FISHER-LEGENDRE (FISHLEG) OPTIMIZATION OF DEEP NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Incorporating second-order gradient information (curvature) into optimization can dramatically reduce the number of iterations required to train machine learning models. In natural gradient descent, such information comes from the Fisher information matrix which yields a number of desirable properties. As exact natural gradient updates are intractable for large models, successful methods such as KFAC and sequels approximate the Fisher in a structured form that can easily be inverted. However, this requires model/layer-specific tensor algebra and certain approximations that are often difficult to justify. Here, we use ideas from Legendre-Fenchel duality to learn a direct and efficiently evaluated model for the product of the inverse Fisher with any vector, in an online manner, leading to natural gradient steps that get progressively more accurate over time despite noisy gradients. We prove that the resulting "Fisher-Legendre" (FishLeg) optimizer converges to a (global) minimum of non-convex functions satisfying the PL condition, which applies in particular to deep linear networks. On standard auto-encoder benchmarks, we show empirically that FishLeg outperforms standard first-order optimization methods, and performs on par with or better than other second-order methods, especially when using small batches. Thanks to its generality, we expect our approach to facilitate the handling of a variety neural network layers in future work.

1 INTRODUCTION & SUMMARY OF CONTRIBUTIONS

The optimization of machine learning models often benefits from the use of second-order gradient (curvature) information, which can dramatically increase the per-iteration progress on the training loss. However, second-order optimizers for machine learning face a number of key challenges:

1. ML models tend to have many parameters, such that estimating the curvature along all dimensions is intractable. There are normally two ways around this: (i) using iterative methods where each iteration only exploits curvature information along a single dimension ("Hessian-free" methods; Martens et al., 2010), and (ii) developing custom curvature approximations that can be efficiently inverted to obtain parameter updates (Martens & Grosse, 2015; Grosse & Martens, 2016; George et al., 2018; Bahamou et al., 2022; Soori et al., 2022). The latter yields state-of-the-art performance in the optimization of deep networks (Goldfarb et al., 2020). However, development of structured curvature approximations must be done on a case-by-case basis for each architecture (e.g. fully-connected or convolutional layer) and requires mathematical assumptions that are difficult to justify.
2. ML datasets tend to be large as well, such that the loss, its gradients, and its curvature can be only stochastically estimated from mini-batches. Noisy gradients hinder the application of traditional quasi-Newton methods (Nocedal & Wright, 1999), because the changes in the gradient that result from parameter updates cannot be estimated precisely (but see Byrd et al., 2016; Moritz et al., 2016). In the current state-of-the-art methods mentioned above, the noise problem is typically mitigated by a combination of large mini-batches and momentum. Nevertheless, estimating the various components of the curvature matrix before inverting it (as opposed to estimating the inverse directly) introduces a bias that can potentially be detrimental.

Here, we focus primarily on the Fisher information matrix (FIM) as a notoriously effective source of curvature information for training ML models. For probabilistic models, preconditioning the

gradient by the inverse FIM yields so-called natural gradient updates (Amari, 1998). We introduce a novel framework for second-order optimization of high-dimensional probabilistic models in the presence of gradient noise. Instead of approximating the FIM in a way that can easily be inverted (e.g. as in KFAC; Martens & Grosse, 2015 and related approaches), we directly parameterize the inverse FIM. We make the following contributions:

- We show that the inverse FIM can be computed through the Legendre-Fenchel conjugate of a cross entropy between model distributions.
- We provide an algorithm (FishLeg) which meta-learns the inverse FIM in an online fashion, and we prove convergence of the corresponding parameter updates.
- We first study its application to deep linear networks, an example of non-convex and pathologically curved loss functions with a Fisher matrix known in closed form (Bernacchia et al., 2018, Huh, 2020), and find that convergence occurs much faster than SGD with momentum or Adam.
- We then show that, in standard auto-encoders benchmarks, FishLeg operating on a block-diagonal Kronecker parameterization of the inverse Fisher performs similarly to – and sometimes outperforms – previous approximate natural gradient methods (Goldfarb et al., 2020), whilst being only twice slower than SGD with momentum in wall-clock time per iteration.

FishLeg can accommodate arbitrary parameterizations of the inverse Fisher, thus we expect to facilitate future applications of natural gradient optimization to a broad range of network architectures, for which manually approximating the FIM in an easily invertible form may otherwise be difficult.

2 BACKGROUND

2.1 FISHER INFORMATION AND THE NATURAL GRADIENT

We consider a probabilistic model parameterized by a vector $\boldsymbol{\theta}$, which attributes a negative log-likelihood $\ell(\boldsymbol{\theta}, \mathcal{D}) = -\log p(\mathcal{D}|\boldsymbol{\theta})$ to any collection $\mathcal{D}$ of data points drawn from a data distribution $p^\star(\mathcal{D})$. This covers a broad range of models, including discriminative models for regression or classification, as well as generative models for density modelling. The goal is to find parameters $\boldsymbol{\theta}^\star$ to approximate the true data distribution $p^\star(\mathcal{D})$ by the model distribution $p(\mathcal{D}|\boldsymbol{\theta}^\star)$.

The Fisher information matrix (FIM) measures how much information can be obtained about parameters $\boldsymbol{\theta}$ after observing data $\mathcal{D}$ under the model $p(\mathcal{D}|\boldsymbol{\theta})$, and captures redundancies between parameters (Rao, 1992). The FIM is defined as

$$\mathcal{I}(\boldsymbol{\theta}) = \mathbb{E}_{\mathcal{D}\sim p(\mathcal{D}|\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \mathcal{D}) \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \mathcal{D})^\top. \tag{1}$$

By this definition, the FIM is a positive semi-definite matrix. It can be shown that under certain regularity conditions, and if ℓ is twice differentiable w.r.t. $\boldsymbol{\theta}$, then the FIM can also be computed as

$$\mathcal{I}(\boldsymbol{\theta}) = \mathbb{E}_{\mathcal{D}\sim p(\mathcal{D}|\boldsymbol{\theta})} \nabla^2_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \mathcal{D}) \tag{2}$$

It is important to note that the average is computed over the model distribution $p(\mathcal{D}|\boldsymbol{\theta})$, *not* the data distribution $p^\star(\mathcal{D})$. Averaging Eq. 1 over $p^\star(\mathcal{D})$ results in the *empirical Fisher* matrix (Kunstner et al., 2019), while averaging Eq. 2 over $p^\star(\mathcal{D})$ results in the Hessian of the loss. The FIM, the empirical Fisher and the Hessian are all different and sometimes confused (Thomas et al., 2020).

We consider cases in which parameters $\boldsymbol{\theta}$ are obtained by maximum likelihood:

$$\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \tag{3}$$

where the population loss function is defined as

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\mathcal{D}\sim p^\star} \ell(\boldsymbol{\theta}, \mathcal{D}) \tag{4}$$

which is in general non-convex, in particular when the model distribution is parameterised by deep neural networks. The natural gradient update takes the form

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\, \mathcal{I}(\boldsymbol{\theta})^{-1} \mathrm{g}(\boldsymbol{\theta}) \quad \text{with} \quad \mathrm{g}(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \tag{5}$$

where η is a learning rate and g is the gradient of the loss. In practice, the true distribution of data $p^\star(\mathcal{D})$ is unknown and we have only access to the empirical loss and a stochastic estimate $\hat{\mathrm{g}}$ of the

gradient based on a finite sample of data $\mathcal{D} \sim p^\star$. Similarly, while the FIM may be sometimes computed exactly (Bernacchia et al., 2018, Huh, 2020), in most circumstances it is also estimated from a mini-batch $\mathcal{D} \sim p$. To guarantee invertibility of the FIM, a small amount of regularization (or "damping") can be added to it when it is nearly singular. Note that Eq. 5 is identical to gradient descent when the FIM equals the identity matrix. The natural gradient has several nice properties that make it an efficient optimizer (Amari, 1998), but computing and inverting the FIM is usually costly.

2.2 THE LEGENDRE-FENCHEL CONJUGATE

The Legendre-Fenchel (LF) conjugate is a useful tool in optimization theory to map pairs of problems via duality (Boyd & Vandenberghe, 2004) and was introduced in the context of mechanics and thermodynamics (Zia et al., 2009). In the following, we deviate from the standard textbook notation to avoid conflicts with the rest of the paper and to facilitate the translation of these classical results to our problem. Consider a twice differentiable function $\mathcal{H}(\boldsymbol{\delta})$ of a vector $\boldsymbol{\delta}$, generally non-convex and assume a minimum of $\mathcal{H}$ exists. The LF conjugate is equal to

$$\mathcal{H}^\star(\mathbf{u}) = \min_{\boldsymbol{\delta}} \mathcal{H}(\boldsymbol{\delta}) - \mathbf{u}^T \boldsymbol{\delta}. \tag{6}$$

The LF conjugate, also known as the convex conjugate, is defined also for non-differentiable functions and is always convex. We summarize here two properties of the convex conjugate $\mathcal{H}^\star$ that we use in our derivations (see chapter 3.3 of Boyd & Vandenberghe, 2004 and Zia et al., 2009 for details):

Property 1 The gradient of the conjugate is equal to

$$\nabla_{\mathbf{u}} \mathcal{H}^\star(\mathbf{u}) = \tilde{\boldsymbol{\delta}}(\mathbf{u}) \qquad \text{where} \quad \tilde{\boldsymbol{\delta}}(\mathbf{u}) = \operatorname{argmin}_{\boldsymbol{\delta}} \mathcal{H}(\boldsymbol{\delta}) - \mathbf{u}^T \boldsymbol{\delta}. \tag{7}$$

The minimizer $\tilde{\boldsymbol{\delta}}(\mathbf{u})$ also satisfies $\nabla_{\boldsymbol{\delta}} \mathcal{H}(\tilde{\boldsymbol{\delta}}(\mathbf{u})) = \mathbf{u}$, implying that $\nabla_{\mathbf{u}} \mathcal{H}^\star$ is a (local) inverse function of $\nabla_{\boldsymbol{\delta}} \mathcal{H}$. The inverse is global when the function $\mathcal{H}$ is strictly convex:

$$\nabla_{\mathbf{u}} \mathcal{H}^\star(\mathbf{u}) = \left(\nabla_{\boldsymbol{\delta}} \mathcal{H}\right)^{-1}(\mathbf{u}). \tag{8}$$

Property 2 The Hessian matrix of the conjugate is equal to the inverse (if it exists) of the Hessian matrix of $\mathcal{H}$, computed at $\tilde{\boldsymbol{\delta}}(\mathbf{u})$:

$$\nabla^2_{\mathbf{u}} \mathcal{H}^\star(\mathbf{u}) = \left(\nabla^2_{\boldsymbol{\delta}} \mathcal{H}(\tilde{\boldsymbol{\delta}}(\mathbf{u}))\right)^{-1}. \tag{9}$$

3 RELATED WORK

Natural gradient The Hessian of non-convex losses is not positive definite, and second-order methods such as Newton's update typically do not converge to a minimum (Dauphin et al., 2014). The natural gradient method substitutes the Hessian with the FIM, which is positive (semi)definite (Amari, 1998). The natural gradient has been applied to deep learning in recent years (Martens & Grosse, 2015, Bernacchia et al., 2018, Huh, 2020, Kerekes et al., 2021), and has been proved approximately equivalent to local loss optimization (Benzing, 2022, Meulemans et al., 2020, Amid et al., 2022). Previous work on natural gradient approximately estimated and inverted the FIM analytically (Martens & Grosse, 2015; Martens et al., 2018). Instead, we use an exact formulation of the inverse FIM as the Hessian of the LF conjugate of the cross entropy, and we provide an algorithm to meta-learn an approximation to the inverse FIM during training.

