Studying K-FAC Heuristics by Viewing Adam through a Second-Order Lens

Ross M. Clarke [1] **José Miguel Hernández-Lobato** [1]

Abstract

Research into optimisation for deep learning is characterised by a tension between the computational efficiency of first-order, gradient-based methods (such as SGD and Adam) and the theoretical efficiency of second-order, curvature-based methods (such as quasi-Newton methods and K-FAC). Noting that second-order methods often only function effectively with the addition of stabilising heuristics (such as Levenberg-Marquardt damping), we ask how much these (as opposed to the second-order curvature model) contribute to second-order algorithms' performance. We thus study *AdamQLR*: an optimiser combining damping and learning rate selection techniques from K-FAC (Martens & Grosse, 2015) with the update directions proposed by Adam, inspired by considering Adam through a second-order lens. We evaluate AdamQLR on a range of regression and classification tasks at various scales and hyperparameter tuning methodologies, concluding K-FAC's adaptive heuristics are of variable standalone general effectiveness, and finding an *untuned* AdamQLR setting can achieve comparable performance vs runtime to *tuned* benchmarks.

1. Introduction

At the heart of any machine learning model is an optimisation problem, and at the heart of any training procedure is an optimisation algorithm. Most frequently seen in the literature are *first-order* optimisers such as SGD, Adam (Kingma & Ba, 2015) and their variants, but exploratory studies have also been performed on *second-order* algorithms such as quasi-Newton methods and K-FAC (Martens & Grosse, 2015). Broadly speaking, second-order algorithms aim to secure more rapid convergence to an optimal value of the objective function by making more principled individual updates, which in turn are more computationally costly than those employed by first-order methods. Combined with a generally more complicated implementation, first-order methods are still preferable to second-order approaches for most practitioners (Anil et al., 2021).

In part, this is a stability issue — by virtue of taking larger individual steps, second-order optimisers carry an increased risk of significantly worsening the objective value if their approximate understanding of curvature in objective space is a poor representation of the true space. Most second-order approaches thus *depend* on additional heuristics (such as curvature damping) for their viability. Heuristics commonly seen in first-order methods, such as weight decay or momentum applied to SGD, improve an already effective optimiser; by contrast, second-order methods' heuristics are *essential* components, without which the optimiser will perform unstably or ineffectively — a point we demonstrate in Appendix B.3.4. It is then natural to ask to what extent these heuristics are responsible for the documented benefits of second-order optimisers, and whether they might similarly improve first-order techniques.

In this paper, we study a damped automatic learning rate strategy, derived by applying K-FAC's damping and learning rate selection techniques to Adam. The resulting algorithm — an optimiser whose default hyperparameters compare favourably with tuned baselines — allows us to investigate the impact of K-FAC's heuristics. After reviewing related work in Section 2, we present our study algorithm in Section 3. We then justify our claims by experiment in Section 4 before Section 5 concludes. Our main contributions are as follows:

- We argue K-FAC's adaptive heuristics are of variable general empirical effectiveness
- We present a novel use of damped, second-order approximate learning rate selection in Adam
- We show our untuned study method often performs similarly to methods using tuned hyperparameters, exhibiting robustness to hyperparameters

[1]University of Cambridge. Correspondence to: Ross M. Clarke <rmc78@cam.ac.uk>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

2. Related Work

First-order methods form the bread and butter of modern machine learning, with SGD and Adam (Kingma & Ba, 2015) being most frequently seen. Adam belongs to a class of *adaptive* first-order methods, which apply some kind of normalisation transformation to the observed gradients; other examples include Adagrad (McMahan & Streeter, 2010; Duchi et al., 2011) and RMSprop (Tieleman & Hinton, 2012). Balles & Hennig (2018) demonstrate that Adam essentially scales gradient signs by their variance. Zhang et al. (2018) show that Adam can be seen as a form of natural gradient mean field variational inference, whose mode-fitting behaviour is known to underestimate variance, corresponding to overestimating curvature in an optimisation task (see e.g. Figure 1.3 in Turner & Sahani (2011)). Zhang et al. (2019) use a noisy quadratic model to argue for the benefits of exponential moving averages and other components found in Adam. These methods achieve computational efficiency by using diagonal approximations or heuristics to understand curvature in the space, so ignore useful information which second-order methods incorporate.

Optimisers employing second-order derivative information are seen more often in the optimisation literature than in practical machine learning projects. The family of *quasi-Newton* methods (Nocedal & Wright, 2006) is inspired by the appearance of the Hessian matrix in a Taylor series truncated at quadratic order; this matrix characterises curvature in the model parameters. Martens (2010) use the Hessian-vector product trick (Pearlmutter, 1994) to work implicitly with the exact Hessian. Other work modifies the Hessian to avoid degeneracies — a particular concern in saddle point-dense high-dimensional spaces (Pascanu & Bengio, 2014; Dauphin et al., 2014) — or introduces spatial and temporal averaging of a diagonal approximation to the Hessian (Yao et al., 2021). Although not explicitly using second derivatives, SHAMPOO (Gupta et al., 2018) learns a factorised set of preconditioned matrices. However, in the non-convex, non-quadratic spaces of ML, the unaltered Hessian may be badly misleading, leading to diverging losses.

Where the system is viewed as a probabilistic model, an alternative curvature characterisation is the Fisher information matrix, which gives rise to the natural gradient family of methods (Amari, 1998). Unlike the Hessian, the Fisher matrix characterises curvature in KL-divergence space between the predicted and ground truth probability distributions. Factorized Natural Gradient (Grosse & Salakhudinov, 2015) approximates the Fisher using a Gaussian graphical model, while the Kronecker-Factored Approximate Curvature (K-FAC) method (Martens & Grosse (2015) after an idea by Heskes (2000)) imposes a block-diagonal approximation to the Fisher and represents each block by a Kronecker product. Extensions to K-FAC include EKFAC (George et al., 2018), which learns the approximate Fisher in an eigenvalue-aligned basis, and a mini-block Fisher variant Bahamou et al. (2023) targets the empirical Fisher by asserting that every diagonal block is itself block-diagonal. TNT (Ren & Goldfarb, 2021) exploits the Kronecker-factored covariance of the assumed tensor normal-distributed sample gradients to approximate the exact probabilistic Fisher matrix, while K-BFGS (Goldfarb et al., 2020) applies a similar factorisation strategy to the Hessian matrix, retaining theoretical guarantees from the classical BFGS optimiser (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970). Although K-FAC can be applied in distributed settings, this is somewhat complex (Osawa et al., 2019), and Fisher curvature expressions must be calculated anew for each different network architecture block.

Another line of work aims to accelerate first-order methods by dynamically adapting the learning rate to match the local optimisation dynamics. Originally this was predominantly done by imposing fixed learning rate schedules (Darken & Moody, 1990; Li & Arora, 2019; Xu et al., 2019; Loshchilov & Hutter, 2017; Smith et al., 2018), but recent developments involve more dynamic adaptations by hypergradients (Franceschi et al., 2017; Micaelli & Storkey, 2020; Donini et al., 2020; Lorraine et al., 2020; Clarke et al., 2022), online Bayesian optimisation (Jin et al., 2023), or explicitly constructing an optimisation framework around the unique characteristics of deep neural networks (Bernstein et al., 2023). Zhang et al. (2019) and Kwatra et al. (2023) adopt a similar quadratic model methodology to our work, but the latter compute a finite-difference approximation to this model rather than using the exact curvature information as we do, and introduces additional hyperparameters controlling an exploration/exploitation trade-off. Niu et al. (2023) uses a parallel approach to ours to incorporate momentum into L-BFGS (Liu & Nocedal, 1989). These methods generally suffer an increased cost over simpler strategies, whether to discover a schedule, compute hypergradients or perform de-facto inline hyperparameter optimisation, in turn requiring a substantial validation dataset to be held aside.

3. AdamQLR

We consider the minimisation of $f(\boldsymbol{\theta})$, representing the loss function of some network parameterised by $\boldsymbol{\theta}$.

3.1. First- and Second-Order Methods

Many optimisation algorithms in ML take the form $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \mathbf{u}(\mathbf{g}_t)$, where α is a learning rate and $\mathbf{u}$ some update function. This $\mathbf{u}$ may depend on an internal state and the current gradient $\mathbf{g}_t$, but not on any higher derivative. As is conventional, we call such algorithms *first-order* optimisers. By contrast, *second-order* optimisers take the form $\boldsymbol{\theta}_t \leftarrow$

$\boldsymbol{\theta}_{t-1} - \mathbf{C}^{-1}\mathbf{u}(\mathbf{g}_t)$, where $\mathbf{C}$ is some curvature matrix (often a damped Hessian, Fisher or Gauss-Newton matrix).

First-order methods broadly provide computational efficiency at the inconvenience of manually selecting α, while second-order methods suffer a large computational cost to dynamically select an implicit α and improved update direction $\mathbf{d}$ using their more powerful objective models. However, a slew of 'adaptive' first-order optimisers (such as Adam (Kingma & Ba, 2015) and relations) blur this distinction by constructing stateful models of the objective, which can often be interpreted as approximating the curvature of $f(\boldsymbol{\theta})$.

Moreover, practical second-order methods for ML are necessarily approximate, as the curvature $\mathbf{C}$ is otherwise intractably large. Further engineering is then required to mitigate the impact of approximate curvature and the inevitable non-convexity of f. For example, in K-FAC, Martens & Grosse (2015) convincingly argue for a particular Kronecker factorisation of a block-diagonal $\mathbf{C}$, but then augment it with a raft of corrections and adaptive heuristics (including multiple periodically-updated damping/factorised Tikhonov regularisation terms, momentum, weight decay, exponential moving averages of curvature statistics and approximate exchange of expectations and Kronecker products). Further, these additions are seemingly *essential* ingredients of a working K-FAC implementation.

A natural question is then whether curvature information or engineering heuristics contribute more to K-FAC's success. In particular, we might ask if accepting first-order methods' inaccurate curvature models and applying second-order stability techniques would blend computational efficiency with optimisation accuracy. We propose to study this possibility by adapting Adam using techniques from K-FAC.

3.2. Adam Revisited

Algorithm 1 restates the Adam optimisation algorithm from Kingma & Ba (2015) applied to f, with some minor notational changes. Our proposed algorithm derives from our anecdotal observation that Adam often makes good choices of update direction, which we notate by $\mathbf{d}_t = \frac{\widehat{\mathbf{m}}_t}{\sqrt{\widehat{\mathbf{v}}_t}+\epsilon}$.

