Fast Approximate Natural Gradient Descent in a Kronecker-factored Eigenbasis

Thomas George*[1], César Laurent*[1], Xavier Bouthillier[1], Nicolas Ballas[2], Pascal Vincent[1,2,3]

* equal contribution

1 Mila, Université de Montréal, McGill

3 CIFAR Canadian Institute for Advanced Research

2 facebook artificial intelligence research

Overview

Preconditioned gradient descent aims at accounting for the curvature of the optimized function. Most preconditioners require inverting a large matrix, which is unfeasible for 15M parameters neural networks.

In this work, we build upon Martens and Grosse (2015) to improve the K-FAC approximation used to efficiently invert the Fisher Information Matrix. We show that our method allows to efficiently track a more accurate approximate than K-FAC, allowing for a speedup in optimization.

Prerequisite: Preconditioned gradient descent

$$\Delta_{GD} = -\lambda \mathbb{E}\left[\nabla_\theta\right]$$

$$\Delta_{PrecGD} = -\lambda \left(G + \epsilon \mathbf{I}\right)^{-1} \mathbb{E}\left[\nabla_\theta\right]$$

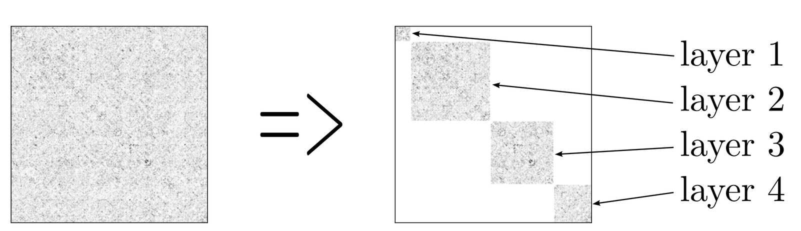
λ : Learning rate

ϵ : Tikhonov damping

G can be:

Hessian	Second moment of gradients ("empirical Fisher")
Gauss-Newton	Fisher Information Matrix

Prerequisite: Block-diagonal approximation



$$G_l = \mathbb{E}\left[\nabla_{\theta_l} \nabla_{\theta_l}^\top\right] = \mathbb{E}\left[\left(x_l \otimes g_l\right)\left(x_l \otimes g_l\right)^\top\right]$$

x_l : Activation from previous layer

g_l : Gradient with respect to outgoing preactivation $\frac{\partial \ell}{\partial h_l}$

From now on we focus on a single layer and we drop the subscript l

Prerequisite: K-FAC (Martens and Grosse 2015, Heskes 2000)

$$G = \mathbb{E}\left[xx^\top \otimes gg^\top\right] = \mathbb{E}\left[xx^\top\right] \otimes \mathbb{E}\left[gg^\top\right] + R$$

approximation error:

$$R = \mathbb{E}\left[\left(xx^\top - \mathbb{E}\left[xx^\top\right]\right) \otimes \left(gg^\top - \mathbb{E}\left[gg^\top\right]\right)\right]$$

Kronecker product property:

$$\underbrace{(A \otimes B)^{-1}}_{mn \times mn} = \underbrace{A^{-1}}_{m \times m} \otimes \underbrace{B^{-1}}_{n \times n}$$

Our PyTorch implementation of K-FAC and EKFAC is available !
https://github.com/Thrandis/EKFAC-pytorch

