Efficient Approximations of the Fisher Matrix in Neural Networks using Kronecker Product Singular Value Decomposition

Abdoulaye Koroko, Ani Anciaux-Sedrakian, Ibtihel Ben Gharbia, Valérie Garès, Mounir Haddou, Quang Huy Tran

EFFICIENT APPROXIMATIONS OF THE FISHER MATRIX IN NEURAL NETWORKS USING KRONECKER PRODUCT SINGULAR VALUE DECOMPOSITION

Abdoulaye Koroko[*,2], Ani Anciaux-Sedrakian[1], Ibtihel Ben Gharbia[1], Valérie Garès[3], Mounir Haddou[3], and Quang Huy Tran[1]

[1]IFP Energies nouvelles, 1 et 4 avenue de Bois Préau, 92852 Rueil-Malmaison Cedex, France
[2]Université Paris-Saclay, Gif-sur-Yvette, France
[3]Univ Rennes, INSA, CNRS, IRMAR - UMR 6625, F-35000 Rennes, France

ABSTRACT

Several studies have shown the ability of natural gradient descent to minimize the objective function more efficiently than ordinary gradient descent based methods. However, the bottleneck of this approach for training deep neural networks lies in the prohibitive cost of solving a large dense linear system corresponding to the Fisher Information Matrix (FIM) at each iteration. This has motivated various approximations of either the exact FIM or the empirical one. The most sophisticated of these is KFAC, which involves a Kronecker-factored block diagonal approximation of the FIM. With only a slight additional cost, a few improvements of KFAC from the standpoint of accuracy are proposed. The common feature of the four novel methods is that they rely on a direct minimization problem, the solution of which can be computed via the Kronecker product singular value decomposition technique. Experimental results on the three standard deep auto-encoder benchmarks showed that they provide more accurate approximations to the FIM. Furthermore, they outperform KFAC and state-of-the-art first-order methods in terms of optimization speed.

1 Introduction

In Deep Learning, the Stochastic Gradient Descent (SGD) method (Robbins & Monro, 1951) and its variants are currently the prevailing methods for training neural networks. To solve the problem

$$\underset{\theta\in\mathbb{R}^p}{\operatorname{argmin}}\, h(\theta) := \frac{1}{n}\sum_{t=1}^{n} L(y_t, f_\theta(x_t)),$$

where h denotes the empirical risk associated with the training data $\mathcal{T} = \{(x_1, y_1), (x_2, y_2), \dots (x_n, y_n)\}$ and the loss function L, the batch SGD method produces iterates

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

where $\alpha_k > 0$ stands for the learning rate and where

$$\nabla_\theta h(\mathcal{S}_k, \theta_k) = \frac{1}{|\mathcal{S}_k|}\sum_{(x_t,y_t)\in\mathcal{S}_k} \nabla_\theta L(y_t, f_{\theta_k}(x_t))$$

is a batch approximation of the full gradient $\nabla_\theta h(\theta_k) = \frac{1}{n}\sum_{t=1}^{n} \nabla_\theta L(y_t, f_{\theta_k}(x_t))$ on a random subset $\mathcal{S}_k \subset \mathcal{T}$.

Despite its ease of implementation and great popularity in the machine learning community, the SGD method, like all other first-order methods, is known to have limited effectiveness (requires many iterations in order to converge or even simply diverges) for non-convex objective functions, as is the case in deep neural networks.

*Corresponding author: abdoulaye.koroko@ifpen.fr

In classical optimization, second-order methods are known for their efficiency in terms of convergence speed compared to first-order methods. A second-order iteration reads

$$\theta_{k+1} = \theta_k - \alpha_k [H(\theta_k)]^{-1} \nabla_\theta h(\theta_k),$$

where $H(\theta_k) \in \mathbb{R}^{p \times p}$ is the curvature matrix of h at θ_k. The matrix H can be the Hessian matrix $\nabla^2_{\theta\theta} h$ as in the Newton-Raphson method, the drawback of which is that $H^{-1}\nabla_\theta h$ is not guaranteed to be a descent direction. It is wiser to replace the Hessian matrix by a surrogate such as the Generalized Gauss-Newton matrix (Schraudolph, 2002) or the Fisher Information Matrix (FIM) (Amari, 1998), which are always positive semi-definite. Unfortunately, second-order methods remain impractical for deep neural networks where the number of parameters can quickly become very large (tens of millions), making it impossible to compute and to store, let alone to invert H.

A first way to avoid assembling and storing the matrix H is the inexact resolution of the linear system by Conjugate Gradient (CG), which requires only matrix-vector products. This *Hessian-free* philosophy (Martens, 2010) is still expensive, since the CG must be run with a significant number of iterations before reaching an acceptable convergence.

An alternative is to consider the direct inversion of a diagonal approximation to H, as in (Becker & Le Cun, 1988) for the Hessian matrix or in (Duchi et al., 2011; Kingma & Ba, 2015; Tieleman & Hinton, 2012) for the empirical FIM. The reader is referred to (Kunstner et al., 2019; Martens, 2014) for the difference between the empirical and the exact FIM (both are estimators of the true FIM but the second one uses sampled outputs from the model distribution). Another approach is to use a low-rank approximation of the Hessian matrix such as BFGS (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970) or its low-memory version L-BFGS (Liu & Nocedal, 1989), which is better suited to deep learning. Nevertheless, the trouble with diagonal and low-rank approximations is that they are very rough and therefore give rise to less efficient algorithms than a well-tuned SGD.

More advanced methods resort to a block-diagonal approximation of a curvature matrix. Le Roux et al. (2008) and Ollivier (2015) use respectively a block-diagonal approximation of the empirical and exact FIM where each block contains the weights associated to a particular neuron. Based on early ideas in (Heskes, 2000; Pascanu & Bengio, 2013; Povey et al., 2014), a new family of methods under the name of KFAC have recently emerged (Ba et al., 2017; George et al., 2018; Grosse & Martens, 2016; Martens & Grosse, 2015; Martens et al., 2018). Thanks to a Kronecker-factored layer-wise block-diagonal approximation of the FIM, the KFAC methods have proven to be more powerful than a well-tuned SGD. Following similar lines of thought, Botev et al. (2017) and Goldfarb et al. (2020) also proposed efficient approximations of respectively GGN and Hessian matrices for training Multi-Layer Perceptrons (MLP).

The fundamental assumption on which KFAC hinges is the independence between activations and pre-activation derivatives. We believe that this premise, the theoretical foundation of which is unclear, is at the root of a poor quality of the FIM approximation. This is why, in this work, we wish to put forward four Kronecker-factored block-diagonal approximations that aim at more accurately representing the FIM by removing this assumption. To this end, we minimize the Frobenius norm of the difference between the original matrix and a prescribed form for the approximation, which is achievable through the Kronecker product singular value decomposition. Tests carried out on the three standard deep auto-encoder benchmarks showed that our proposed methods outperform KFAC both in terms of FIM approximation quality and optimization speed of the objective function.

The paper is organized as follows: Section 2 introduces the natural gradient and KFAC methods. Section 3 proposes the above mentioned novel approximations. In Section 4, we present and comment several numerical experiments. Finally, the conclusion overviews the work undertaken in this research and outlines directions for future study.

2 Background and notation

We consider an ℓ-layer feedforward neural network f_θ parametrized by

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

where $W_i \in \mathbb{R}^{d_i \times (d_{i-1}+1)}$ is the weights matrix associated to layer i and "vec" is the operator that vectorizes a matrix by stacking its columns together. This network transforms an input $x =: a_0 \in \mathbb{R}^{d_0}$ to an output $z = f_\theta(x)$ by the sequence

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

terminated by $z := a_\ell \in \mathbb{R}^{d_\ell}$. Here, $\bar{a}_{i-1} = (1, a_i^T)^T$ is the augmented activation vector (value 1 is used for the bias) and σ_i the activation function at layer i. The number of neurons at layer i is d_i and the total number of parameters is $p = \sum_{i=1}^{\ell} d_i(d_{i-1}+1)$.

For a given input-target pair (x, y), the gradient of the loss $L(y, f_\theta(x))$ w.r.t to the weights is computed by the back-propagation algorithm (LeCun, 1988). For convenience, we adopt the shorthand notation $\mathcal{D}v = \nabla_v L$ for the derivative

of L w.r.t any variable v, as well as the special symbol $g_i = \mathcal{D}s_i$ for the preactivation derivative. Starting from $\mathcal{D}a_\ell = \partial_z L(y, z = a_\ell)$, we perform

$$g_i = \mathcal{D}a_i \odot \sigma_i'(s_i), \quad \mathcal{D}W_i = g_i \bar{a}_{i-1}^T, \quad \mathcal{D}a_{i-1} = W_i^T g_i,$$

for i from ℓ to 1, where $\odot$ denotes the component-wise product. Finally, the gradient $\nabla_\theta L$ is retrieved as

$$\mathcal{D}\theta = [\text{vec}(\mathcal{D}W_1)^T, \text{vec}(\mathcal{D}W_2)^T, \dots, \text{vec}(\mathcal{D}W_\ell)^T]^T.$$

2.1 Natural Gradient Descent

The loss function $L(y, z)$ is now assumed to take the form

$$L(y, z) = -\log(p(y|x, \theta)),$$

where $p(y|x, \theta)$ is the density function of the network's predictive distribution $P_{y|x}(\theta)$. Note that $P_{y|x}(\theta)$ is multivariate normal for the standard square loss function, multinomial for the cross-entropy one. Then, the natural gradient descent method (Amari, 1998) is defined as

$$\theta_{k+1} = \theta_k - \alpha_k \left[F(\theta_k)\right]^{-1} \nabla_\theta h(\theta_k),$$

where

$$F(\theta) = \mathbb{E}_{x\sim Q_x, y\sim P_{y|x}(\theta)}[\mathcal{D}\theta(\mathcal{D}\theta)^T]$$

is the FIM associated to the network parameter θ. The expectation is taken according to the distribution Q_x of the input data x and the conditional distribution $P_{y|x}(\theta)$ of the the network 's output prediction y. For brevity and without any risk of ambiguity, we will omit the subscripts for the expectation and write $\mathbb{E}$ instead of $\mathbb{E}_{x\sim Q_x, y\sim P_{y|x}}(\theta)$.