As we detail in Appendix C, Adam is known to carry a diagonal approximation to the empirical Fisher matrix in $\widehat{\mathbf{v}}_t$. Then, the $\frac{1}{\sqrt{\widehat{\mathbf{v}}_t}+\epsilon}$ term in Algorithm 1 effectively performs a curvature transformation on the averaged gradient $\widehat{\mathbf{m}}_t$ before computing a more traditional gradient-based update for $\boldsymbol{\theta}$. There are widely-known limitations to using the empirical Fisher in place of the true Fisher information matrix (Kunstner et al., 2019), and the square root is motivated only by a desire to be "conservative" (Kingma & Ba, 2015). Indeed, Zhang et al. (2018) show Adam is very similar to one construction of natural gradient mean-field variational inference, a technique which is known to prioritise locally fitting modes of the target probability distribution (Turner & Sahani, 2011). The consequent underestimation of global variance corresponds to overestimating local curvature in optimisation, justifying Kingma & Ba (2015)'s preference for a conservative estimate. Nonetheless, this formulation invites us to view Adam through a second-order optimisation lens, and ask whether common second-order optimiser heuristics might bring similar benefits to Adam.

3.3. Adopting Heuristics from K-FAC

After defining its Kronecker-factored block diagonal approximation to the curvature matrix, K-FAC (Martens & Grosse, 2015) includes three important stabilising heuristics: Levenberg-Marquardt damping, and learning rate and momentum selection according to a local second-order model. Since Adam already implements a momentum correction in $\widehat{\mathbf{m}}_t$, we consider only the first two techniques.

Levenberg-Marquardt damping (Levenberg, 1944; Marquardt, 1963; Roweis, 1996) replaces the curvature matrix $\mathbf{C}$ with the damped version $\mathbf{C} + \lambda\mathbf{I}$. Its interpretations include approximating a trust region, enforcing positive definiteness of $\mathbf{C}$, preventing large updates in low-curvature directions and interpolating between gradient descent and full Newton updates. In effect, it imposes a 'minimum curvature' on the objective to avoid near-zero eigenvalues in $\mathbf{C}$.

Let $M(\boldsymbol{\theta})$ be an approximate second-order model around $\boldsymbol{\theta}_{t-1}$, defined by a truncated Taylor series:

$$M(\boldsymbol{\theta}) = f(\boldsymbol{\theta}_{t-1}) + (\boldsymbol{\theta} - \boldsymbol{\theta}_{t-1})^\mathsf{T}\mathbf{g}_t + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_{t-1})^\mathsf{T}(\mathbf{C} + \lambda\mathbf{I})(\boldsymbol{\theta} - \boldsymbol{\theta}_{t-1}). \quad (1)$$

The damping parameter λ is adapted by comparing the change in objective value predicted by the model $(M(\boldsymbol{\theta}_t) - M(\boldsymbol{\theta}_{t-1}))$ to the actual observed change $(f(\boldsymbol{\theta}_t) - f(\boldsymbol{\theta}_{t-1}))$. This adjustment quantifies the model's reliability by a reduction ratio ρ, incorporating stepping factors[1] $\omega_{\text{dec}}, \omega_{\text{inc}}$:

$$\rho = \frac{f(\boldsymbol{\theta}_t) - f(\boldsymbol{\theta}_{t-1})}{M(\boldsymbol{\theta}_t) - M(\boldsymbol{\theta}_{t-1})}; \lambda \leftarrow \begin{cases} \omega_{\text{dec}}\lambda & \text{if } \rho > \frac{3}{4} \\ \lambda & \text{if } \frac{1}{4} \leq \rho \leq \frac{3}{4} \\ \omega_{\text{inc}}\lambda & \text{if } \rho < \frac{1}{4} \end{cases}. \quad (2)$$

We discuss this formulation further in Appendix A.3.

After choosing an update direction $\mathbf{d}_t$, a learning rate α is selected according to the second-order model M. We minimise $M(\boldsymbol{\theta}_{t-1} - \alpha\mathbf{d}_t)$ with respect to α, which yields

$$\alpha = \frac{\mathbf{g}_t^\mathsf{T}\mathbf{d}_t}{\mathbf{d}_t^\mathsf{T}(\mathbf{C} + \lambda\mathbf{I})\mathbf{d}_t}. \quad (3)$$

[1] In the most general form we allow separate decrease and increase factors, but in practice we will often choose $\omega_{\text{dec}} = \frac{1}{\omega_{\text{inc}}}$ for simplicity. We also require $0 < \omega_{\text{dec}} < 1 < \omega_{\text{inc}}$.

Algorithm 1 Adam (Kingma & Ba, 2015)

$\mathbf{m}_0, \mathbf{v}_0 \leftarrow \mathbf{0}$
for $t = 1, 2, \cdots$ until $\boldsymbol{\theta}$ converged **do**
  $\mathbf{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_{t-1})$
  $\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t$
  $\mathbf{v}_t \leftarrow \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)(\mathbf{g}_t \odot \mathbf{g}_t)$
  $\widehat{\mathbf{m}}_t \leftarrow \frac{\mathbf{m}_t}{1-\beta_1^t}$
  $\widehat{\mathbf{v}}_t \leftarrow \frac{\mathbf{v}_t}{1-\beta_2^t}$
  $\mathbf{d}_t \leftarrow \frac{\widehat{\mathbf{m}}_t}{\sqrt{\widehat{\mathbf{v}}_t}+\epsilon}$
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \mathbf{d}_t$
end for

Algorithm 2 AdamQLR

$\mathbf{m}_0, \mathbf{v}_0 \leftarrow \mathbf{0}$
for $t = 1, 2, \cdots$ until $\boldsymbol{\theta}$ converged **do**
  $\mathbf{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_{t-1})$
  $\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t$
  $\mathbf{v}_t \leftarrow \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)(\mathbf{g}_t \odot \mathbf{g}_t)$
  $\widehat{\mathbf{m}}_t \leftarrow \frac{\mathbf{m}_t}{1-\beta_1^t}$
  $\widehat{\mathbf{v}}_t \leftarrow \frac{\mathbf{v}_t}{1-\beta_2^t}$
  $\mathbf{d}_t \leftarrow \frac{\widehat{\mathbf{m}}_t}{\sqrt{\widehat{\mathbf{v}}_t}+\epsilon}$
  Update learning rate α according to (3)
  Update damping λ according to (2)
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \mathbf{d}_t$
end for

A minor rearrangement shows the large matrix $\mathbf{C}$ only appears in products with vectors. The Jacobian-vector product trick (Pearlmutter, 1994), efficient Fisher decompositions (Martens & Grosse, 2015) and similar techniques compute these quantities using only one additional backward pass per product with $\mathbf{C}$. In practice, the information value of these calculations outweighs this cost.

3.4. Extending Adam

Incorporating K-FAC's damping and learning rate selection strategies into Adam yields Algorithm 2, which is easily implementable as a wrapper around vanilla Adam. We name this family of algorithms *AdamQLR*, where *QLR* indicates an optimiser-agnostic quadratic-model learning rate selection logic, which could be broadly applied (e.g. to SGD). Appendix A.4 gives a sketch of a convergence argument for this algorithm.

One remaining consideration is the choice of a curvature matrix $\mathbf{C}$. We use the (true) Fisher matrix throughout, inspired by its connection with Adam's $\widehat{\mathbf{v}}_t$ buffer (see Appendix C.4), its use at the heart of K-FAC and its positive semi-definite guarantee. In short, we tune the damping parameter λ to create a trust region in which our quadratic approximation — specified by the Fisher — is accurate. Then, given the Adam descent direction and the selected λ, we choose the optimal step size within this trust region. We exploit Jacobian-vector products and the efficient Fisher decomposition described in Martens & Grosse (2015, Appendix C), which computes exact products without explicitly storing $\mathbf{C}$.

Finally, our experiments found AdamQLR's training stability to be most threatened by selecting an unreasonably large α for a particular iteration, causing a divergent parameter update. The problem worsens in larger models with more prevalent low-curvature regions of the space to induce very large update sizes. We found that larger batch sizes improved our curvature estimates, leading to better performance despite the higher cost of each forward pass.

Now, the only remaining hyperparameters are β_1, β_2 and ϵ (from Adam) and an initial damping value λ_0. As Adam's hyperparameters are commonly fixed at the default values suggested by Kingma & Ba (2015), and we show λ_0 to be sufficiently insensitive that a default value can be recommended (Section 4.7), we claim that AdamQLR is robust, even without explicit hyperparameter tuning. In particular, we have encapsulated the learning rate α — arguably the most important hyperparameter to select in many optimisation algorithms. We justify this claim in Section 4.

Compared to Adam, we suffer additional forward and backward passes to compute $M(\boldsymbol{\theta}_t)$ and $(\mathbf{C}+\lambda\mathbf{I})\mathbf{d}_t$. These turn out not to impede empirical performance, though we note a careful implementation would amortise the former cost. Our only significant additional memory cost is storing the vector $(\mathbf{C}+\lambda\mathbf{I})\mathbf{d}_t$, making our approximate memory footprint four times that of SGD (where Adam's is three times SGD).

4. Experiments

We examine the training and test performance of AdamQLR against Adam and K-FAC in a variety of settings:

Rosenbrock (1960) Function Let $a = 1$ and $b = 100$, then $f(x, y) = (a - x)^2 + b(y - x^2)^2$

UCI Energy (Tsanas & Xifara, 2012) on an MLP with one hidden layer of 50 units

UCI Protein (Rana, 2013) on an MLP with one hidden layer of 100 units

Fashion-MNIST (Xiao et al., 2017) on an MLP with one hidden layer of 50 units

SVHN (Netzer et al., 2011) on a ResNet-18 (He et al., 2016)

CIFAR-10 (Krizhevsky, 2009) on a ResNet-18 (He et al., 2016)

We also perform a study on Penn Treebank in Appendix B.1.2. On UCI datasets we generate random splits using the same sizes as Gal & Ghahramani (2016) and use MSE loss; otherwise, we separate the standard test set, randomly choose 1/6 (Fashion-MNIST and SVHN) or 1/10 (CIFAR-10) of the remaining data to form a validation set, and use cross-entropy loss. Code for all our experiments is available at `https://github.com/rmclarke/AdamThroughASecondOrderLens`. We compare (see Appendix A.5 for further notes):

SGD Minimal Classical mini-batched stochastic gradient descent, with tuned learning rate

SGD Full *SGD Minimal* with additional tuned momentum and weight decay

Adam (Kingma & Ba, 2015) with tuned learning rate and fixed defaults for other hyperparameters

Adam (Untuned) *Adam* (Kingma & Ba, 2015), fixing batch sizes[2] and learning rate of 0.001

K-FAC (Martens & Grosse, 2015; Botev & Martens, 2022) with tuned initial damping

K-FAC (Untuned) *K-FAC* (Martens & Grosse, 2015; Botev & Martens, 2022) with initial damping set to a default 1.0 and fixed batch size 3200

AdamQLR (Tuned) Algorithm 2, using Fisher curvature for $\mathbf{C}$. We tune initial damping and damping adjustment factors $\omega_{\text{dec}}, \omega_{\text{inc}}$

AdamQLR (Untuned) *AdamQLR* with fixed batch size 3 200, initial damping 0.001 and $\omega_{\text{dec}} = \frac{1}{\omega_{\text{inc}}} = 0.5$ (justified by Section 4.7 and Appendix B.2)

Except for the Rosenbrock Function and *(Untuned)* variants, we also tune a batch size over $\{50, 100, 200, 400, 800, 1\,600, 3\,200\}$. All hyperparameter tuning uses ASHA (Li et al., 2020) over 200 random initialisations, targeting a fixed number of training epochs, subject to a maximum runtime of 15 minutes (only reached for CIFAR-10; see Appendix B.1.4 for experiments using runtime as the primary constraint). We then perform 50 runs using each of the best hyperparameters found (measured by final validation loss), then plot the mean and standard deviation of the median trends of each of 50 bootstrap samples of the results. Following Botev & Martens (2022), any damping is clipped to $\lambda \geq 10^{-8}$. Except for the Rosenbrock Function, we give a numerical comparison of the end-of-training statistics in Table 5.

