Analysis and Comparison of Two-Level KFAC Methods for Training Deep Neural Networks

Abdoulaye Koroko[a,b], Ani Anciaux-Sedrakian[a], Ibtihel Ben Gharbia[a], Valérie Garès[c], Mounir Haddou[c] and Quang Huy Tran[a]

[a]IFP Energies nouvelles, 1 et 4 avenue de Bois Préau, 92852 Rueil-Malmaison Cedex, France
[b]CentraleSupélec - Université Paris-Saclay, 3 Rue Joliot Curie, 91190 Gif-sur-Yvette, France
[c]Univ. Rennes, INSA, CNRS, IRMAR - UMR 6625, F-35000 Rennes, France

ARTICLE HISTORY
Compiled April 3, 2023

ABSTRACT
As a second-order method, the Natural Gradient Descent (NGD) has the ability to accelerate training of neural networks. However, due to the prohibitive computational and memory costs of computing and inverting the Fisher Information Matrix (FIM), efficient approximations are necessary to make NGD scalable to Deep Neural Networks (DNNs). Many such approximations have been attempted. The most sophisticated of these is KFAC, which approximates the FIM as a block-diagonal matrix, where each block corresponds to a layer of the neural network. By doing so, KFAC ignores the interactions between different layers. In this work, we investigate the interest of restoring some low-frequency interactions between the layers by means of two-level methods. Inspired from domain decomposition, several two-level corrections to KFAC using different coarse spaces are proposed and assessed. The obtained results show that incorporating the layer interactions in this fashion does not really improve the performance of KFAC. This suggests that it is safe to discard the off-diagonal blocks of the FIM, since the block-diagonal approach is sufficiently robust, accurate and economical in computation time.

KEYWORDS
Deep Neural Networks; Natural Gradient Descent; Kronecker Factorization; Two-Level Preconditioning

1. Introduction

Deep learning has achieved tremendous success in many fields such as computer vision [19, 24], speech recognition [44, 46], and natural language processing [5, 14], where its models have produced results comparable to human performance. This was made possible thanks not only to parallel computing resources but also to adequate optimization algorithms, the development of which remains a major research area. Currently, the Stochastic Gradient Descent (SGD) method [43] and its variants [33, 40] are the workhorse methods for training DNNs. Their wide adoption by the machine learning community is justified by their simplicity and their relativeley good behavior on many standard optimization problems. Nevertheless, almost all optimization prob-

Corresponding author: abdoulayekoroko@gmail.fr, abdoulaye.koroko@ifpen.fr

lems arising in deep learning are non-linear and highly non-convex. In addition, the landscape of the objective function may contain huge variations in curvature along different directions [29]. This leads to many challenges in DNNs training, which limit the effectiveness of first-order methods like SGD.

1.1. Approximations of the FIM in NGD methods

By taking advantage of curvature information, second-order methods can overcome the above-mentioned difficulties and speed up the training of DNNs. In such methods, the gradient is rescaled at each iteration with the inverse of a curvature matrix C, whose role is to capture information on the local landscape of the objective function. Several choices of C are available: the well-known Hessian matrix, the Generalized Gauss-Newton matrix (GGN) [45], the FIM [1] or any positive semi-definite approximation of these matrices. The advantage of the GGN and FIM over the Hessian is that they are always positive semi-definite, which is not always guaranteed for the Hessian. Despite their theoretical superiority, second-order methods are unfortunately not practical for training DNNs. This is due to the huge computational and memory requirements for assembling and inverting the curvature matrix C. Several paradigms have therefore been devised to approximate the curvature matrix of DNNs. For example, the Hessian-free approach (HF) [27] eliminates the need to store C by using a Krylov subspace-based Conjugate Gradient (CG) method to solve the linear system involving C. While this approach is memory effective, it remains time-consuming, since one must run at each iteration several steps of CG to converge. Another existing approach is the family of quasi-Newton methods [8, 12, 16, 26, 47] that rely only on gradient information to build a low-rank approximation to the Hessian. Other popular approximations to the curvature matrix are Adagrad [11], RMSprop [49], and Adam [21] which develop diagonal approximations to the empirical FIM. Despite their ease of implementation and scalability to DNNs, both low-rank and diagonal approximations throw away a lot of information and, therefore, are in general less effective than a well-tuned SGD with momentum.

More advanced and sophisticated approximations that have sparked great enthusiasm are the family of Kronecker-factored curvature (KFAC) methods. Evolving from earlier works [20, 25, 36, 38, 41], KFAC methods [18, 30, 31] exploit the network structure to obtain a block-diagonal approximation to the FIM. Each block corresponds to a layer and is further approximated by the Kronecker product of two smaller matrices, cheap to store and easy to invert via the formula $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$. Owing to this attractive feature, KFAC has received a lot of attention and many endeavors have been devoted to improving it. In [3, 37], distributed versions of KFAC were demonstrated to perform well in large-scale settings. The EKFAC method [15] refines KFAC by rescaling the Kronecker factors with a diagonal variance computed in a Kronecker-factored eigenbasis. The TKFAC method [13] preserves a trace-invariance relationship between the approximate and the exact FIM. By removing KFAC's assumption on the independence between activations and pre-activation derivatives, more rigorous Kronecker factorizations can be worked out [22] based on minimization of various errors in the Frobenius norm. Beyond the FIM, the idea of Kronecker factorization can also be extended to the Hessian matrix of DNNs as in KBFGS [17], where the computational complexity is alleviated by approximating the inverse of the Kronecker factors with low-rank updates, as well as the GGN matrix of Multi-Layer Perceptrons (MLP), as shown in [17]. KFAC has also been deployed successfully in the context of Bayesian

deep learning [53], deep reinforcement learning [51] and Laplace approximation [42].

1.2. Enhancement of KFAC by rough layer interaction

For computation and memory purposes, KFAC as well as all related variants use only a block-diagonal approximation of the curvature matrix, where each block corresponds to a layer. This results in a loss of information about the correlations between different layers. The question then naturally arises as to whether it is worth trying to recover some of the lost information in hope of making the approximate FIM closer to the true one, thus improving the convergence speed of the optimizer without paying an excessive price.

To this question, Tselepidis et al. [50] provided an element of answer by considering a "coarse" correction to the inverse of the approximate FIM. This additional term is meant to represent the interaction between layers at a "macroscopic" scale, in contrast with the "microscopic" scale of the interaction between neurons inside each layer. Their approach proceeds by formal analogy with the two-level preconditioning technique in domain decomposition [10], substituting the notion of layer for that of subdomain. The difference with domain decomposition, however, lies in the fact that the matrix at hand does not stem from the discretization of any PDE system, and this prevents the construction of coarse spaces from being correctly guided by any physical sense. Notwithstanding this concern, some ready-made recipes can be blindly borrowed from two-level domain decomposition. In this way, Tselepidis et al. [50] reached a positive conclusion regarding the advisability of enriching the approximate FIM with some reduced information about interactions between layers. Nevertheless, their coarse correction is objectionable in some respects, most notably because of inconsistency in the formula for the new matrix (see §3 for a full discussion), while for the single favorable case on which is based their conclusion, the network architecture selected is a little too simplistic (see §5 for details). Therefore, their claim should not be taken at face value.

Although he did not initially intend to look at the question as formulated above, Benzing [6] recently brought another element of answer that runs counter to the former. By carefully comparing KFAC and the exact natural gradient (as well as FOOF, a method of his own), he came to the astonishingly counterintuitive conclusion that KFAC outperforms the exact natural gradient in terms of optimization performance. In other words, there is no benefit whatsoever in trying to embed any kind of information about the interaction between layers into the curvature matrix, since even the full FIM seems to worsen the situation. While one may not be convinced by his heuristical explanation (whereby KFAC is argued to be a first-order method), his numerical results eloquently speak for themselves. Because Benzing explored a wide variety of networks, it is more difficult to mitigate his findings.

In light of these two contradictory sets of results, we undertook this work in an effort to clarify the matter. To this end, our objective is first to design a family of coarse corrections to KFAC that do not suffer from the mathematical flaws of Tselepidis et al.'s one. This gives rise to a theoretically sound family of approximate FIMs that will next be compared to the original KFAC. This leads to the following outline for the paper. In §2, we introduce notations and recall essential prerequisites on the network model, the natural gradient descent, and the KFAC approximation. In §3, after pointing out the shortcomings of Tselepidis et al.'s corrector, we put forward a series of two-level KFAC methods, the novelty of which is their consistency and their

choices of the coarse space. In §4, we present and comment on several experimental results, which include much more test cases and analysis in order to assess the new correctors as fairly as possible. Finally, in §5, we summarize and discuss the results before sketching out some prospects.

2. Backgrounds on the second-order optimization framework

2.1. Predictive model and its derivatives

We consider a feedforward neural network f_θ, containing ℓ layers and parametrized by

$$\theta = [\mathrm{vec}(W_1)^T, \mathrm{vec}(W_2)^T, \dots, \mathrm{vec}(W_\ell)^T]^T \in \mathbb{R}^p, \tag{2.1}$$

where W_i is the weights matrix associated to layer i and "vec" is the operator that vectorizes a matrix by stacking its columns together. We also consider a training data

$$\mathcal{U} = \big\{(x^{(1)}, y^{(1)}),\ (x^{(2)}, y^{(2)}),\ \dots,\ (x^{(n)}, y^{(n)}) \mid (x^{(b)}, y^{(b)}) \in \mathbb{R}^{d_x \times d_y},\ 1 \le b \le n\big\}$$

and a loss function $L(y, z)$ which measures the discrepancy between the actual target y and the network's prediction $z = f_\theta(x)$ for a given input-target pair $(x, y) \in \mathcal{U}$. The goal of the training problem

$$\operatorname*{argmin}_{\theta \in \mathbb{R}^p} h(\theta) := \frac{1}{n} \sum_{b=1}^{n} L(y^{(b)}, f_\theta(x^{(b)})) \tag{2.2}$$

is to find the optimal value of θ that minimizes the empirical risk $h(\theta)$. In the following, we will designate by $\mathcal{D}v = \nabla_v L$ the gradient of the loss function with respect to any variable v. Depending on the type of network, its output and the gradient of the loss are computed in different ways. Let us describe the calculations for two types of networks.

MLP (Multi-Layer Perceptron). Given an input $x \in \mathbb{R}^{d_x}$, the network computes its output $z = f_\theta(x) \in \mathbb{R}^{d_y}$ through the following sequence, known as forward-propagation: starting from $a_0 := x$, we carry out the iterations

$$s_i = W_i \bar{a}_{i-1}, \qquad a_i = \sigma_i(s_i), \qquad \text{for } i \text{ from 1 to } \ell, \tag{2.3}$$

where $\bar{a}_{i-1} \in \mathbb{R}^{d_{i-1}+1}$ is a_{i-1} concatenated with 1 in order to capture the bias and σ_i is the activation function at layer i. Here, $W_i \in \mathbb{R}^{d_i \times (d_{i-1}+1)}$, with d_i the number of neurons in layer i. The sequence is terminated by $z := a_\ell$. Note that the total number of parameters is necessarily $p = \sum_{i=1}^{\ell} d_i(d_{i-1}+1)$.

The gradient of the loss with respect to the parameters is computed via the back-propagation algorithm: starting from $\mathcal{D}a_\ell = \partial_z L(y, z = a_\ell)$, we perform

$$g_i = \mathcal{D}a_i \odot \sigma_i'(s_i), \quad \mathcal{D}W_i = g_i \bar{a}_{i-1}^T, \quad \mathcal{D}a_{i-1} = W_i^T g_i, \quad \text{for } i \text{ from } \ell \text{ to } 1, \tag{2.4}$$

where the special symbol $g_i := \mathcal{D}s_i$ stands for the preactivation derivative. Note that, in the formula for $\mathcal{D}a_{i-1}$, the last row of W_i^T should be removed so that the product in the right-hand side belongs to $\mathbb{R}^{d_{i-1}}$.

CNN (Convolutional Neural Network). The calculation is governed by the same principle as for MLP, but the practical organization slightly differs from that of MLP. In a convolution layer, the input, which is an image with multiple channels is convolved with a set of filters to produce an output image containing multiple channels. In order to speed up computations, traditional convolution operations are reshaped into matrix-matrix or matrix-vector multiplications using the unrolling approach [9], whereby the input/output data are copied and rearranged into new matrices (see Figure 3).

