R}^p)$ for the batches $B_p, p\in[P]$ additionally need to be aggregated to obtain the FIM. Similar to K-FAC, independence between different batches can be assumed and the FIM could be estimated as $\mathbf{F}_l \approx (\sum_{p=1}^{P}\mathbf{L}^p)\otimes(\frac{1}{P}\sum_{p=1}^{P}\mathbf{R}^p)$, denoted as K-FOC_approx in the following. Empirically, this does not hold in general and the approximation quality worsens for small batch sizes. Another possibility is to collect all factors in one step. In a second step the optimal factors $\hat{\mathbf{L}},\hat{\mathbf{R}} \in \arg\min_{\mathbf{L},\mathbf{R}}\|\sum_{p=1}^{P}\mathbf{L}^p\otimes\frac{1}{P}\mathbf{R}^p-\mathbf{L}\otimes\mathbf{R}\|_F$ can be computed using the power method and the estimate of the FIM is given by $\mathbf{F}_l\approx\hat{\mathbf{L}}\otimes\hat{\mathbf{R}}$. A drawback of this method is that the factors of all batches need to be kept in memory.

A trade-off between approximation quality and memory consumption is approximating the running average $\hat{\mathbf{L}},\hat{\mathbf{R}} \in \arg\min_{\mathbf{L},\mathbf{R}}\|\hat{\mathbf{L}}^{p-1}\otimes\frac{p-1}{p}\hat{\mathbf{R}}^{p-1}+\mathbf{L}^p\otimes\frac{1}{p}\mathbf{R}^p-\mathbf{L}\otimes\mathbf{R}\|_F$ with $\hat{\mathbf{L}}^1=\mathbf{L}^1,\hat{\mathbf{R}}^1=\mathbf{R}^1$ for each batch. The FIM can then be approximated as $\mathbf{F}_l\approx\hat{\mathbf{L}}^P\otimes\hat{\mathbf{R}}^P$ and only the running average and the current matrices need to be kept in memory resulting in the same memory consumption as K-FOC_approx with a small additional computational overhead. We denote this variant of K-FOC as K-FOC_running. Next, we evaluate the proposed approaches empirically.

4 Experiments and Results

We now present our experiments, which are designed to check the approximation quality of the K-FOC when compared to the K-FAC. To compute the FIM, we draw one sample from the model's predictive distribution for each data point and use the same samples for the ground truth block