In Appendix B.1.4, we present analogous results where the hyperparameters are tuned to minimise training or validation losses after a fixed runtime, with no epoch constraint.

[2]We use a 'typical' batch size for each setting: full-batch for UCI Energy, 100 for UCI Protein, 50 for Fashion-MNIST, 256 for SVHN and 128 for CIFAR-10.

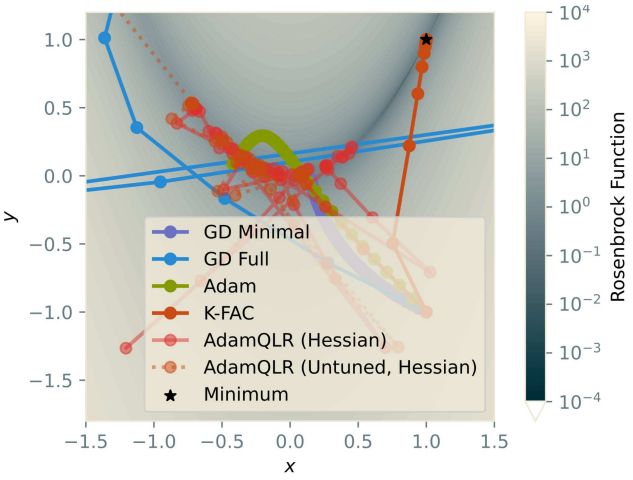


Figure 1: Optimisation trajectories over 200 steps from a fixed initial point on the Rosenbrock Function. Hyperparameter tuning used 200 standard-normal random initial points.

Appendix B.3 details additional ablation studies on Adam, AdamQLR and K-FAC.

4.1. Rosenbrock Function

The Rosenbrock Function (Rosenbrock, 1960) provides a visualisable test bed for optimisation algorithms, containing non-linear correlations between its inputs and anisotropic curvature. We consider 200 optimisation steps, using $\mathcal{N}(\mathbf{0}, \mathbf{I})$-sampled initial (x, y) values during hyperparameter tuning, and evaluate trajectories from the fixed starting point $(1, -1)$ in Figure 1. As there is no probabilistic model, we apply K-FAC to a least-squares formulation (Brunet, 2010, page 38), and we use Hessian curvature in *AdamQLR* for this experiment only. Additionally, we use gradient descent (*GD*) in place of *SGD*. Since there is no separate validation set, we tune hyperparameters on the objective function directly.

Here, *GD Minimal* makes good initial progress into the central 'valley', but its learning rate is too small to continue along the valley floor — without regularisation, GD must select conservative step sizes to avoid diverging from poor initialisations. *GD Full*'s hyperparameters cause it to bounce unstably around the optimisation space — the algorithm is unable to recover from their overly aggressive settings. *Adam*'s adaptive buffers allow it to target the valley more directly, eventually making slow progress along the valley floor, but it takes time to learn the new dynamics in the latter regime, initially 'overshooting' the valley. *K-FAC* demonstrates an impressive understanding of the optimisation space, making direct, rapid progress towards the optimum in spite of the challenging Euclidean curvature.

AdamQLR (Tuned) demonstrates a combination of all these phenomena. It begins by updating in the same direction as

Adam, but with more aggressive step sizes, permitting faster initial progress. This confidence subsequently causes significant steps away from the central valley, but K-FAC's adaptive heuristics allow the algorithm to recover from these errors and return to the valley. Compared to *Adam*, *AdamQLR (Tuned)* performs much more exploration of the central valley, indicating the latter exploits its ability to recover from poor steps by accepting more aggressive updates. *AdamQLR (Untuned)* follows a similar approach, though again with ill-suited hyperparameters for this toy problem. These initial results suggest K-FAC's heuristics do carry potential in their own right for first-order approaches.

4.2. UCI Energy

UCI Energy provides a low-dimensional regression task on a small dataset, which is amenable to hosting long experiments to explore convergence behaviour. We consider 4 000 epochs of training and plot bootstrap-sampled median training and test loss trends in Figure 2a.

Our principal benchmarks fall much as we would expect: *SGD Minimal* makes respectable, if sluggish, progress during optimisation, but is outclassed by the more rapid initial convergence of *SGD Full* and *Adam*. Both these latter methods achieve strong test performance, with *SGD Full* achieving the best final test loss of all methods. Despite its rapid initial progress, *K-FAC* quickly begins overfitting, reaching a final test loss similar to the *AdamQLR* methods.

Generally, *AdamQLR (Tuned)* performs comparably with the tuned *Adam* baseline. The QLR computed learning rates accelerate initial progress, while the addition of damping provides some defence against overfitting, at the cost of a higher final training loss. However, on this trial, *AdamQLR (Untuned)* diverged rapidly beyond the limits of Figure 2a. Since this phenomenon is not apparent with our other trials, and *K-FAC* performs fine, we suppose the latter's curvature matrix smoothing must be important when using second-order information in this problem. Under full hyperparameter tuning, *AdamQLR (Tuned)* does not convincingly improve over *Adam* — potentially the small-scale, full-batch nature of this setting responds poorly to this algorithm.

4.3. UCI Protein

UCI Protein is another low-dimensional regression task, but with far more data points, allowing for a computationally-efficient study of a larger dataset. We show 200 epochs of training in Figure 2b.

Here we see distinct generalisation trends for each algorithm. *SGD Full* improves slightly over *SGD Minimal*, but still lags behind the other methods. *K-FAC* is now clearly the best-performing algorithm, as might perhaps be expected since it computes the most granular curvature approximation when choosing an update direction. However, we see meaningful gains from *AdamQLR*, with the *(Tuned)* variant comfortably outperforming *Adam*. We observe *AdamQLR*'s automatic learning rate selection is capable of outperforming methods which require a sensitive explicit choice of learning rate — the *Untuned* variant is clearly superior to tuned *SGD* on this task and is only slightly worse than tuned *Adam*. The clear ranking here indicates meaningful value is added to *Adam* by including some heuristics from *K-FAC*, but that *K-FAC* offers further advantage even beyond this.

4.4. Fashion-MNIST

Fashion-MNIST provides a first foray into higher-dimensional data, but at a scale still approachable by MLP models. Using a 10-epoch training window, we plot bootstrapped accuracy evolutions in Figure 2c and loss evolutions in Figure 5a.

While *K-FAC* achieves the best final training loss, its loss evolution plots reveal a significant tendency to overfit. While this is a recognised issue with K-FAC (Martens et al., 2018), and the fundamental idea of minimising a test loss by optimising a training loss frustrates the application of naïvely-powerful optimisers, the impact is to question *K-FAC*'s desirability in this application. *SGD Full*, *Adam* and *AdamQLR* all perform very similarly, showing less overfitting but being essentially indistinguishable. *Adam (Untuned)*'s underperformance of *AdamQLR (Untuned)* thus demonstrates that the additional heuristics of the latter improve robustness more than outright performance. Surprisingly, adding further complexity in the form of *K-FAC* seems to *worsen* the overfitting problem.

4.5. SVHN

With SVHN, we progress to a full-colour image dataset and a substantially larger ResNet-18 model, which we tune for 10 epochs and present in Figures 2d (accuracies) and 5b (losses). The periodicity in these evolutions corresponds to individual epochs, and is simply a training artefact.

On this problem, both *AdamQLR* variants achieve accuracy checkpoints slightly faster than *Adam*. *SGD Minimal* again forms a mediocre baseline, but *SGD Full* performs admirably in this setting, only slightly trailing the other algorithms' initial loss convergence. Every method overfits losses in this setting. *K-FAC* generalises less well than its impressive training performance might lead us to hope.

AdamQLR's tuned and untuned generalisation performances are rivalled only by *SGD Full* and *Adam (Untuned)*, the latter over a significantly longer timescale. Interestingly, this selection of heuristics from *K-FAC* again seems to bring benefits which are nullified by the full *K-FAC* setting, raising further questions about the latter's underlying dynamics.