Legendre-Fenchel conjugate For convex losses, Eq. 8 implies that the parameter update $\Delta\boldsymbol{\theta} = -\nabla_{\mathbf{g}} \mathcal{L}^\star\left(\mathbf{g}(\boldsymbol{\theta})\right) + \nabla_{\mathbf{g}} \mathcal{L}^\star\left(\mathbf{0}\right)$ converges to a minimum in one step (Maddison et al., 2021; Chraibi et al., 2019). However, this update is not practical because finding $\mathcal{L}^\star$ is at least as hard as finding a minimum of the loss. Chraibi et al. (2019) propose to learn $\mathcal{L}^\star$ by a neural network. Maddison et al. (2021) prove that, if a surrogate for $\mathcal{L}^\star$ satisfies relative smoothness in dual space, then linear convergence rates can be proven for non-Lipschitz or non-strongly convex losses. Here we consider non-convex losses and we use the LF conjugate of the cross entropy for meta-learning the natural gradient update.

Meta-learning optimizers Many standard optimizers have comparable performance when appropriately tuning a handful of static hyperparameters (Schmidt et al., 2021). *Learned* optimizers often outperform well-tuned standard ones by tuning a larger number of hyperparameters online during training (Chen et al., 2021; Hospedales et al., 2021). Different methods for learning optimizers have been proposed, including hypergradients (Maclaurin et al., 2015; Andrychowicz et al., 2016; Franceschi et al., 2017; Wichrowska et al., 2017; Wu et al., 2018; Park & Oliva, 2019; Micaelli & Storkey, 2021), reinforcement learning (Li & Malik, 2016; Bello et al., 2017), evolution strategies (Metz et al., 2019; Vicol et al., 2021), and implicit gradients (Lorraine et al., 2020; Rajeswaran et al., 2019; Clarke et al., 2021). In our work, hyperparameters do not minimize the loss, instead they minimize an auxiliary loss designed to learn the natural gradient update.

4 FISHER-LEGENDRE OPTIMIZATION

4.1 COMPUTATION OF THE NATURAL GRADIENT VIA THE LEGENDRE-FENCHEL CONJUGATE

We use the properties of the LF conjugate, in particular Eq. 9, to directly learn the *inverse* FIM and use it in the natural gradient update of Eq. 5. This is different from previous work on natural gradient, which aimed at approximating the FIM analytically and then inverting it. We prove the following:

Theorem 1. *Assume that the negative log-likelihood $\ell(\boldsymbol{\theta}, \mathcal{D}) = -\log p(\mathcal{D}|\boldsymbol{\theta})$ is twice differentiable. Assume that $p(\mathcal{D}|\boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta} + \boldsymbol{\delta})$ if and only if $\boldsymbol{\delta} = \mathbf{0}$. Define the cross entropy between $p(\mathcal{D}|\boldsymbol{\theta})$ and $p(\mathcal{D}|\boldsymbol{\theta} + \boldsymbol{\delta})$*

$$\mathcal{H}(\boldsymbol{\theta}, \boldsymbol{\delta}) = \mathbb{E}_{\mathcal{D} \sim p(\mathcal{D}|\boldsymbol{\theta})} \ell(\boldsymbol{\theta} + \boldsymbol{\delta}, \mathcal{D}) \tag{10}$$

and the function

$$\tilde{\boldsymbol{\delta}}(\boldsymbol{\theta}, \mathbf{u}) = \text{argmin}_{\boldsymbol{\delta}} \; \mathcal{H}(\boldsymbol{\theta}, \boldsymbol{\delta}) - \mathbf{u}^T \boldsymbol{\delta}. \tag{11}$$

Then, the inverse Fisher information matrix exists and is equal to

$$\mathcal{I}(\boldsymbol{\theta})^{-1} = \nabla_{\mathbf{u}} \tilde{\boldsymbol{\delta}}(\boldsymbol{\theta}, \mathbf{0}). \tag{12}$$

Proof. Here we provide a sketch of the proof, see Appendix A.2 for the full proof. Lemma 1 proves that the cross-entropy in Eq. 10 has a unique minimum at $\boldsymbol{\delta} = \mathbf{0}$, therefore $\tilde{\boldsymbol{\delta}}(\boldsymbol{\theta}, \mathbf{0}) = \mathbf{0}$. The proof continues by expressing the FIM as the Hessian of the cross-entropy, which is done by reparameterizing Eq. 2. Next, we use Eq. 9 to express the inverse FIM as the Hessian of the LF conjugate of the cross-entropy (seen as a function of $\boldsymbol{\delta}$). From there, the theorem follows from Eq. 7. □

To clarify the connection with the LF conjugate, we note that Eq. 11 represents the gradient of the LF conjugate of the cross-entropy, while Eq. 12 is the Hessian of the LF conjugate – see Appendix A.2 for details, and Appendix A.3 for an informal mathematical argument that helps understanding Theorem 1.

Our goal is to compute the natural gradient update by solving the optimization problem in Eq. 11, and substituting the inverse FIM computed by equation 12 into the definition of the natural gradient step in Eq. 5, yielding

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \, \nabla_{\mathbf{u}} \tilde{\boldsymbol{\delta}}(\boldsymbol{\theta}_t, \mathbf{0}) \mathbf{g}(\boldsymbol{\theta}_t). \tag{13}$$

Importantly, note that Eq. 11 does not need to be solved for all possible pairs of $\boldsymbol{\theta}$ and $\mathbf{u}$. Indeed, given the current value of parameters $\boldsymbol{\theta}_t$, it is sufficient to compute $\tilde{\boldsymbol{\delta}}(\boldsymbol{\theta}_t, \mathbf{u})$ (i) at small values of $\mathbf{u}$, because $\nabla_{\mathbf{u}} \tilde{\boldsymbol{\delta}}$ is evaluated at $\mathbf{u} = \mathbf{0}$ in Eq. 12, and (ii) along the direction $\mathbf{u} \propto \mathbf{g}(\boldsymbol{\theta}_t)$, because $\nabla_{\mathbf{u}} \tilde{\boldsymbol{\delta}}$ is multiplied by $\mathbf{g}(\boldsymbol{\theta}_t)$ in Eq. 13 such that knowing the slope of $\tilde{\boldsymbol{\delta}}(\boldsymbol{\theta}_t, \mathbf{u})$ along any other direction would be irrelevant. Nevertheless, computing Eq. 11 for such restricted $(\boldsymbol{\theta}, \mathbf{u})$ pairs may still not be easier than computing the FIM by standard methods, e.g. by computing Eq. 1 and inverting the resulting matrix. In section 4.2, we therefore propose a practical instantiation of the ideas developed in this section, which yields an online algorithm with provable convergence guarantees.

4.2 ONLINE META-LEARNING OF NATURAL GRADIENT UPDATES

In principle, Theorem 1 provides an exact implementation of natural gradient by solving the optimization problem in Eq. 11 and then applying the update of Eq. 13. However, this approach is impractical due to the complexity of solving Eq. 11. To make it practical, we propose learning an approximation $\overline{\boldsymbol{\delta}}(\boldsymbol{\theta}, \mathbf{u}, \boldsymbol{\lambda})$ of the true $\tilde{\boldsymbol{\delta}}(\boldsymbol{\theta}, \mathbf{u})$, through a set of *auxiliary* parameters $\boldsymbol{\lambda}$. The auxiliary parameters $\boldsymbol{\lambda}$ are updated *online*, during optimization, to push $\overline{\boldsymbol{\delta}}$ towards a solution of Eq. 11 but without requiring convergence. This is akin to a meta-learning approach whereby only one or a few steps are taken towards a desired solution for meta-parameters $\boldsymbol{\lambda}$, which are optimized concurrently with parameters $\boldsymbol{\theta}$ in an inner loop (Hospedales et al., 2021). Here we choose gradient-based meta-learning updates (Finn et al., 2017), but other choices are possible, e.g. implicit differentiation (Lorraine et al., 2020, Rajeswaran et al., 2019) or evolutionary strategies (Metz et al., 2019,Gao & Sener, 2022).