The natural gradient method can be seen as the steepest descent method in the space of model's probability distributions with the metric induced by the Kullback-Leibler (KL) divergence (Amari & Nagaoka, 2000). Indeed, it can be shown that for some constant scaling factor $\lambda > 0$,

$$-\frac{1}{\lambda}[F(\theta)]^{-1}\nabla_\theta h(\theta) = \underset{d:\text{KL}[P_{y|x}(\theta) \,\|\, P_{y|x}(\theta+d)] = c}{\text{argmin}} h(\theta + d).$$

The appealing property of the natural gradient $F^{-1}\nabla h$ is that it has an intrinsic geometric interpretation, regardless of the actual choice of parameters. A thorougher discussion can be found in (Martens, 2014).

It follows from the definition of the FIM that

$$F = \mathbb{E}[\mathcal{D}\theta(\mathcal{D}\theta)^T] = \begin{bmatrix} F_{1,1} & \dots & F_{1,\ell} \\ \vdots & & \vdots \\ F_{\ell,1} & \dots & F_{\ell,\ell} \end{bmatrix},$$

in which the block

$$F_{i,j} = \mathbb{E}[\text{vec}(\mathcal{D}W_i)\text{vec}(\mathcal{D}W_j)^T] = \mathbb{E}[\bar{a}_{i-1}\bar{a}_{j-1}^T \otimes g_i g_j^T]$$

is a $d_i(d_{i-1}+1) \times d_j(d_{j-1}+1)$ matrix. We recall that the Kronecker product $A \otimes B$ between two matrices $A \in \mathbb{R}^{m_A \times n_A}$ and $B \in \mathbb{R}^{m_B \times n_B}$ is the $m_A m_B \times n_A n_B$ matrix

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

The blocks of F can be given the following meaning: $F_{i,i}$ contains second-order statistics of weight derivatives on layer i, while $F_{i,j,i\neq j}$ represents correlation between weight derivatives of layers i and j.

2.2 KFAC method

The Kronecker-factored approximate curvature (KFAC) method introduced by (Martens & Grosse, 2015) is grounded on two assumptions that provide a computationally efficient approximation of F.

The first assumption is that $F_{i,j} = 0$ for $i \neq j$. In other words, weight derivatives in two different layers are uncorrelated. This results in block-diagonal approximation

$$F \approx \text{diag}(F_{1,1}, F_{2,2}, \dots F_{\ell,\ell}).$$

This first approximation is insufficient, insofar as the blocks of $F_{i,i}$ are very large for neural networks with high number of units in layers. A further approximation is in order.

The second assumption is that of independent activations and derivatives (IAD): activations and pre-activation derivatives are independent. i.e $\forall i$, $a_{i-1} \perp\!\!\!\perp g_i$. This allows each block $F_{i,i}$ to be factorized into a Kronecker product of two smaller matrices, i.e.,

$$\begin{aligned} F_{i,i} &= \mathbb{E}[\bar{a}_{i-1}\bar{a}_{i-1}^T \otimes g_i g_i^T] \\ &\approx \mathbb{E}[\bar{a}_{i-1}\bar{a}_{i-1}^T] \otimes \mathbb{E}[g_i g_i^T] \\ &=: \bar{A}_{i-1}^{\text{KFAC}} \otimes G_i^{\text{KFAC}}, \end{aligned} \tag{1}$$

with $\bar{A}_{i-1}^{\text{KFAC}} = \mathbb{E}[\bar{a}_{i-1}\bar{a}_{i-1}^T] \in \mathbb{R}^{(d_{i-1}+1)\times(d_{i-1}+1)}$ and $G_i^{\text{KFAC}} = \mathbb{E}[g_i g_i^T] \in \mathbb{R}^{d_i \times d_i}$.

These two assumptions yield the KFAC approximation

$$F_{\text{KFAC}} = \text{diag}(\bar{A}_0^{\text{KFAC}} \otimes G_1^{\text{KFAC}}, \dots, \bar{A}_{\ell-1}^{\text{KFAC}} \otimes G_\ell^{\text{KFAC}}).$$

KFAC has been extended to convolution neural networks (CNN) by Grosse & Martens (2016). However, due to weight sharing in convolutional layers, it was necessary to add two extra assumptions regarding spatial homogeneity and spatially uncorrelated derivatives.

The decisive advantage of F_{KFAC} is that it can be inverted in a very economical way. Indeed, owing to the properties $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$ and $(A \otimes B)\text{vec}(X) = \text{vec}(BXA^T)$ of the Kronecker product, the approximate natural gradient $F_{\text{KFAC}}^{-1}\nabla h$ can be evaluated as

$$F_{\text{KFAC}}^{-1}\nabla h = \begin{bmatrix} \text{vec}(G_1^{-1}(\nabla_{W_1} h)\bar{A}_0^{-1}) \\ \vdots \\ \text{vec}(G_\ell^{-1}(\nabla_{W_\ell} h)\bar{A}_{\ell-1}^{-1}) \end{bmatrix}, \tag{2}$$

where the KFAC superscripts are dropped from now on to alleviate notations. This drastically reduces computations and memory requirements, since we only need to store, invert and multiply the smaller matrices $\bar{A}_{i-1}$'s and G_i's.

In practice, because the curvature changes relatively slowly (Martens & Grosse, 2015), the factors $(\bar{A}_{i-1}, G_i)$ are computed at every T_1 iterations and their inverses at every T_2 iterations. Moreover, $(\bar{A}_{i-1}, G_i)$ are estimated using exponentially decaying moving average. At iteration k, let $(\bar{A}_{i-1}^{\text{old}}, G_i^{\text{old}})$ be the factors previously computed at iteration $k - T_1$ and $(\bar{A}_{i-1}^{\text{new}}, G_i^{\text{new}})$ be those computed with the current mini-batch. Then, setting $\rho = \min(1 - 1/k, \alpha)$ with $\alpha \in [0, 1]$, we have

$$\begin{aligned} \bar{A}_{i-1} &= \rho\bar{A}_{i-1}^{\text{old}} + (1-\rho)\bar{A}_{i-1}^{\text{new}}, \\ G_i &= \rho G_i^{\text{old}} \ + (1-\rho)G_i^{\text{new}}. \end{aligned}$$

Another crucial ingredient of KFAC is the Tikhonov regularization to enforce invertibility of F_{KFAC}. The straightforward damping $F_{\text{KFAC}} + \lambda I$ deprives us of the possibility of applying the formula $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$. To overcome this issue, Martens & Grosse (2015) advocated the more judicious Kronecker product regularization

$$\widetilde{F}_{i,i} = (\bar{A}_{i-1} + \pi_i \lambda^{1/2} I) \otimes (G_i + \pi_i^{-1} \lambda^{1/2} I)$$

where $\lambda > 0$ and

$$\pi_i = \sqrt{\frac{\text{tr}(\bar{A}_{i-1})/(d_{i-1}+1)}{\text{tr}(G_i)/d_i}}.$$

3 Four novel methods

While staying within the framework of the first assumption (block-diagonal approximation), we now design four new methods that break free from the second hypothesis (IAD) in order to achieve a better accuracy: KPSVD, Deflation, Lanczos-bidiagonalization and KFAC-corrected.

3.1 KPSVD

In our first method, called KPSVD, the factors $(\bar{A}_{i-1}, G_i)$ are specified as the arguments of the best possible approximation of $F_{i,i}$ by a single Kronecker product. Thus,

$$\begin{aligned}(\bar{A}_{i-1}, G_i) &= \operatorname*{argmin}_{(R,S)} \|F_{i,i} - R \otimes S\|_F \\ &= \operatorname*{argmin}_{(R,S)} \|\mathbb{E}[\bar{a}_{i-1}\bar{a}_{i-1}^T \otimes g_i g_i^T] - R \otimes S\|_F,\end{aligned} \tag{3}$$

where $\|\cdot\|_F$ denotes Frobenius norm. Problem (3) can be solved at a low cost by means of the Kronecker product singular value decomposition technique (van Loan, 2000). To write down the solution, we need the following notion. Let

$$M = \begin{bmatrix} M_{1,1} & \dots & M_{1,d} \\ M_{2,1} & \dots & M_{2,d} \\ \vdots & & \vdots \\ M_{d,1} & \dots & M_{d,d} \end{bmatrix} \in \mathbb{R}^{d'd \times d'd}$$

be a uniform block matrix, that is, $M_{\mu,\nu} \in \mathbb{R}^{d' \times d'}$ for all $(\mu, \nu) \in \{1, \dots d\}^2$. The *zigzag rearrangement* operator $\mathcal{Z}$ converts M into the matrix

$$\mathcal{Z}(M) = \begin{bmatrix} \operatorname{vec}(M_{1,1})^T \\ \vdots \\ \operatorname{vec}(M_{d,1})^T \\ \vdots \\ \operatorname{vec}(M_{1,d})^T \\ \vdots \\ \operatorname{vec}(M_{d,d})^T \end{bmatrix} \in \mathbb{R}^{d^2 \times (d')^2},$$

by flattening out each block in a column-wise order and by transposing the resulting vector. This operator is to be applied to each $M = F_{i,i}$ with $d = d_{i-1} + 1$ and $d' = d_i$.