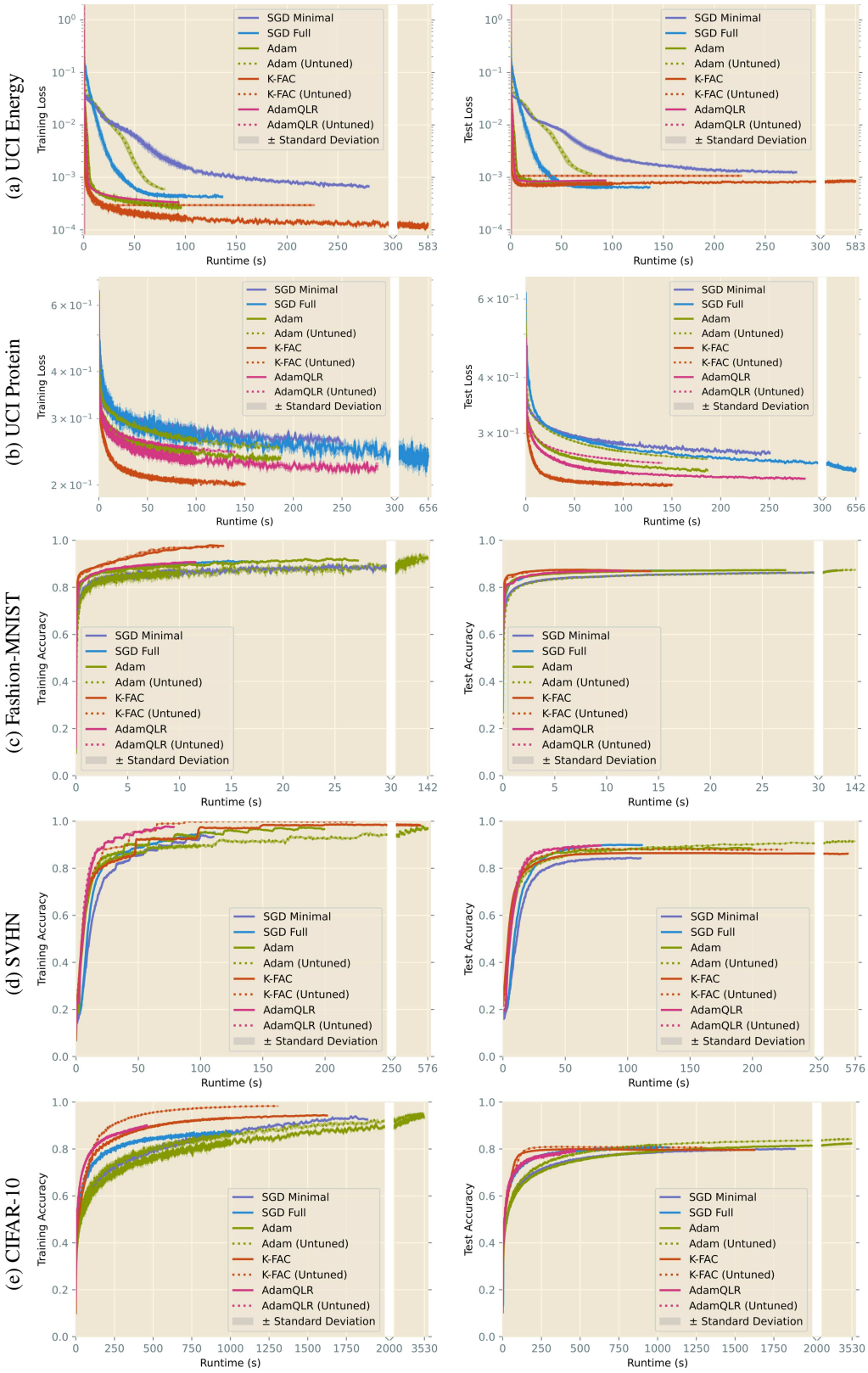


Figure 2: Median training (left) and test (right) performance trajectories, bootstrap-sampled over 50 repetitions per algorithm. Hyperparameters chosen by ASHA over 200 initialisations. Note changes of scale on the time axes. See also results on loss metrics and learning rate evolutions in Figures 5 and 4, and numerical comparison in Table 5.

4.6. CIFAR-10

Finally, in a simulation of larger-scale learning, we train a ResNet-18 on CIFAR-10 over 72 epochs. Here we include conventional data augmentation of 4-pixel padding, random cropping and random left-right flipping, displaying our accuracy results in Figure 2e and loss results in Figure 5c.

Adam is now slower to converge in both training and test accuracy, suggesting this could be an ill-suited setting in which Adam can be expected to underperform (Balles & Hennig, 2018). However, it achieves the highest test accuracy of any method by the end of training, with *Adam (Untuned)* curiously outperforming the tuned variant. This suggests our hyperparameter selection strategy may have been particularly noisy in this setting, in which case much of the difference between algorithms here is negligible. In this trial, *AdamQLR* seems to close most of the performance gap between *Adam* and *K-FAC*, suggesting this problem benefits more from adaptive heuristics than from second-order curvature information.

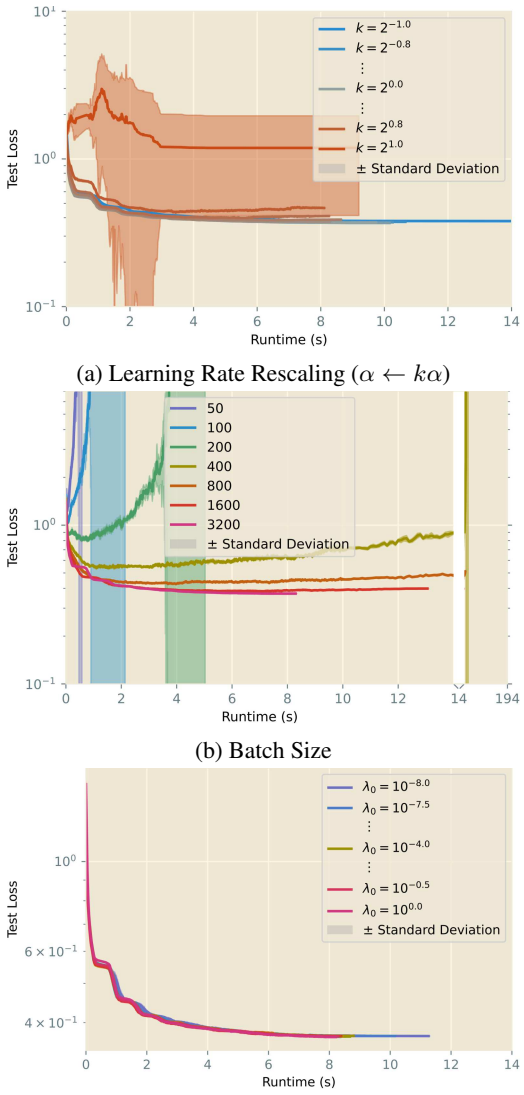
4.7. Sensitivity Studies

In Appendix B.2 we analyse the sensitivity of *AdamQLR* on Fashion-MNIST by repeating the experiments of Section 4.4 with a range of batch sizes, initial damping values and damping adjustment factors, and by replacing the approximately-optimal learning rate α from (3) with the rescaled $k\alpha$, for various $k \in [0.5, 2.0]$. Figure 3 summarises our bootstrapped results for each intervention.

Our results inspire further confidence in AdamQLR as a correctly-configured adaptive method. Generalisation performance is optimised by choosing $k \approx 1$: constant rescaling of our proposed learning rates does not reduce test error, suggesting we adapt well to the local space and select performant update magnitudes for each direction $\mathbf{d}_t$ proposed by Adam. By contrast, AdamQLR is insensitive to the choice of initial damping λ_0 on this dataset, so while our ablation studies in Section B.3.1 indicate damping is an important stabilising feature of our method, it appears the adaptive strategy of (2) selects an appropriate damping magnitude regardless of its starting point. Finally, larger batch sizes increase generalisation performance. Since we depend implicitly on highly-parameterised curvature matrices, larger batch sizes would be expected to give a more performant average, but this also substantially decreases training time, owing to efficient GPU computation. All these results justify our *AdamQLR (Untuned)* hyperparameter choices.

4.8. Learning Rate Evolution

In Figure 4, we plot the trajectories of average learning rates selected by AdamQLR and K-FAC against the fixed values used in SGD and Adam.



(a) Learning Rate Rescaling ($\alpha \leftarrow k\alpha$)

(b) Batch Size

(c) Initial Damping λ_0

Figure 3: Sensitivity studies for *AdamQLR* on Fashion-MNIST over (a) learning rate rescaling, (b) batch size and (c) initial damping, showing test losses.

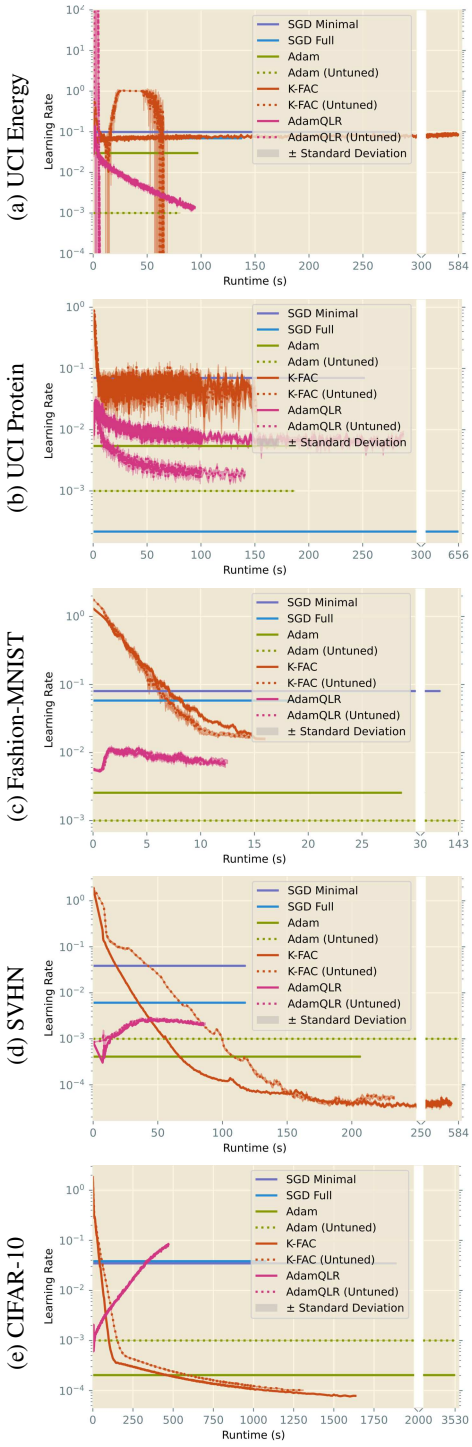


Figure 4: Median learning rate trajectories, bootstrap-sampled over 50 repetitions per algorithm. Hyperparameters chosen by ASHA over 200 initialisations. Note changes of scale on the time axes. See also our numerical presentation in Table 5.

Learning rate schedules are widely known to be important in certain training problems, particularly at larger scales, so it is unsurprising that various algorithms' sense of the 'optimal' learning rate varies over time. For the most part, the chosen schedules give an approximately exponential decay in learning rate, interestingly sometimes excluding the warm-up behaviour commonly specified in manually-designed schedules.

Broadly speaking, AdamQLR and K-FAC seem to adopt more aggressive learning rates than SGD or Adam. It is interesting to note that *AdamQLR (Untuned)* chooses *growing* learning rates on SVHN and CIFAR-10 which differ dramatically from those of other methods, yet achieves similar results in loss and accuracy space. In summation, these results suggest we might do well to explore other approaches to improving machine learning optimisers, beyond focussing on learning rates.

5. Conclusion

In this paper we consider some of the heuristics of K-FAC by studying AdamQLR, an extension to Adam which borrows learning rate selection and adaptive damping strategies from second-order methods. Empirically, the effect of these heuristics seems to vary, being null or negative in some trials while outperforming both Adam and unablated K-FAC in others. This indicates the approach of K-FAC might be less generally applicable than that of common first-order optimisers, perhaps explaining its rare use in practice, and we think there could be great value in future work studying this phenomenon. Of tangential interest, we find an untuned version of AdamQLR, motivated by our sensitivity results, compares broadly favourably with tuned implementations of popular algorithms, while only rarely causing significant detriment to tuned-SGD or Adam baselines. Curiously, this occurs despite its use of large batch sizes conventionally understood to worsen generalisation performance.

We note challenging training-test dynamics from the CIFAR-10 results which merit further investigation, though we leave this to future work. Ultimately, we would like to better understand the workings of second-order methods like K-FAC, such that we can unify the benefits of first- and second-order optimisation to better serve the needs of the ML community, since these significantly differ from those of other optimisation practitioners. In future work, we hope to advance this line of research and better address this fundamental component of ML systems.

Acknowledgements

We thank Baiyu Su for his valuable input to an earlier version of this work, which informed the rewrite presented in this paper.

Ross Clarke acknowledges funding from the Engineering and Physical Sciences Research Council (project reference 2107369, grant EP/S515334/1), and support from Google DeepMind.

José Miguel Hernández-Lobato acknowledges support from a Turing AI Fellowship under grant EP/V023756/1.