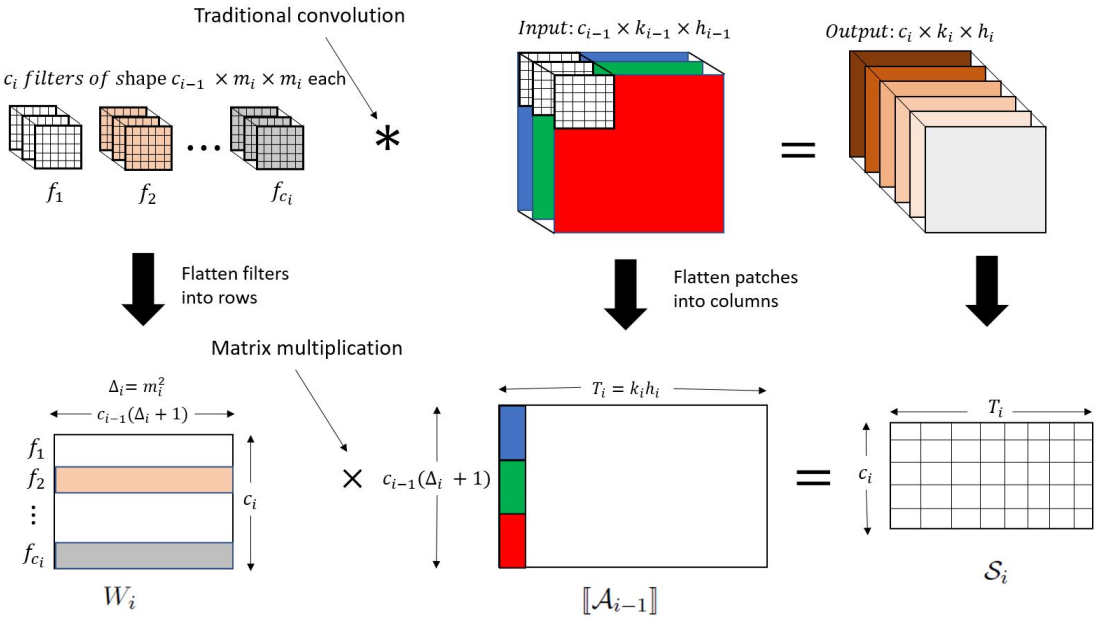


Figure 1. Traditional convolution turned into matrix-matrix multiplication with the *unrolling* approach.

Assume that layer i which receives input $\mathcal{A}_{i-1} \in \mathbb{R}^{c_{i-1} \times T_{i-1}}$, where T_{i-1} denotes the number of spatial locations and c_{i-1} the number of channels. Considering c_i filters, each of which involves Δ_i coefficients, we form a weight matrix W_i of shape $c_i \times (c_{i-1}\Delta_i + 1)$, where each row corresponds to a single filter flattened into a vector. Note that the additional 1 in the column dimension of W_i is required for the bias parameter. Around each position $t \in \{1, \ldots, T_{i-1}\}$, we define the local column vector $a_{i-1,t} \in \mathbb{R}^{c_{i-1}\Delta_i}$ by extracting the patch data from $\mathcal{A}_{i-1}$ (cf. [18] for explicit formulas). The output $\mathcal{A}_i \in \mathbb{R}^{c_i \times T_i}$ is computed by the forward-propagation: for $t \in \{1, \ldots, T_i\}$, the t-th column $\widetilde{a}_{i,t}$ of $\mathcal{A}_i$ is given by

$$s_{i,t} = W_i \bar{a}_{i-1,t}, \qquad \widetilde{a}_{i,t} = \sigma_i(s_{i,t}), \tag{2.5}$$

where $\bar{a}_{i-1,t} \in \mathbb{R}^{c_{i-1}\Delta_i + 1}$ is a_{i-1} concatenated with 1 in order to capture the bias. In matrix form, let $[\![\mathcal{A}_{i-1}]\!] \in \mathbb{R}^{(c_{i-1}\Delta_i + 1) \times T_i}$ be the matrix whose t-th column is $\bar{a}_{i-1,t}$. Then,

$$\mathcal{S}_i = W_i [\![\mathcal{A}_{i-1}]\!], \qquad \mathcal{A}_i = \sigma_i(\mathcal{S}_i). \tag{2.6}$$

The gradient of the loss with respect to the parameters of layer i is computed as

via the back-propagation formulas

$$g_{i,t} = \mathcal{D}\widetilde{a}_{i,t} \odot \sigma_i'(s_{i,t}), \quad \mathcal{D}W_i = \sum_{t=1}^{T_i} g_{i,t}\bar{a}_{i-1,t}^T, \quad \mathcal{D}a_{i-1,t} = W_i^T g_{i,t}, \tag{2.7}$$

for $t \in T_i$, where the special symbol $g_{i,t} := \mathcal{D}s_{i,t}$ stands for the preactivation derivative.

In all cases (MLP or CNN), the gradient $\nabla_\theta L = \mathcal{D}\theta$ of the loss with respect to whole parameter θ is retrieved as

$$\mathcal{D}\theta = [\mathrm{vec}(\mathcal{D}W_1)^T, \mathrm{vec}(\mathcal{D}W_2)^T, \ldots, \mathrm{vec}(\mathcal{D}W_\ell)^T]^T. \tag{2.8}$$

A general descent method to solve the training problem (2.2) is based on the iterates

$$\theta_{k+1} = \theta_k - \alpha_k [C(\theta_k)]^{-1} \nabla_\theta h(\mathcal{S}_k, \theta_k), \tag{2.9}$$

where $\alpha_k > 0$ is the learning rate,

$$\nabla_\theta h(\mathcal{S}_k, \theta_k) = \frac{1}{|\mathcal{S}_k|} \sum_{(x^{(b)}, y^{(b)}) \in \mathcal{S}_k} \nabla_\theta L(y^{(b)}, f_{\theta_k}(x^{(b)})) \tag{2.10}$$

is a batch approximation of the full gradient $\nabla_\theta h(\theta_k) = \frac{1}{n}\sum_{b=1}^n \nabla_\theta L(y^{(b)}, f_{\theta_k}(x^{(b)}))$ on a random subset $\mathcal{S}_k \subset \mathcal{U}$, and $C(\theta_k)$ is an invertible matrix which depends on the method being implemented.

2.2. Natural Gradient Descent

The *Natural Gradient Descent* (NGD) is associated with a particular choice for matrix $C(\theta_k)$, which is well-defined under a mild assumption.

Hypothesis on the loss function. From now on, we take it for granted that there exists a probability density $\wp(y|z)$ on $y \in \mathbb{R}^{d_y}$ such that, up to an additive constant ν, the loss function $L(y, z)$ takes the form

$$L(y, z) = -\log \wp(y|z) + \nu. \tag{2.11}$$

For instance, if the elementary loss corresponds to the least-squares function

$$L(y, z) = \frac{1}{2}\|y - z\|_2^2, \tag{2.12a}$$

then we can take the normal density

$$\wp(y|z) = (2\pi)^{-d_y/2} \exp(-\tfrac{1}{2}\|y - z\|_2^2), \tag{2.12b}$$

so that

$$L(y, z) = -\log \wp(y|z) - \frac{d_y}{2}\log(2\pi). \tag{2.12c}$$

Introduce the notation $p(y|x,\theta) = \wp(y|f_\theta(x))$. Then, the composite loss function

$$L(y, f_\theta(x)) = -\log p(y|x,\theta) \tag{2.13}$$

derives from the density function $p(y|x,\theta)$ of the model's conditional predictive distribution $P_{y|x}(\theta)$. As shown above, $P_{y|x}(\theta)$ is multivariate normal for the standard square loss function. It can also be proved that $P_{y|x}(\theta)$ is multinomial for the cross-entropy one. The learned distribution is therefore $\mathsf{P}_{x,y}(\theta)$ with density

$$\mathsf{p}(x, y|\theta) = q(x)p(y|x,\theta), \tag{2.14}$$

where $q(x)$ is the density of data distribution Q_x over inputs $x \in \mathbb{R}^{d_x}$.

Fisher Information Matrix. The NGD method [1] is defined as the generic algorithm (2.9) in which $C(\theta)$ is set to the *Fisher Information Matrix* (FIM)

$$\begin{aligned} F(\theta) &= \mathbb{E}_{(x,y)\sim\mathsf{P}_{x,y}(\theta)}\{\nabla_\theta \log \mathsf{p}(x,y|\theta)[\nabla_\theta \log \mathsf{p}(x,y|\theta)]^T\} && \text{(2.15a)}\\ &= \mathbb{E}_{(x,y)\sim\mathsf{P}_{x,y}(\theta)}\{\mathcal{D}\theta(\mathcal{D}\theta)^T\} && \text{(2.15b)}\\ &= \mathrm{cov}(\mathcal{D}\theta, \mathcal{D}\theta), && \text{(2.15c)} \end{aligned}$$

where $\mathbb{E}_{(x,y)\sim\mathsf{P}_{x,y}(\theta)}$ denotes the expectation taken over the prescribed distribution $\mathsf{P}_{x,y}(\theta)$ at a fixed θ. To alleviate notations, we shall write $\mathbb{E}$ instead of $\mathbb{E}_{(x,y)\sim\mathsf{P}_{x,y}(\theta)}$ from now on. Likewise, we shall write F instead of $F(\theta)$ or $F(\theta_k)$.

By definition (2.15), the FIM is a covariance matrix and is therefore always positive semi-definite. However, for the iteration

$$\theta_{k+1} = \theta_k - \alpha_k F^{-1}\nabla_\theta h(\mathcal{S}_k, \theta_k) \tag{2.16}$$

to be well-defined, F has to be invertible. This is why, in practice, $C(\theta_k)$ will be taken to be a regularized version of F under the form

$$F_\bullet = F + \lambda I_p, \qquad \lambda > 0. \tag{2.17}$$

The actual NGD iteration is therefore

$$\theta_{k+1} = \theta_k - \alpha_k F_\bullet^{-1}\nabla_\theta h(\mathcal{S}_k, \theta_k). \tag{2.18}$$

In the space of probability distributions $\mathsf{P}_{x,y}(\theta)$ equipped with the *Kullback-Leibler* (KL) divergence, the FIM represents the local quadratic approximation of this induced metric, in the sense that for a small vector $\delta \in \mathbb{R}^p$, we have

$$\mathrm{KL}[\mathsf{P}_{x,y}(\theta) \,\|\, \mathsf{P}_{x,y}(\theta+\delta)] = \frac{1}{2}\delta^T F \delta + O(\|\delta\|^3), \tag{2.19}$$

where KL[P||Q] is the KL divergence between the distributions P and Q As a consequence, the unregularized NGD can be thought of as the steepest descent in this space of probability distributions [2].

Advantages and drawbacks. By virtue of this geometric interpretation, the NGD has the crucial advantage of being intrinsic, that is, invariant with respect to invertible reparameterizations. Put another way, the algorithm will produce matching iterates regardless of how the unknowns are transformed. This is useful in high-dimensional cases where the choice of parameters is more or less arbitrary.

Strictly speaking, invariance with respect to parameters only occurs at the continuous level, i.e., in the limit of $\alpha_k \to 0$. This minor drawback does not undermine the theoretical soundness of the method. In fact, the real drawback of the NGD (2.15)–(2.16) lies in the cost of computing and inverting the Fisher matrix. This is why it is capital to consider suitable approximations of the FIM such as KFAC (cf. §2.3).

Finally and anecdotically, the NGD method can also be viewed as an approximate Newton method since the FIM and the GGN matrix are equivalent when the model predictive distribution $P_{y|x}(\theta)$ belongs to exponential family distributions [28].

2.3. KFAC approximation of the FIM

From equation (2.15), the FIM can be written as a block-diagonal matrix

$$F = \mathbb{E}[\mathcal{D}\theta(\mathcal{D}\theta)^T] = \begin{bmatrix} F_{1,1} & \dots & F_{1,\ell} \\ \vdots & & \vdots \\ F_{\ell,1} & \dots & F_{\ell,\ell} \end{bmatrix} \in \mathbb{R}^{p\times p}, \tag{2.20a}$$

in which each block $F_{i,j}$ is given by

$$F_{i,j} = \mathbb{E}[\mathrm{vec}(\mathcal{D}W_i)\mathrm{vec}(\mathcal{D}W_j)^T]. \tag{2.20b}$$

One can interpret $F_{i,i}$ as being second-order statistics of weight derivatives of layer i, and $F_{i,j}$, $i \neq j$ as representing the interactions between layer i and j.

The KFAC method [31] is grounded on the following two assumptions to provide an efficient approximation to the FIM that is convenient for training DNNs.

(1) The first one is that there are no interactions between two different layers, i.e., $F_{i,j} = 0$ for $i \neq j$. This results in the block-diagonal approximation

$$F \approx \widetilde{F} = \mathrm{diag}(F_{1,1}, F_{2,2}, \dots, F_{\ell,\ell}) \tag{2.21}$$

for the FIM. At this step, computing the inverse of $\widetilde{F}$ is equivalent to computing the inverses of diagonal blocks $F_{i,i}$. Nevertheless, because the diagonal blocks $F_{i,i}$ can be very large (especially for DNNs with large layers), this first approximation remains insufficient.

(2) The second one comes in support of the first one and consists in factorizing each diagonal block $F_{i,i}$ as a Kronecker product of two smaller matrices, namely,

$$F_{i,i} \approx [F_{\mathrm{KFAC}}]_{i,i} = A_i \otimes G_i. \tag{2.22}$$

where the Kronecker product between $A \in \mathbb{R}^{m_A\times n_A}$ and $B \in \mathbb{R}^{m_B\times n_B}$ is the

matrix of size $m_A m_B \times n_A n_B$ given by

$$A \otimes B = \begin{bmatrix} A_{1,1}B & \dots & A_{1,n_A}B \\ \vdots & & \vdots \\ A_{m_A,1}B & \dots & A_{m_A,n_A}B \end{bmatrix}. \tag{2.23}$$

Now, depending on the type of the layer, the computation of the Kronecker factors A_i and G_i may require different other assumptions.

