Fishy: Layerwise Fisher Approximation for Higher-order Neural Network Optimization

Anonymous Author(s)
Affiliation
Address
email

Abstract

We introduce *Fishy*, a local approximation of the Fisher information matrix at each layer for natural gradient descent training of deep neural networks. The true Fisher approximation for deep networks involves sampling labels from the model's predictive distribution at the output layer and performing a full backward pass – *Fishy* defines a Bregman exponential family distribution at each layer, performing the sampling locally. Local sampling allows for model parallelism when forming the preconditioner, removing the need for the extra backward pass. We demonstrate our approach through the Shampoo optimizer, replacing its preconditioner gradients with our locally sampled gradients. Our training results on deep autoencoder and VGG16 image classification models indicate the efficacy of our construction.

1 Introduction

Natural gradient descent (NGD) [1] is a second-order update rule that preconditions the gradient direction with the inverse of the model's Fisher information matrix (FIM). NGD corresponds to the steepest descent direction in the Riemannian space associated with the FIM and is equivariant to any differentiable reparameterization of the model weights. However, this property only holds in the small step limit and whether the benefits extend to finite steps remains an open question [14]. Nevertheless, NGD has proven to be remarkably effective in training deep neural networks, and many standard optimizers in deep learning can be seen as approximations of NGD [15, 9].

NGD is challenging to implement for training deep neural networks. Firstly the FIM requires samples from the model's predictive distribution, with an additional backward pass to calculate the gradients. Secondly, NGD requires calculating the inverse FIM of the whole network, which immediately becomes infeasible even for medium-sized models. K-FAC [15] uses a block diagonal Kronecker factor approximation to address the latter obstacle, considering the FIM for each layer as a Kronecker product of two matrices which can more easily be stored and inverted. However, the Kronecker approximation used in K-FAC becomes challenging to generalize for layers other than fully-connected layers. Moreover, the sampling and additional backward passes keep K-FAC expensive. The Shampoo optimizer [7, 5] approximates the full-matrix AdaGrad preconditioner using the same minibatch gradients of the training examples. Shampoo preconditioning avoids additional sampling or extra backward passes but is no longer related to true Fisher [12, 14] and, as we show experimentally, is outperformed when replaced with true Fisher, which is similar to [17] with several important differences.[1] Nevertheless, the Kronecker product approximation of the preconditioner in Shampoo has the advantage of being applicable to different types of layers. Sun and Nielsen [19] recently proposed a layerwise relative FIM, but do not consider the full form of layers' predictive distributions nor demonstrate scaling to practical architectures.

[1] We will provide more details regarding this approximation in Section 2.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

This paper presents a new layerwise FIM approximation approach, extending the popular Shampoo optimizer to NGD. First, we update Shampoo to an approximate NGD method, sampling from the output layer distribution to construct the preconditioners. Second, inspired by the matching loss [8, 4] construction for LocoProp and layerwise representation learning [3, 2], we define Bregman exponential family distributions for the outputs of each layer based on their activation functions. Sampling locally from these distributions allows preconditioner calculation to be performed in parallel across layers. Third, we compare our approach to standard Shampoo and its Fisher counterpart on a benchmark MNIST deep autoencoder problem and CIFAR100 classification with a VGG16 [18].

2 Methodology

Fisher Information Matrix and Natural Gradient Descent Given a probabilistic model $P(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{\theta})$ with input random variable $\boldsymbol{x}$ and target random variable $\boldsymbol{y}$, the Fisher Information Metric (FIM) is defined in terms of a local Riemannian metric as an approximation to the KL divergence [16],

$$\begin{aligned} D_{\mathrm{KL}}\big(P(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{\theta}), P(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{\theta} + \mathrm{d}\boldsymbol{\theta})\big) &= \int_{\boldsymbol{x},\boldsymbol{y}} p(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{\theta}) \log \frac{p(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{\theta})}{p(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{\theta} + \mathrm{d}\boldsymbol{\theta})} \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}\boldsymbol{y} \\ &\approx {}^1\!/_2 \, \mathrm{d}\boldsymbol{\theta}^\top \underbrace{\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}|\boldsymbol{\theta}}[\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{\theta})^\top]}_{:= \boldsymbol{F}(\boldsymbol{\theta}) \text{ Fisher Information matrix}} \mathrm{d}\boldsymbol{\theta} \,, \end{aligned}$$