Impact Statement

Our work studies a general optimisation algorithm for neural networks, so is unlikely to influence a particular societal problem. However, increasing the effectiveness of optimisation methods makes it easier for both benevolent and malevolent actors to develop systems aligned with their goals, so this class of risk is unavoidable. Of additional concern is that the typical setting of seeking to optimise a test metric by minimising a training metric fundamentally misaligns our algorithms with our objectives, and that misalignment may cause unexpected downstream consequences if poorly understood by the model developer. Finally, it would be naïve to presume any one optimisation algorithm is a panacea for all settings, and any errant belief in this vein may cause promising research directions to be incorrectly dismissed if a supposedly 'universal' optimiser happens to perform poorly on it.

References

Amari, S.-i. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2):251–276, February 1998.

Anil, R., Gupta, V., Koren, T., Regan, K., and Singer, Y. Scalable Second Order Optimization for Deep Learning, March 2021. URL http://arxiv.org/abs/2002.09018.

Babuschkin, I., Baumli, K., Bell, A., Bhupatiraju, S., Bruce, J., Buchlovsky, P., Budden, D., Cai, T., Clark, A., Danihelka, I., Dedieu, A., Fantacci, C., Godwin, J., Jones, C., Hemsley, R., Hennigan, T., Hessel, M., Hou, S., Kapturowski, S., Keck, T., Kemaev, I., King, M., Kunesch, M., Martens, L., Merzic, H., Mikulik, V., Norman, T., Papamakarios, G., Quan, J., Ring, R., Ruiz, F., Sanchez, A., Schneider, R., Sezener, E., Spencer, S., Srinivasan, S., Stokowiec, W., Wang, L., Zhou, G., and Viola, F. The DeepMind JAX Ecosystem, 2020. URL http://github.com/deepmind.

Bahamou, A., Goldfarb, D., and Ren, Y. A Mini-Block Fisher Method for Deep Neural Networks. In *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, pp. 9191–9220. PMLR, April 2023.

Balles, L. and Hennig, P. Dissecting Adam: The Sign, Magnitude and Variance of Stochastic Gradients. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 404–413. PMLR, July 2018.

Baydin, A. G., Cornish, R., Martínez-Rubio, D., Schmidt, M., and Wood, F. Online Learning Rate Adaptation with Hypergradient Descent. In *6th International Conference on Learning Representations, ICLR 2018*, Vancouver, BC, Canada, April 2018. OpenReview.net.

Bernstein, J., Mingard, C., Huang, K., Azizan, N., and Yue, Y. Automatic Gradient Descent: Deep Learning without Hyperparameters, April 2023. URL http://arxiv.org/abs/2304.05187.

Botev, A. and Martens, J. KFAC-JAX, 2022. URL http://github.com/deepmind/kfac-jax.

Botev, A., Ritter, H., and Barber, D. Practical Gauss-Newton Optimisation for Deep Learning. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 557–565. PMLR, July 2017.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Broyden, C. G. The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, March 1970.

Brunet, F. *Contributions to Parametric Image Registration and 3D Surface Reconstruction*. PhD Thesis, Université d'Auvergne, November 2010. URL https://www.brnt.eu/publications/brunet2010phd.pdf.

Clarke, R. M., Oldewage, E. T., and Hernández-Lobato, J. M. Scalable One-Pass Optimisation of High-Dimensional Weight-Update Hyperparameters by Implicit Differentiation. In *The Tenth International Conference on Learning Representations, ICLR 2022*, Virtual Event, April 2022.

Darken, C. and Moody, J. Note on Learning Rate Schedules for Stochastic Optimization. In *Advances in Neural Information Processing Systems*, volume 3. Morgan-Kaufmann, 1990.

Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

Donini, M., Franceschi, L., Frasconi, P., Pontil, M., and Majumder, O. MARTHE: Scheduling the Learning Rate Via Online Hypergradients. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, volume 3, pp. 2119–2125, July 2020.

Duchi, J., Hazan, E., and Singer, Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(61): 2121–2159, 2011.

Fletcher, R. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, January 1970.

Franceschi, L., Donini, M., Frasconi, P., and Pontil, M. Forward and Reverse Gradient-Based Hyperparameter Optimization. In *International Conference on Machine Learning*, pp. 1165–1173, July 2017.

Gal, Y. and Ghahramani, Z. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *International Conference on Machine Learning*, pp. 1050–1059, June 2016.

George, T., Laurent, C., Bouthillier, X., Ballas, N., and Vincent, P. Fast Approximate Natural Gradient Descent in a Kronecker Factored Eigenbasis. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Goldfarb, D. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24 (109):23–26, 1970.

Goldfarb, D., Ren, Y., and Bahamou, A. Practical Quasi-Newton Methods for Training Deep Neural Networks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 2386–2396. Curran Associates, Inc., 2020.

Grosse, R. and Salakhudinov, R. Scaling up Natural Gradient by Sparsely Factorizing the Inverse Fisher Matrix. In *Proceedings of the 32nd International Conference on Machine Learning*, pp. 2304–2313. PMLR, June 2015.

Gupta, V., Koren, T., and Singer, Y. Shampoo: Preconditioned Stochastic Tensor Optimization. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 1842–1850. PMLR, July 2018.

He, K., Zhang, X., Ren, S., and Sun, J. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, June 2016.

Hennigan, T., Cai, T., Norman, T., and Babuschkin, I. Haiku: Sonnet for JAX, 2020. URL `http://github.com/deepmind/dm-haiku`.

Heskes, T. On "Natural" Learning and Pruning in Multilayered Perceptrons. *Neural Computation*, 12(4):881–901, April 2000. Conference Name: Neural Computation.

Jin, Y., Zhou, T., Zhao, L., Zhu, Y., Guo, C., Canini, M., and Krishnamurthy, A. AutoLRS: Automatic Learning-Rate Schedule by Bayesian Optimization on the Fly. In *International Conference on Learning Representations*, January 2023.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

Krizhevsky, A. Learning Multiple Layers of Features from Tiny Images. Master's thesis, University of Toronto, April 2009. URL `https://cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

Kunstner, F., Hennig, P., and Balles, L. Limitations of the empirical Fisher approximation for natural gradient descent. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Kwatra, N., Thejas, V., Iyer, N., Ramjee, R., and Sivathanu, M. AutoLR: A Method for Automatic Tuning of Learning Rate. *Submission to ICLR 2020*, May 2023.

Levenberg, K. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, 2(2):164–168, 1944.

Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Bentzur, J., Hardt, M., Recht, B., and Talwalkar, A. A System for Massively Parallel Hyperparameter Tuning. *Proceedings of Machine Learning and Systems*, 2:230–246, March 2020.

Li, Z. and Arora, S. An Exponential Learning Rate Schedule for Deep Learning. In *8th International Conference on Learning Representations, {ICLR} 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, December 2019.

Liu, D. C. and Nocedal, J. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, August 1989.

Lorraine, J., Vicol, P., and Duvenaud, D. Optimizing Millions of Hyperparameters by Implicit Differentiation. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, pp. 1540–1552. PMLR, June 2020.

Loshchilov, I. and Hutter, F. SGDR: Stochastic Gradient Descent with Warm Restarts. In *5th International Conference on Learning Representations, {ICLR} 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

Marcus, M. P., Santorini, B., Marcinkiewicz, M. A., and Taylor, A. Treebank-3, 1999. URL http://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz.

Marcus, Mitchell P., Santorini, Beatrice, Mary Ann Marcinkiewicz, and Taylor, Ann. Treebank-3, 1999. URL https://catalog.ldc.upenn.edu/LDC99T42.

Marquardt, D. W. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441, June 1963. Publisher: Society for Industrial and Applied Mathematics.

Martens, J. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pp. 735–742, Madison, WI, USA, June 2010. Omnipress.

Martens, J. and Grosse, R. Optimizing Neural Networks with Kronecker-factored Approximate Curvature. In *International Conference on Machine Learning*, pp. 2408–2417, June 2015.

Martens, J., Ba, J., and Johnson, M. Kronecker-factored Curvature Approximations for Recurrent Neural Networks. In *6th International Conference on Learning Representations, ICLR 2018*, Vancouver, BC, Canada, April 2018.

McMahan, H. B. and Streeter, M. Adaptive Bound Optimization for Online Convex Optimization, July 2010. URL http://arxiv.org/abs/1002.4908.

Micaelli, P. and Storkey, A. Non-greedy Gradient-based Hyperparameter Optimization Over Long Horizons, July 2020. URL http://arxiv.org/abs/2007.07869.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading Digits in Natural Images with Unsupervised Feature Learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

Niu, Y., Fabian, Z., Lee, S., Soltanolkotabi, M., and Avestimehr, S. mL-BFGS: A Momentum-based L-BFGS for Distributed Large-Scale Neural Network Optimization. *Transactions on Machine Learning Research*, July 2023.

Nocedal, J. and Wright, S. J. *Numerical Optimization*. Springer, 2006.

Osawa, K., Tsuji, Y., Ueno, Y., Naruse, A., Yokota, R., and Matsuoka, S. Large-Scale Distributed Second-Order Optimization Using Kronecker-Factored Approximate Curvature for Deep Convolutional Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12359–12367, 2019.

Pascanu, R. and Bengio, Y. Revisiting Natural Gradient for Deep Networks. In Bengio, Y. and LeCun, Y. (eds.), *2nd International Conference on Learning Representations, ICLR 2014*, Banff, AB, Canada, April 2014.

Pearlmutter, B. A. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1):147–160, January 1994.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language Models are Unsupervised Multitask Learners. 2019.

Rana, P. S. UCI Machine Learning Repository: Physicochemical Properties of Protein Tertiary Structure Data Set, March 2013. URL https://archive.ics.uci.edu/ml/datasets/Physicochemical+Properties+of+Protein+Tertiary+Structure.

Ren, Y. and Goldfarb, D. Tensor Normal Training for Deep Learning Models. In *Advances in Neural Information Processing Systems*, volume 34, pp. 26040–26052. Curran Associates, Inc., 2021.

Rosenbrock, H. H. An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*, 3(3):175–184, January 1960.

Roweis, S. Levenberg-Marquardt Optimization. Technical Report, New York University, 1996. URL https://cs.nyu.edu/~roweis/notes/lm.pdf.

Shanno, D. F. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, 24 (111):647–656, 1970.

Shi, N., Li, D., Hong, M., and Sun, R. RMSprop converges with proper hyper-parameter. October 2020.

Smith, S. L., Kindermans, P.-J., Ying, C., and Le, Q. V. Don't Decay the Learning Rate, Increase the Batch Size. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

Tieleman, T. and Hinton, G. E. Neural Networks for Machine Learning: Lecture 6. *Coursera*, 2012.

Tsanas, A. and Xifara, A. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, 49: 560–567, June 2012.

Turner, R. E. and Sahani, M. Two problems with variational expectation maximisation for time-series models. In Barber, D., Cemgil, T., and Chiappa, S. (eds.), *Bayesian time series models*, pp. 109–130. Cambridge University Press, 2011. Section: 5.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms, September 2017. URL http://arxiv.org/abs/1708.07747.