MLP layer. When layer i is an MLP, the block $F_{i,i}$ is given by

$$\begin{aligned} F_{i,i} &= \mathbb{E}[\mathrm{vec}(\mathcal{D}W_i)\mathrm{vec}(\mathcal{D}W_j)^T] \\ &= \mathbb{E}[\mathrm{vec}(g_i \bar{a}_{i-1}^T)\mathrm{vec}(g_i \bar{a}_{i-1}^T)^T] \\ &= \mathbb{E}[(\bar{a}_{i-1} \otimes g_i)(\bar{a}_{i-1} \otimes g_i)^T] \\ &= \mathbb{E}[\bar{a}_{i-1}\bar{a}_{i-1}^T \otimes g_i g_i^T]. \end{aligned} \tag{2.24}$$

From the last equality, if one assumes that activations and pre-activation derivatives are independent, that is, $a_{i-1} \perp\!\!\!\perp g_i$, then $F_{i,i}$ can be factorized as

$$F_{i,i} \approx [F_{\mathrm{KFAC}}]_{i,i} = \mathbb{E}[\bar{a}_{i-1}\bar{a}_{i-1}^T] \otimes \mathbb{E}[g_i g_i^T] =: A_i \otimes G_i, \tag{2.25}$$

with $A_i = \mathbb{E}[\bar{a}_{i-1}\bar{a}_{i-1}^T]$ and $G_i = \mathbb{E}[g_i g_i^T]$.

Convolution layer. With such a layer, $F_{i,i}$ is written as

$$\begin{aligned} F_{i,i} &= \mathbb{E}\big[\mathrm{vec}(\mathcal{D}W_i)\mathrm{vec}(\mathcal{D}W_j)^T\big] \\ &= \mathbb{E}\left[\mathrm{vec}\left(\sum_{t=1}^{T_i} g_{i,t}\bar{a}_{i-1,t}^T\right)\mathrm{vec}\left(\sum_{t=1}^{T_i} g_{i,t}\bar{a}_{i-1,t}^T\right)^T\right] \\ &= \mathbb{E}\left[\sum_{t=1}^{T_i}\sum_{t'=1}^{T_i}(\bar{a}_{i-1,t} \otimes g_{i,t})(\bar{a}_{i-1,t'} \otimes g_{i,t'})^T\right] \\ &= \mathbb{E}\left[\sum_{t=1}^{T_i}\sum_{t'=1}^{T_i}\bar{a}_{i-1,t}\bar{a}_{i-1,t'}^T \otimes g_{i,t}g_{i,t'}^T\right] \\ &= \mathbb{E}\left[\sum_{t=1}^{T_i}\sum_{t'=1}^{T_i}\Omega_i(t,t') \otimes \Gamma_i(t,t')\right], \end{aligned} \tag{2.26}$$

with $\Omega_i(t,t') = \bar{a}_{i-1,t}\bar{a}_{i-1,t'}^T$ and $\Gamma_i(t,t') = g_{i,t}g_{i,t'}^T$. In order to factorize $F_{i,i}$ into Kronecker product of two matrices, Grosse and Martens [18] resort to three hypotheses. First, similarly to MLP layers, activations and pre-activation derivatives are assumed to be independent. Secondly, postulating spatial homogeneity, the second-order statistics of the activations and pre-activation derivatives at any two spatial locations t and t' depend only on the difference $t-t'$. Finally, the pre-activation derivatives at any two distinct spatial locations are declared to be uncorrelated, i.e., $\Gamma_i(t,t') = 0$ for $t \neq t'$.

Combining these three assumptions yields the approximation

$$F_{i,i} \approx [F_{\text{KFAC}}]_{i,i} = \mathbb{E}\Big[\sum_{t=1}^{T_i} \Omega_i(t,t)\Big] \otimes \frac{1}{T_i}\mathbb{E}\Big[\sum_{t=1}^{T_i} \Gamma_i(t,t)\Big] =: A_i \otimes G_i, \tag{2.27}$$

with $A_i = \mathbb{E}\big[\sum_{t=1}^{T_i} \Omega_i(t,t)\big]$ and $G_i = \frac{1}{T_i}\mathbb{E}\big[\sum_{t=1}^{T_i} \Gamma_i(t,t)\big]$.

Remark 2.1. It should be mentioned that, in the same spirit, a KFAC-type approximation has been developed for the RNN (Recurrent Neural Network), but with much more assumptions. In this work, we do not consider recurrent layers. The readers interested in KFAC for RNN are referred to [30].

Going back to MLP and CNN layers, the matrices A_i and G_i are estimated using Monte Carlo method, with a mini-batch $\mathcal{B} = \{(x_1, y_1), \dots, (x_B, y_B)\}$, where the targets y_i's are sampled from the model predictive distribution $P_{y|x}(\theta)$. Combining the block-diagonal approximation and Kronecker factorization of each block, the approximate FIM becomes

$$F \approx F_{\text{KFAC}} = \text{diag}(A_1 \otimes G_1,\ A_2 \otimes G_2,\ \dots,\ A_\ell \otimes G_\ell). \tag{2.28}$$

The descent iteration (2.9) with $C(\theta_k) = F_{\text{KFAC}}(\theta_k)$ is now well suited to training DNNs. Indeed, thanks to the Kronecker product properties $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$ and $(A \otimes B)\text{vec}(X) = \text{vec}(BXA^T)$, it is plain that the product

$$F_{\text{KFAC}}^{-1}\nabla_\theta h = \begin{bmatrix} \text{vec}(G_1^{-1}(\nabla_{W_1} h)A_1^{-1}) \\ \vdots \\ \text{vec}(G_\ell^{-1}(\nabla_{W_\ell} h)A_\ell^{-1}) \end{bmatrix} \tag{2.29}$$

only requires to store and to invert matrices of moderately small sizes.

In practice, invertibility of F_{KFAC} must be enforced by a regularization procedure. The usual Tikhonov one, by which we consider $F_{\text{KFAC}} + \lambda I, \lambda > 0$, instead of F_{KFAC}, is equivalent to adding a multiple of the identity matrix of appropriate size to each diagonal block, i.e., $A_i \otimes G_i + \lambda I_i$. Unfortunately, this breaks down the Kronecker factorization structure of the blocks. To preserve the factorized structure, the authors of KFAC [31] advocate a heuristic damping technique in which each Kronecker factor is regularized as

$$[F_{\bullet\,\text{KFAC}}]_{i,i} = (A_i + \pi_i\lambda^{1/2} I_{A_i}) \otimes (G_i + \pi_i^{-1}\lambda^{1/2} I_{G_i}), \tag{2.30a}$$

where I_{A_i} and I_{G_i} denote identity matrices of same size as A_i and G_i respectively, and

$$\pi_i = \sqrt{\frac{\text{tr}(A_i)/(d_{i-1}+1)}{\text{tr}(G_i)/d_i}}. \tag{2.30b}$$

The actual KFAC iteration is therefore

$$\theta_{k+1} = \theta_k - \alpha_k F_{\bullet\,\text{KFAC}}^{-1}\nabla_\theta h(\mathcal{S}_k, \theta_k), \tag{2.31a}$$

with

$$F_{\bullet\,\mathrm{KFAC}} = \mathrm{diag}([F_{\bullet\,\mathrm{KFAC}}]_{1,1},\, [F_{\bullet\,\mathrm{KFAC}}]_{2,2},\, \ldots,\, [F_{\bullet\,\mathrm{KFAC}}]_{\ell,\ell}). \tag{2.31b}$$

3. Two-level KFAC methods

Henceforth, the learning rate is assumed to be $\alpha_k = 1$. Let

$$\zeta_k = \theta_k - \theta_{k+1} = [C(\theta_k)]^{-1} \nabla_\theta h(\mathcal{S}_k, \theta_k) \tag{3.1}$$

be the negative increment of θ at iteration k of the generic descent algorithm (2.9). To further alleviate notations, we shall drop the subscript k and omit the dependence on θ_k. For the regularized NGD (2.18), we have

$$\zeta = F_\bullet^{-1}\, \nabla_\theta h, \tag{3.2}$$

while for the regularized KFAC method, we have

$$\zeta_{\mathrm{KFAC}} = F_{\bullet\,\mathrm{KFAC}}^{-1} \nabla_\theta h, \tag{3.3}$$

being understood that the matrices are regularized whenever necessary.

We want to build a new matrix $F_{\bullet\,\mathrm{KFAC\text{-}2L}}^{-1}$, an augmented version of $F_{\bullet\,\mathrm{KFAC}}^{-1}$, such that the solution

$$\zeta_{\mathrm{KFAC\text{-}2L}} = F_{\bullet\,\mathrm{KFAC\text{-}2L}}^{-1} \nabla_\theta h, \tag{3.4}$$

is a better approximation to ζ than ζ_{KFAC}, namely,

$$\|\zeta_{\mathrm{KFAC\text{-}2L}} - \zeta\|_F \ll \|\zeta_{\mathrm{KFAC}} - \zeta\|_F. \tag{3.5}$$

By "augmented" we mean that, at least partially and at some rough scale, $F_{\mathrm{KFAC\text{-}2L}}^{-1}$ takes into account the information about layer interactions that was discarded by the block-diagonal approximation KFAC. The basic tenet underlying this initiative is the belief that a more accurate approximation to the NGD solution ζ at each descent iteration will help the global optimization process to converge faster.

3.1. Analogy and dissimilarity with domain decomposition

The construction philosophy of $F_{\bullet\,\mathrm{KFAC\text{-}2L}}^{-1}$ proceeds by analogy with insights from domain decomposition. To properly explain the analogy, we first need to cast the matrix $F_{\bullet\,\mathrm{KFAC}}^{-1}$ under a slightly different form.

For each $i \in \{1, \ldots, \ell\}$, let $R_i \in \mathbb{R}^{p_i \times p}$ be the matrix of the restriction operator from $\mathbb{R}^p$, the total space of all parameters, to the subspace of parameters pertaining to layer i, whose dimension is p_i. In other words, for $(\xi, \eta) \in \{1, \ldots, p_i\} \times \{1, \ldots, p\}$,

$$(R_i)_{\xi\eta} = \begin{cases} 1 & \text{if } \ \eta = p_1 + \ldots + p_{i-1} + \xi, \\ 0 & \text{otherwise.} \end{cases} \tag{3.6}$$

The transpose $R_i^T \in \mathbb{R}^{p \times p_i}$ then represents the prolongation operator from the subspace of parameters in layer i to the total space of all parameters. Obviously, the i-th diagonal block of the regularized FIM can be expressed as

$$[F_\bullet]_{i,i} = R_i F_\bullet R_i^T.$$

If there were no approximation of each diagonal block by a Kronecker product, then the block-diagonal approximation of F would give rise to the inverse matrix

$$F_{\bullet\,\text{block-diag}}^{-1} = \sum_{i=1}^{\ell} R_i^T [F_\bullet]_{i,i}^{-1} R_i = \sum_{i=1}^{\ell} R_i^T (R_i F_\bullet R_i^T)^{-1} R_i. \tag{3.7}$$

In the case of KFAC, it follows from (2.28)–(2.29) that

$$\begin{aligned} F_{\bullet\,\text{KFAC}}^{-1} &= \sum_{i=1}^{\ell} R_i^T [F_{\bullet\,\text{KFAC}}]_{i,i}^{-1} R_i \\ &= \sum_{i=1}^{\ell} R_i^T (A_i + \pi_i \lambda^{1/2} I_{A_i})^{-1} \otimes (G_i + \pi_i^{-1} \lambda^{1/2} I_{G_i})^{-1} R_i. \end{aligned} \tag{3.8}$$

In the context of the domain decomposition methods to solve linear systems arising from the discretization of PDEs, the spatial domain of the initial problem is divided into several subdomains. The system is then projected onto the subdomains and the local subproblems are solved independently of each other as smaller systems. In this stage, parallelism can be fully taken advantage of by assigning a processor to each subdomain. This produces a local solution on each subdomain. These local solutions are next combined to create an approximate global solution on the overall domain. Algebraically, the whole process is tantamount to using an inverse matrix of a form similar to (3.7)–(3.8) either within a Schwarz-like iterative procedure or as a preconditioner [10]. The counterparts of $[F_\bullet]_{i,i}^{-1}$ or $[F_{\bullet\,\text{KFAC}}]_{i,i}^{-1}$ are referred to as *local solvers*.

Remark 3.1. The above analogy is not a perfect one. In domain decomposition, the subdomains are allowed (and even recommended!) to overlap each other, so that an unknown can belong to two or more subdomains. In this case, the restriction operators R_i can be much more intricate than the one trivially defined in (3.6).

A well-known issue with domain decomposition methods of the form (3.7)–(3.8) is the disappointingly slow rate of convergence, which results in a lack of *scalability* [10]: the speed-up factor does not grow proportionally with the number of subdomains (and therefore of processors). The reason is that, as the number of subdomains increases, it takes more iterations for an information local to one subdomain to be propagated and taken into account by the others. The common remedy to this problem is to append a "coarse" correction that enables subdomains to communicate with each other in a faster way. The information exchanged in this way is certainly not complete, but only concerns the low frequencies.

Remark 3.2. In domain decomposition, there is a physical problem (represented by the PDE at the continuous level) that serves as a support for the mathematical and numerical reasoning. This is not the case here, where we have to think in a purely algebraic way.

3.2. Multiplicative vs. additive coarse correction

We are going to present the idea of two-level KFAC in a very elementary fashion. Let $m \geq \ell$ be an integer and $R_0 \in \mathbb{R}^{m \times p}$ be a given matrix. The subspace of $\mathbb{R}^p$ spanned by the columns of $R_0^T \in \mathbb{R}^{p \times m}$ is called the *coarse space*. The choice of the coarse space will be discussed later on. For the moment, we can assume that it is known.