For the auxiliary parameterization, we choose a linear function of $\mathbf{u}$,

$$\overline{\boldsymbol{\delta}}(\boldsymbol{\theta}, \mathbf{u}, \boldsymbol{\lambda}) = Q(\boldsymbol{\lambda})\mathbf{u}, \tag{14}$$

where $Q(\boldsymbol{\lambda})$ is a positive definite matrix which will thus effectively estimate the inverse FIM (see equation 12). Appropriate choices for Q should take into account the architecture of the model, as discussed further below. Although the r.h.s. of Eq. 14 does not depend on $\boldsymbol{\theta}$ explicitly, an implicit dependence will arise from us learning the parameters $\boldsymbol{\lambda}$ in a way that depends on the momentary model parameters $\boldsymbol{\theta}$. Indeed, for learning $\boldsymbol{\lambda}$, we perform gradient descent using Adam on the following auxiliary loss $\mathcal{A}$, a choice justified by Eq. 11:

$$\mathcal{A}(\boldsymbol{\theta}, \mathbf{u}, \boldsymbol{\lambda}) = \mathcal{H}\left(\boldsymbol{\theta}, Q(\boldsymbol{\lambda})\mathbf{u}\right)) - \mathbf{u}^T Q(\boldsymbol{\lambda})\mathbf{u}. \tag{15}$$

In summary, we alternate between updating $\boldsymbol{\lambda}$ to minimize the auxiliary loss in Eq. 15, and updating $\boldsymbol{\theta}$ according to Eq. 13 to minimize the main loss,

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \alpha \text{ AdamUpdate}\left(\nabla_{\boldsymbol{\lambda}} \mathcal{A}(\boldsymbol{\theta}_t, \epsilon \mathbf{g}(\boldsymbol{\theta}_t), \boldsymbol{\lambda}_t)\right), \tag{16}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\, Q(\boldsymbol{\lambda}_{t+1})\, \mathbf{g}(\boldsymbol{\theta}_t), \tag{17}$$

where α is a learning rate for the auxiliary parameters and ϵ is a small scalar (both are hyperparameters). Note that the auxiliary loss is computed at $\epsilon\mathbf{g}(\boldsymbol{\theta})$ because, as argued in section 4.1, it is sufficient to compute $\overline{\boldsymbol{\delta}}$ at small values of $\mathbf{u}$ in the direction of $\mathbf{u} \propto \mathbf{g}(\boldsymbol{\theta}_t)$.

For the matrix $Q(\boldsymbol{\lambda})$, we use a Kronecker-factored block-diagonal structure that follows the layer structure of deep neural networks with fully connected layers (section A.5). Although this is similar to parameterizations used in previous studies of natural gradient (Martens & Grosse, 2015), we stress that our approach is more flexible since it learns the inverse FIM rather than approximating and inverting the FIM analytically. In fact, we show in the appendix A.6 that alternative forms of the matrix Q, different from those used before, may provide a better approximation of the inverse FIM (see Fig. 4).

4.3 CONVERGENCE OF ONLINE NATURAL GRADIENT

In this section we prove that the update of Eq. 17 converges to a minimum of the loss for the class of PL functions (Polyak-Lojasiewicz), which are generally non-convex. Charles & Papailiopoulos (2018) proved that deep linear networks, which are non-convex (Saxe et al., 2013), are PL almost everywhere. We study deep linear networks empirically in section 5.1. We stress that the goal of this section is to prove convergence, rather than to compare rates of convergence with other algorithms.

We provide convergence guarantees for both the true gradient $\mathbf{g}$ and stochastic gradients $\hat{\mathbf{g}}$. Theorem 2 is a special case of Theorem 1 in Radhakrishnan et al. (2020), while Theorem 3 is new to our knowledge. In both cases, the crucial assumption is that Q is a positive definite matrix, which holds for our chosen parameterization (see section A.5). We provide theorems statements here, for details on definitions and proofs see appendix A.4.

Theorem 2. *Assume the loss function $\mathcal{L}$ is ξ-smooth and μ-PL. Assume Q is positive definite. Let us use $\lambda_{\min}$ and $\lambda_{\max}$, respectively, for the lower and upper bounds on the eigenvalues of Q. We have the following rate of convergence for optimization using update rule 17 with $\eta = \frac{1}{\xi\lambda_{\max}}$:*

$$\mathcal{L}(\boldsymbol{\theta}_t) - \mathcal{L}(\boldsymbol{\theta}^*) \leq \left(1 - \frac{\mu\lambda_{\min}}{2\xi\lambda_{\max}}\right)^t (\mathcal{L}(\boldsymbol{\theta}_0) - \mathcal{L}(\boldsymbol{\theta}^*)), \tag{18}$$

where $\boldsymbol{\theta}_0$ are the initial parameters.

Theorem 3. *Assume the loss function $\mathcal{L}$ is ξ-smooth and μ-PL. Assume Q is positive definite. Let us use $\lambda_{\min}$ and $\lambda_{\max}$, respectively, for the lower and upper bounds on the eigenvalues of Q. Assume $\mathbb{E}[\|\hat{\mathbf{g}}(\boldsymbol{\theta})\|^2] \leq G^2$. We have the following rate for optimization using update rule 17 with $\eta_t = \frac{1}{\mu\lambda_{\min}(t+1)}$:*

$$\mathcal{L}(\boldsymbol{\theta}_t) - \mathcal{L}(\boldsymbol{\theta}^*) \leq \frac{A}{t+1} \quad \textit{where} \quad A = \max\left\{\frac{\xi\lambda_{\max}^2 G^2}{\mu^2\lambda_{\min}^2}, \mathcal{L}(\boldsymbol{\theta}_0) - \mathcal{L}(\boldsymbol{\theta}^*)\right\}. \tag{19}$$

4.4 Details of the algorithm and further remarks

Here we detail our practical implementation of the online FishLeg algorithm presented in section 4.2, also summarized in Algorithm 1. Our code is available here on GitHub.

Auxiliary optimization Since $\mathcal{H}$ is evaluated on a model-sampled mini-batch, it is a noisy quantity whose variance depends on the hyperparameter ϵ. While this variance could be reduced, for example, by using antithetic estimators (Gao & Sener, 2022), we opted instead for analytically taking the small ϵ limit of the auxiliary loss, which eliminates this free parameter. By Taylor expanding equation 15 (use equations 23, 24), and dropping terms that are constant in $\boldsymbol{\lambda}$, we arrive at

$$\mathcal{A}(\boldsymbol{\theta}, \mathbf{u}, \boldsymbol{\lambda}) \equiv \mathbf{u}^T \left[\frac{1}{2}Q(\boldsymbol{\lambda})\nabla^2_{\boldsymbol{\delta}}\mathcal{H}(\boldsymbol{\theta}, \mathbf{0})Q(\boldsymbol{\lambda}) - Q(\boldsymbol{\lambda})\right]\mathbf{u}. \tag{20}$$

The auxiliary update of Eq. 16 thus becomes $\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \alpha\,\text{AdamUpdate}(\nabla_{\boldsymbol{\lambda}}\mathcal{A}(\boldsymbol{\theta}_t, \mathbf{g}/\|\mathbf{g}\|, \boldsymbol{\lambda}_t))$ where the normalization of $\mathbf{g}$ is important for continuing to learn about curvature even when gradients are small. In order to avoid differentiating through the Hessian-vector product in Eq. 20, we note that the gradient of $\mathcal{A}$ w.r.t. $\boldsymbol{\lambda}$ can be written as

$$\nabla_{\boldsymbol{\lambda}}\mathcal{A}(\boldsymbol{\theta}, \mathbf{u}, \boldsymbol{\lambda}) = \left(\nabla^2_{\boldsymbol{\delta}}\mathcal{H}(\boldsymbol{\theta}, \mathbf{0})Q(\boldsymbol{\lambda})\mathbf{u} - \mathbf{u}\right)^\top \nabla_{\boldsymbol{\lambda}}\left[Q(\boldsymbol{\lambda})\mathbf{u}\right]. \tag{21}$$

Algorithm 1 in the appendix shows how to implement this efficiently using automatic differentiation tools. As a side note, Eq. 21 makes it clear that Q will get to approximate the inverse of $\mathcal{I}(\boldsymbol{\theta})$ (which is also the Hessian of the cross-entropy, see equation 24) at least in the relevant subspace occupied by $\mathbf{u} = \mathbf{g}/\|\mathbf{g}\|$ over iterations. Moreover, note that the noise in $\mathcal{H}$ simply adds stochasticity to the auxiliary loss but will not bias its gradient w.r.t. $\boldsymbol{\lambda}$. Finally, Eq. 20 reveals a connection with Hessian-free optimization which we discuss in Appendix A.7.

Damping Practitioners of second-order optimization in deep learning have consistently found it critical to apply damping when the FIM becomes near-singular. Although adaptive damping schemes exist, here we followed Goldfarb et al. (2020) and used static damping, i.e. we added γI to $\nabla^2_{\boldsymbol{\delta}}\mathcal{H}$ in Eqs. 20-21, and tuned the hyperparameter γ as done in previous work.

Momentum We have found useful to use momentum as many other optimizers do (e.g. KFAC, KBFGS). In most of our experiments, we implemented momentum on $\mathbf{g}$ (as per line 16 of Algorithm 1). For our wall-clock time results (Fig. 3), however, we applied momentum to our natural gradient estimate $Q(\boldsymbol{\lambda})\mathbf{g}$ instead, in order to preserve the low-rank structure of $\mathbf{g}$ at each layer of the neural network. This enabled us to substantially speed up the computation of $Q(\boldsymbol{\lambda})\mathbf{g}$ on small mini-batches, but did not affect training and test errors noticeably.

Initialization of $\boldsymbol{\lambda}$ We find it useful to initialize $\boldsymbol{\lambda}$ in such a way as to make $Q(\boldsymbol{\lambda})$ initially equal to $\frac{\eta_{\text{SGDm}}}{\eta} \times I$, where η_{SGDm} is a learning rate that is known to give good results with SGD-with-momentum. With such initialization, FishLeg therefore initially behaves like SGDm.