Xu, Z., Dai, A. M., Kemp, J., and Metz, L. Learning an Adaptive Learning Rate Schedule. *arXiv:1909.09712 [cs, stat]*, September 2019. arXiv: 1909.09712.

Yao, Z., Gholami, A., Shen, S., Mustafa, M., Keutzer, K., and Mahoney, M. ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12): 10665–10673, May 2021. Number: 12.

Zhang, G., Sun, S., Duvenaud, D., and Grosse, R. Noisy Natural Gradient as Variational Inference. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 5852–5861. PMLR, July 2018.

Zhang, G., Li, L., Nado, Z., Martens, J., Sachdeva, S., Dahl, G., Shallue, C., and Grosse, R. B. Which Algorithmic Choices Matter at Which Batch Sizes? Insights From a Noisy Quadratic Model. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

A. Notes

A.1. Reproducibility Statement

We describe our algorithm fully in Section 3, provide full source code to the reviewers and will publish this code to the community after deanonymisation. The descriptions in this paper describe all the modifications we make to Adam and provide a complete intuitive summary of our contribution, while the source code allows any fine detail of our implementation or experiments to be inspected.

A.2. Limitations

While we have evaluated our algorithm on a range of datasets and models, we have necessarily left many important settings untested. Thus, even though we hope our evaluation approach generalises well to other settings, we should recognise that it has not yet been tested in those settings. In particular, the learning rate selection strategy used by K-FAC and AdamQLR assumes the optimisation space is approximately convex and quadratic, which will not generally be true of machine learning problems — this motivates our use of damping to defend against particularly ill-posed updates. With sufficient damping, we effectively define a 'trust region' beyond which the surface can be non-quadratic without harming our method. Further, since Adam is known not to perform well in certain (poorly-understood) circumstances (Balles & Hennig, 2018), we might expect AdamQLR to have difficulty with the same class of problems.

A.3. Reduction Ratio

Here, we give a more verbose commentary on the damping adjustment mechanism described in (2).

The definition of the reduction ratio ρ is intuitive. When we update the model parameters from $\boldsymbol{\theta}_{t-1}$ to $\boldsymbol{\theta}_t$, we will observe some change in the loss metric $f(\boldsymbol{\theta}_t) - f(\boldsymbol{\theta}_{t-1})$. Similarly, our quadratic model M will have proposed this parameter update predicting the loss will change by $M(\boldsymbol{\theta}_t) - M(\boldsymbol{\theta}_{t-1})$. Ideally, we would like our model to be a good fit for the true optimisation surface, in which case the observed and predicted changes will be similar, and we will find $\rho \approx 1$. Conversely, if the fit is poor, the observation and prediction will be very different, giving $\rho < 1$ if the observed change is much smaller than the model predicted, or $\rho > 1$ if the change is much larger than the model predicted.

If we find the fit of M to be poor, we would like to adjust the damping to help rectify the situation, since a larger damping will generally bias the model towards expecting larger loss changes, thus proposing smaller parameter updates. Broadly speaking, $\rho > 1$ suggests the model is being too conservative, and we would benefit from decreasing damping to better reflect the underlying surface. Conversely, $\rho < 1$ suggests the model expects much more dramatic changes than we actually see, so we should increase damping to 'reign in' the predictive behaviour.

As Martens & Grosse (2015) note in the original presentation of K-FAC, the optimisation dynamics will change during training. In particular, as we approach a local minimum, the loss surface becomes more and more quadratic-like. Under these circumstances, damping slows down convergence by reducing parameter update sizes, without achieving any appreciable benefit. Even away from local minima, damping tends to trade convergence speed for stability. In both cases, there is a natural incentive to be biased towards decreasing damping if at all possible.

In this work, we retain Martens & Grosse (2015)'s damping adjustment thresholds of $\rho > 3/4$ and $\rho < 1/4$, since these choices led to desirable performance from K-FAC. Martens & Grosse articulate their preference for reducing λ if possible, and we can understand their chosen thresholds in that light. It is for this reason that the thresholds are not centred about $\rho = 1$, as might have been our intuitive expectation.

A.4. Convergence of AdamQLR

While we have not studied the convergence of AdamQLR analytically, we believe it will converge under appropriate conditions by the following intuitive argument.

Let the update direction $\mathbf{d}_t$ proposed by Adam be arbitrary. By construction, AdamQLR selects the (possibly negative) learning rate α which minimises the value of some quadratic model $M(\boldsymbol{\theta})$ at the new parameters $\boldsymbol{\theta}_{t-1} - \alpha\mathbf{d}_t$, so we certainly have $M(\boldsymbol{\theta}_{t-1} - \alpha\mathbf{d}_t) \leq M(\boldsymbol{\theta}_{t-1})$. Whether this translates to non-increase of the objective function $f(\boldsymbol{\theta})$ depends on the quality of the approximation $M(\boldsymbol{\theta})$. For any given α, some Lipschitz smoothness condition exists which guarantees $f(\boldsymbol{\theta}_{t-1} - \alpha\mathbf{d}_t) \leq f(\boldsymbol{\theta}_{t-1})$, so we should be able to guarantee convergence for a sufficiently smooth $f(\boldsymbol{\theta})$ if α is clipped to some maximum value. This intuition transfers to the unclipped case if we argue that, in practice, α will generally take values within some finite range.

Shi et al. (2020) prove the convergence of full-batch RMSprop for a diminishing learning rate schedule. While the more arbitrary adaptivity of AdamQLR's learning rate prevents a direct application of this proof, it does lend confidence that AdamQLR should behave as expected in common practical circumstances.

A.5. Choice of Baselines

We now give some explanatory notes justifying our choice of baseline algorithms in Section 4.

Weight decay was only included in *SGD Full*, despite being

theoretically applicable to any algorithm. To clarify the comparison, we studied vanilla versions of SGD, Adam and K-FAC as the most natural baselines for AdamQLR. Since SGD is very commonly used with momentum and weight decay, we include it in *Minimal* and *Full* forms, where the latter includes these additional components. *SGD Full* is included primarily for background context, so its unique use of weight decay does not affect the comparability of the other algorithms. Moreover, since SGD typically underperformed other algorithms in our experiments, any advantage due to weight decay does not affect our results.

Similarly, the use of momentum in *SGD Full* is primarily to provide background context from a common algorithm. *Adam* contains momentum-like behaviour by construction, with the corresponding hyperparameters not usually tuned in practice (a convention we follow), and *K-FAC* uses an adaptive momentum coefficient by default. *AdamQLR* does not incorporate the adaptive momentum of *K-FAC*, so momentum is only used by the inner Adam procedure to select the unscaled update direction, with the magnitude of each update computed independently at each iteration. Thus, any effect from the absence of momentum in AdamQLR is disadvantageous for this algorithm, so our conclusions are not invalidated.

A.6. Hyperparameter Search Space

We use similar hyperparameter search spaces (with unused hyperparameters removed) for each dataset and algorithm combination. These are detailed in Table 1.

A.7. Chosen Hyperparameters

The best hyperparameters selected by ASHA for each setting considered in this work are indicated in Table 2.

A.8. Compute Used

Our experiments were performed on one of the two sets of hardware shown in Table 3. All runtime comparisons were performed on like-for-like hardware. We make use of GPU acceleration throughout, with the JAX (Bradbury et al., 2018), Haiku (Hennigan et al., 2020) and KFAC-JAX (Botev & Martens, 2022) libraries, along with various related components of the DeepMind JAX Ecosystem (Babuschkin et al., 2020).

A.9. Datasets

The datasets we use are all standard in the ML literature; we outline their usage conditions in Table 4.

B. Additional Experiments

B.1. Algorithm Comparisons

In this Section, we provide some additional viewpoints into our main results of Section 4.

B.1.1. Fashion-MNIST, SVHN and CIFAR-10 Losses

In Figure 2, we plotted experimental results in terms of the loss metric used during training. For Fashion-MNIST, SVHN and CIFAR-10, we also present classification accuracy in metrics in Figure 5 and Table 5. These illustrate broadly the same patterns as we discussed in the main body of the paper.

B.1.2. Penn Treebank

As an additional baseline, we consider training the standard Penn Treebank subset (Marcus, Mitchell P. et al., 1999; Marcus et al., 1999) on the GPT-2 model (Radford et al., 2019), as implemented by Hugging Face. We interpret the batch size as the number of token subsequences considered in parallel, chosen over $\{5, 10, 20, 35, 50, 100, 200\}$, and also choose the length of subsequences considered from $\{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$, and we set the HPO runtime limit to 1 hour. Otherwise, our hyperparameter optimisation is identical to that of Section 4. For time efficiency, we perform 10 repetitions of training with the best hyperparameters found, rather than 50 as in Section 4, with each repetition comprising 100 epochs of training, and show our results in Figure 6 and Table 5. Our *AdamQLR (Untuned)* setting uses a batch size of 30 and subsequence length of 70, chosen based on the largest values which fit on our GPUs.

Interestingly, our results reflect the observation that transformer training dynamics are quite different from other NN model classes. The addition of momentum and weight decay to *SGD Full* seems to hinder it in comparison to *SGD Minimal*, with the latter exhibiting superior training and generalisation performance. Both are ultimately beaten by *Adam* on training performance, but the latter shows a greater tendency to overfit, with a gradually increasing test loss after around 1000 s which neither *SGD* algorithm exhibits.