where the expectation is with respect to the model distribution $P(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{\theta})$. FIM corresponds to the Riemannian metric that locally explains the model in the parameter space [13]. This probabilistic model approximates an underlying joint data distribution $P_{\mathrm{d}}(\boldsymbol{x}, \boldsymbol{y})$ from which the data is sampled. Thus, training the model corresponds to finding optimal parameters $\boldsymbol{\theta}^*$ that minimize the KL divergence between the data and model distributions. For a discriminative model that generates $P(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$, the marginal distribution $P(\boldsymbol{x})$ is assumed to be the same as the underlying input data distribution $P_{\mathrm{d}}(\boldsymbol{x})$. Thus, the loss can be written as

$$L(\boldsymbol{\theta}) = \int_{\boldsymbol{x},\boldsymbol{y}} p_{\mathrm{d}}(\boldsymbol{x}, \boldsymbol{y}) \log \frac{p_{\mathrm{d}}(\boldsymbol{x}, \boldsymbol{y})}{p(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{\theta})} \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}\boldsymbol{y} = \underbrace{\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}[\log p_{\mathrm{d}}(\boldsymbol{y}|\boldsymbol{x})]}_{\text{constant}} - \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}[\log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})] \,. \tag{1}$$

In practice, the empirical loss is formed by a Monte Carlo approximation of the expectation,

$$L(\boldsymbol{\theta}|\mathcal{X}) = -\frac{1}{n} \sum_i \log P(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)}, \boldsymbol{\theta}) \,,$$

using training examples $\mathcal{X} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^n$ that are sampled from $P_{\mathrm{d}}(\boldsymbol{x}, \boldsymbol{y})$.

Given the loss $L(\boldsymbol{\theta}|\mathcal{X})$ on the batch of examples $\mathcal{X}$, the *natural gradient* step [1] is defined as the minimizer of the following proximal objective,

$$\begin{aligned} \mathrm{d}\boldsymbol{\theta}_{\mathrm{NGD}} &= \arg\min_{\mathrm{d}\boldsymbol{\theta}} \{L(\boldsymbol{\theta}|\mathcal{X}) + \mathrm{d}\boldsymbol{\theta}^\top \nabla L(\boldsymbol{\theta}|\mathcal{X}) + {}^1\!/_{2\gamma} \, \mathrm{d}\boldsymbol{\theta}^\top \boldsymbol{F}(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta}\} \\ &= -\gamma \boldsymbol{F}(\boldsymbol{\theta})^{-1} \nabla L(\boldsymbol{\theta}|\mathcal{X}) \,, \end{aligned} \tag{2}$$

in which the loss at $L(\boldsymbol{\theta} + \mathrm{d}\boldsymbol{\theta}|\mathcal{X})$ is approximated by its first-order Taylor expansion. The non-negative multiplier $\gamma > 0$ in the regularizer term controls the step size. Specifically, Equation (2) corresponds to minimizing a linear approximation of the loss at the current parameter $\boldsymbol{\theta}$ while the amount of deviation is measured in terms of the local Mahalanobis distance induced by the FIM.

Fishy: Layerwise Fisher Construction Let f_ℓ denote the (elementwise) strictly-increasing activation function in layer $\ell \in [L]$ of an L-layer network. Fishy aims to approximate the Fisher $\boldsymbol{F}$ for each layer *independently* by defining a predictive distribution for $\boldsymbol{y}_\ell$, the layer's output random variable, as

$$p_\ell(\boldsymbol{y}_\ell|\hat{\boldsymbol{y}}_\ell) = \frac{\exp(-D_{F_\ell^*}(\boldsymbol{y}_\ell, \hat{\boldsymbol{y}}_\ell))}{Z(\hat{\boldsymbol{y}}_\ell)} \,, \tag{3}$$

where $\hat{\boldsymbol{y}}_\ell$ is the post-activation of the layer given the input $\hat{\boldsymbol{y}}_{\ell-1}$. Let $\hat{\boldsymbol{a}}_\ell$ denote the pre-activation at layer ℓ. For a fully-connected layer with weights $\boldsymbol{W}_\ell$, we can write $\hat{\boldsymbol{a}}_\ell = \boldsymbol{W}_\ell \, \hat{\boldsymbol{y}}_{\ell-1}$ and $\hat{\boldsymbol{y}}_\ell = f_m(\hat{\boldsymbol{a}}_\ell)$.