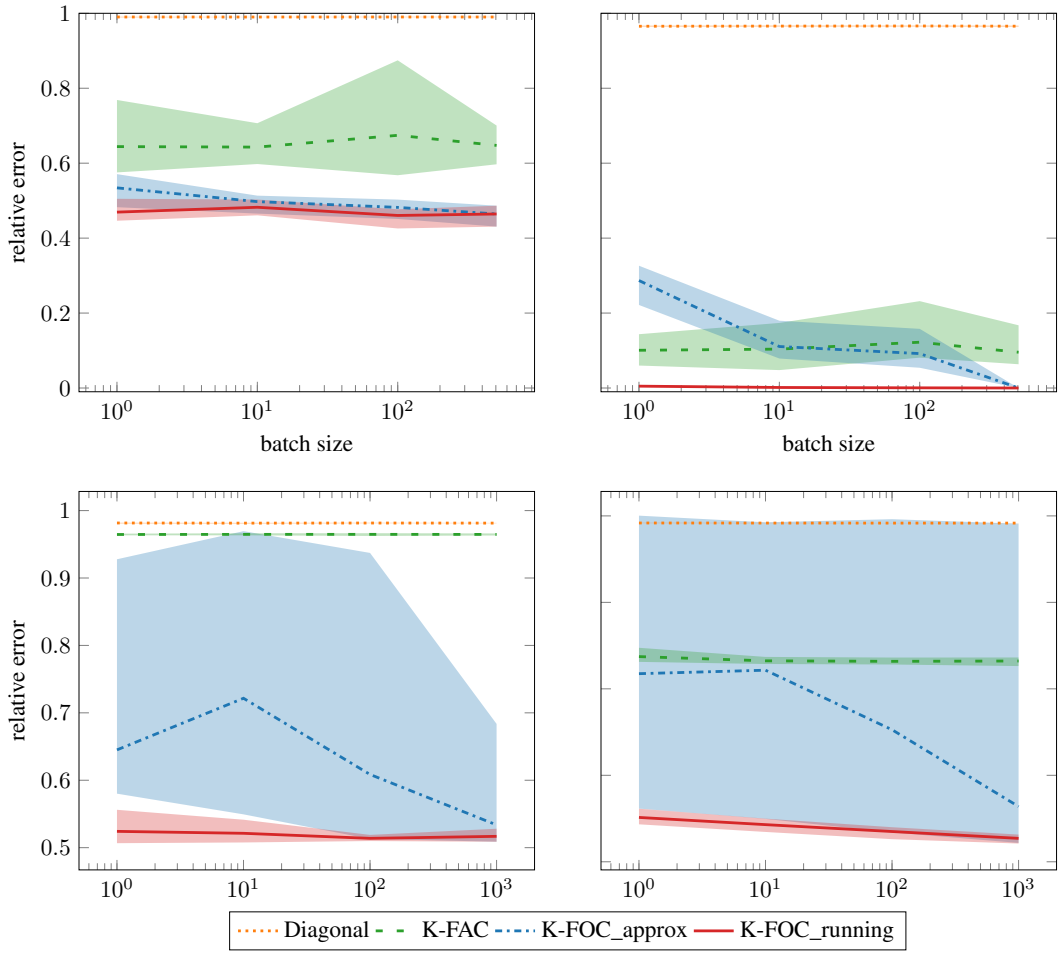

Figure 1: Comparison of the relative Frobenius error for diagonal approximations (orange), K-FAC (green) and K-FOC (blue and red) for different batch sizes for the first (left) and second (right) layer of the relative network architecture. The two fully-connected layers trained on the Boston Housing dataset are in the first row and the convolutional layers learned on MNIST are in the second row. The light area shows the minimal and maximal values among ten independent runs while the thick lines are the mean errors. For K-FOC, both aggregation strategies as described in section 3.2 are depicted.

diagonal FIM as for all approximations. To this end, we compare the relative Frobenius error with respect to the ground-truth FIM. All experiments are implemented in PyTorch [24] on top of the curvature library [17, 10, 30] and are run on an Nvidia RTX 3060 GPU.

4.1 Fully-Connected Layers

Experiment Set-up. To validate the method on fully-connected layers, we train a fully-connected neural network with one hidden layer on the three UCI datasets Boston Housing, Concrete Compression Strength and Energy Efficiency, following Hernández-Lobato and Adams [9] and Lee et al. [17]. Similar to their work, the hidden layer also consists of 50 units. For all three datasets, the FIM is computed over the full dataset.

Results. The first row of Figure 1 shows the results of the fully-connected network for the Boston Housing dataset. The results on the other UCI datasets show a similar qualitative behavior and are shown in Appendix A.3. For each method, the mean error is depicted with a thick line in addition to the minimal and maximal value among the ten runs in the light area. One can observe that K-FAC is strictly better in terms of the Frobenius error compared to the diagonal approximation. The

K-FOC approximation with the independence assumptions (K-FOC_approx) usually improves with an increasing batch size and surpasses K-FAC starting at around a batch size of 10. The K-FOC approximation that estimates the running average (K-FOC_running) performs best even though the difference between the two K-FOC methods diminishes for large batch sizes. The runtime for this experiment is analyzed in Appendix A.4.

4.2 Convolutional Layers

Experiment Set-up. For convolutional layers, the method is validated on the first two convolutional layers of a LeNet-5 [15] architecture on the MNIST dataset [16]. The FIM and its approximations are computed over the training split of the dataset.

Results. In the second row of Figure 1, one can see the relative error on the first two convolutional layers. Similar to the fully-connected experiment, one can observe that K-FOC_approx is usually closer to the block-diagonal matrix than the K-FAC and the diagonal approximation. Nonetheless, the approximation quality highly depends on the specific run and the variance between the runs is in general high. K-FOC_running consistently outperforms all other methods by a large margin and has a lower variance than K-FOC_approx. For both K-FOC methods, the relative error decreases with an increasing batch size, which is not the case for K-FAC or the diagonal method.