The idea is to add to ζ_{KFAC} a correction term that lives in the coarse space, in such a way that the new vector minimizes the error in the $F_\bullet$-norm with respect to the FIM solution $\zeta = F_\bullet^{-1} \nabla_\theta h$. More concretely, this means that for the negative increment, we consider

$$\zeta_{\text{KFAC-2L}} = \zeta_{\text{KFAC}} + R_0^T \beta^*, \tag{3.9}$$

where

$$\beta^* = \underset{\beta \in \mathbb{R}^\ell}{\text{argmin}} \, \|(\zeta_{\text{KFAC}} + R_0^T \beta) - \zeta\|_{F_\bullet}^2 \tag{3.10a}$$

$$= \underset{\beta \in \mathbb{R}^\ell}{\text{argmin}} \, \|(\zeta_{\text{KFAC}} + R_0^T \beta) - F_\bullet^{-1} \nabla_\theta h\|_{F_\bullet}^2. \tag{3.10b}$$

The solution of the quadratic minimization problem (3.10) is given by

$$\beta^* = (R_0 F_\bullet R_0^T)^{-1} R_0 (\nabla_\theta h - F_\bullet \zeta_{\text{KFAC}}), \tag{3.11}$$

provided that the matrix

$$F_{\text{coarse}} := R_0 F_\bullet R_0^T \in \mathbb{R}^{m \times m}, \tag{3.12}$$

representing the *coarse operator*, be invertible. This is a small size matrix, insofar as m will be in practice taken to be equal to ℓ or 2ℓ, and will in any case remain much smaller than p. This is in agreement with domain decomposition where the size of the coarse system is usually equal to the number of subdomains.

As for the vector

$$r_{\text{KFAC}} := \nabla_\theta h - F_\bullet \zeta_{\text{KFAC}}, \tag{3.13}$$

it is referred to as the *residual* associated to the approximate solution ζ_{KFAC}. Plugging (3.11) into (3.9) and recalling that $\zeta_{\text{KFAC}} = F_{\bullet\,\text{KFAC}}^{-1} \nabla_\theta h$, we end up with

$$\zeta_{\text{KFAC-2L}} = F_{\bullet\,\text{KFAC-2L}}^{-1} \nabla_\theta h, \tag{3.14}$$

with

$$F_{\bullet\,\text{KFAC-2L}}^{-1} = F_{\bullet\,\text{KFAC}}^{-1} + R_0^T F_{\text{coarse}}^{-1} R_0 (I - F_\bullet F_{\bullet\,\text{KFAC}}^{-1}). \tag{3.15}$$

The matrix (3.15) that we propose can be checked to be consistent: if $F_{\bullet\,\text{KFAC}}^{-1}$ and $R_0^T F_{\text{coarse}}^{-1} R_0$ are both homogeneous to $F_\bullet^{-1}$, then $F_{\bullet\,\text{KFAC-2L}}^{-1}$ is homogenous to

$$F_\bullet^{-1} + F_\bullet^{-1} - F_\bullet^{-1} F_\bullet F_\bullet^{-1} = F_\bullet^{-1}$$

too. In the language of domain decomposition, the coarse corrector of (3.15) is said to act *multiplicatively*, to the extent that

$$I - F_{\bullet\,\text{KFAC-2L}}^{-1} F_\bullet = [I - (R_0^T F_{\text{coarse}}^{-1} R_0) F_\bullet][I - F_{\bullet\,\text{KFAC}}^{-1} F_\bullet]. \tag{3.16}$$

as can be straightforwarldy verified. If G is an approximation of $F_\bullet^{-1}$, the matrix $I - GF_\bullet$ measures the quality of this approximation. Equality (3.16) shows that the approximation quality of $F_{\bullet\,\text{KFAC-2L}}^{-1}$ is the product of those of $R_0^T F_{\text{coarse}}^{-1} R_0$ and $F_{\bullet\,\text{KFAC}}^{-1}$.

A common practice in domain decomposition is to drop the factor $I - F_\bullet F_{\bullet\,\text{KFAC}}^{-1}$ (which is equivalent to replacing the residual $r_{\text{KFAC}} = \nabla_\theta h - F_\bullet \theta_{\text{KFAC}}$ by $\nabla_\theta h$). This amounts to approximating $F_{\bullet\,\text{KFAC-2L}}^{-1}$ as

$$F_{\bullet\,\text{KFAC-2L}}^{-1} \approx F_{\bullet\,\text{KFAC}}^{-1} + R_0^T F_{\text{coarse}}^{-1} R_0. \tag{3.17}$$

The coarse corrector of (3.17) is said to act *additively* in domain decomposition. Clearly, the resulting matrix is inconsistent with $F_\bullet^{-1}$: in fact, it is consistent with $2F_\bullet^{-1}$! No matter how crude it is, this coarse corrector is actually valid as long as $F_{\bullet\,\text{KFAC-2L}}^{-1}$ is used only as a preconditioner in the resolution of the system $F_\bullet \zeta = \nabla_\theta h$, which means that we solve instead $F_{\bullet\,\text{KFAC-2L}}^{-1} F_\bullet \zeta = F_{\bullet\,\text{KFAC-2L}}^{-1} \nabla_\theta h$ to benefit from a more favorable conditioning but the solution we seek remains the same.

Here, in our problem, $F_\bullet^{-1}$ is directly approximated by $F_{\bullet\,\text{KFAC-2L}}^{-1}$ and therefore the inconsistent additive coarse corrector (3.17) is not acceptable. Note that Tselepidis et al. [50] adopted this additive coarse correction, in which F_{coarse} is approximated as

$$F_{\text{coarse}} \approx R_0 \bar{F}_\bullet R_0^T, \tag{3.18a}$$

where $\bar{F}_\bullet$ is the block-diagonal matrix whose blocks $[\bar{F}_\bullet]_{i,j}$ are given by

$$[\bar{F}_\bullet]_{i,j} = \begin{cases} \mathbb{E}[\bar{a}_{i-1}\bar{a}_{j-1}^T] \otimes \mathbb{E}[g_i g_j^T] & \text{if } \; i \neq j, \\ \mathbb{E}[A_i + \pi_i \lambda^{1/2} I_{A_i}] \otimes \mathbb{E}[G_i + \pi_i^{-1} \lambda^{1/2} I_{G_i}] & \text{if } \; i = j. \end{cases} \tag{3.18b}$$

In this work, we focus to the consistent multiplicative coarse corrector (3.15) and also consider the exact value (3.12) for F_{coarse}.

3.3. Choice of the coarse space R_0^T

By the construction (3.9)–(3.10), we are guaranteed that

$$\|\zeta_{\text{KFAC-2L}} - \zeta\|_F^2 \leq \|\zeta_{\text{KFAC}} - \zeta\|_F^2 \tag{3.19}$$

for any coarse space R_0^T, since the right-hand side corresponds to $\beta = 0$. The choice of R_0^T is a compromise between having a small dimension $m \ll p$ and lowering the new error

$$\left\|\zeta_{\text{KFAC-2L}} - \zeta\right\|_F^2 = \left\| - [I - R_0^T (R_0 F_\bullet R_0^T)^{-1} R_0 F_\bullet][I - F_{\bullet\,\text{KFAC}}^{-1} F_\bullet]\zeta \right\|_F^2 \tag{3.20}$$

as much as possible. But it seems out of reach to carry out the minimization of the latter with respect to the entries of R_0^T.

In the context of the preconditioner, the idea behind a two-level method is to remove first the influence of very large eigenvalues which correspond to high-frequency modes, then remove the smallest eigenvalues thanks to the second level, which affect greatly the convergence. To do so, we need a suitable coarse space to efficiently deal with this second level [32]. Ideally, we would like to choose the deflation subspace which consists of the eigenvectors associated with the small eigenvalues of the preconditioned operator. However, this computation is more costly than solving a linear system itself.

This leads us to choose the coarse space in an a priori way. We consider the a priori form

$$R_0^T = \begin{bmatrix} V_1 & 0 & \dots & \dots & 0 \\ 0 & V_2 & \dots & \dots & 0 \\ \vdots & \vdots & \ddots & & \vdots \\ 0 & 0 & \dots & \dots & V_\ell \end{bmatrix} \in \mathbb{R}^{p \times m}, \tag{3.21}$$

where each block $V_i \in \mathbb{R}^{p_i \times m_i}$ has m_i columns with $m_i \ll p_i$, and

$$m_1 + m_2 + \ldots + m_\ell = m. \tag{3.22}$$

To provide a comparative study, we propose to evaluate several coarse space choices of the form (3.21) that are discussed below.

Nicolaides coarse space. Historically, this is the first [35] coarse space ever proposed in domain decomposition. Transposed to our case, it corresponds to

$$m_1 = \ldots = m_\ell = 1, \qquad m = \ell, \tag{3.23}$$

and for all $i \in \{1, \ldots, \ell\}$,

$$V_i = \begin{bmatrix} 1, \ldots, 1 \end{bmatrix}^T \in \mathbb{R}^{p_i}. \tag{3.24}$$

Originally, the motivation for selecting the vector whose all components are equal to 1 is that it is the discrete version of a continuous constant field, which is the eigenvector associated with the eigenvalue 0 of the operator $-\nabla \cdot (\kappa \nabla)$ (boundary conditions being set aside). Inserting it into the coarse space helps the solver take care of the lowest frequency mode. In our problem, however, there is no reason for 0 to be an eigenvalue of F, nor for 1 to be an eigenvector if this is the case. Hence, there is no justification for the Nicolaides coarse space. Still, this choice remains convenient and practical. This is probably the reason why Tselepidis et al. [50] have opted for it.

Spectral coarse space. This is a slightly refined version of the Nicolaides coarse space. The idea is always to capture the lowest mode [32], but since the lowest eigenvalue and eigenvector are not known in advance, we have to compute them. More specifically, we keep the values (3.23) for the column sizes within R_0^T, while prescribing

$$V_i = \text{eigenvector associated to the smallest eigenvalue of } [F_{\bullet \, \text{KFAC}}]_{i,i} \tag{3.25}$$

for all $i \in \{1, \ldots, \ell\}$. In our case, an advantageous feature of this definition is that the cost of computing the eigenvectors is "amortized" by that of the inverses of $[F_{\text{KFAC}}]_{i,i}$, in the sense that these two calculations can be carried out simultaneously. Indeed, let

$$A_i + \pi_i \lambda^{1/2} I_{A_i} = U_{A_i} \Sigma_{A_i} V_{A_i}^T, \qquad G_i + \pi_i^{-1} \lambda^{1/2} I_{G_i} = U_{G_i} \Sigma_{G_i} V_{G_i}^T \tag{3.26}$$

be the singular value decompositions of $A_i + \pi_i \lambda^{1/2} I_{A_i}$ and $G_i + \pi_i^{-1} \lambda^{1/2} I_{G_i}$ respectively. Then,

$$\begin{aligned} [F_{\bullet\,\text{KFAC}}]_{i,i}^{-1} &= (A_i + \pi_i \lambda^{1/2} I_{A_i})^{-1} \otimes (G_i + \pi_i^{-1} \lambda^{1/2} I_{G_i})^{-1} \\ &= (U_{A_i} \Sigma_{A_i} V_{A_i}^T)^{-1} \otimes (U_{G_i} \Sigma_{G_i} V_{G_i}^T)^{-1} \\ &= (U_{A_i} \Sigma_{A_i}^{-1} V_{A_i}^T) \otimes (U_{G_i} \Sigma_{G_i}^{-1} V_{G_i}^T). \end{aligned} \tag{3.27}$$

Since Σ_{A_i} and Σ_{G_i} are diagonal matrices, their inverses are easy to compute. Now, if V_{A_i} and V_{G_i} are the eigenvectors associated to the smallest eigenvalues of A_i and G_i respectively, then the eigenvector associated to the smallest eigenvalue of $[F_{\bullet\,\text{KFAC}}]_{i,i}$ is given by

$$V_i = V_{A_i} \otimes V_{G_i}. \tag{3.28}$$

Krylov coarse space. If we do not wish to compute the eigenvector associated to the smallest eigenvalue of $[F_{\bullet\,\text{KFAC}}]_{i,i}$, then a variant of the spectral coarse space could be the following. We know that this eigenvector can be obtained by the inverse power method. The idea is then to perform a few iterations of this method, even barely one or two, and to include the iterates into the the coarse subspace. If $m_i - 1 \geq 1$ is the number of inverse power iterations performed for $[F_{\bullet\,\text{KFAC}}]_{i,i}$, then we take

$$V_i = [v_i, \;\; [F_{\bullet\,\text{KFAC}}]_{i,i}^{-1} v_i, \;\; \ldots, \;\; [F_{\bullet\,\text{KFAC}}]_{i,i}^{-(m_i-1)} v_i] \;\in\; \mathbb{R}^{p_i \times m_i} \tag{3.29}$$

where $v_i \in \mathbb{R}^{p_i}$ is an arbitrary vector, assumed to not be an eigenvector of $[F_{\bullet\,\text{KFAC}}]_{i,i}$ to ensure that the columns of V_i are not collinear. By appropriately selecting v_i, we are in a position to use this approach to enrich the Nicolaides coarse space and the residuals coarse space (cf. next construction).