Theorem 3.1. *Any solution of* (3) *is also a solution of the ordinary rank-1 matrix approximation problem*

$$(\operatorname{vec}(\bar{A}_{i-1}^{\text{KPSVD}}), \operatorname{vec}(G_i^{\text{KPSVD}})) = \operatorname*{argmin}_{(R,S)} \|\mathcal{Z}(F_{i,i}) - \operatorname{vec}(R) \operatorname{vec}(S)^T\|_F. \tag{4}$$

Proof. See appendix A.1. □

Problem (4) is solved as follows. Let $U^T \mathcal{Z}(F_{i,i}) V = \Sigma$ be the singular value decomposition (SVD) of $\mathcal{Z}(F_{i,i})$. Let σ_1 be the greatest singular value of $\mathcal{Z}(F_{i,i})$ and (u_1, v_1) be the associated left and right singular vectors. A solution is,

$$\bar{A}_{i-1}^{\text{KPSVD}} = \sqrt{\sigma_1}\, \operatorname{MAT}(u_1), \quad G_i^{\text{KPSVD}} = \sqrt{\sigma_1}\, \operatorname{MAT}(v_1),$$

where "MAT," the converse of "vec," turns a vector into a matrix. The question to be addressed now is how to compute u_1, v_1 and σ_1. We recommend the power SVD algorithm (see appendix B.1), which only requires the matrix-vector multiplications $\mathcal{Z}(F_{i,i})v$ and $\mathcal{Z}(F_{i,i})^T u$. These operations can be performed without explicitly forming $F_{i,i}$ or $\mathcal{Z}(F_{i,i})$, as elaborated on in the upcoming Proposition.

Proposition 3.1. *For all* $u \in \mathbb{R}^{(d_{i-1}+1)^2}$ *and* $v \in \mathbb{R}^{d_i^2}$,

$$\begin{aligned}\mathcal{Z}(F_{i,i})v &= \mathbb{E}[\, g_i^T V g_i \operatorname{vec}(\bar{a}_{i-1}\bar{a}_{i-1}^T)\,], \\ \mathcal{Z}(F_{i,i})^T u &= \mathbb{E}[\, \bar{a}_{i-1}^T U \bar{a}_{i-1} \operatorname{vec}(g_i g_i^T)\,],\end{aligned}$$

with $U = \operatorname{MAT}(u)$ *and* $V = \operatorname{MAT}(v)$.

Proof. See appendix A.2. □

Estimating $\mathcal{Z}(F_{i,i})v$ and $\mathcal{Z}(F_{i,i})^T u$

Let us consider a batch $\mathcal{B} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ drawn from the training data $\mathcal{T}$. We recall that the expectation is taken with respect to both Q_x (data distribution over inputs x) and $P_{y/x}(\theta)$ (predictive distribution of the network). To estimate $\mathcal{Z}\left(F_{i,i}\right) v$ and $\mathcal{Z}\left(F_{i,i}\right)^T u$, we use the Monte-Carlo method as suggested by Martens & Grosse (2015): we first compute the statistics $\bar{a}_{i-1}$'s and g_i's during an additional back-propagation performed using targets y's sampled from $P_{y/x}(\theta)$ and then set

$$\mathcal{Z}(F_{i,i})v \approx \frac{1}{m}\sum_{t=1}^{m} g_{i,t}^T V g_{i,t} \operatorname{vec}(\bar{a}_{i-1,t}\bar{a}_{i-1,t}^T),$$
$$\mathcal{Z}(F_{i,i})^T u \approx \frac{1}{m}\sum_{t=1}^{m} \bar{a}_{i-1,t}^T U \bar{a}_{i-1,t} \operatorname{vec}(g_{i,t}g_{i,t}^T).$$

So far, we have not paid attention to the symmetry of the matrices $(\bar{A}_{i-1}^{\text{KPSVD}}, G_i^{\text{KPSVD}})$ in problem (3). It turns out that symmetry is automatic, while positive semi-definiteness occurs for some solutions to be selected.

Proposition 3.2. *All solutions $(\bar{A}_{i-1}^{\text{KPSVD}}, G_i^{\text{KPSVD}})$ of problem* (3) *are symmetric. Besides, we can select solutions for which these matrices are positive semi-definite.*

Proof. See appendix A.3. □

3.2 Kronecker rank-2 approximation to $F_{i,i}$

Since the KPSVD method of §3.1 is merely a Kronecker rank-1 approximation of $F_{i,i}$, it is most natural to look for higher order approximations. The two methods presented in this section are based on seeking a Kronecker rank-2 approximation $R \otimes S + P \otimes Q$ of $F_{i,i}$ that achieves

$$\min_{(R,S,P,Q)} \|F_{i,i} - (R \otimes S + P \otimes Q)\|_F. \tag{5}$$

Again, the zigzag rearrangement operator $\mathcal{Z}$ enables us to reformulate (5) as an ordinary rank-2 matrix approximation problem. To determine a solution of the latter, there are two techniques in practice: *deflation* (Saad, 2011) and *Lanczos bi-diagonalization* (Golub & Kahan, 1965).

3.2.1 Deflation

The rank-1 factors (R, S) and the rank-2 factors (P, Q) are computed successively, one after another:

1. Apply the power SVD algorithm to $\mathcal{Z}(F_{i,i})$ to compute (R, S) so as to minimize $\|F_{i,i} - R \otimes S\|_F$. The solution is known to be $(R, S) = (\bar{A}_{i-1}^{\text{KPSVD}}, G_i^{\text{KPSVD}})$.
2. Let $\widehat{F}_{i,i} = F_{i,i} - R \otimes S$. Apply the power SVD algorithm to $\mathcal{Z}(\widehat{F}_{i,i})$ to compute (P, Q) so as to minimize $\|\widehat{F}_{i,i} - P \otimes Q\|_F$.
3. Set $F_{i,i} \approx R \otimes S + P \otimes Q$.

In step 2, we need to calculate the matrix-vector products $\mathcal{Z}(\widehat{F}_{i,i})v$ and $\mathcal{Z}(\widehat{F}_{i,i})^T u$. These operations can be done efficiently without explicitly forming $\widehat{F}_{i,i}$ or $\mathcal{Z}(\widehat{F}_{i,i})$. Indeed,

$$\mathcal{Z}(\widehat{F}_{i,i})v = \mathcal{Z}(F_{i,i})v - \mathcal{Z}(R \otimes S)v,$$
$$\mathcal{Z}(\widehat{F}_{i,i})^T u = \mathcal{Z}(F_{i,i})^T u - \mathcal{Z}(R \otimes S)^T u.$$

On one hand, we know how compute $\mathcal{Z}(F_{i,i})v$ and $\mathcal{Z}(F_{i,i})^T u$ from Proposition 3.1. On the other hand, it is not difficult to show that

$$\mathcal{Z}(R \otimes S)v = \langle \operatorname{vec}(S), v \rangle \operatorname{vec}(R),$$
$$\mathcal{Z}(R \otimes S)^T u = \langle \operatorname{vec}(R), u \rangle \operatorname{vec}(S),$$

where $\langle \cdot, \cdot \rangle$ stands for the dot product.

3.2.2 Lanczos bi-diagonalization

In contrast to deflation, the Lanczos bi-diagonalization algorithm (see appendix B.2) computes (R, S) and (P, Q) at the same time. It does so by simulataneously computing the two largest singular values $\sigma_1 \geq \sigma_2$ of $\mathcal{Z}(F_{i,i})$ with the associated singular vectors (u_1, v_1) and (u_2, v_2). Once these singular elements are determined, it remains to set

$$R = \sqrt{\sigma_1}\,\text{MAT}(u_1), \quad S = \sqrt{\sigma_1}\,\text{MAT}(v_1),$$
$$P = \sqrt{\sigma_2}\,\text{MAT}(u_2), \quad Q = \sqrt{\sigma_2}\,\text{MAT}(v_2).$$

Similarly to KPSVD, we only have to perform the matrix-vector multiplications $\mathcal{Z}(F_{i,i})v$ and $\mathcal{Z}(F_{i,i})^T u$ without forming and storing $F_{i,i}$ or $\mathcal{Z}(F_{i,i})$.

In pratice, it is advisable to implement the restarted version of the algorithm (Saad, 2011), which consists of three steps:

1. **Start**: Choose an initial vector $q^{(0)}$ and a dimension K for the Krylov subspace.
2. **Iterate**: Perform Lanczos bidiagonalization algorithm (appendix B.2).
3. **Restart**: Compute the desired singular vectors. If stopping criterion satisfied, stop. Else set $q^{(0)} =$ linear combination of singular vectors and go to 2.

