OPT2020: 12th Annual Workshop on Optimization for Machine Learning

Two-Level K-FAC Preconditioning for Deep Learning

Nikolaos Tselepidis NTSELEPIDIS@STUDENT.ETHZ.CH
Jonas Kohler JONAS.KOHLER@INF.ETHZ.CH
Antonio Orvieto ANTONIO.ORVIETO@INF.ETHZ.CH
ETH Zurich, Switzerland

Abstract

In the context of deep learning, many optimization methods use gradient covariance information in order to accelerate the convergence of Stochastic Gradient Descent. In particular, starting with Adagrad [10], a seemingly endless line of research advocates the use of diagonal approximations of the so-called empirical Fisher matrix in stochastic gradient-based algorithms, with the most prominent one arguably being Adam [15]. However, in recent years, several works cast doubt on the theoretical basis of preconditioning with the empirical Fisher matrix [17, 20, 25], and it has been shown that more sophisticated approximations of the actual Fisher matrix more closely resemble the theoretically well-motivated Natural Gradient Descent [2]. One particularly successful variant of such methods is the so-called K-FAC optimizer [21], which uses a Kronecker-factored block-diagonal Fisher approximation as preconditioner. In this work, drawing inspiration from two-level domain decomposition methods used as preconditioners in the field of scientific computing, we extend K-FAC by enriching it with off-diagonal (i.e. global) curvature information in a computationally efficient way. We achieve this by adding a coarse-space correction term to the preconditioner, which captures the global Fisher information matrix at a coarser scale. We present a small set of experimental results suggesting improved convergence behaviour of our proposed method.

1. Introduction

The question of how to efficiently incorporate curvature information into neural network training has been a long-standing issue in machine learning research. In theory, the use of Hessian information allows to effectively escape saddle points, and improves both local and global convergence rates when used in a regularized Newton framework [7]. However, the obvious drawback of Newton-type methods is that the computation per update step scales unfavorably with the problem dimension d, which can be extremely large in modern deep learning architectures. Although most proposed second-order methods employ Krylov subspace iterations to compute their updates (e.g. [16, 31, 32]), thus making use of efficiently computable Hessian-vector products [27], the worst-case per-iteration complexity still scales as $\mathcal{O}(d^2)$, which is prohibitively large compared to first-order methods. As a result, second-order optimizers are usually much slower in terms of runtime when compared to first-order optimizers (see e.g. [1, 33]). The same bottleneck can be found in a concurrent line of research which proposes the use of generalizations of the Gauss-Newton matrix (GGN) [29] instead of the Hessian (e.g. [8, 19, 25]). These two matrices match asymptotically in zero-residual non-linear least squares problems [24]. Furthermore, the GGN resembles the well-known Fisher information matrix, used in Natural Gradient Descent (NGD), in many neural network settings [2, 20]. GGN-vector products are also computed in $\mathcal{O}(d)$ [29], but since again up to d might be needed per iteration, the total per-step complexity remains $\mathcal{O}(d^2)$.

As a result of the high per-iteration costs, researchers have advocated the use of approximations of the Hessian (or GGN) matrix with algorithms ranging from quasi-Newton [5] to sketched Newton [28] and diagonal approximations [18][1]. The arguably most sophisticated and at the same time most performant method that emerged from this line of research is the so-called K-FAC algorithm [21]. This method makes use of a block-diagonal approximation of the Fisher matrix, which can be used very efficiently within Levenberg-Marquardt schemes, thanks to a Kronecker-factored form that allows inexpensive matrix inversion. K-FAC, and some recent variants such as [11], have been applied successfully for training all kinds of neural networks from Autoencoders [21] over ResNets and CNNs [12] to RNNs [22] and even transformers [34]. The block-diagonal K-FAC approximation has been shown to achieve a good balance between quality of curvature approximation and computational work. As a result, the use of K-FAC preconditioner not only speeds up SGD in terms of iterations, but also in terms of wall-clock time, which is arguably what matters most to practitioners [34]. Nonetheless, the curvature signaled by K-FAC is missing any off-diagonal (cross-layer) information, which suggests possible improvements, especially for very deep networks.

To overcome this drawback, we propose a method for incorporating cross-layer information to the block-diagonal K-FAC approximation. Our approach is inspired from coarse-grid correction techniques that have been widely used in the field of scientific computing [9, 14, 23, 30]. These methods have been shown to improve the convergence behaviour of domain decomposition preconditioners (i.e. block-diagonal approximations) when the number of subdomains (i.e. blocks) is increased [9, 14, 23, 30]. Based on this idea, we introduce a second level to the existing block-diagonal approximation that can effectively capture cross-layer information at a coarser scale.

In Section 2, we give a brief overview of Natural Gradient Descent (NGD) and K-FAC optimizer. In Section 3, we present our two-level approach for enriching K-FAC with off-diagonal covariance information, along with implementation details. In Section 4, we present preliminary experimental results, showing that capturing global covariance information at a coarse scale can indeed improve the convergence of K-FAC in very deep networks.