5 Empirical evaluation

5.1 Deep linear networks

We first studied FishLeg in the context of deep linear networks, whose loss function is non-convex, pathologically curved, and for which the Fisher matrix is known analytically (Bernacchia et al.,

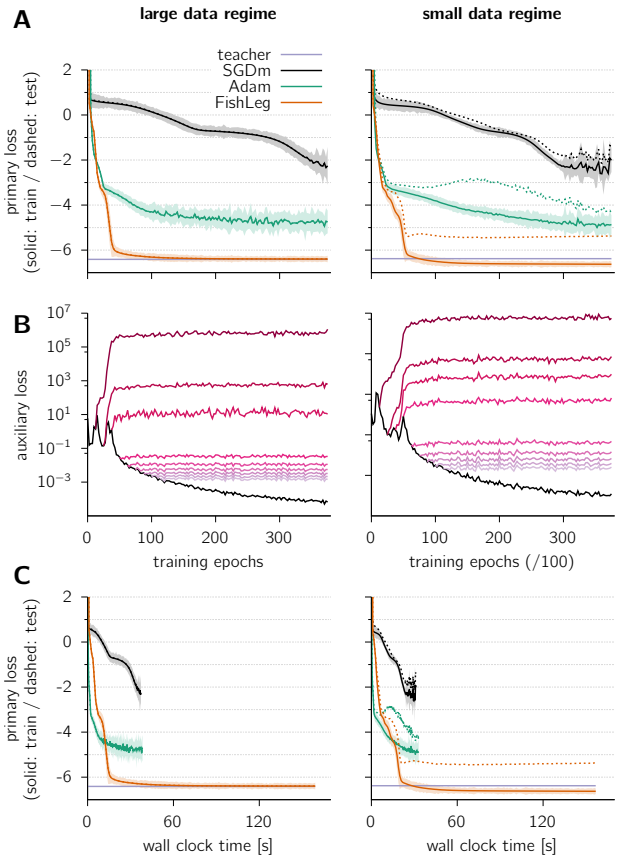

Figure 1: **Application to deep linear networks.** A deep linear network (20 layers of size 20 each) with Gaussian output likelihood is trained on a number of data samples (left: 4000; right: 40) generated by a noisy teacher in the same model class. **(A)** Loss evaluated on the training (solid) and testing set (dashed), as a function of training epoch. The gray line shows the noise floor, i.e. the test loss evaluated on the teacher network. Lines show mean across 10 independent experiments (teacher, initial conditions, etc), shadings show ± 1 stdev. **(B)** Evolution of the auxiliary loss during training (black), with Fisher-matrix-vector products evaluated exactly according to the analytical expression given in (Bernacchia et al., 2018). To assess how well the auxiliary optimization steps minimize the auxiliary loss – a moving target when $\boldsymbol{\theta}$ changes –, the auxiliary parameters $\boldsymbol{\lambda}$ are frozen every 500 iterations and subsequently used to re-evaluate the auxiliary loss (colored lines). **(C)** Same as (A), here as a function of wall-clock time. Parameters: minibatch size = 40, $\eta = 0.04$, $\alpha = 0.001$, $\beta = 0.9$, $\eta_{\text{SGDm}} = 0.002$, $\eta_{\text{Adam}} = 0.0002$.

2018, Huh, 2020). The results presented in Fig. 1 were obtained with a network of 20 (linear) layers of size $n = 20$ each. We generated data by instantiating a teacher network with random Gaussian weights $\mathcal{N}(0, 1/n)$ in each layer and a predictive density with mean equal to the activation of the last layer and isotropic Gaussian noise with variance $\sigma_0^2 = 0.001^2$. Input samples were drawn from a standard normal distribution, and the teacher's predictive distribution was sampled accordingly to obtain output labels. We investigated the behaviour of FishLeg both in the large data regime (4000 training samples) and the small data regime (40 training samples).

In both regimes, FishLeg very rapidly drove the training loss to the minimum set by the noise (σ_0), a minimum which was attained by neither of manually-tuned SGD-m and Adam (Fig. 1A). This is consistent with previous such comparisons in similar settings (Bernacchia et al., 2018). In the small data regime, FishLeg displays some overfitting but continues to compare favourably against first-order methods on test error. In wall-clock time, FishLeg was about 5 times slower than SGDm per training iteration in this case (Fig. 1C), but we show that a significant speed up can be obtained by updating $\boldsymbol{\lambda}$ every 10 iterations at little performance cost (see Fig. 3).

We used this simple experiment to assess the effectiveness of auxiliary loss optimization during the course of training. The lower the auxiliary loss, the closer it is from the Legendre conjugate of the cross-entropy at $\mathbf{g}$, and therefore the closer $Q(\boldsymbol{\lambda})\mathbf{g}$ is from the natural gradient. We assessed how much progress was made on the auxiliary loss, relative to how it would evolve if $\boldsymbol{\lambda}$ was held constant, as $\boldsymbol{\theta}$ is being optimized (Fig. 1B). This revealed two distinct phases of learning. At first, $\boldsymbol{\theta}$ (and therefore the FIM) changes rapidly, but $\boldsymbol{\lambda}$ rapidly adapts to keep the auxiliary loss in check (compare black and dark purple curves) – we speculate that this mostly reflects adaptation to the overall scale of the (inverse) FIM. Later on, as θ changes less rapidly, $\boldsymbol{\lambda}$ begins to learn the more subtle geometry of the FIM, giving rise to rather modest improvements on the auxiliary loss (compare black and lighter purple curves).

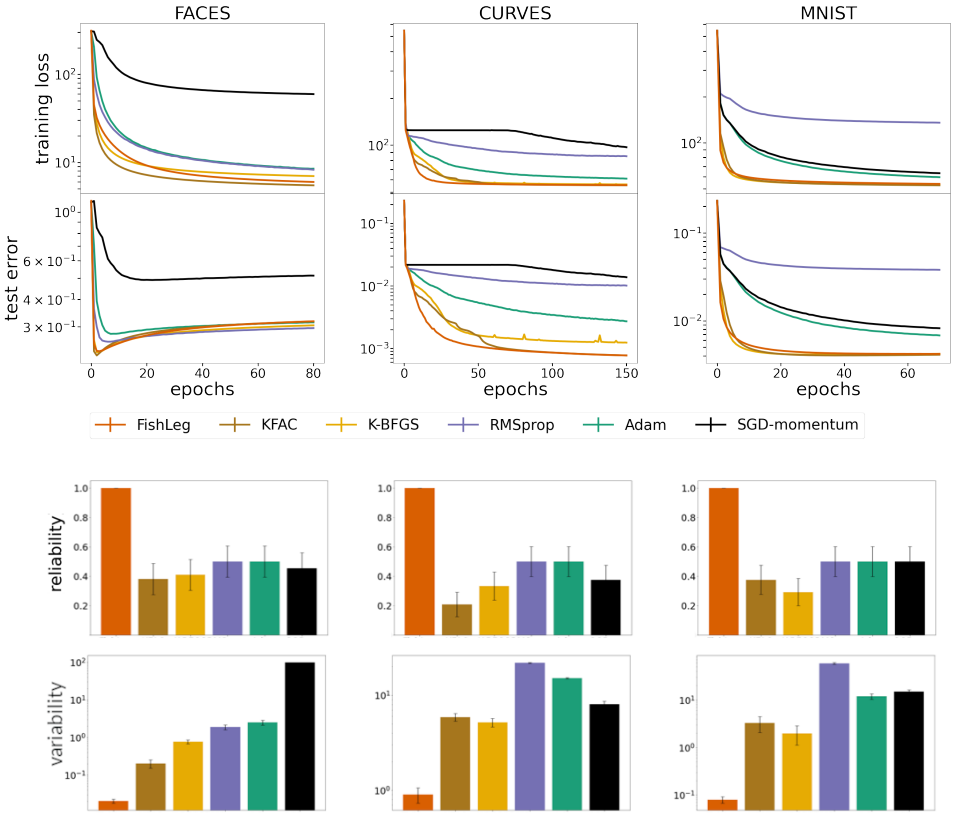

Figure 2: **Comparison between FishLeg and other optimizers** on standard auto-encoders benchmarks (batch size 100). **Top**: Training loss and test error as a function of training epochs on the 3 datasets, averaged over 10 different seeds. **Middle**: reliability of FishLeg across runs. For each optimizer and dataset, the fraction of runs (estimated from 10 runs) that converged properly (i.e. where the loss did not run out to 'nan') is shown; error bars show ± 1 s.e.m. **Bottom**: variability (standard deviation) of the training loss across runs, averaged over epochs (discarding any epochs where the loss might have exploded). See supplementary Fig. 5 for results on batch size 1000.

5.2 Second-order optimization benchmark

We applied FishLeg to the auto-encoders benchmarks previously used to compare second-order optimization methods – the details of these experiments (model architectures, datasets, etc) can be found in (Goldfarb et al., 2020), and hyperparameters specific to FishLeg are Table 1 (Appendix). To compare FishLeg to other optimizers, we used the code provided by Goldfarb et al. which implemented Adam, RMSprop, KFAC and KBFGS.

These autoencoder benchmarks are difficult problems on which the family of second-order methods has been shown to improve substantially over first-order optimizers (e.g. SGDm) and optimizers that exploit second-order information in diagonal form (Adam, RMSprop). Yet, no clear differences in performance has emerged *within* second-order methods. As far as training loss and test errors are concerned, our results confirmed this trend: FishLeg performed similarly to KFAC and KBFGS on the FACES and MNIST datasets, although it did converge within fewer iterations on the CURVES dataset (Fig. 2, top).

Interestingly, however, we found FishLeg to be consistently more robust than other methods, especially when using relatively small batch sizes. In particular, it converged more reliably over repeated runs with different random initializations (Fig. 2, middle). This is perhaps due to other methods achieving their best performance in a near-critical hyperparameter regime in which optimization tends to fail half of the time (these failed runs are often discarded when averaging). In contrast, FishLeg achieved similar or better performance in a more stable hyperparameter regime, and did not

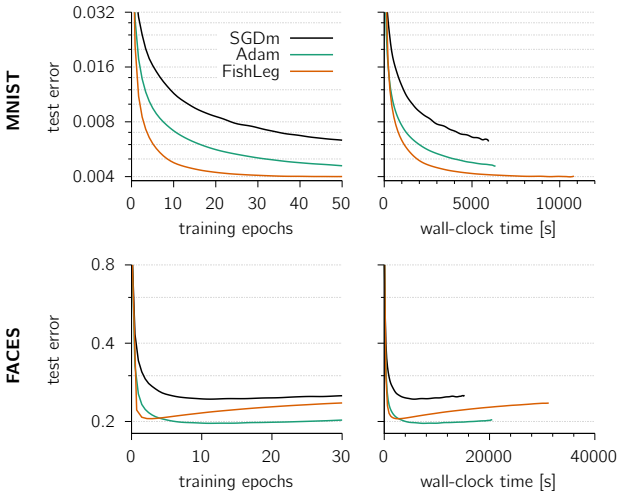

Figure 3: Wall-clock time comparisons on MNIST and FACES, with optimizers all implemented in the same way on CPU (Intel Xeon Platinum 8380H @ 2.90GHz) with OpenBLAS compiled for that architecture and multi-threaded with OpenMP (8 threads). Within each dataset, all curves show the same number of epochs. In FishLeg, the auxiliary parameters were updated only every 10 iterations, with no noticeable drop in performance but large wall-clock speedup, such that FishLeg here is roughly twice slower than SGDm.

fail a single of the 10 runs we performed. Similarly, FishLeg was also more consistent than other methods, i.e. displayed a lower variance in training loss across runs (Fig. 2, bottom).