The increase in the number of columns for V_i is not the price to be paid to avoid the eigenvector calculation: we could have put only the last iterate $[F_{\bullet\,\text{KFAC}}]_{i,i}^{-(m_i-1)} v_i$ into V_i. But since we have computed the previous ones, it seems more cost-effective to use them all to enlarge the coarse space. The larger the latter is, the lower is the minimum value of the objective function. In this work, we consider the simplest case

$$m_1 = \ldots = m_\ell = 2, \qquad m = 2\ell. \tag{3.30}$$

Residuals coarse space. We now introduce a very different philosophy of coarse space, which to our knowledge has never been envisioned before. From the construction (3.9)–(3.10), it is obvious that if the error $\zeta - \zeta_{\text{KFAC}}$ belongs to the coarse space R_0^T, that is, if it can be written as a linear combination $R_0^T \beta^\sharp$ of the coarse matrix columns, then the vector $\zeta_{\text{KFAC}} + R_0^T \beta^\sharp$ coincides with the exact solution ζ and the correction

would be ideally optimal. Although this error $\zeta - \zeta_{\text{KFAC}}$ is unknown, it is connected to the residual (3.13) by

$$\zeta - \zeta_{\text{KFAC}} = F_{\bullet}^{-1} r_{\text{KFAC}}. \tag{3.31}$$

The residual r_{KFAC} is not too expensive to compute. as it consists of a direct matrix-product $F\zeta_{\text{KFAC}}$. Unfortunately, solving a linear system involving F as required by (3.31) is what we want to avoid.

But we can just approximate this error by inverting with $F_{\bullet\,\text{KFAC}}^{-1}$ instead of $F_{\bullet}^{-1}$. Therefore, we propose to build a coarse space that contains $F_{\bullet\,\text{KFAC}}^{-1} r_{\text{KFAC}}$ instead of $F_{\bullet}^{-1} r_{\text{KFAC}}$. To this end, we split $F_{\bullet\,\text{KFAC}}^{-1} r_{\text{KFAC}}$ into ℓ segments, each corresponding to a layer. This amounts to choosing the values (3.23) for the column sizes and set the columns of R_0^T as

$$V_i = [F_{\bullet\,\text{KFAC}}]_{i,i}^{-1} r_{\text{KFAC}}[i] \in \mathbb{R}^{p_i}, \qquad r_{\text{KFAC}}[i] = \text{vec}(\mathcal{D}W_i) - (F_{\bullet}\zeta_{\text{KFAC}})[i] \tag{3.32}$$

for $i \in \{1, \ldots, \ell\}$, where for a vector $\xi \in \mathbb{R}^p$ the notation $\xi[i] = \xi(p_{i-1} + 1 : p_i)$ designates the portion related to layer i. Formulas (3.32) ensure that $F_{\bullet\,\text{KFAC}}^{-1} r_{\text{KFAC}}$ belongs to the coarse space. Indeed, taking $\beta = [1, \ldots, 1]^T \in \mathbb{R}^{\ell}$, we find $R_0^T \beta = F_{\bullet\,\text{KFAC}}^{-1} r_{\text{KFAC}}$.

Taylor coarse space. The previous coarse space is the zeroth-order representative of a family of more sophisticated constructions based on a formal Taylor expansion of $F_{\bullet}^{-1}$, which we now present but which will not be implemented. Setting

$$E = I - F_{\bullet\,\text{KFAC}}^{-1} F_{\bullet} \tag{3.33}$$

and observing that $F_{\bullet} = F_{\bullet\,\text{KFAC}}(I - E)$, we have

$$F_{\bullet}^{-1} = (I - E)^{-1} F_{\bullet\,\text{KFAC}}^{-1} = (I + E + \ldots + E^{q-1} + \ldots) F_{\bullet\,\text{KFAC}}^{-1}. \tag{3.34}$$

The formal series expansion in the last equality rests upon the intuition that E measures the approximation quality of $F_{\bullet}^{-1}$ by $F_{\bullet\,\text{KFAC}}^{-1}$ and therefore can be assumed to be small. Multiplying both sides by the residual r_{KFAC} and stopping the expansion at order $q - 1 \geq 0$, we obtain the approximation

$$(I + E + \ldots + E^{q-1}) F_{\bullet\,\text{KFAC}}^{-1} r_{\text{KFAC}} \tag{3.35}$$

for the error $F_{\bullet}^{-1} r_{\text{KFAC}} = \zeta - \zeta_{\text{FAC}}$, which is also the ideal correction term. As earlier, we impose that this approximate correction vector (3.35) must be contained in the coarse space R_0^T. This suggests to extract the components in layer i of the vectors

$$\left\{ F_{\bullet\,\text{KFAC}}^{-1} r_{\text{KFAC}},\ E F_{\bullet\,\text{KFAC}}^{-1} r_{\text{KFAC}},\ \ldots,\ E^{q-1} F_{\bullet\,\text{KFAC}}^{-1} r_{\text{KFAC}} \right\}$$

and assign them to the columns of V_i. In view of (3.33), the space spanned by the above vectors is the same as the one spanned by

$$\left\{ F_{\bullet\,\text{KFAC}}^{-1} r_{\text{KFAC}},\ (F_{\bullet\,\text{KFAC}}^{-1} F_{\bullet}) F_{\bullet\,\text{KFAC}}^{-1} r_{\text{KFAC}},\ \ldots,\ (F_{\bullet\,\text{KFAC}}^{-1} F_{\bullet})^{q-1} F_{\bullet\,\text{KFAC}}^{-1} r_{\text{KFAC}} \right\}.$$

Consequently, we can take

$$m_1 = \ldots = m_\ell = q, \qquad m = q\ell, \tag{3.36}$$

and

$$V_i = [w_1[i],\ w_2[i],\ \ldots,\ w_q[i]] \in \mathbb{R}^{p_i \times m_i} \tag{3.37}$$

where

$$w_1 = F_{\bullet\,\mathrm{KFAC}}^{-1} r_{\mathrm{KFAC}} \in \mathbb{R}^p, \qquad w_{j+1} = F_{\bullet\,\mathrm{KFAC}}^{-1} F_\bullet\, w_j \in \mathbb{R}^p, \tag{3.38}$$

for $1 \le j \le q-1$. The case $q = 1$ degenerates to the residuals coarse space. From (3.38), we see that upgrading to the next order is done by multiplying by $F_\bullet$, an operation that mixes the layers.

For the practical implementation of these coarse spaces, we need efficient computational methods for two essential building blocks, namely, the matrix-vector product $F_\bullet u$ and the coarse operator F_{coarse}. These will be described in appendix §A.

3.4. Pseudo-code for two-level KFAC methods

Algorithm 1 summarizes the steps for setting up a two-level KFAC method.

Algorithm 1: High-level pseudo-code for a two-level KFAC method

Input: θ_0 (Initial point), $k_{\max}$ (maximum number of iterations), and α (learning rate)

Output: $\theta_{k_{\max}}$

for $k = 0, 1, \ldots, k_{\max} - 1$ **do**

- Compute an estimate $\nabla_\theta h(\mathcal{S}_k, \theta_k)$ of the gradient on a mini-batch $\mathcal{S}_k$ randomly sampled from the training data;
- Compute $\theta_{\mathrm{KFAC}} = F_{\bullet\,\mathrm{KFAC}}^{-1} \nabla_\theta h(\mathcal{S}_k, \theta_k)$;
- Choose a coarse space R_0^T and compute the associated coarse correction $R_0\beta^* = R_0^T (F_{\mathrm{coarse}})^{-1} R_0 r_{\mathrm{KFAC}}$;
- Compute $\theta_{\text{KFAC-2L}} = \theta_{\mathrm{KFAC}} + R_0\beta^*$;
- Update $\theta_{k+1} = \theta_k - \alpha\theta_{\text{KFAC-2L}}$;

end

4. Numerical results

In this section, we compare the new two-level KFAC methods designed in §3 with the standard KFAC [18, 31] from the standpoint of convergence speed. For a thorough analysis, we also include the two-level KFAC version of Tselepidis et al. [50] and baseline optimizers (ADAM and SGD).

We run a series of experiments to investigate the optimization performance of deep auto-encoders, CNNs, and deep linear networks. Since our primary focus is on convergence speed rather than generalization, we shall only be concerned with the ability of optimizers to minimize the objective function. In particular, we report only training losses for each optimizer. To equally treat all methods, we adopt the following rules.

We perform a Grid Search and select hyper-parameters that give the best reduction to the training loss. Learning rates for all methods and damping parameters for KFAC and two-level KFAC methods are searched in the range

$$\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}, 10^{4}\}.$$

For each optimizer, we apply the Early Stopping technique with patience of 10 epochs i.e. we stop training the network when there is no decrease in the training loss during 10 consecutive epochs). We also include weight decay with a coefficient of 10^{-3} for all optimizers.

All experiments presented in this work are performed with PyTorch framework [39] on a supercomputer with Nvidia Ampere A100 GPU and AMD Milan@2.45GHz CPU. For ease of reading, the following table explains all abbreviations of two-level KFAC methods that we will use in the figure legends.

Table 1. Name abbreviations of two-level KFAC optimizers.

Optimizer	Name abbreviation
Two-level KFAC with Nicolaides coarse space	NICO
Two-level KFAC with spectral coarse space	SPECTRAL
Two-level KFAC with residuals coarse space	RESIDU
Two-level KFAC with Krylov Nicolaides coarse space	KRY-NICO
Two-level KFAC with Krylov residuals coarse space	KRY-RESIDU
Two-level KFAC of Tselepidis et al. [50]	PREVIOUS

4.1. Deep auto-encoder problems

The first set of experimental tests performed is the optimization of three different deep auto-encoders, each trained with a different dataset (CURVES, MNIST, and FACES). Note that due to the difficulty of optimizing the underlying networks, these three auto-encoder problems are commonly used as benchmarks for evaluating new optimization methods in the deep learning community [7, 22, 27, 31, 48]. For each problem, we train the network with three different batch sizes.

Figure 2 shows the obtained results. The first observation is that, as expected, natural gradient-based methods (KFAC and two-level KFAC methods) outperform baseline optimizers (ADAM and SGD). The second and most important observation is that, for each of the three problems, regardless of the batch size, the training curve of KFAC and those of all two-level KFAC methods (the one of Tselepidis et al. [50] and those proposed in this work) are overlaid, which means that taking into account the extra-diagonal terms of the Fisher matrix through two-level decomposition methods does not improve the convergence speed of KFAC method. This second observation is quite puzzling, since theoretically two-level methods are supposed to offer a better approximation to the exact natural gradient than KFAC does and therefore should at least slightly outperform KFAC in terms of optimization performance. Note that we repeated these experiments on three different random seeds and obtained very similar results.

These surprising results are in line with the findings of Benzing [6], according to which KFAC outperforms the exact natural gradient in terms of optimization performance. This suggests that extra-diagonal blocks of the FIM do not contribute to improving the optimization performance, and sometimes even affect it negatively.

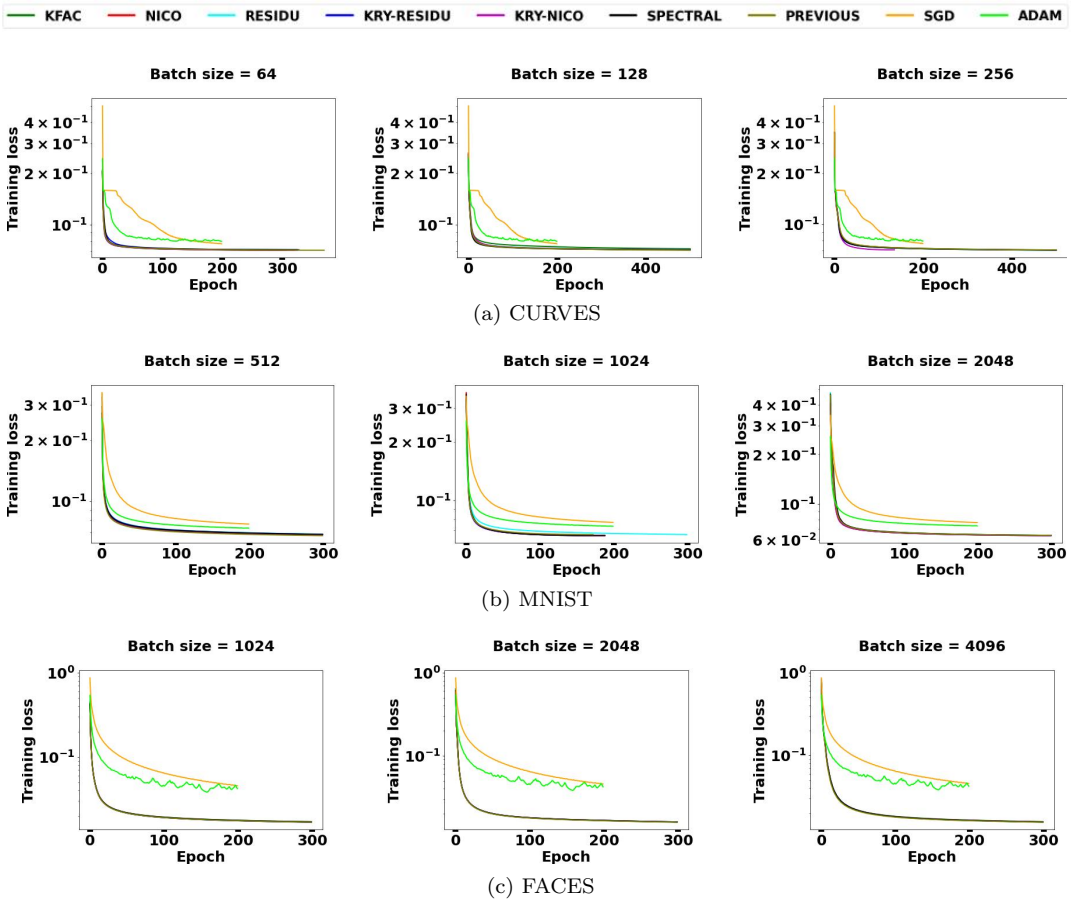


Figure 2. Comparison of KFAC against two-level KFAC methods on the three deep auto-encoder problems (CURVES **top** row, MNIST **middle** row and FACES **bottom** row). Three different batch sizes are considered for each problem (each column corresponds to a different batch size).