2. Background on K-FAC Optimizer

Natural Gradient Descent. While standard Gradient Descent (GD) follows the directions of steepest descent in the parameter space, Natural Gradient Descent (NGD) preconditions the gradient using the Fisher information matrix in order to proceed along the directions of the steepest descent in the distribution space, with metric induced by the KL divergence. More precisely, given tuples of labeled data $(x, y) \sim Q_{x,y}$, as well as an underlying parametric probabilistic model $p(y|x, \theta)$

1. In some sense, most of the well known adaptive gradient methods such as RMSprop and Adam, can be ranked among such methods, but one must note that the applied diagonal preconditioners are approximations of the *empirical* Fisher which may differ strongly from the real Fisher [17].

(e.g. a neural network), the Fisher information matrix (often denoted by F, for a formal definition see [19]) describes the local curvature of the KL divergence between $p(x, y|\theta)$ and $p(x, y|\theta + \delta)$, in the sense that $\mathrm{KL}(p(x, y|\theta)|p(x, y|\theta + \delta)) = \frac{1}{2}\delta^\intercal F\delta + \mathcal{O}(\delta^3)$. NGD can be written as an iterative procedure with the update rule: $\theta^{t+1} \leftarrow \theta^t - \eta F^{-1}\nabla_\theta \mathcal{L}(\theta^t)$, where θ^t are the model parameters at iteration t, $\mathcal{L}$ is the objective we aim to optimize, and $\eta > 0$ is the learning rate.

NGD and K-FAC as Second-Order Methods. Interestingly, the Fisher information matrix coincides with the Generalized Gauss-Newton (GGN) matrix in many neural network settings [20, 25]. Since the GGN can be seen as a positive definite approximation of the Hessian[2], K-FAC is often regarded as a second-order algorithm that leverages curvature information. In this regard, K-FAC computes an estimate of the natural gradient $F^{-1}\nabla\mathcal{L}$, using a block-diagonal approximation $\hat{F}$ of the Fisher F, where each block can be expressed as the Kronecker product of two factors of reduced order. This factorization leads to substantial savings in computation and memory, and thus yields a highly efficient approximate second-order method with inherent parallelism [4, 21].

2.1. K-FAC on Neural Networks

Let us now consider the case of training a feed-forward neural network using K-FAC.

Notation. Such a network is a function $f : \mathbb{R}^{d_0} \to \mathbb{R}^{d_L}$, parametrized by θ, that maps a given input $a_0 \in \mathbb{R}^{d_0}$ to an output $a_L \in \mathbb{R}^{d_L}$, through a sequence of affine, and element-wise non-linear transformations. In compact form, a feed-forward neural network can be written as $f(x, \theta) = W_L \cdot \varphi_{L-1}\left(W_{L-1} \cdot \varphi_{L-2}\left(\ldots W_2 \cdot \varphi_1\left(W_1 \cdot x\right)\right)\ldots\right)$, where $W_i \in \mathbb{R}^{d_i \times d_{i-1}}, i = 1, \ldots, L$, denote the affine maps, and φ_i are the element-wise non-linearities, also termed as activations. For a given layer i, we denote its pre- and post-activations by $s_i = W_i \cdot \bar{a}_{i-1}$ and $a_i = \varphi_i\left(s_i\right)$, respectively. $\bar{a}_i$ is formed by appending to a_i an homogeneous coordinate with value one, so that every affine transformation can be expressed as a single matrix-vector product. We gather all model parameters in a vector θ, which is defined as $\theta = [\mathrm{vec}(W_1)^\intercal, \mathrm{vec}(W_2)^\intercal, \ldots, \mathrm{vec}(W_L)^\intercal]^\intercal$. As usual, the operator $\mathrm{vec}(\cdot)$ vectorizes matrices by stacking their columns together. Moreover, let $\mathcal{L}$ denote a loss function of the form $\mathcal{L}\left(y, f(x, \theta)\right) = -\log r(y|f(x, \theta))$, which is associated with a predictive distribution $R_{y|f(x,\theta)} := P_{y|x}(\theta)$ used at the model's output, as well as the model distribution $P_{x,y}(\theta)$, with r and p being the respective probability density functions. Examples of such functions are the standard least-squares-, as well as the cross-entropy loss (for a proof, see [20]).