3.3 KFAC-CORRECTED

Another idea is to simply add an *ad hoc* correction to the KFAC approximation. Put another way, we consider

$$F_{i,i} \approx \bar{A}_{i-1}^{\text{KFAC}} \otimes G_i^{\text{KFAC}} + \bar{A}_{i-1}^{\text{corr.}} \otimes G_i^{\text{corr.}},$$

using the best possible correctors, that is,

$$(\bar{A}_{i-1}^{\text{corr.}}, G_i^{\text{corr.}}) = \underset{(P,Q)}{\text{argmin}} \|F_{i,i} - \bar{A}_{i-1}^{\text{KFAC}} \otimes G_i^{\text{KFAC}} - P \otimes Q\|_F. \tag{6}$$

Again, the solution of (6) can be computed by applying the power SVD algorithm to the matrix $\mathcal{Z}(F_{i,i} - \bar{A}_{i-1}^{\text{KFAC}} \otimes G_i^{\text{KFAC}})$. The matrix-vector multiplications required can be done in the same way as in the *deflation* method without explicitly forming and storing the matrices.

3.4 Inversion of $A \otimes B + C \otimes D$

For each of the last three methods, we need to solve a linear system of the form $(A \otimes B + C \otimes D)u = v$ in an efficient way. This is far from obvious, since due to the sum, the well-known and powerful identities $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$ and $(A \otimes B)^{-1}\text{vec}(X) = \text{vec}(B^{-1}XA^{-T})$ can no longer be applied.

There are many good methods to compute u, but the most appropriate for our problem is that of Martens & Grosse (2015), since it takes advantage of symmetry and definiteness of the matrices. Below is a summary of the algorithm, the full details of which are in (Martens & Grosse, 2015).

1. Compute $A^{-1/2}$, $B^{-1/2}$ and the symmetric eigen/SVD-decompositions
$$A^{-1/2}CA^{-1/2} = E_1 S_1 E_1^T,$$
$$B^{-1/2}DB^{-1/2} = E_2 S_2 E_2^T,$$
where $S_{1,2}$ are diagonal and $E_{1,2}$ are unitary.
2. Set $K_1 = A^{-1/2}E_1$, $K_2 = B^{-1/2}E_2$. Then,
$$u = \text{vec}(K_2[(K_2^T V K_1) \oslash (\mathbf{1}\mathbf{1}^T + s_2 s_1^T)]K_1^T),$$
where $E \oslash F$ denotes the Hadamard or element-wise division of E by F, $s_{1,2} = \text{diag}(S_{1,2})$, $\mathbf{1}$ vector of ones and $V = \text{MAT}(v)$. Note that $K_{1,2}$, $s_{1,2}$ can be stored and reused for different choices of v.

4 Experiments

We have evaluated our proposed methods as well as KFAC, SGD and ADAM on the three standard deep-auto-encoder problems used for benchmarking neural network optimization methods (Botev et al., 2017; Martens, 2010; Martens &

Grosse, 2015; Sutskever et al., 2013). The benchmarks consist of training three different auto-encoder architectures with CURVES, MNIST and FACE datasets respectively. See appendix D for a complete description of the network architectures and datasets. In our experiments, all our proposed methods as well as KFAC use approximations of the exact FIM F. Experiments were performed with PyTorch framework (Paszke et al., 2019) on supercomputer with Nividia Ampere A100 GPU and AMD Milan@2.45GHz CPU.

The precision value ϵ for power SVD and Lanczos bi-diagonalization algorithm was set to 10^{-6}. Also for these two algorithms, we used a warm-start technique which means that the final results of the previous iteration are used as a starting point (instead of a random point) for the current iteration. This has resulted in a faster convergence. In all experiments, the batch sizes used are 256, 512 and 1024 for CURVES, MNIST and FACES datasets respectively.

We first evaluate the approximation qualities of the FIM and then report the results on performance of the optimization objective.

4.1 Approximation qualities of the FIM

We investigated how well our proposed methods and KFAC approximate blocks of the exact FIM. To do so, we computed for each of the problems the exact FIM and its different approximations of the 5th layer of the network. For a fair comparison, the exact FIM as well as its different approximations were computed during the same optimization process with an independent optimizer (SGD or ADAM). We ran two independent tests with SGD and ADAM optimizers respectively and ended up with the same results. We therefore decided to report only the results obtained with ADAM. Let F be the exact FIM of the 5th layer of the network and $\hat{F}$ be any approximation to F ($\hat{F}$ is in the form $A \otimes B$ for KFAC and KPSVD, and $R \otimes S + P \otimes Q$ for KFAC corrected, Deflation and Lanczos). We measured the following two types of error:

- **Error 1**: Frobenius norm error between F and $\hat{F}$: $\|F - \hat{F}\|_F / \|F\|_F$;
- **Error 2**: ℓ_2 norm error between the spectra of F and $\hat{F}$: $\|\text{spec}(F) - \text{spec}(\hat{F})\|_2 / \|\text{spec}(F)\|_2$ where $\text{spec}(M)$ denotes the spectrum of M and $\|\cdot\|_2$ is the ℓ_2 norm.

Note that here the Fisher matrices were estimated without the exponentially decaying averaging scheme which means that only the mini-batch at iteration k is used to computed the Fisher matrices at this iteration.

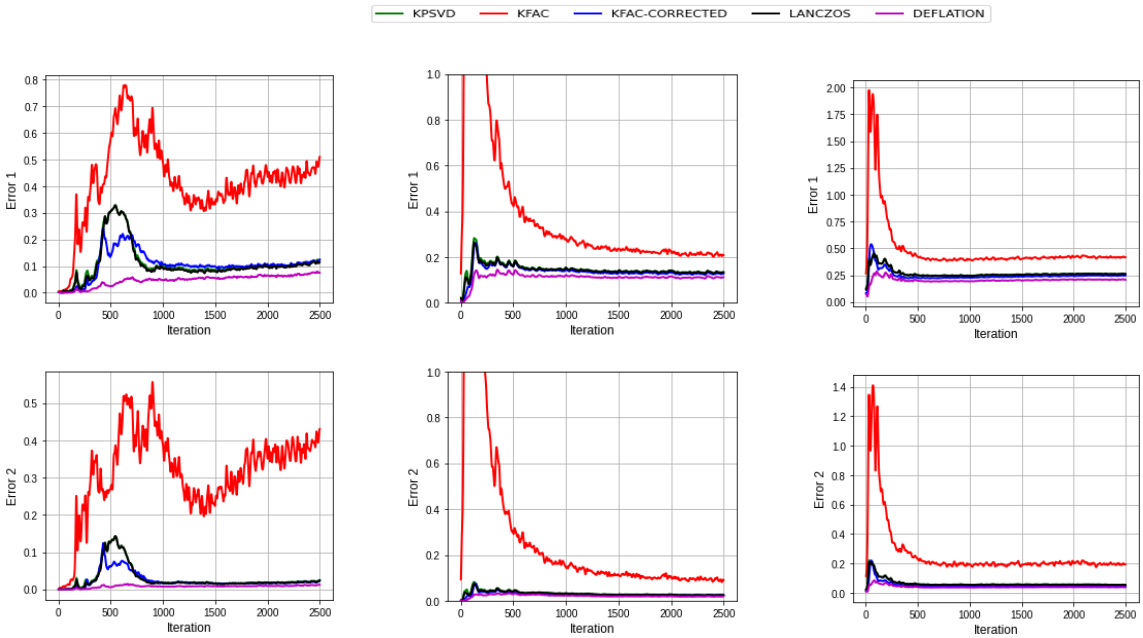
As we can see in Figure 1, for each of the problems, the Deflation method gives the best approximation, followed by the other methods. The **Error 1** and **Error 2** made by our different methods remain lower than those caused by KFAC throughout the optimization process. This suggests that our methods give a better approximation to the Fisher than KFAC, and that increasing the rank does improve the quality of approximation. One can go further in this direction if there is no prohibitive extra cost.

4.2 Optimization performance

We now consider the network optimization in each of the three problems. We have evaluated our methods against KFAC and the baselines (SGD and ADAM). Here the different approximations to the FIM were computed using the exponentially decaying technique as described in §2.2. The decay factor α was set to 0.95 as in (Martens & Grosse, 2015). Since the goal of KFAC as well as our methods is optimization performance rather than generalization, we performed Grid Search for each method and selected hyperparameters that gave a better reduction to the training loss. The learning rate η and the damping parameter λ are in range $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 3 \cdot 10^{-1}, 3 \cdot 10^{-2}, 3 \cdot 10^{-3}, 3 \cdot 10^{-4}\}$, and the clipping parameter c belongs to $\{10^{-2}, 10^{-3}\}$ (see appendix E for definition of c). Note that damping and clipping are used only in KFAC and our proposed methods. Update frequencies T_1 and T_2 were set to 100. The momentum parameters were $\beta = 0.9$ for SGD and $(\beta_1, \beta_2) = (0.9, 0.999)$ for ADAM.