AdamQLR (Tuned) performs very similarly to *SGD Minimal* on this setting, albeit now with a slight tendency to overfit towards the end of training. Further, it achieves similar final training losses to *Adam*, though the latter reaches these losses much faster. On the other hand, *AdamQLR (Untuned)* shows a much greater distinction from *AdamQLR (Tuned)* here than in our other experiments, suggesting the default hyperparameters we propose are not as immediately applicable to transformer models. However, it is reassuring that this algorithm achieves monotonically decreasing training

Table 1: Hyperparameter search spaces for Section 4

Hyperparameter	Search Range
Batch Size	Uniform in $\{50, 100, 200, 400, 800, 1\,600, 3\,200\}$
Learning Rate α	*SGD*: Logarithmic in $[10^{-6}, 10^{-1}]$ *Adam*: Logarithmic in $[10^{-6}, 1]$
Momentum	Logarithmic in $[10^{-4}, 0.3]$, subtracted from 1
Weight Decay	Logarithmic in $[10^{-10}, 1]$
Initial Damping λ_0	Logarithmic in $[10^{-8}, 1]$
Damping Decrease Factor ω_{dec}	Logarithmic in $[0.5, 1.0]$
Damping Increase Factor ω_{inc}	Logarithmic in $[1.0, 4.0]$

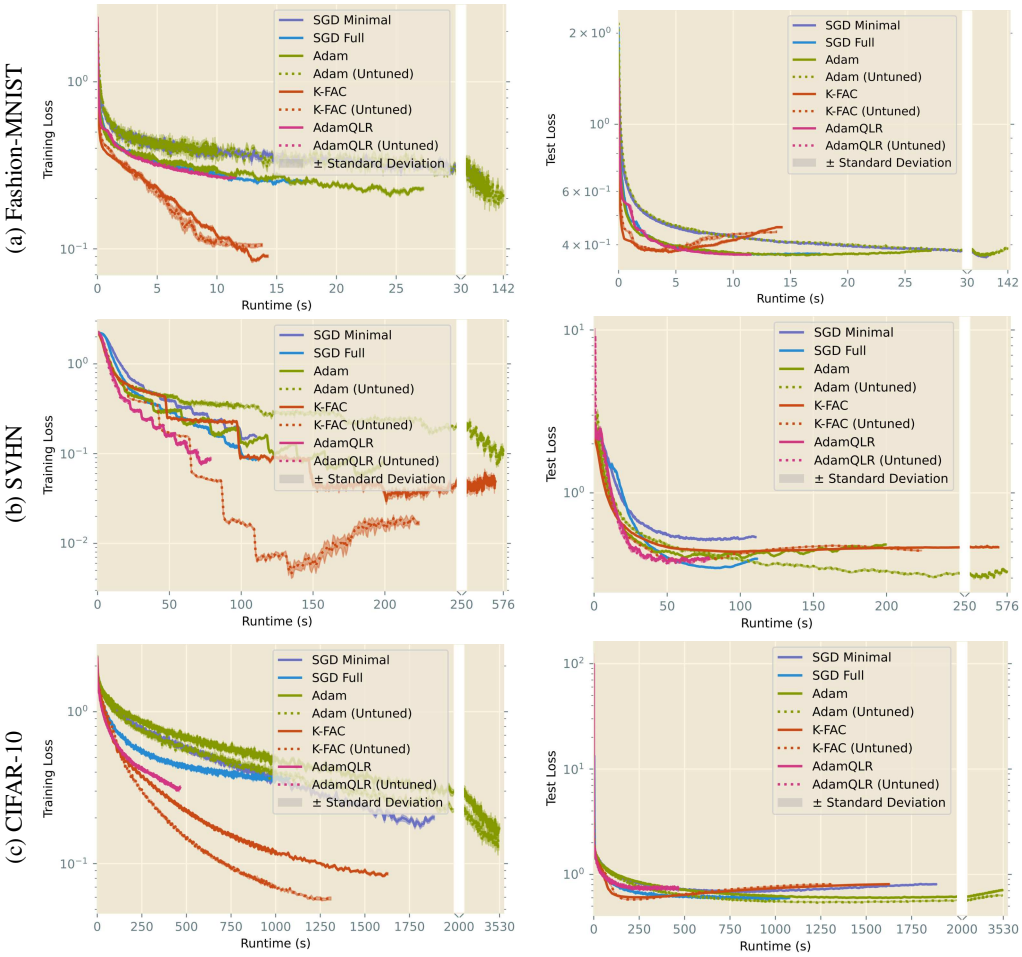


Figure 5: Median training (left) and test (right) loss trajectories, bootstrap-sampled over 50 repetitions per algorithm. Hyperparameters chosen by ASHA over 200 initialisations. Note changes of scale on the time axes. See also our numerical comparison in Table 5.

Table 2: Optimal hyperparameters used to produce the results of Section 4, Appendix B.1.2 and Appendix B.3

Dataset	Algorithm	Batch Size	Learning Rate	Momentum	Weight Decay	Initial Damping	Damping Decrease Factor	Damping Increase Factor
Rosenbrock	GD Minimal	—	—	—	—	—	—	—
	GD Full	—	—	—	—	—	—	—
	Adam	—	9.88×10^{-2}	—	—	—	—	—
	Adam (Untuned)	—	—	—	—	—	—	—
	K-FAC	—	—	—	—	2.70×10^{-7}	—	—
	AdamQLR (Hessian)	—	—	—	—	4.14×10^{-1}	0.7	1.1
	AdamQLR (Untuned, Hessian)	—	—	—	—	1.00	0.9	1.1
UCI Energy	SGD Minimal	100	9.88×10^{-2}	—	—	—	—	—
	SGD Full	400	6.92×10^{-2}	0.996	1.29×10^{-4}	—	—	—
	Adam	800	2.99×10^{-2}	—	—	—	—	—
	Adam (Untuned)	588	—	—	—	—	—	—
	Adam (Tuned ϵ)	1600	—	—	—	—	—	—
	K-FAC	50	—	—	—	1.00×10^{-2}	—	—
	K-FAC (Untuned)	3200	—	—	—	—	—	—
	K-FAC (Unadaptive)	1600	—	—	—	—	—	—
	AdamQLR	800	—	—	—	5.39×10^{-3}	0.6	1.3
	AdamQLR (Untuned)	3200	—	—	—	—	0.5	2.0
	Baydin SGD	50	—	—	—	—	—	—
UCI Protein	SGD Minimal	400	7.00×10^{-2}	—	—	—	—	—
	SGD Full	100	2.17×10^{-4}	0.997	1.54×10^{-8}	—	—	—
	Adam	800	5.42×10^{-3}	—	—	—	—	—
	Adam (Untuned)	1000	—	—	—	—	—	—
	Adam (Tuned ϵ)	800	—	—	—	—	—	—
	K-FAC	3200	—	—	—	2.11×10^{-1}	—	—
	K-FAC (Untuned)	3200	—	—	—	—	—	—
	K-FAC (Unadaptive)	1600	—	—	—	—	—	—
	AdamQLR	400	—	—	—	1.49×10^{-4}	0.6	1.7
	AdamQLR (Untuned)	3200	—	—	—	—	0.5	2.0
	Baydin SGD	400	—	—	—	—	—	—
Fashion-MNIST	SGD Minimal	100	8.01×10^{-2}	—	—	—	—	—
	SGD Full	800	5.81×10^{-2}	0.929	1.65×10^{-8}	—	—	—
	Adam	400	2.56×10^{-3}	—	—	—	—	—
	Adam (Untuned)	50	—	—	—	—	—	—
	Adam (Tuned ϵ)	800	—	—	—	—	—	—
	K-FAC	3200	—	—	—	1.92×10^{-1}	—	—
	K-FAC (Untuned)	3200	—	—	—	—	—	—
	K-FAC (Unadaptive)	400	—	—	—	—	—	—
	AdamQLR (Hessian)	3200	—	—	—	3.04×10^{-2}	0.8	1.9
	AdamQLR (Undamped)	3200	—	—	—	—	—	—
	AdamQLR	3200	—	—	—	1.17×10^{-5}	0.7	1.8
	AdamQLR (Untuned)	3200	—	—	—	—	0.5	2.0
	Baydin SGD	50	—	—	—	—	—	—
SVHN	SGD Minimal	1600	3.86×10^{-2}	—	—	—	—	—
	SGD Full	1600	6.10×10^{-3}	0.986	8.61×10^{-7}	—	—	—
	Adam	800	4.10×10^{-4}	—	—	—	—	—
	Adam (Untuned)	256	—	—	—	—	—	—
	Adam (Tuned ϵ)	400	—	—	—	—	—	—
	K-FAC	800	—	—	—	6.40×10^{-1}	—	—
	K-FAC (Untuned)	3200	—	—	—	—	—	—
	K-FAC (Unadaptive)	1600	—	—	—	—	—	—
	AdamQLR	3200	—	—	—	4.73×10^{-1}	0.6	1.1
	AdamQLR (Untuned)	3200	—	—	—	—	0.5	2.0
	Baydin SGD	800	—	—	—	—	—	—
CIFAR-10	SGD Minimal	200	3.47×10^{-2}	—	—	—	—	—
	SGD Full	400	3.83×10^{-2}	0.920	8.74×10^{-4}	—	—	—
	Adam	100	2.04×10^{-4}	—	—	—	—	—
	Adam (Untuned)	128	—	—	—	—	—	—
	K-FAC	1600	—	—	—	9.03×10^{-1}	—	—
	K-FAC (Untuned)	3200	—	—	—	—	—	—
	AdamQLR (Hessian)	3200	—	—	—	6.00×10^{-1}	0.9	3.7
	AdamQLR (Undamped)	1600	—	—	—	—	—	—
	AdamQLR	3200	—	—	—	2.99×10^{-5}	0.7	1.3
	AdamQLR (Untuned)	3200	—	—	—	—	0.5	2.0
Penn Treebank	SGD Minimal	20	4.61×10^{-2}	—	—	—	—	—
	SGD Full	35	5.01×10^{-2}	0.891	1.24×10^{-3}	—	—	—
	Adam	50	2.54×10^{-4}	—	—	—	—	—
	Adam (Untuned)	30	—	—	—	—	—	—
	AdamQLR (Untuned)	30	—	—	—	1.00×10^{-3}	0.5	2.0
	AdamQLR (Tuned)	20	—	—	—	7.54×10^{-1}	0.7	2.2
	AdamQLR	35	—	—	—	3.53×10^{-2}	0.9	2.9
	AdamQLR (Untuned)	30	—	—	—	—	0.5	2.0

Table 3: System configurations used to run our experiments.

Type	CPU	GPU (NVIDIA)	Python	JAX	CUDA	cuDNN
Consumer Desktop	Intel Core i7-3930K	RTX 2080GTX	3.10.11	0.3.25	11.4	8.05
Local Cluster	Intel Core i9-10900X	RTX 2080GTX	3.10.11	0.3.25	11.8	8.05

Table 4: Licences under which we use datasets in this work

Dataset	Licence	Source	Input	Output	Total Size
UCI Energy	Creative Commons Attribution 4.0 International (CC BY 4.0)	Tsanas & Xifara (2012); Gal & Ghahramani (2016)	8-Vector	Scalar	692
UCI Protein	None specified	Rana (2013); Gal & Ghahramani (2016)	9-Vector	Scalar	45 730
Fashion-MNIST	MIT	Xiao et al. (2017)	28×28 Image	Class (from 10)	60 000
CIFAR-10	None specified	Krizhevsky (2009)	32×32 Image	Class (from 10)	60 000
SVHN	None specified	Netzer et al. (2011)	32×32 Image	Class (from 10)	99 289

and test losses — combined with *AdamQLR (Untuned)*'s robustness on other experiments, this leads us to suspect that a transformer-specific choice of default hyperparameters would provide similar robustness in this setting. We leave an investigation of these alternative defaults to future work.

B.1.3. Numerical Results

In Table 5, we give a numerical presentation of the results in Figures 2, their corresponding loss plots from Figure 5 (Appendix B.1.1) and our additional Penn Treebank study from Figure 6 (Appendix B.1.2). We use a similar bootstrapping technique to Section 4 to give estimates for typical runtimes and numbers of steps completed.