Kronecker-Factored Approximate Fisher. Let us denote with $\mathcal{D}v$ the gradient of the loss $\mathcal{L}$ with respect to the quantity v. The Fisher information matrix of a neural network parametrized by θ can be written as $F = E_{P_{y|x}(\theta),Q_x}\left[\mathcal{D}\theta\mathcal{D}\theta^\intercal\right]$. The layer-wise ordering of the parameters in θ induces an $L \times L$ block structure of the Fisher matrix F, where the (i, j)-th block is defined as $F_{i,j} = E\left[\mathrm{vec}(\mathcal{D}W_i)\mathrm{vec}(\mathcal{D}W_j)^\intercal\right]$. Here, DW_i is the gradient of $\mathcal{L}$ with respect to the weights W_i of the i-th layer of the network, and $g_i = \mathcal{D}s_i$ denotes the associated back-propagated loss derivatives, i.e. $\mathcal{D}W_i = g_i\bar{a}_{i-1}^\intercal$. Using Kronecker products, it can be shown (see [21]) that each Fisher block

2. In fact, it is the Hessian of the local linearization of $\mathcal{L}$ as populated in the NTK literature [13]. Furthermore, it has been shown that in networks with piecewise linear activations, the GGN and Hessian agree on the diagonal blocks [6].

can be rewritten as:

$$F_{i,j} = E\left[(\bar{a}_{i-1} \otimes g_i)(\bar{a}_{j-1} \otimes g_j)^\intercal\right] = E\left[(\bar{a}_{i-1} \otimes a_{j-1}^\intercal)(g_i \otimes g_j^\intercal)\right]. \tag{1}$$

For the derivation of K-FAC, it is assumed that the products of the input activations are statistically independent with the products of the back-propagated derivatives [21]. Hence, $F_{i,j}$ is approximated by $\tilde{F}_{i,j}$ as follows:

$$\tilde{F}_{i,j} = E\left[\bar{a}_{i-1}\bar{a}_{j-1}^\intercal\right] \otimes E\left[g_i g_j^\intercal\right] = \bar{A}_{i-1,j-1} \otimes G_{i,j}, \tag{2}$$

where $\bar{A}_{i,j} = E[\bar{a}_i \bar{a}_j^\intercal]$ and $G_{i,j} = E[g_i g_j^\intercal]$. Next, in order to efficiently compute $\tilde{F}^{-1}$, K-FAC approximates $\tilde{F}$ either as block-diagonal or as block-tridiagonal, leading to two different variants [21]. In our work, we only consider the block-diagonal variant, which has attracted the interest of the deep learning community because of its inherent parallelism [4]. In this case, the inverse of every diagonal block $\tilde{F}_{i,i}$ is computed as $\tilde{F}_{i,i}^{-1} = \bar{A}_{i-1,i-1}^{-1} \otimes G_{i,i}^{-1}$, without the need of explicitly forming and inverting $\tilde{F}_{i,i}$, hence reducing the computational work and memory requirements.

3. Two-Level K-FAC

In order to compute an estimate of the natural gradient $F^{-1}\nabla\mathcal{L}$, the original one-level K-FAC utilizes a block-diagonal approximation $\hat{F}^{-1}$ of $\tilde{F}^{-1}$ (see equation (2)), and computes:

$$F^{-1}\nabla\mathcal{L} \approx \hat{F}^{-1}\nabla\mathcal{L} = \text{diag}\left(\tilde{F}_{1,1}^{-1}, \tilde{F}_{2,2}^{-1}, \ldots, \tilde{F}_{L,L}^{-1}\right)\nabla\mathcal{L}. \tag{3}$$

Approximating $\tilde{F}^{-1}$ as block-diagonal is equivalent to approximating $\tilde{F}$ as block-diagonal. Therefore, the original one-level K-FAC utilizes only the intra-layer approximate covariances $\tilde{F}_{i,i}$, and ignores all off-diagonal blocks that represent the inter-layer covariances $\tilde{F}_{i,j},\ i \neq j$. In this section, motivated from this observation, we propose a method to incorporate inter-layer information into the one-level K-FAC preconditioner, in an attempt to improve convergence behaviour, especially for cases where cross-layer information is very important, such as (presumably) in very deep networks.

K-FAC as a Subspace Projection Method. Let us consider a neural network f as in Section 2.1, with L layers, and n_i weights per layer, including the bias terms. We denote the total number of parameters of the network with $n = \sum_{i=1}^{L} n_i$. Let us consider the restriction matrices $V_i \in \{0,1\}^{n_i \times n}$ that project a vector of $\mathbb{R}^n$ onto the layer $i,\ i = 1, \ldots, L$, respectively. Each restriction matrix V_i is comprised of the subset of the rows e_j of the identity matrix $I \in \mathbb{R}^{n \times n}$, where the j-th parameter belongs to the i-th layer of the network. Using this notation, equation (3) can be rewritten as follows:

$$F^{-1}\nabla\mathcal{L} \approx \hat{F}^{-1}\nabla\mathcal{L} = \sum_{i=1}^{L} V_i^\intercal \tilde{F}_{i,i}^{-1} V_i \nabla\mathcal{L}. \tag{4}$$

It can be observed, that the one-level block-diagonal K-FAC preconditioner projects the gradient vector $\nabla\mathcal{L}$ on every layer independently, i.e. $y_i = V_i\nabla\mathcal{L}$, then scales the corresponding components of y_i using local curvature information computed only from the intra-layer covariances, i.e. $\hat{y}_i = \tilde{F}_{i,i}^{-1} y_i$, and finally back-projects $\hat{y}_i$ onto the network f, combining the contributions from different layers to form the approximate natural gradient $\hat{F}^{-1}\nabla\mathcal{L}$.