Figure 2 shows the performance of the different optimizers on the three studied problems. The first observation is that in each problem, KFAC as well as our methods optimize the training loss function faster than SGD and ADAM both with respect to epoch and time. Although our methods may seem much more computationally expensive than KFAC since at each iteration we perform the power SVD or Lanczos bi-diagonalization to estimate the Fisher matrix, they actually have the same order of magnitude in computational cost as KFAC. See appendix C for a comparison of the computational costs. For each of the three problems, we observe that KFAC and KPSVD perform about the same while the DEFLATION, LANCZOS and KFAC-CORRECTED methods have the ability to optimize the objective function much faster both with respect to epoch and time.

Although this is not our object of study, we observe that for each of the three problems, our proposed methods also maintain a good generalization.



(a) CURVES (b) MNIST (c) FACES

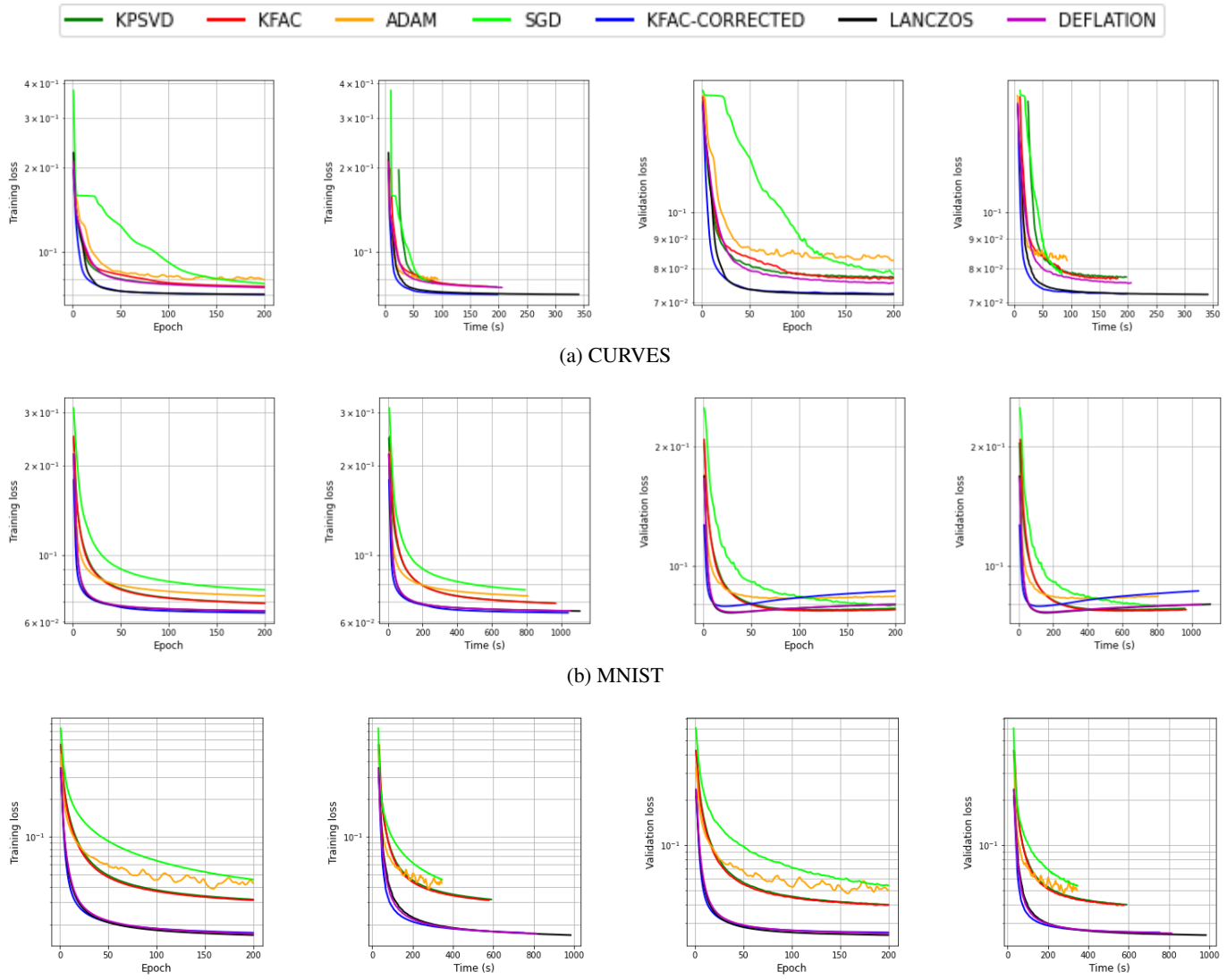
Figure 1: Comparison between FIM approximation qualities of our methods and KFAC. For each problem, at each training iteration of the network with ADAM optimizer, the exact FIM and its different approximations are computed for layer 5 of the network. **Error 1** and **Error 2** described in subsection 4.1 are measured. For the sake of visual comparison between different methods, the display scale in the axis of ordinates was deliberately restricted to $[0, 1]$ for the MNIST problem. It thus seems that the error curves for KFAC, whose peak amplitudes are about 6.5, are truncated.

We evaluated our methods on others MLP architectures and obtained good results. However, when considering CNN architectures, we did not observe any gain in performance compared to KFAC. This can be explained by the fact that IAD is not the only assumption made by KFAC for CNNs (Grosse & Martens, 2016) and therefore steering clear from this hypothesis alone is insufficient to reach a better performance.

5 Conclusion and perspectives

In this work, we proposed a series of novel Kronecker factorizations to the blocks of the Fisher of multi-layer perceptrons using the Kronecker product Singular Value Decomposition technique. Tests realized on the three standard deep auto-encoder problems showed that our proposed methods outperform KFAC both in terms of Fisher approximation quality and in terms of optimization speed of the objective function. These facts are even more noticeable for the methods using high rank approximations.

KFAC as well as our methods use a block-diagonal approximation of FIM where each block corresponds to a layer. This results in ignoring the correlations between the layers. Future works will focus on incorporating cross-layer information, as was attempted by Tselepidis et al. (2020) with a two-level KFAC preconditioning approach.



(a) CURVES

(b) MNIST

(c) FACES

Figure 2: Comparison of optimization performance of different algorithms on each of the 3 problems (CURVES **top** row, MNIST **middle** row and FACES **bottom** row). For each problem, from left to right, **first** figure displays training loss *vs* epoch, **second** one represents training loss *vs* time, the **third** depicts validation loss *vs* epoch and the **last** displays validation loss *vs* time.

References

Amari, S.-I. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998. doi: 10.1162/089976698300017746.

Amari, S.-I. and Nagaoka, H. *Methods of Information Geometry*, volume 191 of *Translations of Mathematical Monographs*. American Mathematical Society, Providence, Rhode Island, 2000. ISBN 9780821843024.

Ba, J., Grosse, R., and Martens, J. Distributed second-order optimization using Kronecker-factored approximations. In *5th International Conference on Learning Representations, Conference Track Proceedings*, Toulon, France, 2017. URL https://openreview.net/forum?id=SkkTMpjex.

Becker, S. and Le Cun, Y. Improving the convergence of back-propagation learning with second order methods. In Touretzky, D., Hinton, G., and Sejnowski, T. (eds.), *Proceedings of the 1988 Connectionist Models Summer School*, pp. 29–37. Morgan Kaufman, 1988.

Botev, A., Ritter, H., and Barber, D. Practical Gauss-Newton optimisation for deep learning. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp. 557–565, Sydney, Australia, 06–11 Aug 2017. PMLR. doi: 10.5555/3305381.3305439.

Broyden, C. G. The convergence of a class of double-rank minimization algorithms 1. General considerations. *IMA Journal of Applied Mathematics*, 6(1):76—90, 1970. doi: 10.1093/imamat/6.1.76.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12(61):2121–2159, 2011. URL http://www.jmlr.org/papers/v12/duchi11a.html.

Fletcher, R. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 1970. doi: 10.1093/comjnl/13.3.317.

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

Goldfarb, D. A family of variable-metric methods derived by variational means. *Mathematics of computation*, 24(109): 23–26, 1970. doi: 10.1090/S0025-5718-1970-0258249-6.

Goldfarb, D., Ren, Y., and Bahamou, A. Practical quasi-Newton methods for training deep neural networks, 2020. URL https://arxiv.org/pdf/2006.08877.pdf. arXiv:2006.08877.

Golub, G. and Kahan, W. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics: Series B, Numerical Analysis*, 2(2):205–224, 1965. doi: 10.1137/0702016.

Grosse, R. and Martens, J. A Kronecker-factored approximate Fisher matrix for convolution layers. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48, pp. 573–582, New York, USA, 2016. URL http://proceedings.mlr.press/v48/grosse16.html.

Heskes, T. On "natural" learning and pruning in multilayered perceptrons. *Neural Computation*, 12(4):881–901, 2000. doi: 10.1162/089976600300015637.

Hinton, G. E. and Salakhutdinov, R. R. Reducing the dimensionality of data with neural netwoks. *Science*, 313(5786): 504–507, 2006. doi: 10.1126/science.1127647.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *Proceedings of the 3rd International Conference on Learning Representations*, San Diego, California, 2015. URL http://arxiv.org/abs/1412.6980.

Kunstner, F., Hennig, P., and Balles, L. Limitations of the empirical Fisher approximation for natural gradient descent. In Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 4156–4167. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/8669-limitations-of-the-empirical-fisher-approximation-for-natural-gradient-descent.pdf.

