FROG: Fisher Row-wise Preconditioning Technical Overview

Nikita Breskanu
nbreskanu73@gmail.com

Abstract

FROG is a second-order optimizer based on row-wise Fisher preconditioning. It uses joint Conjugate Gradient solves to approximate natural-gradient updates with low computational overhead. Fisher trace–based normalization ensures scale-free updates. The method is applicable to linear and convolutional layers and requires only a small number of CG iterations in practice. Implementation is available at https://github.com/Fullfix/frog-optimizer.

1 Motivation

K-FAC [4] is a well-known second-order optimizer that approximates the Fisher information matrix (FIM) using Kronecker factorization. While effective, it suffers from several practical limitations: (i) Kronecker decoupling introduces approximation error; (ii) per-iteration Fisher preconditioning is computationally expensive; (iii) natural-gradient updates of the form $\eta F^{-1} g$ scale inversely with gradient magnitude, resulting in strong sensitivity to learning-rate choice.

To address these issues, we propose FROG (Fisher ROw-wise preconditioninG). FROG decouples output rows (or output channels in convolutional layers) and applies Fisher preconditioning independently per row. Empirically, the cross-row Fisher terms are often small compared to within-row terms, which makes this approximation reasonable. Decoupling reduces the Fisher matrix from size $mn \times mn$ to $n \times n$, enabling efficient Conjugate Gradient (CG) inversion with only a few iterations. To further reduce overhead, the Fisher matrix is estimated using only a small subset of activations in a batch. Finally, Fisher is normalized by its mean diagonal value $\mathrm{tr}(F)/D$, which yields scale-free updates similar to first-order optimizers such as RMSprop [6] and Adam [2].

2 Method

2.1 Scale-normalized natural gradient

We model the network as a conditional distribution $p(y \mid x, \theta)$ and define the Fisher information matrix

$$F = \mathbb{E}\left[gg^\top\right], \quad g = \nabla_\theta \log p(y \mid x, \theta),$$

where the expectation is taken over individual samples.

For a small parameter update h, the quadratic form $\langle Fh, h\rangle$ approximates the second-order Taylor expansion of the KL divergence between $p(y \mid x, \theta)$ and $p(y \mid x, \theta + h)$ [1]. We therefore consider the constrained optimization problem

$$\langle g, h\rangle \to \min_h \quad \text{s.t.} \quad \langle \hat{F}h, h\rangle \le \eta^2, \quad \text{where} \quad \hat{F} = \frac{F}{\mathrm{tr}(F)/D} + \tau I. \tag{1}$$

Normalizing by $\mathrm{tr}(F)/D$ removes sensitivity to loss scaling, while damping τI stabilizes updates along directions with small Fisher eigenvalues. We refer to $\hat{F}$ as the **normalized Fisher**. The solution of (1) is

$$h^* = -\eta \frac{\hat{F}^{-1} g}{\sqrt{\langle \hat{F}^{-1} g, g\rangle}}. \tag{2}$$

Ignoring damping, h^* corresponds to a rescaled natural-gradient update.

2.2 Row-wise Fisher structure

Consider a linear layer $W \in \mathbb{R}^{R\times D}$ and a single row $w_j \in \mathbb{R}^D$, where $j = 1, \ldots, R$ indexes output rows. Let x_i denote input activations and $\delta L/\delta y_{ij}$ the gradient of the loss with respect to the j-th output for sample i, where $i = 1, \ldots, B$. In practice, B denotes a small subset of a batch used for Fisher estimation. The per-sample gradient is

$$g_{ij} = \frac{\delta L}{\delta y_{ij}}\, x_i.$$

The corresponding row-wise Fisher matrix is

$$F_j = \mathbb{E}[g_j g_j^\top] = \frac{1}{B}\sum_{i=1}^{B}\left(\frac{\delta L}{\delta y_{ij}}\right)^2 x_i x_i^\top = X^\top \Lambda_j X, \qquad \Lambda_j = \operatorname{diag}\left(\frac{1}{B}\left(\frac{\delta L}{\delta y_{ij}}\right)^2\right),$$

where $X \in \mathbb{R}^{B\times D}$ is the activation matrix.

The trace of F_j is

$$\operatorname{tr}(F_j) = \frac{1}{B}\sum_{i=1}^{B}\left(\frac{\delta L}{\delta y_{ij}}\right)^2 \|x_i\|^2. \tag{3}$$

The normalized Fisher matrix is therefore

$$\hat{F}_j = X^\top \hat{\Lambda}_j X + \tau I_D, \qquad \hat{\Lambda}_j = \frac{\Lambda_j}{\operatorname{tr}(F_j)/D}. \tag{4}$$

2.3 Efficient joint conjugate gradient

Explicit construction of the normalized Fisher matrices $\hat{F}_j$ is infeasible for large R. However, all $\hat{F}_j$ share the same activation matrix X, which enables efficient joint matrix–vector products.

Let $V = [v_1, \ldots, v_R] \in \mathbb{R}^{D\times R}$ and collect the row-wise diagonal weights into $\hat{\Lambda} \in \mathbb{R}^{B\times R}$. A joint matrix–vector product is given by

$$V \;\mapsto\; X^\top\big(\hat{\Lambda} \odot (XV)\big) + \tau V,$$

which yields $\hat{F}_j v_j$ for each column j.