Enriching K-FAC with a Coarse-Space Correction. Let us now consider a restriction matrix $Z \in \{0,1\}^{L\times n}$, that projects a vector of $\mathbb{R}^n$ onto the "coarse" network f_{coarse} with L layers and only a single weight per layer. This matrix is defined as follows:

$$z_{i,j} = \begin{cases} 1 & \text{if the } j\text{-th component of } \theta \text{ belongs to the } i\text{-th layer of } f \\ 0 & \text{otherwise} \end{cases}. \tag{5}$$

To the coarse network, we can associate the coarse representation of the Fisher, $F_{\text{coarse}} = ZFZ^\intercal \in \mathbb{R}^{L\times L}$, which captures the global covariance information at a coarse scale. Since in K-FAC we consider the approximate Kronecker-factored Fisher $\tilde{F}$, we also introduce the associated approximation $\tilde{F}_{\text{coarse}} = Z\tilde{F}Z^\intercal \in \mathbb{R}^{L\times L}$, which equivalently can be rewritten as $[\tilde{F}_{\text{coarse}}]_{i,j} = \sum_{k,l}[\tilde{F}_{i,j}]_{k,l}$.

Based on this observation, we enrich K-FAC with an additional correction term, that operates on the global but coarse parameter space, capturing inter-layer covariance information, i.e.:

$$F^{-1}\nabla\mathcal{L} \approx \breve{F}^{-1}\nabla\mathcal{L} = \sum_{i=1}^{L} V_i^\intercal \tilde{F}_{i,i}^{-1} V_i \nabla\mathcal{L} + Z^\intercal \tilde{F}_{\text{coarse}}^{-1} Z \nabla\mathcal{L}. \tag{6}$$

Thus, in order to compute an estimate of the natural gradient $F^{-1}\nabla\mathcal{L}$, our two-level approach additionally projects the gradient vector $\nabla\mathcal{L}$ onto the space associated with the coarse network f_{coarse}, scales it using the inverse of the coarse but global covariance, and projects it back to the space of the fine network f, to shift the independently preconditioned gradients by a different scalar for each layer. Intuitively, the component $\sum_{i=1}^{L} V_i^T \tilde{F}_{i,i}^{-1} V_i$ can be seen as a smoother that eliminates the high frequencies of the error $||F^{-1}\nabla\mathcal{L} - \breve{F}^{-1}\nabla\mathcal{L}||$, while the component $Z^\intercal \tilde{F}_{\text{coarse}}^{-1} Z$ operates on a coarser level, capturing the global trend of the natural gradient at a low resolution, and thus effectively eliminating the low frequencies of the error, without substantially increasing the computational work. In particular, it only requires the computation and inversion of an $L \times L$ matrix, whenever the preconditioner is updated. Similar ideas have been widely used for designing scalable parallel preconditioned iterative methods for solving large sparse linear systems in the field of scientific computing [9, 14, 23, 30]. We give an efficient algorithm for computing the coarse Fisher matrix $\tilde{F}_{\text{coarse}}$, along with implementation details in Appendix A.

4. Experimental Results

In this section, we present some preliminary empirical evidence that enriching K-FAC with inter-layer information, using a coarse-space correction term, can indeed improve the approximation quality of the preconditioner and lead to faster convergence. In particular, we compare our two-level approach with the standard one-level K-FAC in two neural network settings. To put the results into perspective, we also benchmark SGD and Adam. Regarding hyperparameters, we fixed the batch size for each optimizer and then grid-searched both learning rate ($\eta \in \{10^{-2}, 10^{-3}, 10^{-4}\}$) and momentum ($\mu \in \{0, 0.9\}$). Regarding the K-FAC variants, we tried the values 10^{-2} and 10^{-3} both for the damping parameter λ and for the KL-clipping parameter κ. We trained using the PyTorch framework [26] on a single NVIDIA GTX 1080 Ti with 11GB memory.