Given the heterogeneity of software systems in which the various methods were implemented (e.g. JAX vs PyTorch), we decided to run a clean wallclock-time comparison between SGDm, Adam and FishLeg using a unified CPU-only implementation. This ensured e.g. that the loss and its gradients were computed in exactly the same way across methods. We performed this for the FACES and MNIST benchmarks. Overall, one iteration of vanilla FishLeg was ~ 5 times slower than one iteration of SGDm. However, we were able to bring this down to only twice slower by updating $\boldsymbol{\lambda}$ every 10 iterations, which did not significantly affect performance. Combined with FishLeg's faster progress per-iteration, this meant that FishLeg retained a significant advantage in wall-clock time over SGD (Fig.3). In practice we think that it might make sense to update $\boldsymbol{\lambda}$ more frequently at the beginning of training, and let these updates become sparser as optimization progresses – we leave this to future investigations.

6 Discussion

We provided a general framework for approximating the natural gradient through online meta-learning of the LF conjugate of a specific cross-entropy. Our framework is general: different choices can be made for how to meta-learn the LF conjugate (Eq. 11), parameterize its gradient (Eq. 14) and evaluate/differentiate the auxiliary loss (Eq. 15). Beyond our specific implementation, future work will study alternative choices that may be more efficient. For example, implicit differentiation (Lorraine et al., 2020, Rajeswaran et al., 2019), or evolutionary strategies (Metz et al., 2019,Gao & Sener, 2022) may be used to meta-learn the LF conjugate. The auxiliary loss could also be evaluated without taking the small ϵ limit but using antithetic estimators to reduce variance (Gao & Sener, 2022). Alternative parameterizations could be used for $\overline{\boldsymbol{\delta}}$ or Q. Indeed, we have preliminary results showing that more expressive choices for Q achieve a better approximation of the inverse FIM (see Fig. 4 in appendix A.6), and future work will determine whether this observation translates into better performance.

Previous work on natural gradient aimed at computing and inverting analytically an approximation to the FIM, which was done on a case-by-case basis for dense (Martens & Grosse, 2015) and convolutional (Grosse & Martens, 2016) layers, and standard recurrent networks (Martens et al., 2018). For example, to our knowledge no approximation has been derived for transformers neural networks. By parameterizing the inverse FIM directly, our approach allow the user to express their assumptions about the structure of parameter *precisions*, which is typically easier to reason about than the structure of parameter *covariances*. We therefore expect it will facilitate future applications of natural gradient optimization to a broader range of network architectures.

REFERENCES

Shun-ichi Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2):251–276, February 1998. ISSN 0899-7667, 1530-888X. doi: 10.1162/089976698300017746. URL http://www.mitpressjournals.org/doi/10.1162/089976698300017746.

Ehsan Amid, Rohan Anil, and Manfred Warmuth. Locoprop: Enhancing backprop via local loss optimization. In *International Conference on Artificial Intelligence and Statistics*, pp. 9626–9642. PMLR, 2022.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

Achraf Bahamou, Donald Goldfarb, and Yi Ren. An Adaptive Mini-Block Fisher Method for Deep Neural Networks, July 2022. URL http://arxiv.org/abs/2202.04124. arXiv:2202.04124 [cs].

Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V Le. Neural optimizer search with reinforcement learning. In *International Conference on Machine Learning*, pp. 459–468. PMLR, 2017.

Frederik Benzing. Gradient Descent on Neurons and its Link to Approximate Second-order Optimization. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 1817–1853. PMLR, June 2022. URL https://proceedings.mlr.press/v162/benzing22a.html. ISSN: 2640-3498.

Alberto Bernacchia, Máté Lengyel, and Guillaume Hennequin. Exact natural gradient in deep linear networks and its application to the nonlinear case. *Advances in Neural Information Processing Systems*, 31, 2018.

Stephen P. Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, Cambridge, UK ; New York, 2004. ISBN 978-0-521-83378-3.

Richard H Byrd, Gillian M Chin, Will Neveitt, and Jorge Nocedal. On the use of stochastic Hessian information in optimization methods for machine learning. *SIAM Journal on Optimization*, 21 (3):977–995, 2011.

Richard H Byrd, Samantha L Hansen, Jorge Nocedal, and Yoram Singer. A stochastic quasi-Newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2):1008–1031, 2016.

Zachary Charles and Dimitris Papailiopoulos. Stability and Generalization of Learning Algorithms that Converge to Global Optima. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 745–754. PMLR, July 2018. URL https://proceedings.mlr.press/v80/charles18a.html. ISSN: 2640-3498.

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to Optimize: A Primer and A Benchmark. *arXiv:2103.12828 [cs, math, stat]*, July 2021. URL http://arxiv.org/abs/2103.12828. arXiv: 2103.12828.

Sélim Chraibi, Adil Salim, Samuel Horváth, Filip Hanzely, and Peter Richtárik. Learning to Optimize via Dual space Preconditioning. December 2019. URL https://openreview.net/forum?id=rklx-gSYPS.

Ross M Clarke, Elre T Oldewage, and José Miguel Hernández-Lobato. Scalable one-pass optimisation of high-dimensional weight-update hyperparameters by implicit differentiation. *arXiv preprint arXiv:2110.10461*, 2021.

Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *Advances in neural information processing systems*, 27, 2014.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *International Conference on Machine Learning*, pp. 1165–1173. PMLR, 2017.

Katelyn Gao and Ozan Sener. Generalizing Gaussian smoothing for random search. In *International Conference on Machine Learning*, pp. 7077–7101. PMLR, 2022.

Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast approximate natural gradient descent in a Kronecker factored eigenbasis. *Advances in Neural Information Processing Systems*, 31, 2018.

Donald Goldfarb, Yi Ren, and Achraf Bahamou. Practical quasi-newton methods for training deep neural networks. *Advances in Neural Information Processing Systems*, 33:2386–2396, 2020.

Roger Grosse and James Martens. A Kronecker-factored approximate Fisher matrix for convolution layers. In *International Conference on Machine Learning*, pp. 573–582. PMLR, 2016.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9): 5149–5169, 2021.

Dongsung Huh. Curvature-corrected learning dynamics in deep neural networks. *ICML*, pp. 9, 2020.

Anna Kerekes, Anna Mészáros, and Ferenc Huszár. Depth Without the Magic: Inductive Bias of Natural Gradient Descent. *arXiv:2111.11542 [cs, stat]*, November 2021. URL `http://arxiv.org/abs/2111.11542`. arXiv: 2111.11542.

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.

Ke Li and Jitendra Malik. Learning to Optimize. *arXiv:1606.01885 [cs, math, stat]*, June 2016. URL `http://arxiv.org/abs/1606.01885`. arXiv: 1606.01885.

Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pp. 1540–1552. PMLR, 2020.

David JC MacKay, David JC Mac Kay, et al. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International conference on machine learning*, pp. 2113–2122. PMLR, 2015.

Chris J Maddison, Daniel Paulin, Yee Whye Teh, and Arnaud Doucet. Dual space preconditioning for gradient descent. *SIAM Journal on Optimization*, 31(1):991–1016, 2021.

James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417. PMLR, 2015.

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

James Martens et al. Deep learning via Hessian-free optimization. In *ICML*, volume 27, pp. 735–742, 2010.

Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 4556–4565. PMLR, May 2019. URL `https://proceedings.mlr.press/v97/metz19a.html`. ISSN: 2640-3498.

Alexander Meulemans, Francesco Carzaniga, Johan Suykens, João Sacramento, and Benjamin F Grewe. A theoretical framework for target propagation. *Advances in Neural Information Processing Systems*, 33:20024–20036, 2020.

Paul Micaelli and Amos J Storkey. Gradient-based hyperparameter optimization over long horizons. *Advances in Neural Information Processing Systems*, 34:10798–10809, 2021.

Philipp Moritz, Robert Nishihara, and Michael Jordan. A Linearly-Convergent Stochastic L-BFGS Algorithm. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pp. 249–258. PMLR, May 2016. URL https://proceedings.mlr.press/v51/moritz16.html. ISSN: 1938-7228.

Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

Eunbyung Park and Junier B Oliva. Meta-curvature. *Advances in Neural Information Processing Systems*, 32, 2019.

Adityanarayanan Radhakrishnan, Mikhail Belkin, and Caroline Uhler. Linear convergence of generalized mirror descent with time-dependent mirrors. *arXiv preprint arXiv:2009.08574*, 2020.

Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. Meta-learning with implicit gradients. *Advances in neural information processing systems*, 32, 2019.

C Radhakrishna Rao. Information and the accuracy attainable in the estimation of statistical parameters. In *Breakthroughs in statistics*, pp. 235–247. Springer, 1992.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Robin M. Schmidt, Frank Schneider, and Philipp Hennig. Descending through a Crowded Valley - Benchmarking Deep Learning Optimizers. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 9367–9376. PMLR, July 2021. URL https://proceedings.mlr.press/v139/schmidt21a.html. ISSN: 2640-3498.

Saeed Soori, Bugra Can, Baourun Mu, Mert Gürbüzbalaban, and Maryam Mehri Dehnavi. TENGraD: Time-Efficient Natural Gradient Descent with Exact Fisher-Block Inversion, March 2022. URL http://arxiv.org/abs/2106.03947. arXiv:2106.03947 [cs].

Valentin Thomas, Fabian Pedregosa, Bart Merriënboer, Pierre-Antoine Manzagol, Yoshua Bengio, and Nicolas Le Roux. On the interplay between noise and curvature and its effect on optimization and generalization. In *International Conference on Artificial Intelligence and Statistics*, pp. 3503–3513. PMLR, 2020.