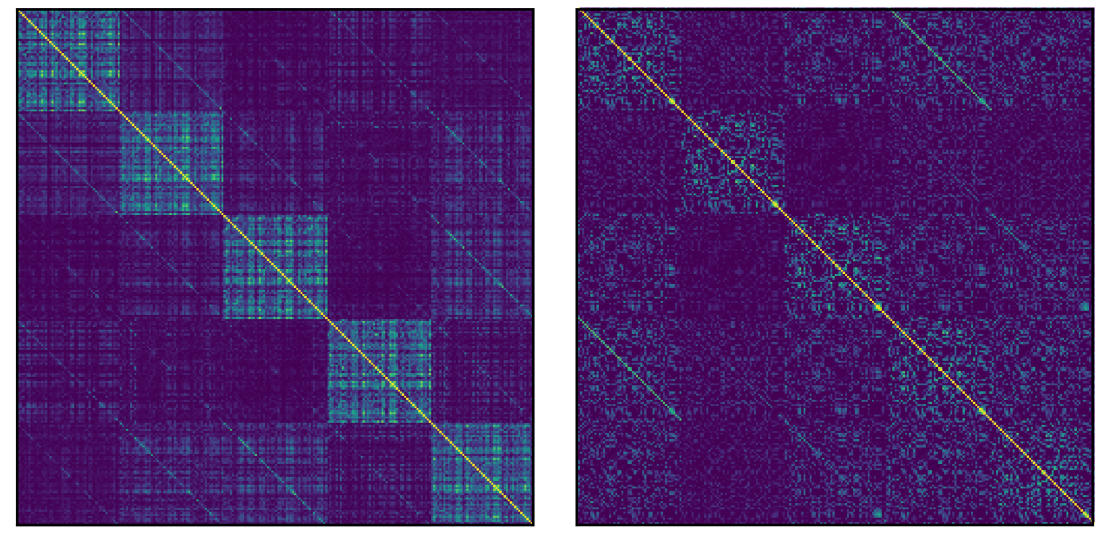
Kronecker-factored Eigenbasis (KFE)

if $A \otimes B \approx G$ then the SVD of $A \otimes B$ must be a good approximate of the SVD of G.

$$\left.\begin{aligned} A &= U_A \Lambda_A U_A^\top \\ B &= U_B \Lambda_B U_B^\top \end{aligned}\right\} \text{SVD of } A \text{ and } B$$

$$A \otimes B = \underbrace{\left(U_A \otimes U_B\right)}_{U} \left(\Lambda_A \otimes \Lambda_B\right) \left(U_A \otimes U_B\right)^\top$$

U approximate eigenbasis for G **= KFE**



Correlation of the gradients of a single layer for a MLP on MNIST

Eigenvalue corrected K-FAC (EKFAC)

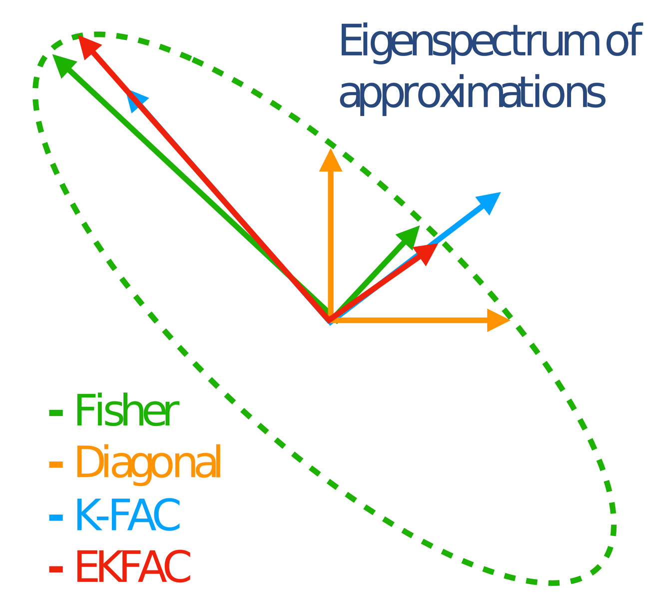
Proposal: use the KFE but rescale the eigenvalues so that they exactly match the diagonal in that basis.

$$\Lambda = \operatorname{diag}\left(U^\top G U\right)$$

diag keeps the diagonal values and puts all other elements to 0.