Le Roux, N., Manzagol, P.-A., and Bengio, Y. Topmoumoute online natural gradient algorithm. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, pp. 849–856, Vancouver, Canada, 2008. URL https://dl.acm.org/citation.cfm?id=2981669.

LeCun, Y. A theoretical framework for back-propagation. In Touretzky, D., Hinton, G., and Sejnowski, T. (eds.), *Proceedings of the 1988 Connectionist Models Summer School*, volume 1, pp. 21–28, Pittsburgh, Philadelphia, 1988.

Liu, D. C. and Nocedal, J. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3):503–528, 1989.

Martens, J. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, volume 27, pp. 735–742, Haifa, Israel, 2010. URL http://www.cs.toronto.edu/~jmartens/docs/Deep_HessianFree.pdf.

Martens, J. New insights and perspectives on the natural gradient method. arXiv:1412.1193, 2014. URL https://arxiv.org/abs/1412.1193.

Martens, J. and Grosse, R. Optimizing neural networks with Kronecker-factored approximate curvature. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pp. 2408–2417, Lille, France, 2015. URL http://proceedings.mlr.press/v37/martens15.html.

Martens, J., Ba, J., and Johnson, M. Kronecker-factored curvature approximations for recurrent neural networks. In *Proceedings of the 6th International Conference on Learning Representations*, Vancouver, Canada, 2018. URL https://openreview.net/forum?id=HyMTkQZAb.

Ollivier, Y. Riemannian metrics for neural networks I: feedforward networks. *Information and Inference: A Journal of the IMA*, 4(2):108–153, 2015. doi: 10.1093/imaiai/iav006.

Pascanu, R. and Bengio, Y. Revisiting natural gradient for deep networks. arXiv:1301.3584, 2013. URL https://arxiv.org/abs/1301.3584v4.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in neural information processing systems*, volume 32, pp. 8026–8037. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

Povey, D., Zhang, X., and Khudanpur, S. Parallel training of DNNs with natural gradient and parameter averaging. rXiv:1410.7455, 2014. URL https://arxiv.org/abs/1410.7455.

Robbins, H. and Monro, S. A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407, 1951. URL https://www.jstor.org/stable/2236626.

Saad, Y. *Numerical Methods for Large Eigenvalue Problems*, volume 6. Society for Industrial and Applied Mathematics, Philadelphia, 2011.

Schraudolph, N. N. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14 (7):1723–1738, 2002. doi: 10.1162/08997660260028683.

Shanno, D. F. Conditioning of quasi-Newton methods for function minimization. *Mathematics of computation*, 24 (111):647–656, 1970. doi: 10.1090/S0025-5718-1970-0274029-X.

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

Tieleman, T. and Hinton, G. Lecture 6.5 RMSProp: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2):26–31, 2012.

Tselepidis, N., Kohler, J., and Orvieto, A. Two-level K-FAC preconditioning for deep learning. In *OPT2020: 12th Annual Workshop on Optimization for Machine Learning*, 2020. URL https://arxiv.org/abs/2011.00573.

van Loan, C. F. The ubiquitous Kronecker product. *J. Comput. Appl. Math.*, 123(1):85–100, 2000. doi: 10.1016/S0377-0427(00)00393-9.

A Proofs

A.1 Proof of Theorem 3.1

Proof. We are going to derive the identity

$$\|F_{i,i} - R \otimes S\|_F = \|\mathcal{Z}(F_{i,i}) - \mathrm{vec}(R)\mathrm{vec}(S)^T\|_F \tag{7}$$

for all $R \in \mathbb{R}^{(d_{i-1}+1)\times(d_{i-1}+1)}$ and $S \in \mathbb{R}^{d_i \times d_i}$, from which Theorem 3.1 will follow. For notational convenience, let

$$M = F_{i,i}, \qquad d = d_{i-1} + 1, \qquad d' = d_i.$$

We recall that M has the block structure

$$M = \begin{bmatrix} M_{1,1} & \dots & M_{1,d} \\ M_{2,1} & \dots & M_{2,d} \\ \vdots & & \vdots \\ M_{d,1} & \dots & M_{d,d} \end{bmatrix} \in \mathbb{R}^{d'd \times d'd},$$

where each block $M_{\mu,\nu}$, $(\mu,\nu) \in \{1,\dots,d\}^2$, is of size $d' \times d'$. By definition of the Frobenius norm,

$$\begin{aligned} \|M - R \otimes S\|_F^2 &= \sum_{\mu=1}^{d} \sum_{\nu=1}^{d} \|M_{\mu,\nu} - R_{\mu,\nu} S\|_F^2 \\ &= \sum_{\mu=1}^{d} \sum_{\nu=1}^{d} \|\mathrm{vec}(M_{\mu,\nu}) - R_{\mu,\nu} \mathrm{vec}(S)\|_2^2 \\ &= \sum_{\mu=1}^{d} \sum_{\nu=1}^{d} \|\mathrm{vec}(M_{\mu,\nu})^T - R_{\mu,\nu} \mathrm{vec}(S)^T\|_2^2, \end{aligned} \tag{8}$$

where $R_{\mu,\nu}$ is the (μ,ν)-scalar entry of R and $\|\cdot\|_2$ denotes the Euclidean norm. By virtue of

$$\mathcal{Z}(M) = \begin{bmatrix} \mathrm{vec}(M_{1,1})^T \\ \vdots \\ \mathrm{vec}(M_{d,1})^T \\ \vdots \\ \mathrm{vec}(M_{1,d})^T \\ \vdots \\ \mathrm{vec}(M_{d,d})^T \end{bmatrix}, \qquad \mathrm{vec}(R)\mathrm{vec}(S)^T = \begin{bmatrix} R_{1,1}\mathrm{vec}(S)^T \\ \vdots \\ R_{d,1}\mathrm{vec}(S)^T \\ \vdots \\ R_{1,d}\mathrm{vec}(S)^T \\ \vdots \\ R_{d,d}\mathrm{vec}(S)^T \end{bmatrix},$$

the last equality of (8) also reads $\|M - R \otimes S\|_F^2 = \|\mathcal{Z}(M) - \mathrm{vec}(R)\mathrm{vec}(S)^T\|_F^2$, which proves (7). □

A.2 Proof of Proposition 3.1

Proof. Using the shorthand notations

$$\mathtt{A} = \bar{a}_{i-1}\bar{a}_{i-1}^T, \qquad \mathtt{G} = g_i g_i^T, \qquad d = d_{i-1} + 1, \qquad d' = d_i$$

we have

$$F_{i,i} = \mathbb{E}[\mathtt{A} \otimes \mathtt{G}] = \mathbb{E}\left(\begin{bmatrix} \mathtt{A}_{1,1}\mathtt{G} & \dots & \mathtt{A}_{1,d}\mathtt{G} \\ \vdots & & \vdots \\ \mathtt{A}_{d,1}\mathtt{G} & \dots & \mathtt{A}_{d,d}\mathtt{G} \end{bmatrix} \right) \in \mathbb{R}^{d'd \times d'd}.$$

Hence,

$$\mathcal{Z}(F_{i,i}) = \mathbb{E}\left(\begin{bmatrix} \mathrm{vec}(\mathtt{A}_{1,1}\mathtt{G})^T \\ \vdots \\ \mathrm{vec}(\mathtt{A}_{d,1}\mathtt{G})^T \\ \vdots \\ \mathrm{vec}(\mathtt{A}_{1,d}\mathtt{G})^T \\ \vdots \\ \mathrm{vec}(\mathtt{A}_{d,d}\mathtt{G})^T \end{bmatrix} \right) \in \mathbb{R}^{d^2 \times (d')^2}$$

For all $v \in \mathbb{R}^{(d')^2}$,

$$\mathcal{Z}(F_{i,i})v = \mathbb{E}\left(\begin{bmatrix}\mathrm{vec}(\mathtt{A}_{1,1}\mathtt{G})^T\\ \vdots \\ \mathrm{vec}(\mathtt{A}_{d,1}\mathtt{G})^T \\ \vdots \\ \mathrm{vec}(\mathtt{A}_{1,d}\mathtt{G})^T \\ \vdots \\ \mathrm{vec}(\mathtt{A}_{d,d}\mathtt{G})^T\end{bmatrix}\right) v = \mathbb{E}\left(\begin{bmatrix}\mathtt{A}_{1,1}\mathrm{vec}(\mathtt{G})^T v\\ \vdots \\ \mathtt{A}_{d,1}\mathrm{vec}(\mathtt{G})^T v \\ \vdots \\ \mathtt{A}_{1,d}\mathrm{vec}(\mathtt{G})^T v \\ \vdots \\ \mathtt{A}_{d,d}\mathrm{vec}(\mathtt{G})^T v\end{bmatrix}\right) = \mathbb{E}[\,(\mathrm{vec}(\mathtt{G})^T v)\,\mathrm{vec}(\mathtt{A})].$$

The scalar quantity $\mathrm{vec}(\mathtt{G})^T v$ can be further detailed as

$$\mathrm{vec}(\mathtt{G})^T v = (\mathrm{vec}(g_i g_i^T))^T v = (g_i \otimes g_i)^T v = (g_i^T \otimes g_i^T)\,\mathrm{vec}(\mathrm{MAT}(v)),$$

owing to the identities $\mathrm{vec}(xy^T) = y \otimes x$ and $(A \otimes B)^T = A^T \otimes B^T$. Invoking now $(A \otimes B)\,\mathrm{vec}(X) = \mathrm{vec}(BXA^T)$, we end up with

$$\mathrm{vec}(\mathtt{G})^T v = \mathrm{vec}(g_i^T\,\mathrm{MAT}(v)\,g_i) = \mathrm{vec}(g_i^T V g_i).$$