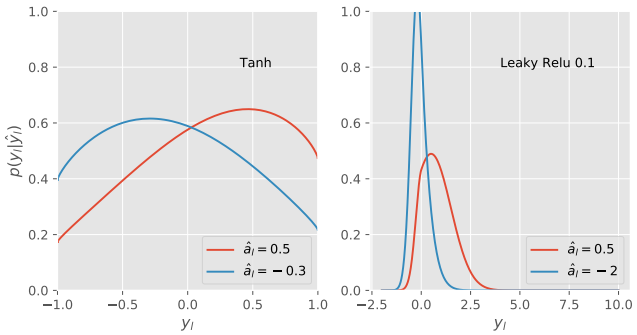

Figure 1: Example of layerwise predictive distributions for tanh and leaky relu activation functions.

Also, $Z(\hat{\boldsymbol{y}}_\ell)$ is the normalizing partition function that ensures the distribution sums to one. The distribution in Equation (3) belongs to a Bregman exponential family [6] in which the Bregman divergence is induced by the convex function F_ℓ^*. We apply the layerwise loss construction in [3] to obtain $D_{F_\ell^*}$, which corresponds to the matching loss [8, 11, 4] of the activation function f_ℓ induced by its integral function. Intuitively, $D_{F_\ell^*}$ encodes the layer's local geometry. Correcting for the gradient direction using a local Fisher approximation can stabilize training and improve convergence. For the linear, sigmoid, and softmax activations, Equation (3) yields the standard Gaussian, Bernoulli, and categorical distributions, respectively. Figure 1 gives some example distributions for tanh and leaky ReLU activations.

In order to calculate the layer FIM, we need to calculate the gradient of the *score* function $\log p_\ell(\boldsymbol{y}_\ell|\hat{\boldsymbol{y}}_\ell)$. Taking the gradient with respect to the pre-activation $\hat{\boldsymbol{a}}_\ell$, we have

$$\nabla_{\hat{\boldsymbol{a}}_\ell} \log p_\ell(\boldsymbol{y}_\ell|\hat{\boldsymbol{y}}_\ell) = \boldsymbol{y}_\ell - \hat{\boldsymbol{y}}_\ell - \nabla_{\boldsymbol{a}_\ell} \log Z(\boldsymbol{\theta}) = \boldsymbol{y}_\ell - \hat{\boldsymbol{y}}_\ell - \mathop{\mathbb{E}}_{P_\ell(\boldsymbol{y}_\ell|\hat{\boldsymbol{y}}_\ell)} [\boldsymbol{y}_\ell - \hat{\boldsymbol{y}}_\ell] \,. \tag{4}$$

The last term in Equation (4) is non-zero in general for asymmetric distributions induced by activation functions such as tanh and leaky ReLU (see Figure 1). Nevertheless, in our approximation, we discard the last term in Equation (4) to form an upper bound of the local FIM approximation. For a fully-connected layer with $\boldsymbol{\theta}_\ell = \mathrm{vec}(\boldsymbol{W}_\ell)$, we can write

$$\boldsymbol{F}(\boldsymbol{\theta}_\ell) \preccurlyeq \mathop{\mathbb{E}}_{P(\boldsymbol{x},\boldsymbol{y}_\ell)} \left[\big((\boldsymbol{y}_\ell - \hat{\boldsymbol{y}}_\ell) \otimes \hat{\boldsymbol{y}}_{\ell-1}\big) \big((\boldsymbol{y}_\ell - \hat{\boldsymbol{y}}_\ell) \otimes \hat{\boldsymbol{y}}_{\ell-1}\big)^T \right] , \tag{5}$$

where $P(\boldsymbol{x}, \boldsymbol{y}_\ell) = P_{\mathrm{d}}(\boldsymbol{x})\, P_\ell(\boldsymbol{y}_\ell|\hat{\boldsymbol{y}}_\ell(\boldsymbol{x}))$. The FIM approximation in Equation (5) corresponds to a block-diagonal approximation of the full FIM as the layers are treated independently. This is a common procedure for calculating FIM for large neural networks [15, 19]. However, instead of sampling at each layer, we can divide the network into multiple disjoint sections and perform the sampling at the output layer of each section. The sampled gradients are then backpropagated through the corresponding section. In the extreme case where the network is treated as a single section, Equation (5) reduces to the common procedure of approximating the block diagonal FIM by sampling from the model's output predictive distribution [15]. Next, we discuss the Shampoo approximation to apply the inverse of Equation (5) as the preconditioner at each layer.