Paul Vicol, Luke Metz, and Jascha Sohl-Dickstein. Unbiased Gradient Estimation in Unrolled Computation Graphs with Persistent Evolution Strategies. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 10553–10563. PMLR, July 2021. URL https://proceedings.mlr.press/v139/vicol21a.html. ISSN: 2640-3498.

Olga Wichrowska, Niru Maheswaranathan, Matthew W. Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. Learned Optimizers that Scale and Generalize. In *International Conference on Machine Learning*, pp. 3751–3760. PMLR, July 2017. URL https://proceedings.mlr.press/v70/wichrowska17a.html. ISSN: 2640-3498.

Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding Short-Horizon Bias in Stochastic Meta-Optimization. *arXiv:1803.02021 [cs, stat]*, March 2018. URL http://arxiv.org/abs/1803.02021. arXiv: 1803.02021.

Royce KP Zia, Edward F Redish, and Susan R McKay. Making sense of the legendre transform. *American Journal of Physics*, 77(7):614–622, 2009.

A APPENDIX

A.1 ALGORITHM

Algorithm 1 FishLeg algorithm (online setting)

1: **function** UPDATE_AUX($\boldsymbol{\theta}$, $\mathbf{g}$, $\boldsymbol{\lambda}$, `Adam_state`)
2: $\tilde{\mathbf{g}} \leftarrow \mathbf{g}/\|\mathbf{g}\|$ ▷ *normalize gradient*
3: initialize the adjoint of $\boldsymbol{\lambda}$ to prepare for automatic differentiation reverse pass (c.f. line 9).
4: $\bar{\boldsymbol{\delta}} \leftarrow Q(\boldsymbol{\lambda})\tilde{\mathbf{g}}$ ▷ *can exploit fast matrix-vector products without forming Q*
5: ▷ *Hessian-vector product on the cross-entropy $\mathcal{H}$ (with $\bar{\boldsymbol{\delta}}$ taken off the automatic differentiation tape!) evaluated on a mini-batch different from the one used to obtain $\mathbf{g}$.* ◁
6: $\mathbf{h} \leftarrow$ `Hess_vec_prod(fun` $\boldsymbol{\delta} \mapsto \mathcal{H}(\boldsymbol{\theta}, \boldsymbol{\delta})$, at $\boldsymbol{\delta} = 0$, along $\mathbf{v} =$`stop_gradient(`$\bar{\boldsymbol{\delta}}$`))`
7: $\mathbf{d} \leftarrow \mathbf{h} - \tilde{\mathbf{g}}$
8: `aux_loss` $\leftarrow \frac{1}{2}(d - \tilde{g})^\top$`stop_gradient(`$\bar{\boldsymbol{\delta}}$`)` ▷ *log the value of the auxiliary loss*
9: $\Delta_\lambda \leftarrow$ adjoint of $\boldsymbol{\lambda}$ after completion of reverse pass on surrogate auxiliary loss $d^\top \bar{\boldsymbol{\delta}}$
10: $\boldsymbol{\lambda}$, `Adam_state` $\leftarrow$ Adam_update($\boldsymbol{\lambda}, \Delta_\lambda, \alpha,$ `Adam_state`) ▷ *α is the learning rate*
11: **return** `aux_loss`, λ, `Adam_state`
12:
13: $t \leftarrow 0$, initialize $\boldsymbol{\theta}_0$ and $\boldsymbol{\lambda}_0$
14: **while** not converged **do**
15: $\mathcal{L}, \mathbf{g} \leftarrow$ value and gradient of negative log likelihood evaluated at $\boldsymbol{\theta}_t$ on a minibatch
16: $\bar{\mathbf{g}} \leftarrow \beta\bar{\mathbf{g}} + (1 - \beta)\mathbf{g}$ ▷ *momentum*
17: ▷ *Note: the following step needs not be performed at every iteration – wall-clock time speedups can be obtained by running this every 10 iterations, though this might depend on the problem.* ◁
18: `aux_loss`, $\boldsymbol{\lambda}_{t+1}$, `Adam_state` $\leftarrow$ UPDATE_AUX($\boldsymbol{\theta}_t, \bar{\mathbf{g}}, \boldsymbol{\lambda}_t,$ `Adam_state`)
19: $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta Q(\boldsymbol{\lambda}_{t+1})\bar{\mathbf{g}}$
20: $t \leftarrow t + 1$

A.2 PROOF OF THEOREM 1

We start by stating the following Lemma, which is also known as Gibbs' inequality (see e.g. Chapter 2.6 of MacKay et al., 2003)

Lemma 1. *Assume that $p(\mathcal{D}|\boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta} + \boldsymbol{\delta})$ if and only if $\boldsymbol{\delta} = \mathbf{0}$. Then, for a fixed $\boldsymbol{\theta}$, the cross entropy between $p(\mathcal{D}|\boldsymbol{\theta})$ and $p(\mathcal{D}|\boldsymbol{\theta} + \boldsymbol{\delta})$*

$$\mathcal{H}(\boldsymbol{\theta}, \boldsymbol{\delta}) = \mathbb{E}_{\mathcal{D}\sim p(\mathcal{D}|\boldsymbol{\theta})}\ell(\boldsymbol{\theta} + \boldsymbol{\delta}, \mathcal{D}) \tag{22}$$

has a unique global minimum at $\boldsymbol{\delta} = \mathbf{0}$.

Proof of theorem 1. We start by computing the gradient and Hessian of the cross-entropy $\mathcal{H}(\boldsymbol{\theta}, \boldsymbol{\delta})$ with respect to $\boldsymbol{\delta}$, computed at $\boldsymbol{\delta} = 0$.

$$\nabla_{\boldsymbol{\delta}}\mathcal{H}(\boldsymbol{\theta}, \mathbf{0}) = \mathbb{E}_{\mathcal{D}\sim p(\mathcal{D}|\boldsymbol{\theta})}\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}, \mathcal{D}) = \mathbf{0} \tag{23}$$

$$\nabla^2_{\boldsymbol{\delta}}\mathcal{H}(\boldsymbol{\theta}, \mathbf{0}) = \mathbb{E}_{\mathcal{D}\sim \mathbf{p}(\mathcal{D}|\boldsymbol{\theta})}\nabla^{\mathbf{2}}_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}, \mathcal{D}) = \mathcal{I}(\boldsymbol{\theta}) \tag{24}$$

The gradient is equal to the expectation of the score function, which is known to be equal to zero (as also implied by lemma 1). The Hessian is equal to the FIM, as expressed by equation 2. By Lemma 1, $\boldsymbol{\delta} = \mathbf{0}$ is the unique minimum of $\mathcal{H}$, therefore the FIM $\mathcal{I}$ is positive definite and its inverse exists.

Using property 9 of the LF conjugate, we express the *inverse* Fisher matrix as the Hessian of the LF conjugate of $\mathcal{H}$. We denote by $\mathcal{H}^\star(\boldsymbol{\theta}, \mathbf{u})$ the LF conjugate of $\mathcal{H}(\boldsymbol{\theta}, \boldsymbol{\delta})$, as a function of its second argument $\boldsymbol{\delta}$. Using the definition of LF conjugate 6, that is equal to

$$\mathcal{H}^\star(\boldsymbol{\theta}, \mathbf{u}) = \min_{\boldsymbol{\delta}} \mathcal{H}(\boldsymbol{\theta}, \boldsymbol{\delta}) - \mathbf{u}^T\boldsymbol{\delta} \tag{25}$$

We denote by $\tilde{\boldsymbol{\delta}}(\boldsymbol{\theta}, \mathbf{u})$ the minimizer of this expression

$$\tilde{\boldsymbol{\delta}}(\boldsymbol{\theta}, \mathbf{u}) = \operatorname{argmin}_{\boldsymbol{\delta}} \mathcal{H}(\boldsymbol{\theta}, \boldsymbol{\delta}) - \mathbf{u}^T\boldsymbol{\delta} \tag{26}$$

Using the property of LF conjugate, equation 9, we have that

$$\nabla^2_{\mathbf{u}}\mathcal{H}^\star(\boldsymbol{\theta},\mathbf{u}) = \left(\nabla^2_{\boldsymbol{\delta}}\mathcal{H}(\boldsymbol{\theta},\tilde{\boldsymbol{\delta}}(\boldsymbol{\theta},\mathbf{u}))\right)^{-1} \tag{27}$$

Comparing with the expression 24 of the FIM, the right hand side of this equation is equal to the inverse FIM when $\tilde{\boldsymbol{\delta}}(\boldsymbol{\theta},\mathbf{u})$ is equal to zero. By lemma 1, we have that $\tilde{\boldsymbol{\delta}}(\boldsymbol{\theta},\mathbf{0}) = \mathbf{0}$ and therefore

$$\mathcal{I}(\boldsymbol{\theta})^{-1} = \nabla^2_{\mathbf{u}}\mathcal{H}^\star(\boldsymbol{\theta},\mathbf{0}) \tag{28}$$

Finally, using the properties of LF conjugate, equation 7, we have that $\tilde{\boldsymbol{\delta}}(\boldsymbol{\theta},\mathbf{u}) = \nabla_{\mathbf{u}}\mathcal{H}^\star(\boldsymbol{\theta},\mathbf{u})$ and the inverse FIM is equal to

$$\mathcal{I}(\boldsymbol{\theta})^{-1} = \nabla_{\mathbf{u}}\tilde{\boldsymbol{\delta}}(\boldsymbol{\theta},\mathbf{0}) \tag{29}$$

□

A.3 Informal argument for theorem 1

Theorem 1 suggests that computation of the inverse FIM requires computing $\tilde{\boldsymbol{\delta}}(\boldsymbol{\theta},\mathbf{u})$ near $\mathbf{u} = \mathbf{0}$. By lemma 1, we have that $\tilde{\boldsymbol{\delta}}(\boldsymbol{\theta},\mathbf{0}) = \mathbf{0}$, therefore we may hypothesize that $\tilde{\boldsymbol{\delta}}(\boldsymbol{\theta},\mathbf{u})$ is near zero when $\mathbf{u}$ is also near zero. Under this assumption, and using equations 23, 24, we may approximate the cross entropy $\mathcal{H}(\boldsymbol{\theta},\boldsymbol{\delta})$ by a second order Taylor expansion in $\boldsymbol{\delta}$:

$$\mathcal{H}(\boldsymbol{\theta},\boldsymbol{\delta}) - \mathbf{u}^T\boldsymbol{\delta} \;\simeq\; \mathcal{H}(\boldsymbol{\theta},\mathbf{0}) + \frac{1}{2}\boldsymbol{\delta}^T\mathcal{I}(\boldsymbol{\theta})\boldsymbol{\delta} - \mathbf{u}^T\boldsymbol{\delta} \tag{30}$$

Minimizing this expression with respect to $\boldsymbol{\delta}$ results in

$$\tilde{\boldsymbol{\delta}}(\boldsymbol{\theta},\mathbf{u}) = \mathcal{I}(\boldsymbol{\theta})^{-1}\mathbf{u} \tag{31}$$

which implies the statement of the theorem, $\mathcal{I}(\boldsymbol{\theta})^{-1} = \nabla_{\mathbf{u}}\tilde{\boldsymbol{\delta}}(\boldsymbol{\theta},\mathbf{0})$.