5 Conclusion

In this paper, we propose the Kronecker-Factored Optimal Curvature (K-FOC) in order to compute the Fisher Information Matrix (FIM) more accurately, while preserving the Kronecker structure. Our algorithm relies on an optimization formulation over the sums of Kronecker products. Concretely, we show that this formulation is equivalent to the best rank-one approximation problem and prove that the power iteration method can converge to an optimal rank-one solution. While the analysis of computational complexities is detailed in the paper, we perform experiments to demonstrate that the K-FOC can yield higher accuracy than the widely adopted K-FAC method.

Acknowledgments and Disclosure of Funding

We would like to acknowledge the support of Helmholtz Association, the project ARCHES (contract number ZT-0033), the Initiative and Networking Fund (INF) under the Helmholtz AI platform grant agreement (ID ZT-I-PF-5-1) and lastly, the EU Horizon 2020 project RIMA under the grant agreement number 824990.

References

[1] J. Ba, R. B. Grosse, and J. Martens. Distributed second-order optimization using kronecker-factored approximations. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[2] D. Bindel. Power iteration, 2016. URL `https://www.cs.cornell.edu/~bindel/class/cs6210-f16/lec/2016-10-17.pdf`.

[3] A. Blum, J. Hopcroft, and R. Kannan. *Best-Fit Subspaces and Singular Value Decomposition (SVD)*, page 29–61. Cambridge University Press, 2020. doi: 10.1017/9781108755528.003.

[4] F. Dangel, F. Kunstner, and P. Hennig. Backpack: Packing more into backprop. In *International Conference on Learning Representations*, 2020.

[5] E. Daxberger, A. Kristiadi, A. Immer, R. Eschenhagen, M. Bauer, and P. Hennig. Laplace redux–effortless bayesian deep learning. *arXiv preprint arXiv:2106.14806*, 2021.

[6] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1 (3):211–218, 1936. ISSN 0033-3123. doi: 10.1007/BF02288367. URL `https://link.springer.com/article/10.1007%2FBF02288367`.

[7] J. Gawlikowski, C. R. N. Tassi, M. Ali, J. Lee, M. Humt, J. Feng, A. Kruspe, R. Triebel, P. Jung, R. Roscher, et al. A survey of uncertainty in deep neural networks. *arXiv preprint arXiv:2107.03342*, 2021.

[8] R. Grosse and J. Martens. A kronecker-factored approximate fisher matrix for convolution layers. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 573–582, New York, New York, USA, 2016. PMLR. URL `http://proceedings.mlr.press/v48/grosse16.html`.

[9] J. M. Hernández-Lobato and R. P. Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 1861–1869. JMLR.org, 2015.

[10] M. Humt, J. Lee, and R. Triebel. Bayesian optimization meets laplace approximation for robotic introspection. *arXiv preprint arXiv:2010.16141*, 2020.

[11] T.-C. Kao, K. T. Jensen, A. Bernacchia, and G. Hennequin. Natural continual learning: success is a journey, not (just) a destination. *arXiv preprint arXiv:2106.08085*, 2021.

[12] R. Karakida, S. Akaho, and S.-i. Amari. Universal statistics of fisher information in deep neural networks: Mean field approach. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1032–1041. PMLR, 2019.

[13] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[14] A. Kristiadi, M. Hein, and P. Hennig. Learnable uncertainty under laplace approximations. *arXiv preprint arXiv:2010.02720*, 2020.

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. doi: 10.1162/neco.1989.1.4.541.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. ISSN 00189219. doi: 10.1109/5.726791.

[17] J. Lee, M. Humt, J. Feng, and R. Triebel. Estimating model uncertainty of neural networks in sparse information form. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5702–5713. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/lee20b.html`.

[18] J. Lee, J. Feng, M. Humt, M. G. Müller, and R. Triebel. Trust your robots! predictive uncertainty estimation of neural networks with sparse gaussian processes. In *5th Annual Conference on Robot Learning*, 2021.

[19] A. Ly, M. Marsman, J. Verhagen, R. P. Grasman, and E.-J. Wagenmakers. A tutorial on fisher information. *Journal of Mathematical Psychology*, 80:40–55, 2017.

[20] D. J. MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3): 448–472, 1992.

[21] J. Martens and R. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2408–2417, Lille, France, 2015. PMLR. URL `http://proceedings.mlr.press/v37/martens15.html`.