Therefore, $\mathcal{Z}(F_{i,i})v = \mathbb{E}[(g_i^T V g_i)\,\mathrm{vec}(\mathtt{A})]$. The proof of $\mathcal{Z}(F_{i,i})^T u = \mathbb{E}[(\bar{a}_{i-1}^T U \bar{a}_{i-1})\,\mathrm{vec}(\mathcal{G}_i)]$ for all $u \in \mathbb{R}^{d^2}$ goes along the same lines. □

A.3 Proof of Proposition 3.2

Proof.
▷ *Symmetry*. By construction and up to a choice of sign,

$$\mathrm{vec}(\bar{A}_{i-1}^{\mathrm{KPSVD}}) = \sqrt{\sigma_1}u_1, \qquad \mathrm{vec}(G_i^{\mathrm{KPSVD}}) = \sqrt{\sigma_1}v_1,$$

where σ_1 is the largest singular value of $\mathcal{Z}(F_{i,i})$ associated with left and right singular vectors (u_1, v_1). From the standard SVD properties

$$\mathcal{Z}(F_{i,i})v_1 = \sigma_1 u_1, \qquad \mathcal{Z}(F_{i,i})^T u_1 = \sigma_1 v_1,$$

we infer that

$$\sqrt{\sigma_1}\mathrm{vec}(\bar{A}_{i-1}^{\mathrm{KPSVD}}) = \mathcal{Z}(F_{i,i})v_1 = \mathbb{E}[\,(g_i^T\mathrm{MAT}(v_1)g_i)\,\mathrm{vec}(\bar{a}_{i-1}\bar{a}_{i-1}^T)\,],$$

the last equality being a consequence of Proposition 3.1. The scalar quantity $g_i^T\mathrm{MAT}(v_1)g_i$ can be moved into the argument of the "vec" operator, after which we can permute $\mathbb{E}$ and "vec" to obtain

$$\sqrt{\sigma_1}\mathrm{vec}(\bar{A}_{i-1}^{\mathrm{KPSVD}}) = \mathbb{E}[\,\mathrm{vec}((g_i^T\mathrm{MAT}(v_1)g_i)\,\bar{a}_{i-1}\bar{a}_{i-1}^T)\,] = \mathrm{vec}(\mathbb{E}[\,(g_i^T\mathrm{MAT}(v_1)g_i)\,\bar{a}_{i-1}\bar{a}_{i-1}^T\,]).$$

Hence, upon taking the "MAT" operator,

$$\sqrt{\sigma_1}\bar{A}_{i-1}^{\mathrm{KPSVD}} = \mathbb{E}[\,(g_i^T\mathrm{MAT}(v_1)g_i)\,\bar{a}_{i-1}\bar{a}_{i-1}^T\,].$$

Since each $(g_i^T\mathrm{MAT}(v_1)g_i)\,\bar{a}_{i-1}\bar{a}_{i-1}^T$ is a symmetric matrix, their expectation is also symmetric. The symmetry of G_i^{KPSVD} is proven in a similar fashion.

▷ *Positive and semi-definiteness*. Since $\bar{A}_{i-1}^{\mathrm{KPSVD}}$ and G_i^{KPSVD} are symmetric, they can be diagonalized as

$$\begin{aligned}\bar{A}_{i-1}^{\mathrm{KPSVD}} &= U^T D U, \qquad D = \mathrm{diag}(\alpha_1, \alpha_2, \ldots, \alpha_{d_{i-1}+1}),\\ G_i^{\mathrm{KPSVD}} &= V^T E V, \qquad E = \mathrm{diag}(\beta_1, \beta_2, \ldots, \beta_{d_i}),\end{aligned}$$

with orthogonal matrices U and V. Because $\bar{a}_{i-1}\bar{a}_{i-1}^T$ is positive semi-definite, all the α's must have the same sign (as $g_i^T\mathrm{MAT}(v_1)g_i$). Likewise, all the β's must have the same sign. We are going to show that it is possible to modify the matrices, while preserving minimality of the Frobenius norm, so that the sign of the α's is equal to that of the β's.

To this end, we first observe that

$$\bar{A}_{i-1}^{\mathrm{KPSVD}} \otimes G_i^{\mathrm{KPSVD}} = (U^T D U) \otimes (V^T E V) = (U \otimes V)^T (D \otimes E)(U \otimes V),$$

which leads us to introduce

$$C = (U \otimes V) F_{i,i}\,(U \otimes V)^T.$$

By unitary invariance of the Frobenius norm, we have

$$\|F_{i,i} - \bar{A}_{i-1}^{\text{KPSVD}} \otimes G_i^{\text{KPSVD}}\|_F^2 = \|(U \otimes V)^T (C - D \otimes E)(U \otimes V)\|_F^2 = \|C - D \otimes E\|_F^2.$$

The last quantity can be expressed as

$$\|C - D \otimes E\|_F^2 = \sum_{\omega=1}^{d_i(d_{i-1}+1)} (C_{\omega,\omega} - (D \otimes E)_\omega)^2 + \sum_{\xi \neq \eta} C_{\xi,\eta}^2 = \sum_{\omega=1}^{d_i(d_{i-1}+1)} (C_{\omega,\omega} - \alpha_{\mu(\omega)}\beta_{\tau(\omega)})^2 + \sum_{\xi \neq \eta} C_{\xi,\eta}^2,$$

where $\mu(\omega) \in \{1, \ldots, d_{i-1}+1\}$ and $\tau(\omega) \in \{1, \ldots, d_i\}$ can be uniquely determined[2] from $\omega \in \{1, \ldots, d_i(d_{i-1}+1)\}$ in such a way that $\omega = (\mu(\omega) - 1)d_i + \tau(\omega)$. Since the α's have the same sign and the β's have the same sign, the $\alpha_{\mu(\omega)}\beta_{\tau(\omega)}$'s appearing in the above equality must all have the same sign. On the other hand, because $F_{i,i}$ is positive semi-definite, C is also positive semi-definite, which implies that $C_{\omega,\omega} \geq 0$.

If $\alpha_{\mu(\omega)}\beta_{\tau(\omega)} \geq 0$ for all ω, we have what we claim. Assume that $\alpha_{\mu(\omega)}\beta_{\tau(\omega)} \leq 0$ for all ω. Then, it is readily checked that for all ω,

$$|C_{\omega,\omega} + \alpha_{\mu(\omega)}\beta_{\tau(\omega)}|^2 \leq |C_{\omega,\omega} - \alpha_{\mu(\omega)}\beta_{\tau(\omega)}|^2.$$

This means that if we set, for instance,

$$R = U^T D U = \bar{A}_{i-1}^{\text{KPSVD}}, \qquad S = V^T(-E)V = -G_i^{\text{KPSVD}},$$

then

$$\|F_{i,i} - R \otimes S\|_F^2 \leq \|F_{i,i} - \bar{A}_{i-1}^{\text{KPSVD}} \otimes G_i^{\text{KPSVD}}\|_F^2.$$

If the inequality were strict, minimality of $(\bar{A}_{i-1}^{\text{KPSVD}}, G_i^{\text{KPSVD}})$ would be contradicted. Therefore, we must have equality. This entails that (R, S) is another minimizer for which the eigenvalues of R have the same sign as those of S. In such a case, we select this pair for the factors $(\bar{A}_{i-1}^{\text{KPSVD}}, G_i^{\text{KPSVD}})$. This procedure allows us to assume that the α's and the β's all have the same sign. In other words, either both matrices are positive semi-definite or both of them are negative semi-definite. Since

$$\bar{A}_{i-1}^{\text{KPSVD}} \otimes G_i^{\text{KPSVD}} = (-\bar{A}_{i-1}^{\text{KPSVD}}) \otimes (-G_i^{\text{KPSVD}}),$$

we have the freedom to choose the sign so that both matrices are positive semi-definite. □

B Algorithms

B.1 Power SVD algorithm

Algorithm to compute the dominant singular value $\sigma_1 = \sigma_{\max}$ of a real rectangular matrix and associated right and left singular vectors.

Algorithm 1: SVD Power algorithm

Input: $A \in \mathbb{R}^{m \times n}$, $v^{(0)} \in \mathbb{R}^m$, ϵ (precision), $k_{\max}$ (maximum iteration).
Output: σ_1, u_1 and v_1 ($Av_1 = \sigma_1 u_1$, $A^T u_1 = \sigma_1 v_1$

for $k = 1, 2, \ldots, k_{\max}$ **do**
  $w^{(k)} = Av^{(k-1)}$; $u^{(k)} = w^{(k)}/\|w^{(k)}\|_2$;
  $z^{(k)} = A^T u^{(k)}$; $v^{(k)} = z^{(k)}/\|z^{(k)}\|_2$;
  $\sigma^{(k)} = \|z^{(k)}\|_2$;
  $\texttt{error} = \|Av^{(k)} - \sigma^{(k)} u^{(k)}\|_2$;
  if $\texttt{error} \leq \epsilon$ **then**
    Break;
  end
end

[2]The solution is given by $\mu(\omega) = 1 + \lfloor(\omega - 1)/d_i\rfloor$ and $\tau(\omega) = d_i\lfloor(\omega - 1)/d_i\rfloor$, where $\lfloor\cdot\rfloor$ is the integer part, but this does not matter here.