4.2. Convolution neural networks

The second set of experiments concerns the optimization of three different CNNs namely Resnet 18 [19], Cuda-convnet and Resnet 34 [19]. We consider in particular Cuda-convnet which is the architecture used to evaluate the original KFAC method in [18]. It must be mentioned that it contains 3 convolution layers and one MLP layer. We train Cuda-convnet on CIFAR10 dataset [23] with a batch size equal to 256, and Resnet 18 on CIFAR100 [23] with a batch size equal to 128. Finally, we train Resnet 34 on the SVHN dataset [34] with a batch size equal to 512.

For these CNNs (see Figure 3), we arrive at quite similar observations and conclusions to those we mention for deep auto-encoder problems. In particular, like in [50], when considering CNNs, we do not observe any significant gain in the convergence speed of KFAC when we enrich it with cross-layer information through two-level decomposition methods. Once again, these results corroborate the claims of Benzing [6] and suggest that we do not need to take into account the extra diagonal blocks of the FIM.

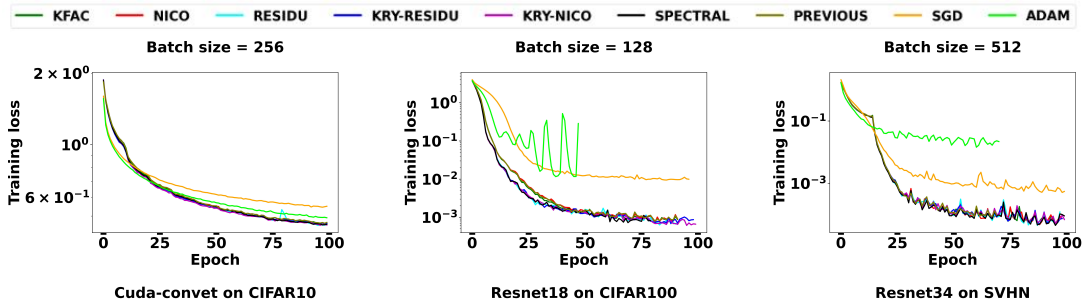
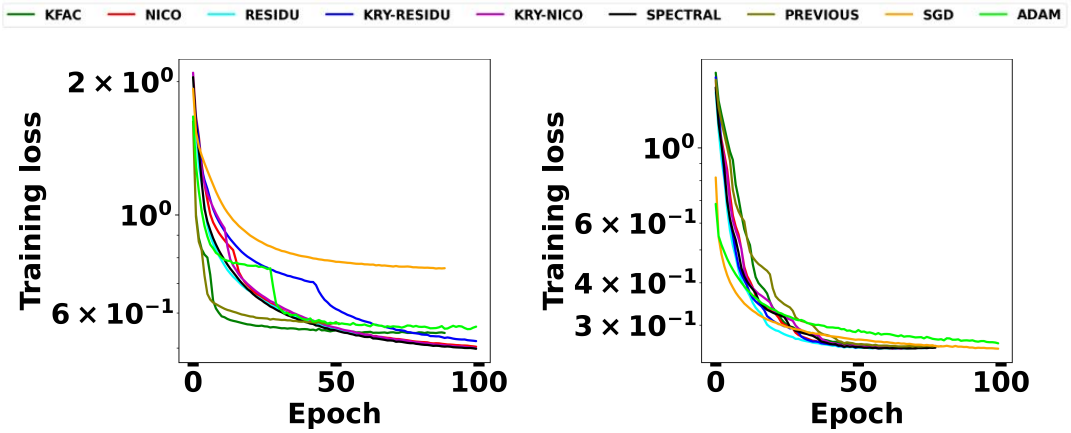


Figure 3. Optimization performance evaluation of KFAC and two-level KFAC methods on three different CNNs.

4.3. Deep linear networks

The last experiments concern relatively simple optimization problems: linear networks optimization. We consider two deep linear networks. These tests are motivated by the results obtained by Tselepidis et al. [50] for their two-level method. Indeed, for an extremely simple linear network with 64 layers (each layer contains 10 neurons and a batch normalization layer) trained with randomly generated ten-size input vectors, they outperform KFAC in terms of optimization performance. Here, we first consider the same architecture but train the network on the Fashion MNIST dataset [52](since we could not use the same dataset). Then, we consider another linear network that contains 14 layers with batch normalization, with this time much larger layers. More precisely we consider the following architecture: $784-1000-900-800-700-600-500-400-300-200-100-50-20-10$. We train this second network on the MNIST dataset. Both networks are trained with a batch size of 512.

Figure 4 shows the training curves obtained in both cases. Here we observe like in [50] an improvement in the optimization performance of two-level optimizers over KFAC. However, this gain remains too small and only concerns simple linear networks that are not used for practical applications. We therefore do not encourage enriching KFAC with two-level methods that require additional computational costs.



(a) Deep linear network of 64 layers, each layer containing 10 neurons, trained on Fashion MNIST.

(b) Deep linear network of 14 larger layers, trained on MNIST.

Figure 4. Optimization performance evaluation of KFAC and Tow-level KFAC optimizers on two different deep linear networks.

4.4. Verification of error reduction for linear systems

In the above experiments, two-level methods do not seem to outperform KFAC in terms of optimization performance. We thus wish to check that at each descent iteration, the negative increment $\zeta_{\text{KFAC-2L}}$ obtained with the coarse correction is indeed closer to that of the regularized natural gradient one ζ than the negative increment ζ_{KFAC} corresponding to the original KFAC. In other words, we want to make sure that inequality (3.19) holds numerically.

For $\beta \in \mathbb{R}^m$, let

$$\mathfrak{E}(\beta) = \|\zeta_{\text{KFAC}} + R_0^T\beta - \zeta\|_{F_\bullet}^2 \tag{4.1}$$

be the function to be minimized at fixed R_0^T in the construction (3.9)–(3.10), where it is recalled that

$$\zeta = F_\bullet^{-1}\nabla_\theta h, \qquad \zeta_{\text{KFAC}} = F_{\bullet\,\text{KFAC}}^{-1}\nabla_\theta h.$$

Note that

$$\mathfrak{E}(0) = \|\zeta_{\text{KFAC}} - \zeta\|_{F_\bullet}^2 \tag{4.2}$$

is the squared $F_\bullet$-distance between the KFAC increment and that natural gradient one, regardless of R_0^T. Meanwhile, if β^* is taken to be the optimal value (3.11), then

$$\mathfrak{E}(\beta^*) = \|\zeta_{\text{KFAC-2L}} - \zeta\|_{F_\bullet}^2. \tag{4.3}$$

To see whether (3.19) is satisfied, the idea is to compute the difference $\mathfrak{E}(\beta^*) - \mathfrak{E}(0)$ and check that it is negative. The goal of the game, however, is to avoid using the unknown natural gradient solution ζ. Owing to the identity $\|a\|^2 - \|b\|^2 = (a-b, a+b)$ for the $F_\bullet$-dot product, this difference can be transformed into

$$\begin{aligned}
\mathfrak{E}(\beta^*) - \mathfrak{E}(0) &= \|\zeta_{\text{KFAC-2L}} - \zeta\|_{F_\bullet}^2 - \|\zeta_{\text{KFAC}} - \zeta\|_{F_\bullet}^2 \\
&= (\zeta_{\text{KFAC-2L}} - \zeta_{\text{KFAC}},\, \zeta_{\text{KFAC-2L}} + \zeta_{\text{KFAC}} - 2\zeta)_{F_\bullet} \\
&= \|\zeta_{\text{KFAC-2L}} - \zeta_{\text{KFAC}}\|_{F_\bullet}^2 + 2(\zeta_{\text{KFAC-2L}} - \zeta_{\text{KFAC}},\, \zeta_{\text{KFAC}} - \zeta)_{F_\bullet} \\
&= \|R_0^T\beta^*\|_{F_\bullet}^2 + 2(R_0^T\beta^*,\, \zeta_{\text{KFAC}} - \zeta)_{F_\bullet} \\
&= \big\langle F_\bullet R_0^T\beta^*, R_0^T\beta^*\big\rangle + 2\big\langle R_0^T\beta^*,\, F_\bullet(\zeta_{\text{KFAC}} - \zeta)\big\rangle,
\end{aligned} \tag{4.4}$$

where $\langle\cdot,\cdot\rangle$ denotes the Euclidean dot product. But

$$F_\bullet(\zeta_{\text{KFAC}} - \zeta) = F_\bullet\zeta_{\text{KFAC}} - \nabla_\theta h = -r_{\text{KFAC}} \tag{4.5}$$

is the opposite of the residual (3.13), which can be computed without knowing ζ. Finally, the desired difference can also be computed as

$$\mathfrak{E}(\beta^*) - \mathfrak{E}(0) = \big\langle R_0 F_\bullet R_0^T\beta^*,\, \beta^*\big\rangle - 2\big\langle R_0^T\beta^*,\, r_{\text{KFAC}}\big\rangle. \tag{4.6}$$

For the two-level method of Tselepidis-Kohler-Orvieto [50], the correction reads

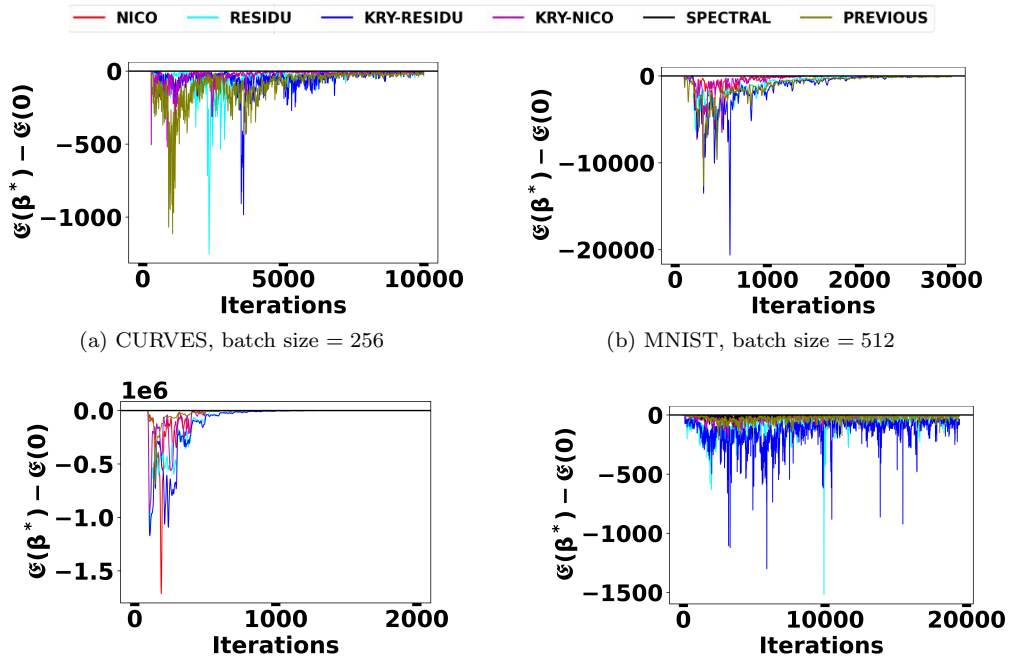
$$\zeta_{\text{TKO}} = \zeta_{\text{KFAC}} + R_0^T \beta^*_{\text{TKO}} \tag{4.7a}$$

with

$$\beta^*_{\text{TKO}} = (R_0 F_\bullet R_0^T)^{-1} R_0 \nabla_\theta h \tag{4.7b}$$

instead of β^*, the KFAC-2L value (3.11). The difference $\mathfrak{E}(\beta^*_{TKO}) - \mathfrak{E}(0)$ is then given by a formula similar to (4.6) in which β^* is simply replaced by β^*_{TKO}.

We compute the error $\mathfrak{E}(\beta^*) - \mathfrak{E}(0)$ associated to various two-level methods in the experiments conducted above. More specifically, we do it for the three deep auto-encoder problems and also for a CNN (cuda-convnet). The results obtained are shown in Figure 5. The observation is that all two-level methods proposed in this work as well as the TKO two-level method [50] have the gap have negative gaps $\mathfrak{E}(\beta^*) - \mathfrak{E}(0)$ throughout the optimization process. This implies that two-level methods solve the linear system (3.2) more accurately than KFAC does. It also means that the approximate natural gradients obtained with Two-level methods are closer to the exact natural gradient than the one obtained with KFAC is.



(a) CURVES, batch size = 256

(b) MNIST, batch size = 512

(c) FACES, batch size=1024

(d) Cuda-convnet, batch size=256

Figure 5. Evolution of $\mathfrak{E}(\beta^*) - \mathfrak{E}(0)$ during training for each of the two-level methods considered. All methods proposed in this work as well as the TKO two-level method [50] have the gap $\mathfrak{E}(\beta^*) - \mathfrak{E}(0)$ negative throughout the training process.

5. Conclusion and discussion

In this study, we sought to improve KFAC by incorporating extra-diagonal blocks using two-level decomposition methods. To this end, we proposed several two-level KFAC methods, with a careful design of coarse corrections. Through several experiments, we came to the conclusion that two-level KFAC methods do not generally outperform the original KFAC method in terms of optimization performance of the objective function. This implies that taking into account the interactions between the layers is not useful for the optimization process.