This structure enables batched Conjugate Gradient [5] solves for all rows simultaneously. Empirically, 4–5 CG iterations are sufficient; additional iterations tend to overfit noisy low-eigenvalue directions of the empirical Fisher.

For convolutional layers, the operators $X(\cdot)$ and $X^\top(\cdot)$ correspond to forward and backward convolutions, respectively. In this case, diagonal weights have shape $\hat{\Lambda} \in \mathbb{R}^{B\times C_o\times H_o\times W_o}$, activations have shape $X \in \mathbb{R}^{B\times C_i\times H_i\times W_i}$, and CG is performed over $C_i \times k_h \times k_w$ dimensions.

2.4 Momentum and weight Decay

We apply momentum *before* Fisher preconditioning, following an Adam-style [2] formulation. Specifically, we precondition the momentum buffer

$$m_t \leftarrow \beta m_{t-1} + g_t,$$

and compute $\hat{F}^{-1} m_t$ using Conjugate Gradient (CG).

Weight decay is applied in a decoupled manner, as in AdamW [3]. The resulting per-row parameter update is

$$\theta_t \leftarrow (1-\rho\eta)\theta_{t-1} \;-\; \eta\frac{\hat{F}_t^{-1} m_t}{\sqrt{\langle \hat{F}_t^{-1} m_t, m_t\rangle}}. \tag{5}$$

3 The FROG Algorithm

We now summarize the resulting optimizer in algorithmic form.

Algorithm 1 FROG for linear and convolutional layers

Input: parameters Θ, learning rate η, damping τ, weight decay ρ, momentum coefficient β, Fisher sample size r, number of CG iterations k.
Initialize momentum buffer $M \leftarrow 0$
for $t = 1$ **to** T **do**
Compute averaged gradient $G \leftarrow \nabla_\Theta L(\Theta)$
Update momentum: $M \leftarrow \beta M + G$
Sample r input activations X and corresponding output gradients Y
Compute row-wise normalized Fisher weights:

$$\hat{\Lambda}_{ij} \leftarrow \frac{Y_{ij}^2/r}{\frac{1}{rD}\sum_{i'=1}^{r} Y_{i'j}^2 \|X_{i'}\|^2}$$

Define Fisher matvec operator:

$$\mathcal{F}(V) = X^\top\big(\hat{\Lambda} \odot (XV)\big) + \tau V$$

Solve $\mathcal{F}(Z) = M$ using k iterations of batched CG, starting from $Z^{(0)} = 0$
Normalize per row:

$$Z_j \leftarrow \frac{Z_j}{\sqrt{\langle Z_j, M_j\rangle}}$$

Update parameters:

$$\Theta \leftarrow (1 - \rho\eta)\Theta - \eta Z$$

end for

Implementation details. For linear layers, $X(\cdot)$ and $X^\top(\cdot)$ correspond to standard matrix multiplication. For convolutional layers, they denote the forward and backward convolution operators, respectively. In this case, the dimension D corresponds to the flattened kernel dimension $D = C_{\text{in}} \times k_h \times k_w$, and Conjugate Gradient is performed over this space for each output channel.

When using PyTorch with `reduction='mean'` in the loss function, output gradients are implicitly scaled by the inverse batch size. To maintain consistent Fisher scaling and stable learning-rate behavior, it is recommended to rescale output gradients as $Y \leftarrow B \cdot Y$.

4 Practical Considerations

At present, FROG has been evaluated primarily on convolutional architectures. The following empirical guidelines were found to work well in practice.

Applicable layers. For convolutional networks, applying FROG to all convolutional layers is recommended. Applying FROG to the final classification layer is not recommended, as it often becomes unstable toward the end of training, likely due to strong cross-class correlations.

Batch size. Larger batch sizes (typically $B \geq 128$) are recommended, as they reduce the relative computational overhead of Fisher estimation: the Fisher matrix is computed using a fixed-size subset of the batch, so its cost is amortized over more samples.

Number of CG iterations. A small number of Conjugate Gradient iterations (typically $k = 3$–5) is sufficient. Increasing k further often degrades performance, likely due to over-inversion of noisy low-eigenvalue directions of the empirical Fisher.

Fisher sample size. The Fisher matrix can be estimated using a small subset of batch samples (e.g., $r = 16$–64). Larger sample sizes increase computational cost with limited benefit.

Damping. The damping parameter τ stabilizes inversion along directions with small Fisher eigenvalues. In practice, values in the range $\tau \in [10^{-6}, 10^{-2}]$ work well, with larger values typically preferred in later stages of training.

Learning rate. Learning rates comparable to those used with SGD can be employed; in practice, values approximately $2\times$–$4\times$ larger than the SGD learning rate work well.

Momentum. Standard momentum values (e.g., $\beta = 0.9$) are effective.

Weight decay. Lower weight decay than typically used with SGD is preferred. Values in the range 10^{-5} to 5×10^{-5} work well.

References

[1] Shun-ichi Amari and Hiroshi Nagaoka. *Methods of information geometry*, volume 191. American Mathematical Soc., 2000.

[2] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[3] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

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

[5] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain. 1994.

[6] Tijmen Tieleman. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26, 2012.