B.2 Lanczos bidiagonalization algorithm

Algorithm 2: Lanczos bidiagonalization algorithm

Input: $A \in \mathbb{R}^{m\times n}$, $q^{(0)} \in \mathbb{R}^m$, $\|q^{(0)}\| = 1$, K (dimension of Krylov subspace), ϵ (precision)
Output: Matrices $P \in \mathbb{R}^{n\times K}$, $Q \in \mathbb{R}^{m\times K}$, $H \in \mathbb{R}^{K\times K}$ and $P \in \mathbb{R}^{K\times K}$
Start:
$w^{(0)} = Aq^{(0)}$
$\alpha^{(0)} = \|w^{(0)}\|$
$p^{(0)} = w^{(0)}/\alpha^{(0)}$
$H[0,0] = \alpha_0$
$P[:,0] = p^{(0)}$
$Q[:,0] = q^{(0)}$
for $k = 0, 1, \ldots, K-1$ **do**
  $z^{(k)} = A^T p^{(k)} - \alpha^{(k)} q^{(k)}$
  $\beta^{(k)} = \|z^{(k)}\|$
  if $\beta^{(k)} \leq \epsilon$ **then**
    Break
  else
    $q^{(k+1)} = z^{(k)}/\beta^{(k)}$;
    $w^{(k+1)} = Aq^{(k+1)} - \beta^{(k)} p^{(k)}$;
    $\alpha^{(k+1)} = \|w^{(k+1)}\|$;
    $p^{(k+1)} = w^{(k+1)}/\alpha^{(k+1)}$;
    $H[k+1,k+1] = \alpha^{(k+1)}$;
    $H[k,k+1] = \beta^{(k)}$;
    $P[:,k+1] = p^{(k+1)}$;
    $Q[:,k+1] = q^{(k+1)}$;
  end
end

Consider input A and outputs H, P, Q of Algorithm **??**. The truncated SVD factorization of H yields the rank-k approximation

$$H \approx X_k \Sigma_k Y_k^T = \sum_{i=1}^{k} \sigma_i x_i y_i^T, \qquad \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_K.$$

Let $U_k = PX_k$ and $V_k = QY_k$. Then,

$$A \approx A_k = U_k \Sigma_k V_k^T$$

is a rank-k approximation of A (Golub & Kahan, 1965).

C Computational costs

Here we estimate the computation costs required to compute $\hat{F}$ (estimate of F), $\hat{F}^{-1}$ and $\hat{F}^{-1}\nabla h$ of our proposed methods compared to KFAC. We recall that here d denotes the number of neurons in each layer, ℓ denotes the number of network layers and m the mini-batch size. Table 1 summarizes orders of computational costs required by each method. We did not include forwards and backwards/additional backwards costs as they are the same for all methods. K is the dimension of Krylov subspace in Lanczos bi-diagonalization algorithm (see §B.2). k_1 and k_2 represent the number of iterations at which the corresponding algorithm has converged (power SVD or Lanczos bi-diagonalization algorithm). In our experiments, we found that they are of the order of tens. As for c_1 and c_2 they denote implementation constants.

As we can see in Table 1, our proposed methods are of the same order of magnitude as KFAC in terms of computation costs.

Table 1: Range of the computational costs per update.

	$\hat{F}$	$\hat{F}^{-1}$	$\hat{F}^{-1}\nabla h$
KFAC	$2\ell md^2$	$2\ell d^3$	$2\ell d^3$
KPSVD	$4k_1\ell md^2$	$2\ell d^3$	$2\ell d^3$
DEFLATION	$2\ell md^2 + 4k_1\ell md^2$	$c_1\ell d^3$	$c_2\ell d^3$
LANCZOS	$4k_2\ell md^2 + \ell K^3 + 2\ell Kd^2$	$c_1\ell d^3$	$c_2\ell d^3$
KFAC-CORRECTED	$2\ell md^2 + 4k_1\ell md^2$	$c_1\ell d^3$	$c_2\ell d^3$

Explanation of the entries of Table 1

- **KFAC**: To compute $\hat{F}$, we need to compute 2ℓ terms $\bar{A}_{i-1} = \mathbb{E}[\bar{a}_{i-1}\bar{a}_{i-1}^T]$ and $G_i = \mathbb{E}[g_i g_i^T]$ of computational costs $O(md^2)$ each. For $\hat{F}^{-1}$, the inverses of the ℓ pairs $\bar{A}_{i-1}$ and G_i are required. The computational cost of each $\bar{A}_{i-1}^{-1}$ or G_i^{-1} is $O(d^3)$. As for $\hat{F}^{-1}\nabla h$, we need to perform ℓ matrix-matrix multiplications $G_i^{-1}\nabla_W h A_{i-1}^{-1}$ (see equation (2)).
- **KPSVD**: The computation of $\hat{F}$ requires to apply the power SVD algorithm. if k_1 is the iteration number of convergence, then for each layer i, we need to perform k_1 matrix-vector multiplications $\mathcal{Z}(F_{i,i})v = \mathbb{E}[(g_i^T V g_i)\text{vec}(\bar{a}_{i-1}\bar{a}_{i-1}^T)]$ and $\mathcal{Z}(F_{i,i})^T u = \mathbb{E}[(\bar{a}_{i-1}^T U \bar{a}_{i-1})\text{vec}(g_i g_i^T)]$. The computational cost of $\mathcal{Z}(F_{i,i})v$ or $\mathcal{Z}(F_{i,i})^T u$ is $O(md^2)$. The computational costs required for $\hat{F}^{-1}$ and $\hat{F}^{-1}\nabla h$ are the same as in KFAC.
- **DEFLATION and KFAC-CORRECTED**: The computation of $\hat{F}$ is a combination of the computation of $\hat{F}$ in KFAC and in KPSVD so the complexity is the sum of the complexity in KFAC and KPSVD. As for $\hat{F}^{-1}$ and $\hat{F}^{-1}\nabla h$ the technique described in subsection 3.4 is used and the complexities are $O(c_1\ell d^3)$ for $\hat{F}^{-1}$ (SVD and matrix-matrix multiplications) and $O(c_2\ell d^3)$ for $\hat{F}^{-1}\nabla h$ (matrix-matrix multiplications).
- **LANCZOS**: To compute $\hat{F}$, the Lanczos bi-diagonalization algorithm is applied for each layer. Like in KPSVD, if k_2 is the iteration number of convergence then k_2 $\mathcal{Z}(F_{i,i})v$ and $\mathcal{Z}(F_{i,i})^T u$ (in $O(md^2)$ each) were necessary for each layer. At the end Lancozs bi-diagonalization algorithm, we need to perform for each layer, the SVD of matrix $H \in \mathbb{R}^{K\times K}$ (in $O(K^3)$) and matrix-matrix operations PX_k (in $O(Kd^2)$) and QY_k (in $O(Kd^2)$).The computational costs required for $\hat{F}^{-1}$ and $\hat{F}^{-1}\nabla h$ are the same as in DEFLATION or KFAC-CORRECTED.

D Network architectures and Datasets

We describe here the datasets and network architectures (Hinton & Salakhutdinov, 2006) used in our tests.

- **Auto-encoder problem 1**
  - Network architecture: $784 - 1000 - 500 - 250 - 30 - 250 - 500 - 1000 - 784$
  - Activations functions: ReLU $-$ ReLU $-$ ReLU $-$ ReLU $-$ ReLU $-$ ReLU $-$ ReLU $-$ Sigmoid
  - Data : MNIST (images of shape 28×28 of handwritten digits. 50000 training images and 10000 validation images).
  - Loss function: binary cross entropy
- **Auto-encoder problem 2**
  - Network architecture: $625 - 2000 - 1000 - 500 - 30 - 500 - 1000 - 2000 - 625$
  - Activation functions: ReLU $-$ ReLU $-$ ReLU $-$ ReLU $-$ ReLU $-$ ReLU $-$ ReLU $-$ Linear
  - Data : FACES (images of shape 25×25 people. 82800 training images and 20700 validation images).
  - Loss function: mean square error.
- **Auto-encoder problem 3**
  - Network architecture: $784 - 400 - 200 - 100 - 50 - 25 - 6 - 25 - 50 - 100 - 200 - 400 - 784$
  - Activations functions: ReLU $-$ ReLU $-$ ReLU $-$ ReLU $-$ ReLU $-$ ReLU $-$ ReLU $-$ ReLU $-$ ReLU $-$ ReLU $-$ Sigmoid
  - Data : CURVES (images of shape 28×28 of simulated handdrawn curves. 16000 training images and 4000 validation images).
  - Loss function: binary cross entropy.

E Gradient clipping

We applied the KL-clipping technique (Ba et al., 2017): after preconditioning the gradients, we scaled them by a factor ν given by

$$\nu = \min\left(1, \sqrt{\frac{c}{\sum_{i=1}^{\ell} |\mathcal{G}_i^T \nabla h(W_i)|}}\right),$$

where $\mathcal{G}_i$ denotes the preconditioned gradient and c is a constant that represents the maximum clipping parameter.