Fishy via Shampoo Approximation For a fully-connected layer with gradient $\boldsymbol{G} \in \mathbb{R}^{d_i \times d_o}$, Shampoo approximates the outer-product of the vectorized gradients $\boldsymbol{g} = \mathrm{vec}(\boldsymbol{G}) \in \mathbb{R}^{d_i d_o}$ as a Kronecker product of two factors

$$\varepsilon\, \boldsymbol{I}_{d_i d_o} + \frac{1}{r}\, \boldsymbol{g}\boldsymbol{g}^\top \preccurlyeq \boldsymbol{L} \otimes \boldsymbol{R} = (\epsilon \boldsymbol{I}_{d_i} + \boldsymbol{G}\boldsymbol{G}^\top)^{\frac{1}{2}} \otimes (\epsilon \boldsymbol{I}_{d_o} + \boldsymbol{G}^\top \boldsymbol{G})^{\frac{1}{2}} \,, \tag{6}$$

where r is an upper-bound on the rank of the gradients (see [7] for further details), and $\boldsymbol{L}$ and $\boldsymbol{R}$ correspond to the left and right preconditioners. The construction in Equation (6) is more general and can be applied to other types of layers. However, notice that $\boldsymbol{g}$ in Equation (6) corresponds to the average gradient of a batch (and not the per example gradient).

To approximate Equation (5) via Shampoo, we first sample $\boldsymbol{y}_\ell^{(i)}$ from the predictive distribution in Equation (3) given the input sample $\boldsymbol{x}^{(i)}$ in the batch $\{\boldsymbol{x}^{(i)}\}_{i=1}^n$ and the corresponding post-activations $\hat{\boldsymbol{y}}_{\ell-1}$ and $\hat{\boldsymbol{y}}_\ell$ at layer $\ell - 1$ and ℓ, respectively. Next, instead of the per example sampled

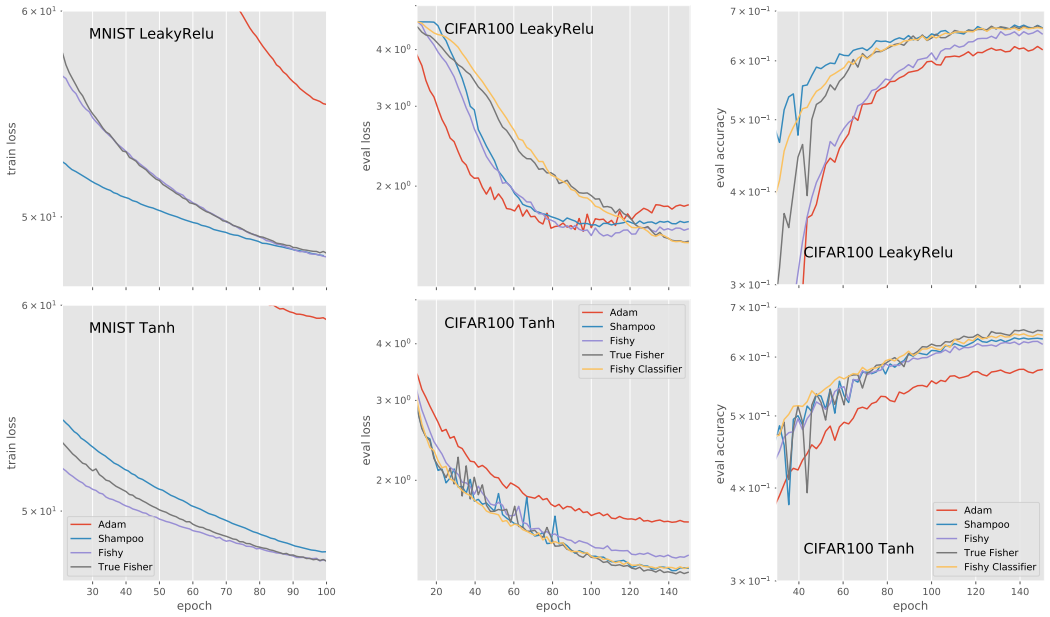

Figure 2: Experimental results on Deep Autoencoder on MNIST daraset and VGG16 convolutional network on CIFAR100 for image classification.

gradient $\boldsymbol{g}_s^{(i)} = (\boldsymbol{y}_\ell^{(i)} - \hat{\boldsymbol{y}}_\ell^{(i)}) \otimes \hat{\boldsymbol{y}}_{\ell-1}^{(i)}$, we pass the average sampled gradient $\boldsymbol{g}_s = {}^1\!/_n \sum_i \boldsymbol{g}_s^{(i)}$ into Shampoo to form the preconditioners. A similar averaging is performed in K-FAC [15] on the sampled gradients before forming the preconditioners. Shampoo then applies the (EMA of the) left $\boldsymbol{L}_s$ and right $\boldsymbol{R}_s$ Fishy preconditioners on the gradient of the training examples $\boldsymbol{G}$,

$$\boldsymbol{G}_{\text{Fishy-NGD}} = \boldsymbol{L}_s^{-1} \, \boldsymbol{G} \, \boldsymbol{R}_s^{-1} \, . \tag{7}$$

Notice that the exponents in Equation (7) are different than the original Shampoo formulation as we are interested in the inverse of the local FIM (and not its inverse square root).