[22] K. Osawa, S. Swaroop, M. E. E. Khan, A. Jain, R. Eschenhagen, R. E. Turner, and R. Yokota. Practical deep learning with bayesian principles. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[23] 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. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12359–12367, 2019.

[24] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[25] C. S. Perone, R. P. Silveira, and T. Paula. L2m: Practical posterior laplace approximation with optimization-driven second moment estimation. *arXiv preprint arXiv:2107.04695*, 2021.

[26] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical Mathematics*, volume 37. Springer, 01 2007. ISBN 978-1-4757-7394-1. doi: 10.1007/b98885.

[27] H. Ritter, A. Botev, and D. Barber. Online structured laplace approximations for overcoming catastrophic forgetting. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[28] H. Ritter, A. Botev, and D. Barber. A scalable laplace approximation for neural networks. In *6th International Conference on Learning Representations, ICLR 2018-Conference Track Proceedings*, volume 6. International Conference on Representation Learning, 2018.

[29] A. Sharma, N. Azizan, and M. Pavone. Sketching curvature for efficient out-of-distribution detection for deep neural networks. *arXiv preprint arXiv:2102.12567*, 2021.

[30] K. Shinde, J. Lee, M. Humt, A. Sezgin, and R. Triebel. Learning multiplicative interactions with bayesian neural networks for visual-inertial odometry. *arXiv preprint arXiv:2007.07630*, 2020.

[31] Z. Tang, F. Jiang, M. Gong, H. Li, Y. Wu, F. Yu, Z. Wang, and M. Wang. Skfac: Training neural networks with faster kronecker-factored approximate curvature. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13479–13487, 2021.

[32] G. Zhang, S. Sun, D. Duvenaud, and R. Grosse. Noisy natural gradient as variational inference. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5852–5861. PMLR, 10–15 Jul 2018.

A Appendix

A.1 Proof of Lemma 3.1

Lemma 3.1. *Let $M, N, K \in \mathbb{N}$, $\mathbf{L}^k \in \mathbb{R}^{M\times M}$ and $\mathbf{R}^k \in \mathbb{R}^{N\times N}$ for $k \in [K]$. Then*

$$\|\sum_{k=1}^{K} \mathbf{L}^k \otimes \mathbf{R}^k - \mathbf{L}\otimes\mathbf{R}\|_F = \|\sum_{k=1}^{K} \operatorname{vec}(\mathbf{L}^k)\operatorname{vec}(\mathbf{R}^k)^T - \operatorname{vec}(\mathbf{L})\operatorname{vec}(\mathbf{R})^T\|_F. \tag{11}$$

Proof. Let $i, j \in [MN]$. Then the i, j-th entry of the left matrix is

$$\begin{aligned}\left(\sum_{k=1}^{K} \mathbf{L}^k \otimes \mathbf{R}^k - \mathbf{L}\otimes\mathbf{R}\right)_{i,j} &= \sum_{k=1}^{K} \mathbf{L}^k_{i_1,j_1}\mathbf{R}^k_{i_2,j_2} - \mathbf{L}_{i_1,j_1}\mathbf{R}_{i_2,j_2} \\ &= \left(\sum_{k=1}^{K} \operatorname{vec}(\mathbf{L}^k)\operatorname{vec}(\mathbf{R}^k)^T - \operatorname{vec}(\mathbf{L})\operatorname{vec}(\mathbf{R})^T\right)_{M(i_1-1)+j_1, N(i_2-1)+j_2}, \end{aligned} \tag{16}$$

with $i = N(i_1-1)+i_2$, $j = N(j_1-1)+j_2$. Hence, both matrices have the same entries and only the order of the entries is in general different. Therefore, the sum over the squared entries and thus the Frobenius norm is the same