A.4 Proofs of convergence

Definition 1 (PL Condition)**.** *We say that a function satisfies the PL condition with parameter $\mu \in \mathbb{R}_{>0}$, if the following holds*

$$\|\nabla\mathcal{L}(\boldsymbol{\theta})\|^2 \geq \mu(\mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}^\star)), \quad \forall\boldsymbol{\theta} \tag{32}$$

We use the notation μ-PL to denote the class of functions satisfying the PL condition with parameter μ.

This condition implies that every stationary point is a global minimum. It however does not imply either uniqueness of the global minimum nor convexity.

Definition 2 (Smoothness)**.** *We say that a function is smooth with parameter ξ, if the following holds*

$$\mathcal{L}(\boldsymbol{\theta}') \leq \mathcal{L}(\boldsymbol{\theta}) + \nabla\mathcal{L}(\boldsymbol{\theta})^T(\boldsymbol{\theta}' - \boldsymbol{\theta}) + \frac{\xi}{2}\|\boldsymbol{\theta}' - \boldsymbol{\theta}\|^2, \quad \forall\boldsymbol{\theta},\boldsymbol{\theta}' \tag{33}$$

We use the notation ξ-smooth to denote the class of smooth functions with parameter ξ.

Proof of theorem 2. We show that the error at each iteration $t+1$ is linearly related to the error at iteration t.

$$
\begin{aligned}
& \mathcal{L}(\boldsymbol{\theta}_{t+1}) - \mathcal{L}(\boldsymbol{\theta}_t) \\
\leq \quad & -\eta \mathbf{g}(\boldsymbol{\theta}_t)^\top Q \mathbf{g}(\boldsymbol{\theta}_t) + \frac{\xi \eta^2}{2} \mathbf{g}(\boldsymbol{\theta}_t)^\top Q^2 \mathbf{g}(\boldsymbol{\theta}_t) && (34) \\
\leq \quad & -\eta \mathbf{g}(\boldsymbol{\theta}_t)^\top Q \mathbf{g}(\boldsymbol{\theta}_t) + \frac{\xi \lambda_{\max} \eta^2}{2} \mathbf{g}(\boldsymbol{\theta}_t)^\top Q \mathbf{g}(\boldsymbol{\theta}_t) && (35) \\
= \quad & -\eta (1 - \frac{\xi \lambda_{\max} \eta}{2}) \mathbf{g}(\boldsymbol{\theta}_t)^\top Q \mathbf{g}(\boldsymbol{\theta}_t) && (36) \\
\leq \quad & -\frac{\eta \lambda_{\min}}{2} \|\mathbf{g}(\boldsymbol{\theta}_t)\|^2 && (37) \\
\leq \quad & -\frac{\eta \mu \lambda_{\min}}{2} (\mathcal{L}(\boldsymbol{\theta}_t) - \mathcal{L}(\boldsymbol{\theta}^*)) && (38) \\
= \quad & -\frac{\mu \lambda_{\min}}{2 \xi \lambda_{\max}} (\mathcal{L}(\boldsymbol{\theta}_t) - \mathcal{L}(\boldsymbol{\theta}^*)) && (39)
\end{aligned}
$$

The first line is a result of smoothness assumption and the update rule 17. The second line follows from the definition of $\lambda_{\max}$. The third line is only a rearrangement of terms. The fourth line follows from the definition of $\lambda_{\min}$ and the choice of $\eta = \frac{1}{\xi \lambda_{\max}}$. The fifth line holds by PL condition. The last line follows from replacing the value of $\eta = \frac{1}{\xi \lambda_{\max}}$.

We thus have

$$
\mathcal{L}(\boldsymbol{\theta}_{t+1}) - \mathcal{L}(\boldsymbol{\theta}^*) \leq (1 - \frac{\mu \lambda_{\min}}{2 \xi \lambda_{\max}}) \left(\mathcal{L}(\boldsymbol{\theta}_t) - \mathcal{L}(\boldsymbol{\theta}^*) \right).
$$

Applying this recursive relation over t, starting from $\boldsymbol{\theta}_0$, we arrive at

$$
\mathcal{L}(\boldsymbol{\theta}_t) - \mathcal{L}(\boldsymbol{\theta}^*) \leq (1 - \frac{\mu \lambda_{\min}}{2 \xi \lambda_{\max}})^t \left(\mathcal{L}(\boldsymbol{\theta}_0) - \mathcal{L}(\boldsymbol{\theta}^*) \right).
$$

□

Proof of theorem 3. We show that the error at each iteration $t+1$ is related to the error at iteration t as follows.

$$
\begin{aligned}
& \mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}_{t+1}) - \mathcal{L}(\boldsymbol{\theta}_t) | \boldsymbol{\theta}_t \right] \\
\leq \quad & \mathbb{E}\left[-\eta_t \mathbf{g}(\boldsymbol{\theta}_t)^\top Q \hat{\mathbf{g}}(\boldsymbol{\theta}_t) + \frac{\xi \eta_t^2}{2} \hat{\mathbf{g}}(\boldsymbol{\theta}_t)^\top Q^2 \hat{\mathbf{g}}(\boldsymbol{\theta}_t) | \boldsymbol{\theta}_t \right] && (40) \\
\leq \quad & -\eta_t \mathbf{g}(\boldsymbol{\theta}_t)^\top Q \mathbf{g}(\boldsymbol{\theta}_t) + \mathbb{E}\left[\frac{\xi \lambda_{\max}^2 \eta_t^2}{2} \|\hat{\mathbf{g}}(\boldsymbol{\theta}_t)\|^2 | \boldsymbol{\theta}_t \right] && (41) \\
\leq \quad & -\eta_t \lambda_{\min} \|\mathbf{g}(\boldsymbol{\theta}_t)\|^2 + \frac{\xi \eta_t^2 \lambda_{\max}^2 G^2}{2} && (42) \\
\leq \quad & -\eta_t \mu \lambda_{\min} (\mathcal{L}(\boldsymbol{\theta}_t) - \mathcal{L}(\boldsymbol{\theta}^*)) + \frac{\xi \eta_t^2 \lambda_{\max}^2 G^2}{2} && (43)
\end{aligned}
$$

The first line is a result of smoothness assumption and the update rule 17. The second line follows from the definition of $\lambda_{\max}$. The third line follows from the upper bound assumption on the norm of gradient. The fourth line holds by PL condition.

We thus have

$$
\mathcal{L}(\boldsymbol{\theta}_{t+1}) - \mathcal{L}(\boldsymbol{\theta}^*) \leq (1 - \eta_t \mu \lambda_{\min})(\mathcal{L}(\boldsymbol{\theta}_t) - \mathcal{L}(\boldsymbol{\theta}^*)) + \frac{\xi \eta_t^2 \lambda_{\max}^2 G^2}{2}
$$

Recall the choice of $\eta_t = \frac{1}{\mu\lambda_{\min}(t+1)}$. By definition of A, we have $\mathcal{L}(\boldsymbol{\theta}_t) - \mathcal{L}(\boldsymbol{\theta}^*) \leq \frac{A}{t+1}$ when $t = 0$. Using induction, we prove the same for $\mathcal{L}(\boldsymbol{\theta}_{t+1}) - \mathcal{L}(\boldsymbol{\theta}^*)$.

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}_{t+1}) - \mathcal{L}(\boldsymbol{\theta}^*) &\leq (1 - \frac{1}{t+1})\frac{A}{(t+1)} + \frac{\xi\lambda_{\max}^2 G^2}{2\mu^2\lambda_{\min}^2(t+1)^2} && (44)\\
&\leq \frac{A}{t+1} - \frac{A}{2(t+1)^2} && (45)\\
&= A(\frac{1}{t+1} - \frac{1}{2(t+1)^2}) && (46)\\
&\leq \frac{A}{t+2}. && (47)
\end{aligned}$$

That completes the proof.