3 Experiments

We perform experiments on two architectures, and datasets, a deep MNIST autoencoder with fully connected hidden layers [1000, 500 $\times$ 8, 250, 30, 250, 500 $\times$ 8, 1000], a standard benchmark for second-order methods, and a VGG16 CNN architecture applied to CIFAR100. We compare Adam [10], standard Shampoo [5], FIM Shampoo (true Fisher), and our method Fishy. For VGG16, we include a variant of Fishy that only approximates the local FIM for the final fully-connected layers (called Fishy classifier), letting earlier convolutional layers inherit gradients from the first fully-connected layer's predictive distribution. Hyperparameters for each method are tuned over 400 trials with a Bayesian optimization toolbox, and the trial with the best final objective value is displayed. Shampoo and Fishy share the same hyperparameters. We find significant improvements over Shampoo with Fishy, especially for the tanh activations. Fishy even outperforms true Fisher on some problems. We plan to open-source the code to reproduce these experiments.

4 Conclusion

We introduced Fishy, a layerwise Fisher construction method that allows model parallelism for approximating the FIM. We apply our construction to Shampoo, which only requires simple adjustments but can improve performance on some problems. In the future, we would like to perform more extensive experiments comparing K-FAC and derive a full expectation version without requiring sampling. Further treatment of layerwise predictive distributions on their own can also yield interesting research directions.

References

[1] Shun-ichi Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2):251–276, February 1998.

[2] Ehsan Amid, Rohan Anil, Christopher Fifty, and Manfred K. Warmuth. Layerwise Bregman Representation Learning with Applications to Knowledge Distillation, September 2022. arXiv:2209.07080 [cs].

[3] Ehsan Amid, Rohan Anil, and Manfred Warmuth. LocoProp: Enhancing BackProp via Local Loss Optimization. In *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, pages 9626–9642. PMLR, May 2022. ISSN: 2640-3498.

[4] Ehsan Amid, Manfred K. K Warmuth, Rohan Anil, and Tomer Koren. Robust Bi-Tempered Logistic Loss Based on Bregman Divergences. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[5] Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. Scalable second order optimization for deep learning. *arXiv preprint arXiv:2002.09018*, 2020.

[6] Arindam Banerjee, Srujana Merugu, Inderjit S Dhillon, and Joydeep Ghosh. Clustering with Bregman divergences. *Journal of machine learning research*, 6(Oct):1705–1749, 2005.

[7] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned Stochastic Tensor Optimization. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1842–1850. PMLR, July 2018.

[8] D.P. Helmbold, J. Kivinen, and M.K. Warmuth. Relative loss bounds for single neurons. *IEEE Transactions on Neural Networks*, 10(6):1291–1304, November 1999.

[9] Mohammad Emtiyaz Khan and Håvard Rue. The bayesian learning rule. *arXiv preprint arXiv:2107.04562*, 2021.

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

[11] J. Kivinen and M. K. Warmuth. Relative Loss Bounds for Multidimensional Regression Problems. *Machine Learning*, 45(3):301–329, December 2001.

[12] Frederik Kunstner, Philipp Hennig, and Lukas Balles. Limitations of the empirical fisher approximation for natural gradient descent. *Advances in neural information processing systems*, 32, 2019.

[13] John M Lee. *Riemannian manifolds: an introduction to curvature*, volume 176. Springer Science & Business Media, 2006.

[14] James Martens. New Insights and Perspectives on the Natural Gradient Method. *Journal of Machine Learning Research*, 21(146):1–76, 2020.

[15] James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 2408–2417, Lille, France, July 2015. JMLR.org.

[16] Frank Nielsen. An elementary introduction to information geometry. *Entropy*, 22(10):1100, 2020.

[17] Yi Ren and Donald Goldfarb. Tensor normal training for deep learning models. *Advances in Neural Information Processing Systems*, 34:26040–26052, 2021.

[18] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, April 2015. arXiv:1409.1556 [cs].

[19] Ke Sun and Frank Nielsen. Relative Fisher Information and Natural Gradient for Learning Large Modular Models. In *Proceedings of the 34th International Conference on Machine Learning*, pages 3289–3298. PMLR, July 2017.