$$\begin{aligned}\|\sum_{k=1}^{K} \mathbf{L}^k \otimes \mathbf{R}^k - \mathbf{L}\otimes\mathbf{R}\|_F^2 &= \sum_{i=1}^{M^2}\sum_{j=1}^{N^2}\left(\sum_{k=1}^{K} \mathbf{L}^k \otimes \mathbf{R}^k - \mathbf{L}\otimes\mathbf{R}\right)^2_{i,j} \\ &= \sum_{i_1,j_1=1}^{N}\sum_{i_2,j_2=1}^{M}\left(\sum_{k=1}^{K} \mathbf{L}^k \otimes \mathbf{R}^k - \mathbf{L}\otimes\mathbf{R}\right)^2_{N(i_1-1)+i_2, N(j_1-1)+j_2} \\ &= \sum_{i_1,j_1=1}^{N}\sum_{i_2,j_2=1}^{M}\left(\sum_{k=1}^{K} \mathbf{L}^k_{i_1,j_1} \otimes \mathbf{R}^k_{i_2,j_2} - \mathbf{L}_{i_1,j_1}\otimes\mathbf{R}_{i_2,j_2}\right)^2 \\ &= \sum_{i_1,j_1=1}^{N}\sum_{i_2,j_2=1}^{M}\left(\sum_{k=1}^{K} \operatorname{vec}(\mathbf{L}^k)\operatorname{vec}(\mathbf{R}^k)^T - \operatorname{vec}(\mathbf{L})\operatorname{vec}(\mathbf{R})^T\right)^2_{M(i_1-1)+j_1, N(i_2-1)+j_2} \\ &= \|\sum_{k=1}^{K} \operatorname{vec}(\mathbf{L}^k)\operatorname{vec}(\mathbf{R}^k)^T - \operatorname{vec}(\mathbf{L})\operatorname{vec}(\mathbf{R})^T\|_F^2 \end{aligned} \tag{17}$$

□

A.2 Proof of Lemma 3.2

Lemma 3.2. *Let $\mathbf{A} = \sum_{k=1}^{K} \operatorname{vec}(\mathbf{L}^k)\operatorname{vec}(\mathbf{R}^k)^T$ and $\mathbf{A} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i\mathbf{v}_i^T$ be its singular value decomposition with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ and $\mathbf{u}_i^T\mathbf{u}_j = \mathbf{v}_i^T\mathbf{v}_j = \mathbb{1}[i=j]$. Then there is a solution of equation 10 with*

$$\operatorname{vec}(\hat{\mathbf{L}}) = \mathbf{u}_1, \operatorname{vec}(\hat{\mathbf{R}}) = \sigma_1\mathbf{v}_1. \tag{12}$$

If $\sigma_1 > \sigma_2$, the solution is unique up to changing the sign of both factors and Algorithm 1 converges almost surely to this solution.

Proof. The main idea of the proof is to use Lemma 3.1 to identify the problem with a best rank-one approximation. The algorithm then corresponds to the power method that utilizes the Kronecker factorization for a faster and memory-efficient computation of the matrix-vector products in the Kronecker matrix space.
By the Eckart–Young–Mirsky theorem [6], an optimal rank-one approximation for A in the Frobenius norm is

$$\sigma_1\mathbf{u}_1\mathbf{v}_1^T \in \underset{\hat{\mathbf{A}}\in\mathbb{R}^{M^2\times N^2}:\operatorname{rank}(\hat{\mathbf{A}})=1}{\arg\min} \|\mathbf{A} - \hat{\mathbf{A}}\|_F, \tag{18}$$

which is unique up to changing the sign of both factors if $\sigma_1 > \sigma_2$.
Therefore, the matrices $\hat{\mathbf{L}}$ and $\hat{\mathbf{R}}$ that satisfy equation (12) are optimal solutions of equation (10). Moreover, the left factor is normalized, e.g. $\|\hat{\mathbf{L}}\|_F = \|\mathbf{u}_1\|_2 = 1$.
The equivalence of Algorithm 1 with the power method can be seen by multiplying $\mathbf{A}\mathbf{A}^T$ with $\operatorname{vec}(\mathbf{L}^{(n-1)})$ for $\mathbf{L}^{(n-1)} \in \mathbb{R}^{N^2}$:

$$\mathbf{A}^T \operatorname{vec}(\mathbf{L}^{(n-1)}) = \sum_{k=1}^{K} \operatorname{vec}(\mathbf{R}^k) \operatorname{vec}(\mathbf{L}^k)^T \operatorname{vec}(\mathbf{L}^{(n-1)}) \tag{19}$$

$$= \sum_{k=1}^{K} \langle \mathbf{L}^k, \mathbf{L}^{(n-1)} \rangle_F \operatorname{vec}(\mathbf{R}^k) \tag{20}$$