We also numerically verfied that, at the level of the linear system of each iteration, the increment provided by any two-level method is much closer to the exact natural gradient solution than that obtained with KFAC, in a norm naturally associated with the FIM. This reveals that closeness to the exact natural gradient does not necessarily results in a more efficient algorithm. This observation is consistent with Benzing's previous claim [6] that KFAC outperforms the exact natural gradient in terms of optimization performance.

The fact that incorporating extra-diagonal blocks does not improve or often even hurts the optimization performance of the initial diagonal approximation could be explained by a negative interaction between different layers of the neural network. This suggests ignoring extra-diagonal blocks of the FIM and keeping the block-diagonal approximation, and if one seeks to improve the block-diagonal approximation, one should focus on diagonal blocks as attempted in many recent works [7, 13, 15, 22].

It is worth pointing out that the conclusion of Tpselepedis et al. [50] on the performance of their proposed two-level method seems a little hasty. Indeed, the authors only ran two different experiments: the optimization of a CNN and a simple linear network. For the CNN network, they did not observe any improvement. For the linear network they obtain some improvement in the optimization performance. Their conclusion is therefore based on this single observation.

Finally, we recall that as is the case for almost every previous work related to natural gradient and KFAC methods [6, 7, 18, 31], the one undertaken in this paper is limited to the optimization performance of the objective function. It will thus be interesting to investigate the generalization capacity of these methods (including KFAC). Since the study of generalization requires a different experimental framework [6, 54, 55], we leave it as a prospect. Our findings and those of Benzing [6] imply that it can be interesting to explore the use of even simpler approximations of the FIM. More precisely, after approximating the FIM by a block diagonal matrix as in KFAC, one can further approximate each full diagonal block by an inner sub-blocks diagonal matrix (see for instance [4]). This approach will save computational time and probably maintain the same level of optimization performance.

References

[1] S.I. Amari, *Natural gradient works efficiently in learning*, Neur. Comput. 10 (1998), pp. 251–276.

[2] S.I. Amari and H. Nagaoka, *Methods of Information Geometry*, Translations of Mathematical Monographs Vol. 191, American Mathematical Society, Providence, Rhode Island, 2000.

[3] J. Ba, R. Grosse, and J. Martens, *Distributed Second-Order Optimization using Kronecker-Factored Approximations*, in *5th International Conference on Learning Representations,*

Conference Track Proceedings, 24–26 Apr, Toulon, France. 2017. Available at https://openreview.net/forum?id=SkkTMpjex.

[4] A. Bahamou, D. Goldfarb, and Y. Ren, *A mini-block fisher method for deep neural networks* (2022). Available at https://arxiv.org/abs/2202.04124.

[5] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, in *3rd International Conference on Learning Representations*, Y. Bengio and Y. LeCun, eds., May, San Diego, California. 2015. Available at https://arxiv.org/abs/1409.0473.

[6] F. Benzing, *Gradient Descent on Neurons and its Link to Approximate Second-Order Optimization*, in *Proceedings of the 39th International Conference on Machine Learning*, Vol. 162, Baltimore, Maryland. 2022. Available at https://proceedings.mlr.press/v162/benzing22a/benzing22a.pdf.

[7] A. Botev, H. Ritter, and D. Barber, *Practical Gauss-Newton Optimisation for Deep Learning*, in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y.W. Teh, eds., Proceedings of Machine Learning Research Vol. 70, 06–11 Aug, Sydney, Australia. 2017, pp. 557–565.

[8] C.G. Broyden, *The convergence of a class of double-rank minimization algorithms 1. General considerations*, IMA J. Appl. Math. 6 (1970), pp. 76–90.

[9] K. Chellapilla, S. Puri, and P.Y. Simard, *High Performance Convolutional Neural Networks for Document Processing*, in *10th International Workshop on Frontiers in Handwriting Recognition*, G. Lorette, H. Bunke, and L. Schomaker, eds., 23–26 Oct, La Baule, France. 2006. Available at https://hal.inria.fr/inria-00112631.

[10] V. Dolean, P. Jolivet, and F. Nataf, *An Introduction to Domain Decomposition Methods: Algorithms, Theory, and Parallel Implementation*, Society for Industrial and Applied Mathematics, Philadelphia, 2015, Available at https://www.ljll.math.upmc.fr/nataf/OT144DoleanJolivetNataf_full.pdf.

[11] J. Duchi, E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, J. Mach. Learn. Res. 12 (2011), pp. 2121–2159. Available at https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf.

[12] R. Fletcher, *A new approach to variable metric algorithms*, Comput. J. 13 (1970), pp. 317–322.

[13] K.X. Gao, X.L. Liu, Z.H. Huang, M. Wang, Z. Wang, D. Xu, and F. Yu, *A trace-restricted Kronecker-factored approximation to natural gradient* (2020). Available at https://arxiv.org/abs/2011.10741.

[14] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y.N. Dauphin, *Convolutional sequence to sequence learning*, in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y.W. Teh, eds., Vol. 70, 06–11 Aug, Sydney, Australia. 2017, pp. 1243–1252. Available at https://proceedings.mlr.press/v70/gehring17a.html.

[15] T. George, C. Laurent, X. Bouthillier, N. Ballas, and P. Vincent, *Fast approximate natural gradient descent in a Kronecker factored eigenbasis*, in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds., Curran Associates, Inc., Montréal, Canada, 2018, pp. 9550–9560. Available at http://papers.nips.cc/paper/8164-fast-approximate-natural-gradient-descent-in-a-kronecker-factored-eigenbasis.pdf.

[16] D. Goldfarb, *A family of variable-metric methods derived by variational means*, Math. Comp. 24 (1970), pp. 23–26.

[17] D. Goldfarb, Y. Ren, and A. Bahamou, *Practical quasi-Newton methods for training deep neural networks*, arXiv:2006.08877 (2020). Available at https://arxiv.org/pdf/2006.08877.pdf.

[18] R. Grosse and J. Martens, *A Kronecker-factored approximate Fisher matrix for convolution layers*, in *Proceedings of the 33rd International Conference on Machine Learning*, M.F. Balcan and K.Q. Weinberger, eds., Vol. 48, 19–24 Jun, New York. 2016, pp. 573–582. Available at http://proceedings.mlr.press/v48/grosse16.html.

[19] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*, in

Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition, 27–30 Jun, Las Vegas, Nevada. 2016, pp. 770–778.

[20] T. Heskes, *On "natural" learning and pruning in multilayered perceptrons*, Neur. Comput. 12 (2000), pp. 881–901.

[21] D.P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, in *Proceedings of the 3rd International Conference on Learning Representations*, Y. Bengio and Y. LeCun, eds., 7–9 May, San Diego, California. 2015. Available at http://arxiv.org/abs/1412.6980.

[22] A. Koroko, A. Anciaux-Sedrakian, I. Ben Gharbia, V. Garès, M. Haddou, and Q.H. Tran, *Efficient approximations of the Fisher matrix in neural networks using Kronecker product singular value decomposition*, arXiv:2201.10285 (2022). Available at https://arxiv.org/abs/2201.10285.

[23] A. Krizhevsky, *Learning multiple layers of features from tiny images*, Tech. Rep., University of Toronto, Toronto, Ontario, 2009. Available at https://www.cs.toronto.edu/~kriz/cifar.html.

[24] A. Krizhevsky, I. Sutskever, and G.E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, in *Advances in Neural Information Processing System 25*, F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, eds., 03–08 Dec, Lake Tahoe, California. Curran Associates, Inc., 2012, pp. 1097–1105. Available at https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[25] N. Le Roux, P.A. Manzagol, and Y. Bengio, *Topmoumoute online natural gradient algorithm*, in *Proceedings of the 20th International Conference on Neural Information Processing Systems*, J.C. Platt, D. Koller, Y. Singer, and S.T. Roweis, eds., 03–06 Dec, Vancouver, Canada. 2007, pp. 849–856.

[26] D.C. Liu and J. Nocedal, *On the limited memory BFGS method for large scale optimization*, Math. Prog. 45 (1989), pp. 503–528.

[27] J. Martens, *Deep learning via Hessian-free optimization*, in *Proceedings of the 27th International Conference on Machine Learning*, Vol. 27, 21–24 Jun, Haifa, Israel. 2010, pp. 735–742.

[28] J. Martens, *New insights and perspectives on the natural gradient method*, arXiv:1412.1193 (2014). Available at https://arxiv.org/abs/1412.1193.

[29] J. Martens, *Second-order optimization for neural networks*, Ph.D. diss., University of Toronto, Ontario, Canada, 2016. Available at http://hdl.handle.net/1807/71732.

[30] J. Martens, J. Ba, and M. Johnson, *Kronecker-factored curvature approximations for recurrent neural networks*, in *Proceedings of the 6th International Conference on Learning Representations*, 30 Apr–3 May, Vancouver, Canada. 2018. Available at https://openreview.net/forum?id=HyMTkQZAb.

[31] J. Martens and R. Grosse, *Optimizing neural networks with Kronecker-factored approximate curvature*, in *Proceedings of the 32nd International Conference on Machine Learning*, Vol. 37, 06–11 Jul, Lille, France. 2015, pp. 2408–2417. Available at http://proceedings.mlr.press/v37/martens15.html.

[32] F. Nataf, H. Xiang, V. Dolean, and N. Spillane, *A coarse space construction based on local Dirichlet-to-Neumann maps*, SIAM J. Sci. Comput. 33 (2011), pp. 1623–1642.

[33] Y.E. Nesterov, *A method for solving the convex programming problem with convergence rate* $O(1/k^2)$, Dokl. Akad. Nauk SSSR 269 (1983), pp. 543–547. Available at http://mi.mathnet.ru/dan4600.

[34] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng, *Reading Digits in Natural Images with Unsupervised Feature Learning*, in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, December, Granada. 2011. Available at http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf.

[35] R.A. Nicolaides, *Deflation of conjugate gradients with applications to boundary value problems*, SIAM J. Numer. Anal. 24 (1987), pp. 355–365.

[36] Y. Ollivier, *Riemannian metrics for neural networks I: feedforward networks*, Inform. Infer. 4 (2015), pp. 108–153.

[37] 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 2019 IEEE Conference on Computer Vision and Pattern Recognition*, 15–20 Jun, Long Beach, California. 2019, pp. 12359–12367. Available at http://openaccess.thecvf.com/content_CVPR_2019/html/Osawa_Large-Scale_Distributed_Second-Order_Optimization_Using_Kronecker-Factored_Approximate_Curvature_for_Deep_CVPR_2019_paper.html.

[38] R. Pascanu and Y. Bengio, *Revisiting natural gradient for deep networks*, arXiv preprint arXiv:1301.3584 (2013). Available at https://arxiv.org/abs/1301.3584v4.

[39] 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 *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, eds., 08–14 Dec, Vancouver, Canada. Curran Associates, Inc., 2019, pp. 8026–8037. Available at https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

[40] B. Polyak, *Some methods of speeding up the convergence of iteration methods*, USSR Comput. Math. Math. Phys. 4 (1964), pp. 1–17.

[41] D. Povey, X. Zhang, and S. Khudanpur, *Parallel training of DNNs with natural gradient and parameter averaging*, arXiv:1410.7455 (2014). Available at https://arxiv.org/abs/1410.7455.

[42] H. Ritter, A. Botev, and D. Barber, *A Scalable Laplace Approximation for Neural Networks*, in *6th International Conference on Learning Representations*, ICLR Conference Track Proceedings Vol. 6, Vancouver, Canada. 2018. Available at https://openreview.net/forum?id=Skdvd2xAZ.

[43] H. Robbins and S. Monro, *A stochastic approximation method*, Ann. Math. Statist. 22 (1951), pp. 400–407. Available at https://www.jstor.org/stable/2236626.

[44] H. Sak, A. Senior, and F. Beaufays, *Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling*, in *15th Annual Conference of the International Speech Communication Association. Celebrating the Diversity of Spoken Languages*, H. Li and P. Ching, eds., 14–18 Sep, Singapore. Curran Associates, Inc., 2014, pp. 338–342. Available at https://arxiv.org/abs/1402.1128.

[45] N.N. Schraudolph, *Fast curvature matrix-vector products for second-order gradient descent*, Neur. Comput. 14 (2002), pp. 1723–1738.

[46] T. Sercu, C. Puhrsch, B. Kingsbury, and Y. LeCun, *Very deep multilingual convolutional neural networks for LVCSR*, in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing*, 20–25 Mar, Shanghai, China. 2016, pp. 4955–4959.

[47] D.F. Shanno, *Conditioning of quasi-Newton methods for function minimization*, Math. Comp. 24 (1970), pp. 647–656.

[48] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, *On the importance of initialization and momentum in deep learning*, in *Proceedings of the 30th International Conference on Machine Learning*, S. Dasgupta and D. McAllester, eds., Proceedings of Machine Learning Research Vol. 28, 17–19 Jun, Atlanta, Georgia. 2013, pp. 1139–1147. Available at https://proceedings.mlr.press/v28/sutskever13.html.

[49] T. Tieleman and G. Hinton, *Lecture 6.5 RMSProp: Divide the gradient by a running average of its recent magnitude*, COURSERA: Neural Networks for Machine Learning 4 (2012), pp. 26–31.

[50] N. Tselepidis, J. Kohler, and A. Orvieto, *Two-Level K-FAC Preconditioning for Deep Learning*, in *12th Annual Workshop on Optimization for Machine Learning*, online. 2020. Available at https://arxiv.org/abs/2011.00573.