Theorem: It is the optimal diagonal scaling in this basis in the sense that it minimizes $\|G - U\Lambda U^\top\|_F$

Corollary: $\|G - G_{EKFAC}\|_F \leq \|G - G_{KFAC}\|_F$



Parameter update

$$\left(G + \epsilon \mathbf{I}\right)^{-1} \mathbb{E}\left[\nabla_\theta\right] = U \left(\Lambda + \epsilon \mathbf{I}\right)^{-1} U^\top \mathbb{E}\left[\nabla_\theta\right]$$

$$\begin{aligned} \Lambda &= \operatorname{diag}\left(U^\top \mathbb{E}\left[\nabla_\theta \nabla_\theta^\top\right] U\right) \\ &= \mathbb{E}\left[\operatorname{diag}\left(U^\top \nabla_\theta \nabla_\theta^\top U\right)\right] \\ &= \mathbb{E}\left[\left(U^\top \nabla_\theta\right)^2\right] \end{aligned}$$

$$\mathbb{E}\left[U^\top \nabla_\theta\right]$$

$U^\top \nabla_\theta = \tilde{\nabla}_\theta$ gradient projected in the KFE

2 interpretations:

- EKFAC is a rescaled K-FAC preconditioner in the parameter space
- EKFAC is a diagonal method in the KFE

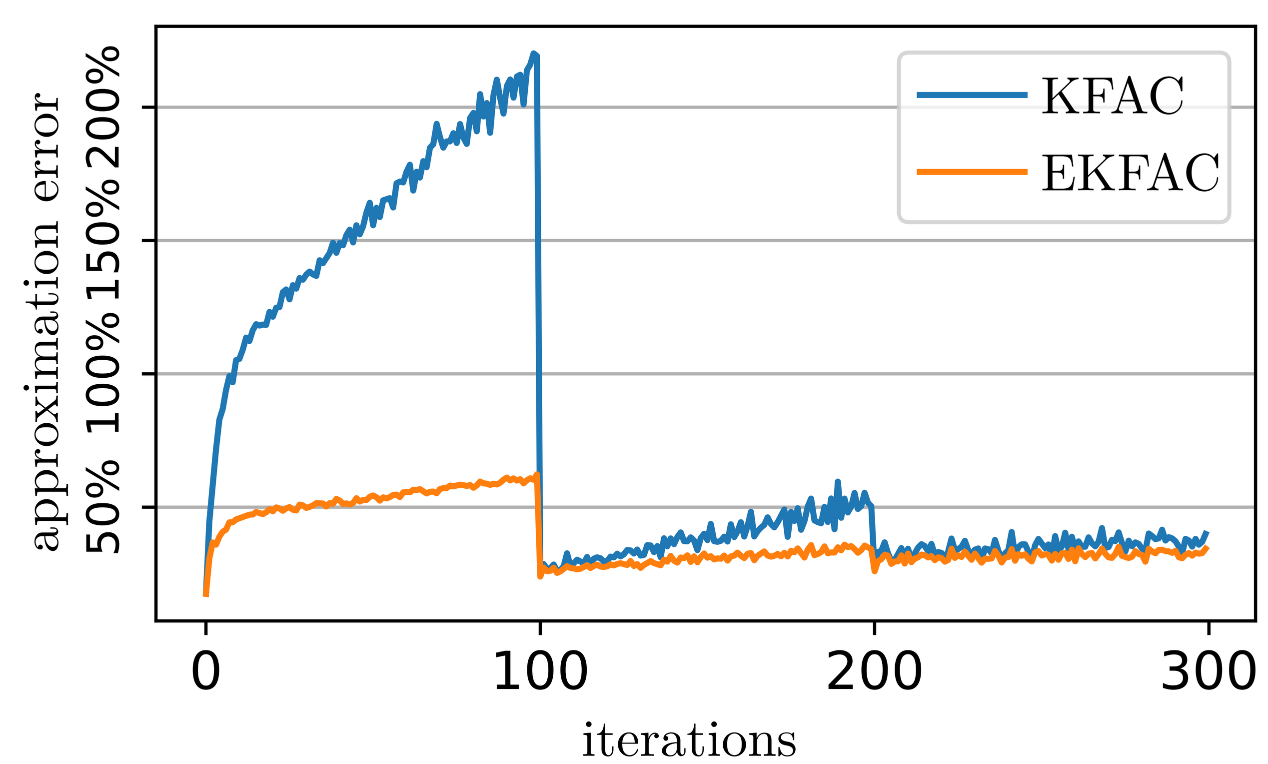
Advantages of EKFAC over K-FAC

▷ Eigendecompositions can be amortized every n updates, then Λ is cheap to compute:

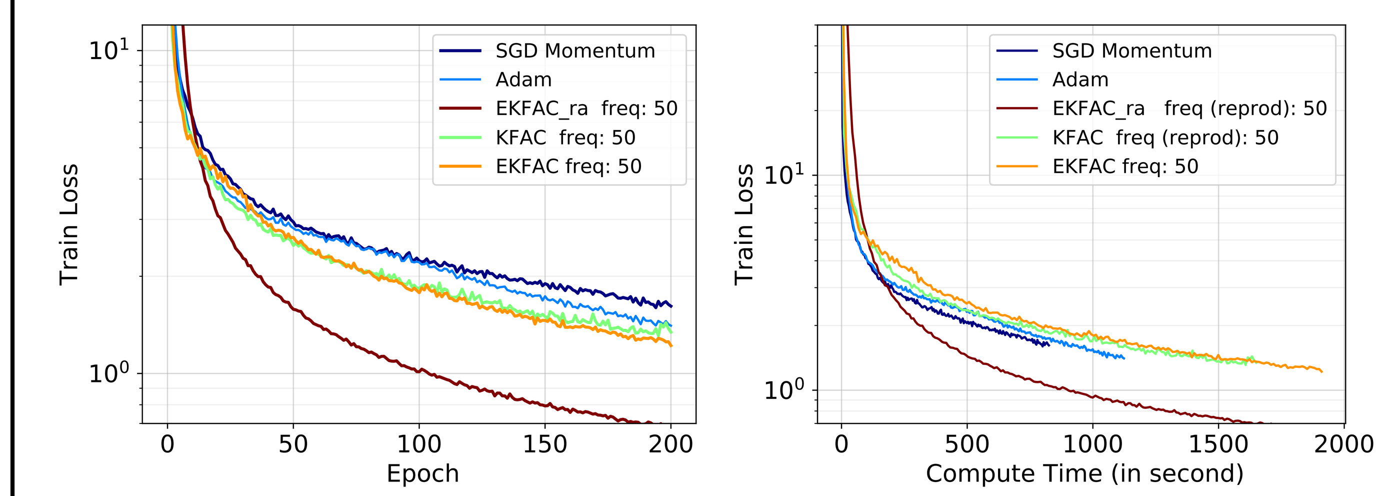
- using intra minibatch 2nd moment
- using a running average estimate

▷ Allows go keep a better approximate during training compared to amortized K-FAC