$$= \operatorname{vec}(\bar{\mathbf{R}}^{(n)}) \tag{21}$$

and

$$\mathbf{A}\mathbf{A}^T \operatorname{vec}(\mathbf{L}^{(n-1)}) = \sum_{k=1}^{K} \operatorname{vec}(\mathbf{L}^k) \operatorname{vec}(\mathbf{R}^k)^T \operatorname{vec}(\bar{\mathbf{R}}^{(n)}) \tag{22}$$

$$= \sum_{k=1}^{K} \langle \mathbf{R}^k, \bar{\mathbf{R}}^{(n)} \rangle_F \operatorname{vec}(\mathbf{L}^k) \tag{23}$$

$$= \|\bar{\mathbf{R}}^{(n)}\|_F \sum_{k=1}^{K} \langle \mathbf{R}^k, \mathbf{R}^{(n)} \rangle_F \operatorname{vec}(\mathbf{L}^k) \tag{24}$$

$$= \|\bar{\mathbf{R}}^{(n)}\|_F \operatorname{vec}(\bar{\mathbf{L}}^{(n)}). \tag{25}$$

Hence, Algorithm 1 computes the same iterations as the standard power method:

$$\frac{\mathbf{A}\mathbf{A}^T \operatorname{vec}(\mathbf{L}^{(n-1)})}{\|\mathbf{A}\mathbf{A}^T \operatorname{vec}(\mathbf{L}^{(n-1)})\|_2} = \frac{\|\bar{\mathbf{R}}^{(n)}\|_F \operatorname{vec}(\bar{\mathbf{L}}^{(n)})}{\|\bar{\mathbf{R}}^{(n)}\|_F \|\bar{\mathbf{L}}^{(n)}\|_F} = \frac{\operatorname{vec}(\bar{\mathbf{L}}^{(n)})}{\|\bar{\mathbf{L}}^{(n)}\|_F} = \operatorname{vec}(\mathbf{L}^{(n)}). \tag{26}$$

The final right factor then corresponds to $\mathbf{A}^T \operatorname{vec}(\mathbf{L}^{(n)}) \approx \sigma_1 \mathbf{v}_1$.
For $\sigma_1 > \sigma_2$, the convergence properties are inherited from the power method, see *e.g.* [2]. □

Remark. Even in the case when the first singular value is not (much) larger than the other singular values and no convergence is achieved, the resulting matrices of Algorithm 1 are with high probability in the span of the singular vectors corresponding to the set of large singular values [3]. Hence, in this case, the approximation will still converge to good Kronecker factors with high probability.