First, we trained a deep linear network with 64 hidden layers (10 neurons per layer), and batch normalization, on five randomly generated datasets, with 10-dimensional input samples drawn from

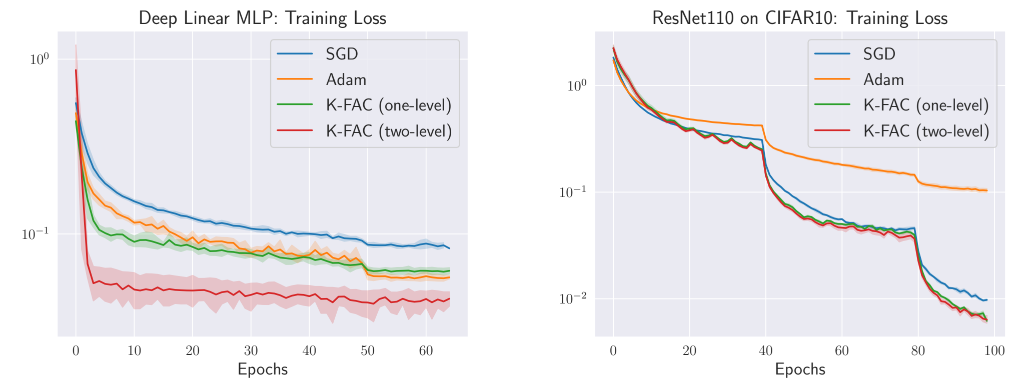


Figure 1: Convergence of SGD, Adam, as well as one- and two-level K-FAC, when training a 64-layer linear MLP with planted targets (left), and ResNet110 on CIFAR10 (right). Mean and 95% confidence interval of 5 independent runs.

a Gaussian distribution ($25k$ training and $2.5k$ test samples), and with binary targets coming from a separate randomly initialized one-layer linear network of the same width. In this experiment, all optimizers operated on mini-batches of 512 samples. Further details on parameters can be found in Appendix B.1. As can be seen in Fig. 1 (left), the proposed two-level K-FAC clearly outperforms its block-diagonal counterpart, as well as Adam and SGD. In order to generalize this finding beyond simple linear networks, we also trained a ResNet110 on CIFAR10, using the cross-entropy loss. Here, Adam and SGD were set to operate on batches of 64 samples, while one- and two-level K-FAC used batch size 128 (further details again in Appendix B.1). As can be seen in Fig. 1 (right), the coarse-space correction still works quite well in this setting, but the margin to one-level K-FAC is significantly reduced compared to case of the deep linear MLP[3].

In summary, our findings suggest that two-level K-FAC can indeed enhance convergence in deep neural networks, but further investigation is needed to identify settings where off-diagonal Hessian information is particularly useful. We consider this to be an interesting direction of future research.

5. Conclusion

We proposed a two-level extension to K-FAC that incorporates a coarse-space correction term in order to efficiently capture the global structure of the Fisher information matrix and improve the convergence behaviour of the optimizer. Our experiments show that the use of off-diagonal covariance information can indeed yield enhanced optimization performance of K-FAC in the case of (very) deep networks. Going forward, we believe that the identification of more such settings, where cross-layer information is important to consider for optimizers, is an interesting direction of future research. In particular, it is yet to be understood how advanced network architectures such as normalization layers, residual connections, and attention layers alter layer dependencies, and hence off-diagonal Hessian information.

3. At this point, we can only hypothesize, but one possible explanation could be that the network is highly over-parametrized for the simple task of CIFAR10 classification, which usually makes optimization much easier [3].

References

[1] Leonard Adolphs, Jonas Kohler, and Aurelien Lucchi. Ellipsoidal trust region methods and the marginal value of hessian information for neural network training. *arXiv preprint arXiv:1905.09201*, 2019.

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

[3] Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. *arXiv preprint arXiv:1802.06509*, 2018.

[4] Jimmy Ba, Roger Grosse, and James Martens. Distributed second-order optimization using kronecker-factored approximations, 2016.

[5] Raghu Bollapragada, Dheevatsa Mudigere, Jorge Nocedal, Hao-Jun Michael Shi, and Ping Tak Peter Tang. A progressive batching l-bfgs method for machine learning. *arXiv preprint arXiv:1802.05374*, 2018.

[6] Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical gauss-newton optimisation for deep learning. *arXiv preprint arXiv:1706.03662*, 2017.

[7] Coralia Cartis, Nicholas IM Gould, and Philippe L Toint. Adaptive cubic regularisation methods for unconstrained optimization. part ii: worst-case function-and derivative-evaluation complexity. *Mathematical programming*, 130(2):295–319, 2011.

[8] Olivier Chapelle and Dumitru Erhan. Improved preconditioner for hessian free optimization. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, volume 201. Sierra Nevada Spain, 2011.

[9] Victorita Dolean, Frédéric Nataf, Robert Scheichl, and Nicole Spillane. Analysis of a two-level schwarz method with coarse spaces based on local dirichlet-to-neumann maps. *Computational Methods in Applied Mathematics*, 12(4):391–414, 2012.

[10] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[11] Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast approximate natural gradient descent in a kronecker factored eigenbasis. In *Advances in Neural Information Processing Systems*, pages 9550–9560, 2018.

[12] Roger Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pages 573–582, 2016.