Approximation error: $\dfrac{\|G - \hat{G}\|_F}{\|G\|_F}$



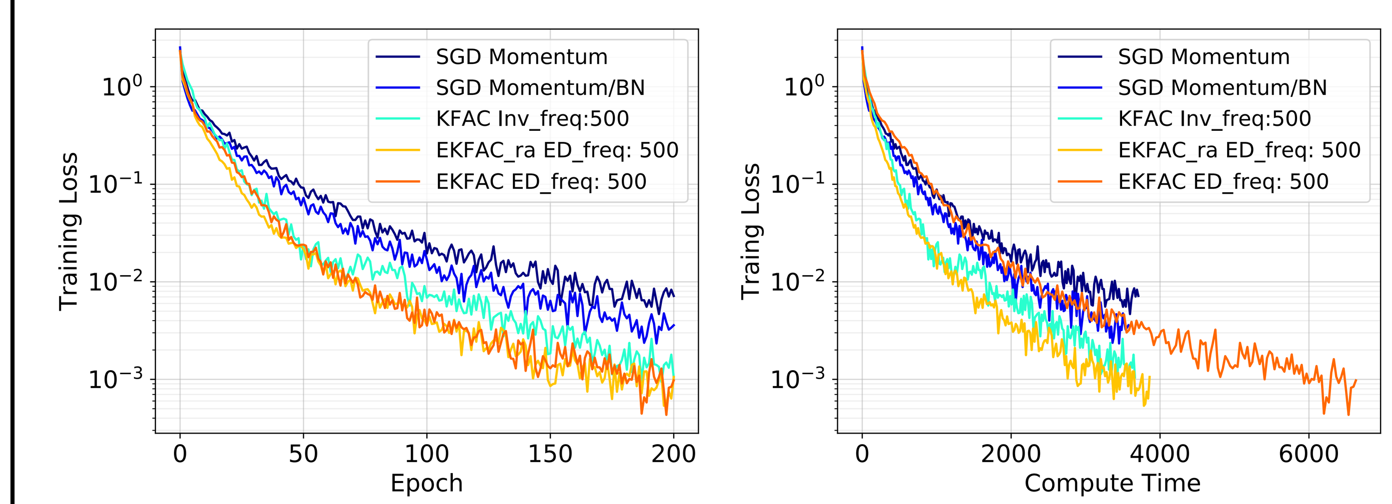
Experiment: MNIST Deep Auto-Encoder



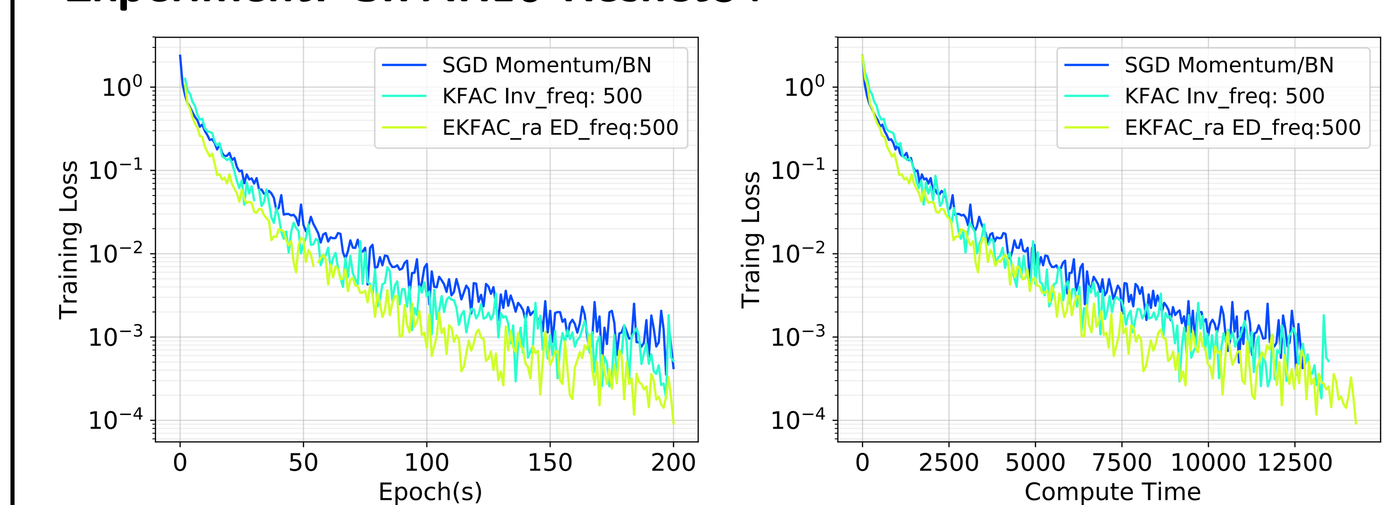
▷ Both EKFAC and EKFAC_ra show an optimization benefit compared to amortized K-FAC and other baselines

▷ This transfers to faster training (in wall clock time) for EKFAC_ra

Experiment: CIFAR10 VGG11



Experiment: CIFAR10 Resnet34



▷ We did not find one set of hyperparameter for which EKFAC is below K-FAC for all epochs (and vice versa)

▷ However, if we do the model selection for each epoch, the best EKFAC will always outperform the best K-FAC

▷ K-FAC and EKFAC are very sensitive to the learning rate and the Tikhonov damping hyperparameters.

Conclusion

▷ EKFAC can optimize networks fast succesfully without BN

▷ Future work: apply other diagonal methods in the KFE (RMSProp, SignSGD, ...)

▷ Develop regularization and improve robustness to hyperparameters

References

James Martens and Roger Grosse. "Optimizing neural networks with kronecker-factored approximate curvature." ICML 2015.

Tom Heskes. "On natural learning and pruning in multilayered perceptrons." Neural Computation, 2000