A.3 Further Results

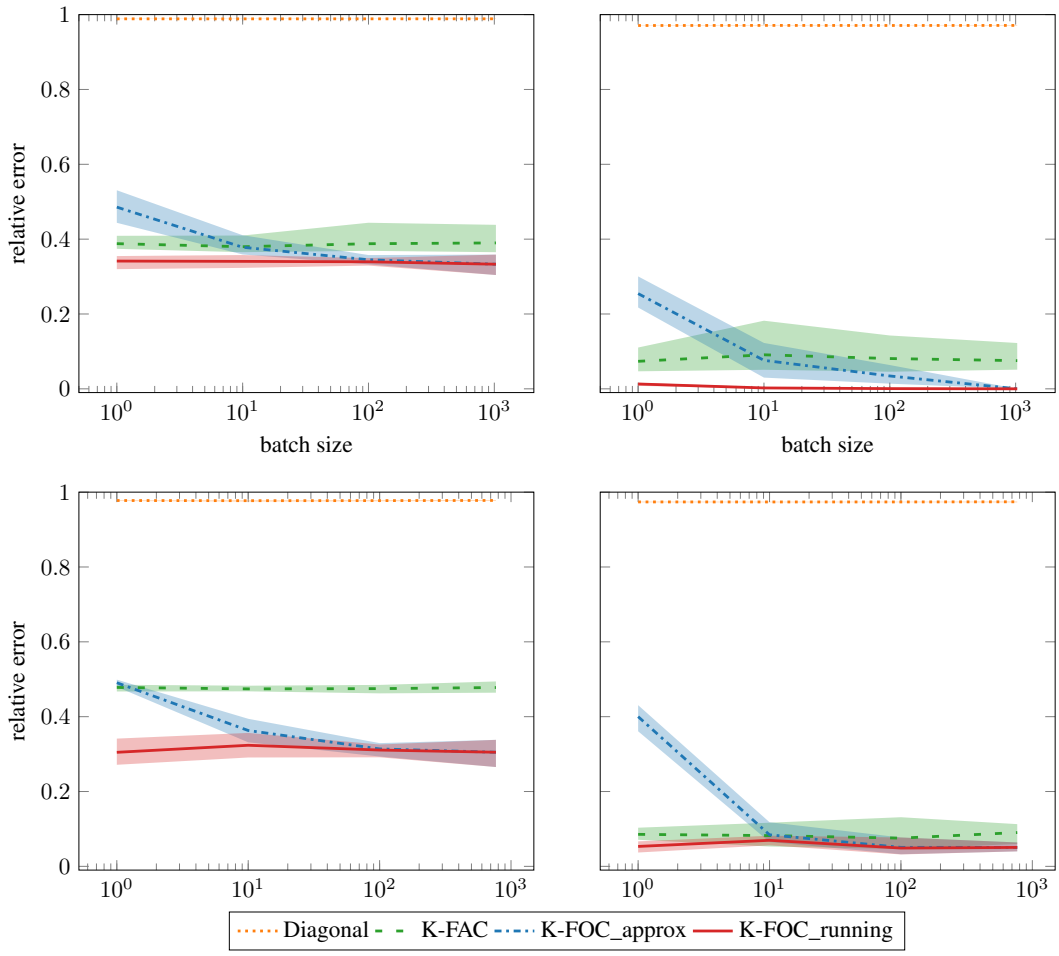

Figure 2: Comparison of the relative Frobenius error for diagonal approximations (orange), K-FAC (green) and K-FOC (blue and red) for different batch sizes for the first (left) and second (right) layer of the relative network architecture. The rows correspond to the fully-connected layers trained on the Concrete Compression Strength (top) and Energy Efficiency dataset (bottom), respectively. The light area shows the minimal and maximal values among ten independent runs while the thick lines are the mean errors. For K-FOC, both aggregation strategies as described in section 3.2 are depicted.

Table 1: Runtime in ms

Name	Approximation	Batch size 1	10	100	full
Boston Housing	K-FAC	180.7 ± 5.5	18.5 ± 1.7	3.2 ± 0.8	1.3 ± 1.1
	K-FOC_approx	727.1 ± 12.8	118.0 ± 7.5	22.9 ± 10.8	3.0 ± 0.8
	K-FOC_running	1262.6 ± 14.5	175.6 ± 6.8	25.2 ± 7.4	2.6 ± 1.0
Concrete Compression Strength	K-FAC	366.8 ± 19.0	35.8 ± 2.9	4.0 ± 1.1	0.2 ± 2.1
	K-FOC_approx	1462.3 ± 17.0	225.2 ± 17.8	37.2 ± 10.8	1.4 ± 1.3
	K-FOC_running	2503.6 ± 34.6	314.6 ± 11.9	55.0 ± 17.6	1.5 ± 2.7
Energy Efficiency	K-FAC	266.7 ± 20.4	27.2 ± 3.4	3.6 ± 1.6	0.7 ± 1.5
	K-FOC_approx	1072.7 ± 15.8	256.7 ± 40.1	28.4 ± 8.1	4.5 ± 5.4
	K-FOC_running	1908.1 ± 20.4	346.5 ± 29.0	41.1 ± 11.9	4.5 ± 4.7

A.4 Runtime

Table 1 shows the runtime of each approximation method on the UCI datasets. K-FOC_approx and K-FOC_running need around seven and nine times as much computational time compared to K-FAC, respectively. In general, each iteration of the power iteration has a similar runtime as K-FAC but usually multiple iterations are needed to converge which then corresponds to a multiple of the runtime of K-FAC. In return, this shows that usually much less iterations are needed than the maximal number of steps $n^{max} = 100$. The additional runtime from K-FOC_running compared to K-FOC_approx comes from the aggregation of the factors for different batches utilizing again the power method to compute an estimate of the running mean. Still, both K-FOC algorithms are in the same complexity classes as K-FAC and only have a small linear overhead compared to K-FAC.