[13] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[14] Pierre Jolivet, Frédéric Hecht, Frédéric Nataf, and Christophe Prud'Homme. Scalable domain decomposition preconditioners for heterogeneous elliptic problems. *Scientific Programming*, 22(2):157–171, 2014.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Jonas Moritz Kohler and Aurelien Lucchi. Sub-sampled cubic regularization for non-convex optimization. *arXiv preprint arXiv:1705.05933*, 2017.

[17] Frederik Kunstner, Philipp Hennig, and Lukas Balles. Limitations of the empirical fisher approximation for natural gradient descent. In *Advances in Neural Information Processing Systems*, pages 4156–4167, 2019.

[18] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[19] James Martens. Deep learning via hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010.

[20] James Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

[21] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

[22] James Martens, Jimmy Ba, and Matt Johnson. Kronecker-factored curvature approximations for recurrent neural networks. In *International Conference on Learning Representations*, 2018.

[23] Artem Napov and Yvan Notay. An algebraic multigrid method with guaranteed convergence rate. *SIAM journal on scientific computing*, 34(2):A1079–A1109, 2012.

[24] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[25] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

[26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

[27] Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1): 147–160, 1994.

[28] Mert Pilanci and Martin J Wainwright. Newton sketch: A near linear-time optimization algorithm with linear-quadratic convergence. *SIAM Journal on Optimization*, 27(1):205–245, 2017.

[29] Nicol N Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738, 2002.

[30] Jok Man Tang, Reinhard Nabben, Cornelis Vuik, and Yogi A Erlangga. Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods. *Journal of scientific computing*, 39(3):340–370, 2009.

[31] Zhe Wang, Yi Zhou, Yingbin Liang, and Guanghui Lan. Stochastic variance-reduced cubic regularization for nonconvex optimization. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2731–2740. PMLR, 2019.

[32] Peng Xu, Fred Roosta, and Michael W Mahoney. Newton-type methods for non-convex optimization under inexact hessian information. *Mathematical Programming*, pages 1–36, 2019.

[33] Peng Xu, Fred Roosta, and Michael W Mahoney. Second-order optimization for non-convex machine learning: An empirical study. In *Proceedings of the 2020 SIAM International Conference on Data Mining*, pages 199–207. SIAM, 2020.

[34] Guodong Zhang, Lala Li, Zachary Nado, James Martens, Sushant Sachdeva, George Dahl, Chris Shallue, and Roger B Grosse. Which algorithmic choices matter at which batch sizes? insights from a noisy quadratic model. In *Advances in Neural Information Processing Systems*, pages 8196–8207, 2019.

Appendix A. Implementation Details

Coarse Fisher Matrix Computation. The (i,j)-th element of $\tilde{F}_{\text{coarse}}$ is equal to the sum of all elements in the (i,j)-th block of $\tilde{F}$. Moreover, every block $\tilde{F}_{i,j}$ is equal to $\tilde{F}_{i,j} = \bar{A}_{i-1,j-1} \otimes G_{i,j}$. Explicitly computing $\tilde{F}_{i,j}$, and then summing up all the elements requires substantial computational work and excessive memory requirements. To overcome this issue, we directly compute the required sum without explicitly forming $\tilde{F}_{i,j}$, as follows:

$$\sum_{k,l} \left[\tilde{F}_{i,j}\right]_{k,l} = \sum_{k} \left[\tilde{F}_{i,j} \cdot \mathbb{1}\right]_{k} = \sum_{k} \left[\left(\bar{A}_{i-1,j-1} \otimes G_{i,j}\right) \cdot \mathbb{1}\right]_{k} = \sum_{k,l} \left[G_{i,j} \cdot \mathbb{1}_{m\times n} \cdot \bar{A}_{i-1,j-1}\right]_{k,l}, \tag{7}$$

where $\mathbb{1}$ is a vector of ones, and $\mathbb{1}_{m\times n}$ is the same vector reshaped into an $m \times n$ matrix, where m and n are such that the dimensions match for the matrix multiplications. Equation (7) provides an efficient way for computing every element of $\tilde{F}_{\text{coarse}}$, and since $\tilde{F}_{\text{coarse}}$ is symmetric (similarly to $\tilde{F}$), one only needs to compute its upper or lower triangular part. The algorithm for the computation of the coarse Fisher matrix $\tilde{F}_{\text{coarse}}$ is given below:

Algorithm 1: Coarse Fisher Matrix Computation

```
in : Input activations $\bar{a}_{i-1}^{(t)}$ and back-propagated gradients $g_i^{(t)}$ for $i = 1, \ldots, L$, at iteration $t$.
out: Coarse approximate Fisher information matrix $\tilde{F}_{\text{coarse}}$
1  begin
2      Set $\epsilon = \min(1 - 1/t, 0.95)$                # statistical decay;
3      for $i = 1, \ldots, L$ do
4          for $j = 1, \ldots, i$ do
5              Compute $\bar{A}_{i-1,j-1}^{(t)} = E\left[\bar{a}_{i-1}^{(t)} (\bar{a}_{j-1}^{(t)})^\intercal\right]$   # downsample $\bar{a}_{i-1}^{(t)}$ or $\bar{a}_{j-1}^{(t)}$ if needed;
6              Update $\bar{A}_{i-1,j-1} = \epsilon \bar{A}_{i-1,j-1} + (1-\epsilon)\bar{A}_{i-1,j-1}^{(t)}$;
7              Compute $G_{i,j}^{(t)} = E\left[g_i^{(t)} (g_j^{(t)})^\intercal\right]$   # downsample $g_i^{(t)}$ or $g_j^{(t)}$ if needed;
8              Update $G_{i,j} = \epsilon G_{i,j} + (1-\epsilon) G_{i,j}^{(t)}$;
9              Compute $\left[\tilde{F}_{\text{coarse}}\right]_{i,j} = \sum_{k,l} \left[\tilde{F}_{i,j}\right]_{k,l}$ without forming $\bar{A}_{i-1,j-1} \otimes G_{i,j}$ and
                 summing up all elements, but using the formula $\sum_{k,l} \left[G_{i,j} \cdot \mathbb{1}_{m\times n} \cdot \bar{A}_{i-1,j-1}\right]_{k,l}$,
                 where $m$ and $n$ are such that the dimensions match;
10         end
11     end
12     $\tilde{F}_{\text{coarse}} = \tilde{F}_{\text{coarse}} + \tilde{F}_{\text{coarse}}^\intercal - \text{diag}(\tilde{F}_{\text{coarse}})$   # here, diag(·) yields a diagonal matrix;
13 end
```

It should be noted, that in the case of convolutional layers the dimensions of the input activations $\bar{a}_{i-1}^{(t)}$ and back-propagated gradients $g_i^{(t)}$, between different layers, i.e. for $i \neq j$, may not match, and thus the inter-layer covariances $\bar{A}_{i-1,j-1}^{(t)}$ and $G_{i,j}^{(t)}$ cannot be computed. To tackle this issue, we downsample the feature maps of larger dimensions, so that we can compute the required covariances. In our implementation, we use the nearest-neighbor downsampling algorithm as implemented in PyTorch. Finally, we need to mention that we keep running estimates of the statistics $\bar{A}_{i-1,j-1}$ and $G_{i,j}$ as shown in lines 6 and 8 of Algorithm 1.

Damping. Although, in theory the block-diagonal K-FAC preconditioner can be inverted block-wise with the inverse of each block being $\tilde{F}_{i,i}^{-1} = \bar{A}_{i-1,i-1}^{-1} \otimes G_{i,i}^{-1}$, in practice, a damping term λI is usually added to every $\tilde{F}_{i,i}$ in order to account for the inaccuracies of the approximation and ill-conditioning of the diagonal blocks. The addition of this term makes the use of the previous formula impossible. Martens and Grosse [21] proposed two methods to resolve this issue; (i) an exact method that is based on the eigenvalue decomposition of the diagonal blocks, and (ii) an approximate but more computationally efficient approach, which they refer to as factored Tikhonov regularization. In the latter case, every block $\tilde{F}_{i,i} = \bar{A}_{i-1,i-1} \otimes G_{i,i} + \lambda I$ is approximated by:

$$\tilde{F}_{i,i} \approx \left(\bar{A}_{i-1,i-1} + \pi_i \lambda^{1/2} I\right) \otimes \left(G_{i,i} + \frac{1}{\pi_i}\lambda^{1/2} I\right) \text{ where } \pi_i = \sqrt{\frac{\operatorname{tr}(\bar{A}_{i-1,i-1})/(d_{i-1}+1)}{\operatorname{tr}(G_{i,i})/d_i}}. \quad (8)$$

Here, d_i is the dimension (number of units) in layer i. Therefore, using this approach as well as the formula $(A \otimes B)\operatorname{vec}(X) = \operatorname{vec}(BXA^\intercal)$, the i-th block of the natural gradient can be computed as:

$$(\tilde{F}_{i,i} + \lambda I)^{-1} \nabla \mathcal{L}_i = \operatorname{vec}\left(\left(G_{i,i} + \frac{1}{\pi_i}\lambda^{1/2} I\right)^{-1} \nabla \mathcal{L}_i^{\text{reshaped}} \left(\bar{A}_{i-1,i-1} + \pi_i \lambda^{1/2} I\right)^{-1}\right). \quad (9)$$

Concerning the inversion of the coarse Fisher matrix $\tilde{F}_{\text{coarse}}$, we actually compute $(\tilde{F}_{\text{coarse}} + \lambda I)^{-1}$, where λ is equal to the damping parameter used when inverting the diagonal blocks.