[51] Y. Wu, E. Mansimov, R.B. Grosse, S. Liao, and J. Ba, *Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation*, in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, I. Guyon, U.V.

Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., 04–09 Dec, Long Beach, California. Curran Associates, Inc., 2017, pp. 5285–5294. Available at https://proceedings.neurips.cc/paper/2017/file/361440528766bbaaaa1901845cf4152b-Paper.pdf.

[52] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms* (2017). Available at https://arxiv.org/abs/1708.07747.

[53] G. Zhang, S. Sun, D. Duvenaud, and R. Grosse, *Noisy Natural Gradient as Variational Inference*, in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, eds., Proceedings of Machine Learning Research Vol. 80, 10–15 Jul. 2018, pp. 9308–9321. Available at https://proceedings.mlr.press/v80/zhang18l.html.

[54] G. Zhang, C. Wang, B. Xu, and R. Grosse, *Three mechanisms of weight decay regularization* (2018). Available at https://arxiv.org/abs/1810.12281.

[55] J. Zhang, *Gradient descent based optimization algorithms for deep learning models training* (2019). Available at https://arxiv.org/abs/1903.03614.

Appendix A. Efficient computation of $F_\bullet u$ and F_{coarse}

A.1. Notations

We consider the same network architectures (MLP and CNN) and notations introduced in §2. For two matrices A et B of same sizes, $A \odot B$ denotes the Hadamard (element-wise) product of A and B. We will also write $\langle u, v\rangle$ for the inner (dot) product between vectors u and v. We recall that "vec" is the operator that turns a matrix into a vector by stacking its columns together and "MAT," the converse of "vec," turns a vector into a matrix.

For a big vector $u \in \mathbb{R}^p$, where p is the number of parameters contained in θ, the notation $u[i] \in \mathbb{R}^{p_i}$ stands for the part of u corresponding to layer i, whose number of parameters is p_i. For example, if $u = \theta = [\text{vec}(W_1)^T, \text{vec}(W_2)^T, \dots, \text{vec}(W_\ell)^T]^T$, then $u[i] = \text{vec}(W_i)$ for all $i \in \{1, \dots, \ell\}$.

A.2. Computation of $F_\bullet u$

In view of the regularization (2.17), it is plain that

$$F_\bullet u = Fu + \lambda u. \tag{A1}$$

Since the regularization term does not cause any trouble, we only have to deal with Fu. It is notoriously knwon that the matrix-vector product involving the Fisher matrix can be carried out without explicitly forming F thanks to an algorithm by Schraudolph [45]. However, this approach requires to perform additional forward/backward passes. Here, we present an efficient way to evaluate Fu by re-using the quantities computed during the traditional backward/forward pass.

Suppose that we have a mini-batch $\mathcal{B} = \{(x_1, y_1), \dots, (x_B, y_B)\}$ where targets y_i's are sampled from the model predictive distribution $P_{y|x}(\theta)$. Then, F is computed using a Monte Carlo estimation, i.e.,

$$F = [\mathcal{D}\theta(\mathcal{D}\theta)^T] \approx \frac{1}{B}\sum_{b=1}^{B}(\mathcal{D}\theta)^{(b)}((\mathcal{D}\theta)^{(b)})^T = \frac{1}{B}JJ^T, \tag{A2}$$

where $J \in \mathbb{R}^{p \times B}$ is the matrix whose b-th column is $(\mathcal{D}\theta)^{(b)}$. Thus,

$$Fu = \frac{1}{B} J J^T u. \tag{A3}$$

The computation of Fu can therefore be divided into two steps: (1) matrix-vector product $v = J^T u$; (2) matrix-vector product $\frac{1}{B} Jv$.

Step 1: multiplying by J^T. Since $J^T \in \mathbb{R}^{B \times p}$ and $u \in \mathbb{R}^p$, we have $v = J^T u \in \mathbb{R}^B$. For all $b \in \{1, \dots, B\}$, the b-th entry v_b of v is none other than the dot product between the b-th column $(\mathcal{D}\theta)^{(b)}$ of J and u. This dot product can be split into layer-wise dot products and then summed up together. Formally, we have

$$v_b = \left\langle (\mathcal{D}\theta)^{(b)}, u \right\rangle = \sum_{i=1}^{\ell} \left\langle \mathrm{vec}((\mathcal{D}W_i)^{(b)}), u[i] \right\rangle. \tag{A4}$$

We now distinguish two cases according to the type of layer i.

(1) If layer i is an MLP, we have $\mathcal{D}W_i = g_i \bar{a}_{i-1}^T$ and then

$$\begin{aligned} \left\langle \mathrm{vec}((\mathcal{D}W_i)^{(b)}), u_{[i]} \right\rangle &= \left\langle \mathrm{vec}(g_i^{(b)} (\bar{a}_{i-1}^{(b)})^T), u_{[i]} \right\rangle \\ &= \left\langle \bar{a}_{i-1}^{(b)} \otimes g_i^{(b)}, u_{[i]} \right\rangle \\ &= \left((\bar{a}_{i-1}^{(b)})^T \otimes (g_i^{(b)})^T \right) u_{[i]} \\ &= (g_i^{(b)})^T \mathrm{MAT}(u_{[i]}) \bar{a}_{i-1}^{(b)}. \end{aligned} \tag{A5}$$

(2) If layer i is a CNN layer, we have $\mathcal{D}W_i = \sum_{t=1}^{T_i} g_{i,t} \bar{a}_{i-1,t}^T$ and therefore

$$\begin{aligned} \left\langle \mathrm{vec}((\mathcal{D}W_i)^{(b)}), u_{[i]} \right\rangle &= \left\langle \mathrm{vec}\left(\sum_{t=1}^{T_i} g_{i,t}^{(b)} (\bar{a}_{i-1,t}^{(b)})^T \right), u_{[i]} \right\rangle \\ &= \left\langle \sum_{t=1}^{T_i} \mathrm{vec}(g_{i,t}^{(b)} (\bar{a}_{i-1,t}^{(b)})^T), u_{[i]} \right\rangle \\ &= \sum_{t=1}^{T_i} (g_{i,t}^{(b)})^T \mathrm{MAT}(u_{[i]}) \bar{a}_{i-1,t}^{(b)}. \end{aligned} \tag{A6}$$

Step 2: multiplying by J. Here, we detail how to compute Jv, but one should not forget to multiply the result by the scaling factor $1/B$. Let $v \in \mathbb{R}^B$. Jv is a weighted sum of the columns of J with the weight coefficients corresponding to the entries of v. In other words,

$$Jv = \sum_{b=1}^{B} v_b (\mathcal{D}\theta)^{(b)}. \tag{A7}$$

For all $i \in \{1, \dots, \ell\}$, the part of Jv corresponding to layer i is a linear combination of those of columns of J corresponding to layer i, that is,

$$(Jv)[i] = \sum_{b=1}^{B} v_b \operatorname{vec}((\mathcal{D}W_i)^{(b)}). \tag{A8}$$

As in step 1, we have to distinguish two cases according the type of layer i.

(1) If layer i is an MLP, we have

$$(Jv)[i] = \sum_{b=1}^{B} v_b \operatorname{vec}(g_i^{(b)}(\bar{a}_{i-1}^{(b)})^T) = \operatorname{vec}\Big(\sum_{b=1}^{B} v_b g_i^{(b)}(\bar{a}_{i-1}^{(b)})^T\Big), \tag{A9}$$

and then

$$\operatorname{MAT}\big((Jv)_{[i]}\big) = \sum_{b=1}^{B} v_b g_i^{(b)}(\bar{a}_{i-1}^{(b)})^T = \big[(\mathbb{1}v^T) \odot \hat{\mathcal{G}}_i\big]\hat{\mathcal{A}}_{i-1}^T, \tag{A10}$$

where $\mathbb{1} \in \mathbb{R}^{d_i}$ is a vector of all one's, $\hat{\mathcal{A}}_{i-1} = (\bar{a}_{i-1}^{(1)}, \dots, \bar{a}_{i-1}^{(B)}) \in \mathbb{R}^{d_{i-1} \times B}$ is the matrix of activations, and $\hat{\mathcal{G}}_i = (g_i^{(1)}, \dots, g_i^{(B)}) \in \mathbb{R}^{d_i \times B}$ is the matrix containing pre-activations derivatives. Note that the last equality in (A10) is due to the fact that for two matrices $A = [A_1, \dots, A_m]$ and $B = [B_1, \dots, B_m]$, we have

$$AB^T = \sum_k A_k B_k^T. \tag{A11}$$

(2) If layer i is a CNN, we have

$$\begin{aligned}(Jv)_{[i]} &= \sum_{b=1}^{B} v_b \operatorname{vec}\Big(\sum_{t=1}^{T_i} g_{i,t}^{(b)}(\bar{a}_{i-1,t}^{(b)})^T\Big) \\ &= \sum_{b=1}^{B} \operatorname{vec}\Big(\sum_{t=1}^{T_i} v_b g_{i,t}^{(b)}(\bar{a}_{i-1,t}^{(b)})^T\Big),\end{aligned} \tag{A12}$$

and then

$$\begin{aligned}\operatorname{MAT}\big((Jv)_{[i]}\big) &= \sum_{b=1}^{B} v_b \sum_{t=1}^{T_i} g_{i,t}^{(b)}(\bar{a}_{i-1,t}^{(b)})^T \\ &= \big[(\mathbb{1}V^T) \odot \hat{\mathcal{G}}_i\big][\![\hat{\mathcal{A}}_{i-1}]\!]^T,\end{aligned} \tag{A13}$$

where here $\mathbb{1}$ is a vector of all one's of size c_i (the number of output channels),

whereas

$$V = (v_1, \dots v_1|, \dots, |v_B, \dots v_B)^T \tag{A14a}$$

$$[\![\hat{\mathcal{A}}_{i-1}]\!] = \big(\bar{a}^{(1)}_{i-1,1}, \dots, \bar{a}^{(1)}_{i-1,T_i} | \bar{a}^{(2)}_{i-1,1}, \dots, \bar{a}^{(2)}_{i-1,T_i} | \dots, | \bar{a}^{(B)}_{i-1,1}, \dots, \bar{a}^{(B)}_{i-1,T_i}\big), \tag{A14b}$$

$$\hat{\mathcal{G}}_i = \big(g^{(1)}_{i,1}, \dots, g^{(1)}_{i,T_i} | g^{(2)}_{i,1}, \dots, g^{(2)}_{i,T_i} | \dots, | g^{(B)}_{i,1}, \dots, g^{(B)}_{i,T_i}\big), \tag{A14c}$$

are respectively in $\mathbb{R}^{T_i B}$ (a duplicated version of v), $\mathbb{R}^{(c_{i-1}\Delta_i+1)\times T_i B}$ and $\mathbb{R}^{c_i \times T_i B}$.

A.3. Computation of F_{coarse}

From definition (3.12) of the coarse operator and the a priori form (3.21) of the coarse space, it follows that

$$[F_{\text{coarse}}]_{i,j} = V_i^T [F_\bullet]_{i,j} V_j \tag{A15}$$

for all $(i,j) \in \{1,\dots,\ell\} \times \{1,\dots,\ell\}$. In view of the regularization (2.17), the entry (A15) becomes

$$[F_{\text{coarse}}]_{i,j} = \begin{cases} V_i^T F_{i,j} V_j & \text{if } i \neq j, \\ V_i^T F_{i,i} V_i + \lambda V_i^T V_i & \text{if } i = j. \end{cases} \tag{A16}$$

Since the regularization term $\lambda V_i^T V_i$ does not cause any trouble, we only have to deal with the elementary products $v^T F_{i,j} w$, where $v \in \mathbb{R}^{p_i}$ (a column of V_i) and $w \in \mathbb{R}^{p_j}$ (a column of V_j). We reall that $F_{i,j} \in \mathbb{R}^{p_i \times p_j}$ is computed using a Monte Carlo estimation on a mini-batch, i.e.,

$$F_{i,j} \approx \frac{1}{B} \sum_{b=1}^{B} \text{vec}(\mathcal{D}W_i)^{(b)} (\text{vec}(\mathcal{D}W_j)^{(b)})^T = \frac{1}{B} J_i J_j^T, \tag{A17}$$

with

$$J_i = \big(\text{vec}(\mathcal{D}W_i)^{(1)}, \text{vec}(\mathcal{D}W_i)^{(2)}, \dots, \text{vec}(\mathcal{D}W_i)^{(B)}\big) \in \mathbb{R}^{p_i \times B}, \tag{A18a}$$

$$J_j = \big(\text{vec}(\mathcal{D}W_j)^{(1)}, \text{vec}(\mathcal{D}W_j)^{(2)}, \dots, \text{vec}(\mathcal{D}W_j)^{(B)}\big) \in \mathbb{R}^{p_j \times B}. \tag{A18b}$$

Then, $v^T F_{i,j} w$ is given by

$$v^T F_{i,j} w = \frac{1}{B} v^T J_i J_j^T w. \tag{A19}$$

The computation of $v^T F_{i,j} w$ can therefore be performed in three steps: (1) matrix-vector product $\mu = J_j^T w$; (2) matrix-vector product $\gamma = J_i \mu$; (3) dot product $\frac{1}{B}\langle v, \gamma \rangle$. The computation of $\mu = J_j^T v$ is done in the same way as $J^T u$ in the previous subsection §A.2. The only difference is that here we do not sum over layers. Likewise, the computation of $\gamma = J_i \mu$ is done exactly in the same way as $(Jv)[i]$ in §A.2. Finally, step 3 is the classical dot product and is straightforward.