□

A.5 Parameterization of the matrix Q

Consider a multi-layer perceptron (MLP) neural network with L layers. The activation a_i^ℓ of neuron i at layer ℓ is equal to

$$a_i^\ell = \sigma_\ell\left(\sum_{j=1}^{N_{\ell-1}+1} W_{ij}^\ell a_j^{\ell-1}\right) \quad \text{for } 1 \leq i \leq N_\ell \tag{48}$$

$$\tag{49}$$

where layer ℓ has N_ℓ neurons and $\ell = 1, \ldots, L$. The symbol W^ℓ denotes the weight matrix from layer $\ell - 1$ to layer ℓ, and has size $N_\ell \times (N_{\ell-1} + 1)$. This weight matrix includes a bias, by setting the $N_\ell + 1$ activation at each layer equal to one:

$$a_{N_\ell+1}^\ell = 1 \tag{50}$$

The function σ_ℓ can be any nonlinearity such as ReLU or a Softmax in case of the last layer. The output of the neural network is defined as the activation in the last layer, a^L. For convenience of notation, the input to the neural network, denoted by x and of dimension N_0, is defined as activation at layer 0:

$$a_i^0 = x_i \quad \text{for } 1 \leq i \leq N_0 \tag{51}$$

$$a_{N_0+1}^0 = 1 \tag{52}$$

We structure the matrix Q as block-diagonal, each block corresponds to one layer and has the same number of rows and columns, equal to $N_\ell(N_{\ell-1}+1)$. For layer l, We parameterize the corresponding block, denoted by Q_l as

$$Q_\ell = (R_\ell R_\ell^T \otimes L_\ell L_\ell^T) \tag{53}$$

where the matrix R_ℓ has size $(N_{\ell-1} + 1) \times (N_{\ell-1} + 1)$ while the matrix L_ℓ has size $N_\ell \times N_\ell$. Both matrices L_ℓ, R_ℓ are lower triangular. This parameterization ensures that the matrix Q is positive definite. The auxiliary parameters $\boldsymbol{\lambda}$ are represented by the matrices L_ℓ, R_ℓ for all layers $\ell = 1, \ldots, L$, for a total of $\frac{1}{2}\sum_{\ell=1}^{L}\left[N_\ell(N_\ell + 1) + (N_{\ell-1} + 1)(N_{\ell-1} + 2)\right]$ auxiliary parameters. This number is much smaller than the total number of entries of the matrix Q, which is equal to $\left(\sum_{\ell=1}^{L} N_\ell(N_{\ell-1} + 1)\right)^2$

A.6 Preliminary investigations of more flexible approximations of the inverse FIM

Directly parameterizing and learning the inverse FIM lends more flexibility than parameterizing and learning the FIM in an easy-to-invert form. We conducted preliminary experiments with alternative parameterizations $Q(\boldsymbol{\lambda})$ of the inverse FIM (i.e. beyond the one described in Appendix A.5), all constrained to afford fast $Q(\boldsymbol{\lambda})\mathbf{g}$ products. In particular, we experimented with:

- a simple diagonal matrix Q (constrained to be positive definite), mostly included as a useful baseline
- the block Kronecker used in the paper and described in Appendix A.5, known to perform much better than a diagonal matrix;
- a modification of the block Kronecker form that introduces full inner and outer diagonal rescaling: at each layer ℓ, we took

$$Q_\ell = A_\ell(R_\ell \otimes L_\ell)B_\ell^2(R_\ell^T \otimes L_\ell^T)A_\ell$$

  where A_ℓ and B_ℓ are two diagonal matrices of the appropriate size. The presence of B_ℓ makes this similar to EKFAC (George et al., 2018), but A_ℓ makes it even more general;
- a sum of block Kronecker approximations: at each layer ℓ, we took

$$Q_\ell = (R_\ell^{(1)}{R_\ell^{(1)}}^T) \otimes (L_\ell^{(1)}{L_\ell^{(1)}}^T) + (R_\ell^{(2)}{R_\ell^{(2)}}^T) \otimes (L_\ell^{(2)}{L_\ell^{(2)}}^T)$$

All these approximations lend themselves to efficient $Q\mathbf{g}$ products (e.g. by exploiting standard properties of the Kronecker product).

To assess how well these approximations could approximate the inverse FIM at a point in optimization where high-quality curvature information matters, we pre-trained an MLP with layer sizes $[784, 300, 100, 30, 100, 300, 784]$ for 200 iterations on MNIST autoencoding using SGD-with-momentum. The resulting model parameters $\boldsymbol{\theta}$ were subsequently frozen, and the auxiliary loss was minimized to convergence (asymptotic value shown here) under each of the four different parameterizations of $Q(\boldsymbol{\lambda})$ described above. We found that diagonal rescaling improved only slightly over the vanilla block Kronecker approximation. However, there was a larger improvement going from the block Kronecker approximation to the sum-of-block-Kronecker approximation – indeed, the improvement on the auxiliary loss was a sizeable fraction of the improvement going from a diagonal approximation to single block-Kronecker terms. The latter is known to be highly consequential for training, and so we speculate that sum-of-Kronecker approximations might improve natural gradient descent in future work.

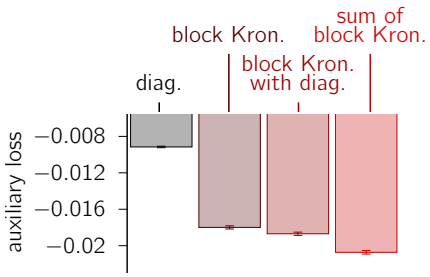

Figure 4: **Comparison of various forms of $Q(\boldsymbol{\lambda})$.** See text in Appendix A.6 for details.

A.7 FISHLEG AS AMORTIZATION OF CG STEPS IN HESSIAN-FREE OPTIMIZATION

FishLeg can be thought as a way of gradually amortizing the conjugate gradients (CG)-based inner loop in Hessian-free optimization. Looking back at Eq. 20, one might wonder: why not directly optimizing over the product $\mathbf{v} = Q(\boldsymbol{\lambda})\mathbf{g}$, instead of optimizing over $\boldsymbol{\lambda}$? Treating $\mathbf{v}$ as free parameters, this would entail minimizing $\frac{1}{2}\mathbf{v}^\top\mathcal{I}(\boldsymbol{\theta})\mathbf{v} - \mathbf{v}^\top\mathbf{g}$, which is in fact exactly what Hessian-free optimization does using CG (Byrd et al., 2011; Martens et al., 2010; though $\mathcal{I}$ is sometimes replaced by the Gauss-Newton approximation to the Hessian). Although each step of CG has a cost comparable to the computation of $\mathbf{g}$, in practice several iterations are required which tends to remove any advantage on wallclock time. By learning $Q(\boldsymbol{\lambda})$ instead of $Q(\boldsymbol{\lambda})\mathbf{g}$, FishLeg learns to amortize those CG steps over a progressively growing subspace of noisy gradients.

	FACES		CURVES		MNIST	
	100	1000	100	1000	100	1000
Damping γ	1.0	1e-3	1e-2	1.0	3e-1	3e-2
η	5e-2	5e-2	1e-2	1e-1	7e-2	5e-2
α	1e-4	1e-4	1e-5	5e-4	1e-4	1e-4
η_{SGDm}	1e-2	1e-2	3e-2	1e-2	1e-2	1e-3

Table 1: Optimal hyperparameter values for FishLeg, identified as the result of a grid search over the space shown in Table 2. These hyperparameters were chosen to minimise the training loss.

	FACES		CURVES		MNIST	
	100	1000	100	1000	100	1000
Damping γ	1e-2, 1e-1, 1.0, 10, 100	1e-3, 1e-2, 1e-1, 1.0, 10, 100	1e-4, 1e-3, 3e-2, 1e-2, 3e-1, 1.0, 10	1e-4, 1e-3, 1e-2, 1e-1, 1.0	1e-4, 1e-2, 1e-1, 3e-1, 1.0, 100	1e-4, 1e-2, 3e-2, 1.0, 100
η	1e-3, 5e-3, 1e-2, 5e-2	5e-4, 5e-3, 1e-3, 1e-2, 5e-2	1e-3, 5e-3, 1e-2, 5e-2, 7e-2, 1e-1, 2e-1, 5e-1	1e-3, 1e-2, 2e-2, 5e-2, 1e-1	1e-3, 5e-3, 1e-2, 5e-2, 7e-2, 1e-1, 2e-1, 5e-1	1e-3, 5e-3, 1e-2, 2e-2, 5e-2, 7e-2, 1e-1, 5e-1
α	5e-5, 8e-5, 1e-4, 5e-4, 1e-3, 5e-3	5e-5, 1e-4, 5e-4, 1e-3, 1e-2, 1e-1	1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3	1e-5, 1e-4, 5e-4, 1e-3, 1e-2	1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3	5e-5, 1e-4, 5e-4, 1e-3

Table 2: Entire space of hyperparameters explored to optimize the training loss for each task. All combinations were explored, in a grid fashion, and each combination was run three times with different seeds. Finally, the combination of hyperparameters with the lowest training loss was selected.

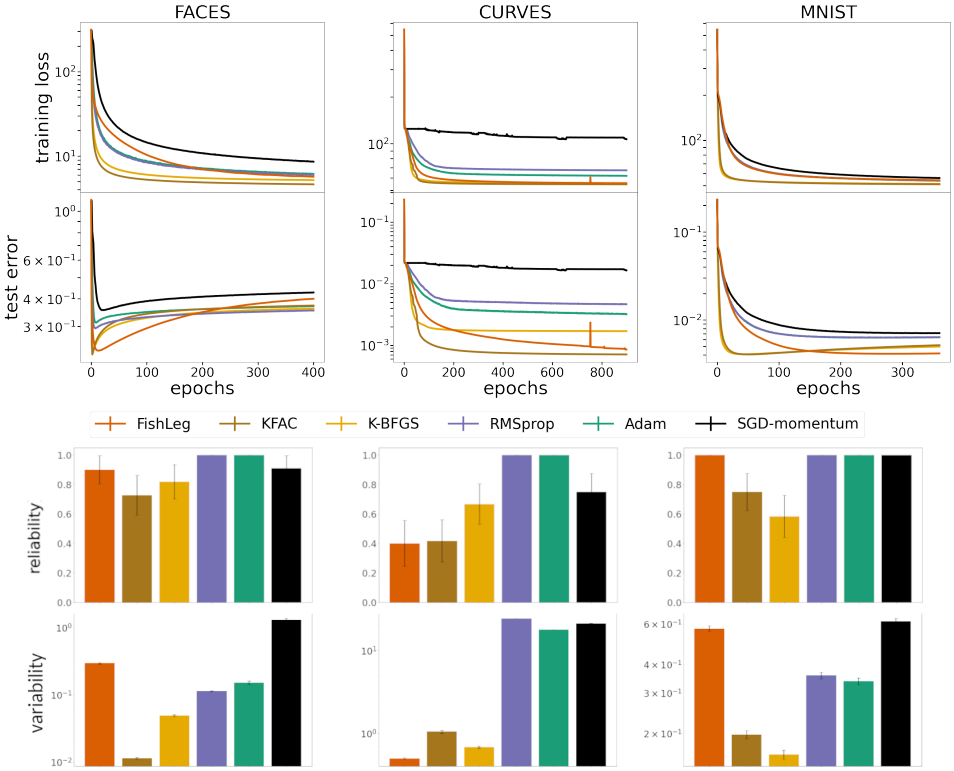

Figure 5: Same as Fig. 2 with a batch size of 1000.