KL-clipping. After preconditioning the gradients, we scale them by a factor ν which is given by the equation:

$$\nu = \min\left(1, \sqrt{\frac{\kappa}{\eta^2 \sum_{i=1}^{n} |\mathcal{G}_i^\intercal \nabla \mathcal{L}_i(\theta_i)|}}\right), \quad (10)$$

where $\mathcal{G}$ is the preconditioned gradient, η is the learning rate, and κ is a user defined parameter. We choose κ so that the square Fisher norm is at most κ [21].

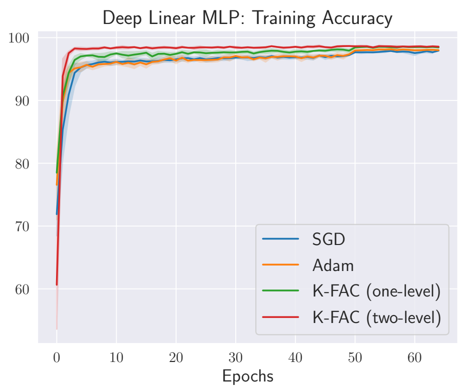
Appendix B. Experiments

B.1. Details on Settings and Parameters

Deep Linear MLP. In this experiment, all optimizers operated on mini-batches of 512 samples, with learning rate $\eta = 10^{-3}$, momentum $\mu = 0.9$, and weight decay $\beta = 10^{-3}$. The one- and two-level K-FAC, were configured so that they update the running estimates of the covariances every 10 iterations, and recompute the preconditioner every 100 iterations. Moreover, the damping parameter λ was set to 10^{-2}, and the parameter κ used for KL-clipping was set to 10^{-3}. The inverses of the diagonal blocks in one- and two-level K-FAC were computed using eigen-decomposition [21], since it seemed to be more robust for the case of the deep linear MLP.

ResNet110. Similarly to the previous experiment, we configured all optimizers to use momentum with $\mu = 0.9$ and weight decay with $\beta = 10^{-3}$. The learning rate was set to 10^{-2} for all optimizers except for Adam who seemed to work better with 10^{-3}, since setting a larger learning rate led to a substantial increase in the test loss, reducing generalization. Moreover, a learning rate schedule was chosen so that the learning rate η is reduced by a factor of 10, on epochs 40 and 80. For the one- and two-level K-FAC the damping parameter λ was set to 10^{-3}, while the KL-clipping parameter κ was set to 10^{-2}. It should be mentioned, that in order to keep the computational work at low levels, we configured one- and two-level K-FAC to update the running estimates of the covariances every 200 iterations, and recompute the preconditioner every 2000 iterations. Moreover, the inverses of the diagonal blocks were computed using the factored Tikhonov regularization technique (see Appendix A).

B.2. Additional Experimental Results



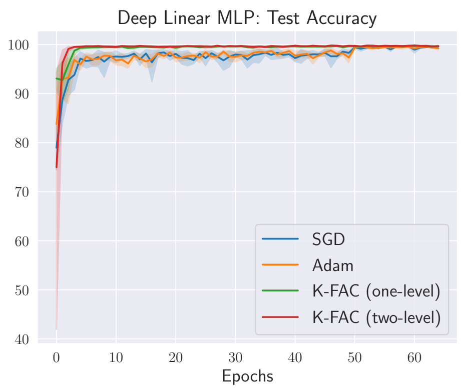
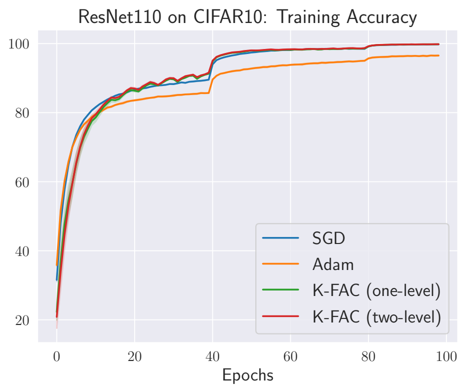


Figure 2: Training and test accuracy per epoch for SGD, Adam, as well as one- and two-level K-FAC, when training a linear 64-layer MLP with planted targets.



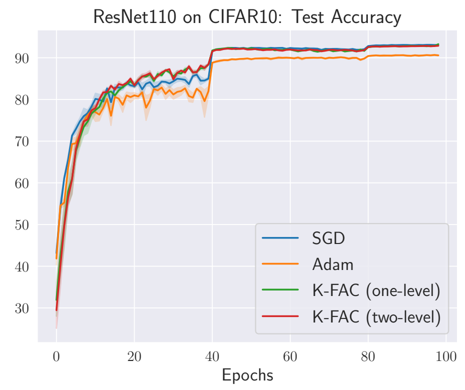


Figure 3: Training and test accuracy per epoch for SGD, Adam, as well as one- and two-level K-FAC, when training ResNet110 on CIFAR10.