B.1.4. Fixed-Runtime Comparisons

Our main results in Section 4 impose a primary constraint of a fixed number of epochs, with a secondary constraint of a runtime limit. Further, since our hyperparameter optimisation sought a minimal validation loss, the training loss evolutions display an early-stopping-like behaviour in which they do not fully converge. To develop additional context on AdamQLR's performance, we repeat these experiments without the primary number-of-epochs constraint, such that our hyperparameter tuning directly optimises for the best loss attained after the 15 minute runtime limit, and the algorithms are evaluated on the same metric. Figure 7 and Table 6 show results where we optimised for final validation loss, while Figure 8 and Table 7 show results where the hyperparameters were optimised to minimise final *training* loss. This latter setting allows us to compare the naïve power of each optimiser to optimise the given objective in isolation.

These results display an interesting tendency for *K-FAC* to wildly diverge in the later phases of training on Fashion-MNIST, an effect which *AdamQLR* is largely able to avoid. Broadly speaking, *AdamQLR* gives competitive generalisation performance on UCI Energy and UCI Protein in Figure 7, with a more pronounced overfitting behaviour on larger datasets. However, on CIFAR-10 *AdamQLR (Tuned)* diverges severely. We additionally see an effective demonstration of *AdamQLR*'s optimisation power in Figure 8 — although training performance on Fashion-MNIST again lags behind *Adam* in this setting, larger datasets achieve particularly strong training loss evolutions, though CIFAR-10 remains unstable.

B.2. Sensitivity Studies

To justify our configurations and further demonstrate the utility of our algorithm, we conduct a range of sensitivity experiments for *AdamQLR (Tuned)* trained on Fashion-MNIST under the same conditions as in Section 4.4. All hyperparameters except for the one under investigation are fixed at the best values found for ASHA in those experiments. Again, our plots show the averages of median trends of bootstrap-sampled sets of 50 repetitions for each configuration considered.

B.2.1. Learning Rate Rescaling

Firstly, we analyse the accuracy of our learning rate selection strategy by executing our algorithm as normal, but setting $\alpha \leftarrow k\alpha$ for each k in $\{2^{-1.0}, 2^{-0.8}, 2^{-0.6}, \cdots, 2^{1.0}\}$. In effect, we investigate the potential for systemic bias in our learning rate selection by asking if our results would improve with a constant scaling factor on those learning rates.

Our results in Figure 9 show the $k = 2^{1.0}$ case exhibiting large variance due to unstable runs, while the best training

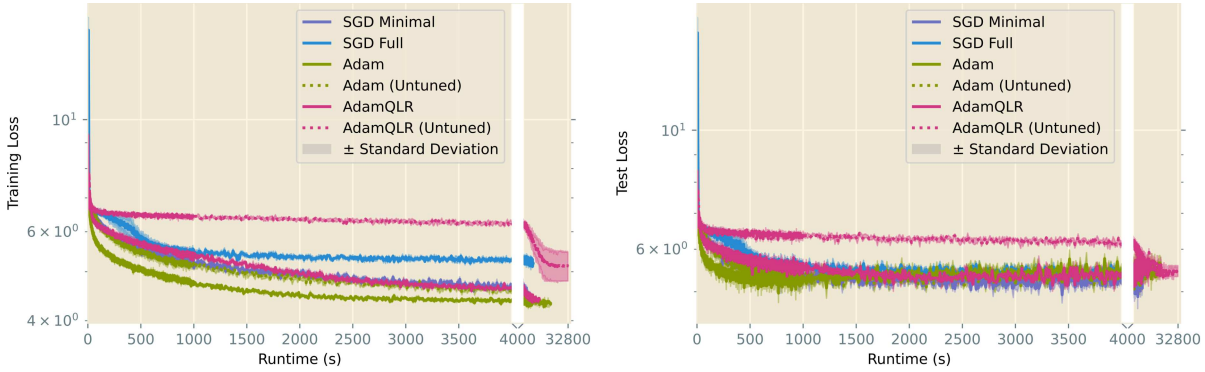


Figure 6: Median training (left) and test (right) loss trajectories for Penn Treebank on GPT-2, bootstrap-sampled over 10 repetitions per algorithm. Hyperparameters chosen by ASHA over 200 initialisations. Note changes of scale on the time axes. See also our numerical presentation in Table 5.

Table 5: Numerical study of the results shown in Figures 2, 5 and 6: final statistics after epoch-constrained training on our benchmark tasks.

Dataset	Algorithm	Training Loss	Training Accuracy	Test Loss	Test Accuracy	Generalisation Gap	Total Steps	Total Time (s)
UCI Energy	SGD Minimal	$0.000\,674 \pm 0.000\,031$	—	$0.001\,245 \pm 0.000\,049$	—	$0.000\,571 \pm 0.000\,081$	$24\,000 \pm 0$	276.79 ± 0.30
	SGD Full	$0.000\,431 \pm 0.000\,019$	—	$0.000\,657 \pm 0.000\,016$	—	$0.000\,226 \pm 0.000\,035$	8000 ± 0	134.37 ± 0.34
	Adam	$0.000\,282 \pm 0.000\,026$	—	$0.000\,768 \pm 0.000\,029$	—	$0.000\,486 \pm 0.000\,054$	4000 ± 0	94.05 ± 0.11
	Adam (Untuned)	$0.000\,596 \pm 0.000\,021$	—	$0.001\,147 \pm 0.000\,058$	—	$0.000\,550 \pm 0.000\,079$	4000 ± 0	77.92 ± 0.49
	K-FAC	$0.000\,134\,9 \pm 0.000\,005\,6$	—	$0.000\,843 \pm 0.000\,027$	—	$0.000\,708 \pm 0.000\,032$	$48\,000 \pm 0$	580.38 ± 0.46
	K-FAC (Untuned)	$0.000\,295\,3 \pm 0.000\,009\,4$	—	$0.001\,069 \pm 0.000\,039$	—	$0.000\,774 \pm 0.000\,048$	1027 ± 20	64.7 ± 1.1
	AdamQLR	$0.000\,333\,0 \pm 0.000\,008\,7$	—	$0.000\,873 \pm 0.000\,021$	—	$0.000\,540 \pm 0.000\,030$	4000 ± 0	92.49 ± 0.10
	AdamQLR (Untuned)	$1.6 \times 10^{29} \pm 5.3 \times 10^{28}$	—	$1.3 \times 10^{29} \pm 4.2 \times 10^{28}$	—	$-3.6 \times 10^{28} \pm 9.5 \times 10^{28}$	175.0 ± 2.6	10.82 ± 0.32
UCI Protein	SGD Minimal	0.2583 ± 0.0041	—	0.2714 ± 0.0019	—	0.0132 ± 0.0060	$17\,600 \pm 0$	249.16 ± 0.75
	SGD Full	0.2401 ± 0.0088	—	0.2497 ± 0.0016	—	0.010 ± 0.010	$70\,000 \pm 0$	648.69 ± 0.52
	Adam	0.2370 ± 0.0029	—	0.2475 ± 0.0011	—	0.0105 ± 0.0040	8800 ± 0	185.62 ± 0.56
	Adam (Untuned)	0.2511 ± 0.0014	—	$0.262\,49 \pm 0.000\,52$	—	0.0114 ± 0.0019	7000 ± 0	182.31 ± 0.75
	K-FAC	0.2015 ± 0.0010	—	$0.230\,18 \pm 0.000\,77$	—	0.0287 ± 0.0018	2200 ± 0	146.89 ± 0.60
	K-FAC (Untuned)	0.2010 ± 0.0010	—	$0.229\,89 \pm 0.000\,90$	—	0.0289 ± 0.0020	2200 ± 0	148.47 ± 0.66
	AdamQLR	0.2241 ± 0.0048	—	$0.237\,33 \pm 0.000\,76$	—	0.0132 ± 0.0056	$17\,600 \pm 0$	284.14 ± 0.30
	AdamQLR (Untuned)	0.2451 ± 0.0017	—	$0.257\,32 \pm 0.000\,56$	—	0.0122 ± 0.0022	2200 ± 0	138.13 ± 0.45
Fashion-MNIST	SGD Minimal	0.244 ± 0.011	0.9070 ± 0.0066	0.3678 ± 0.0013	$0.872\,22 \pm 0.000\,42$	0.124 ± 0.013	5000 ± 0	79.671 ± 0.064
	SGD Full	0.2530 ± 0.0042	0.9088 ± 0.0023	0.3738 ± 0.0016	$0.871\,15 \pm 0.000\,56$	0.1209 ± 0.0058	630 ± 0	15.16 ± 0.37
	Adam	0.2292 ± 0.0052	0.9142 ± 0.0020	0.3822 ± 0.0015	$0.873\,20 \pm 0.000\,72$	0.1530 ± 0.0067	1250 ± 0	26.286 ± 0.057
	Adam (Untuned)	0.214 ± 0.011	0.9214 ± 0.0045	0.3899 ± 0.0011	$0.874\,37 \pm 0.000\,74$	0.176 ± 0.012	$10\,000 \pm 0$	140.47 ± 0.26
	K-FAC	0.0904 ± 0.0016	$0.976\,12 \pm 0.000\,89$	0.4572 ± 0.0028	$0.868\,92 \pm 0.000\,64$	0.3668 ± 0.0044	160 ± 0	13.378 ± 0.082
	K-FAC (Untuned)	0.1054 ± 0.0023	$0.970\,58 \pm 0.000\,72$	0.4416 ± 0.0033	$0.870\,70 \pm 0.000\,47$	0.3361 ± 0.0056	160 ± 0	9.66 ± 0.35
	AdamQLR	0.2664 ± 0.0023	0.9076 ± 0.0012	$0.371\,00 \pm 0.000\,78$	$0.870\,40 \pm 0.000\,21$	0.1046 ± 0.0030	160 ± 0	10.53 ± 0.20
	AdamQLR (Untuned)	0.2692 ± 0.0029	0.9050 ± 0.0010	$0.370\,28 \pm 0.000\,60$	$0.871\,00 \pm 0.000\,28$	0.1011 ± 0.0035	160 ± 0	10.656 ± 0.075
SVHN	SGD Minimal	0.1544 ± 0.0019	0.9337 ± 0.0013	0.5344 ± 0.0042	$0.842\,87 \pm 0.000\,81$	0.3800 ± 0.0061	390 ± 0	110.28 ± 0.22
	SGD Full	0.0871 ± 0.0021	$0.952\,25 \pm 0.000\,58$	0.3939 ± 0.0024	$0.898\,83 \pm 0.000\,56$	0.3068 ± 0.0044	390 ± 0	110.52 ± 0.15
	Adam	0.0776 ± 0.0033	$0.966\,36 \pm 0.000\,86$	0.4827 ± 0.0038	0.8842 ± 0.0010	0.4051 ± 0.0071	770 ± 0	198.50 ± 0.13
	Adam (Untuned)	0.1109 ± 0.0097	0.9696 ± 0.0031	0.3275 ± 0.0037	$0.914\,72 \pm 0.000\,60$	0.217 ± 0.013	2390 ± 0	570.6 ± 3.8
	K-FAC	0.0480 ± 0.0038	0.9828 ± 0.0011	0.4638 ± 0.0034	$0.862\,70 \pm 0.000\,67$	0.4158 ± 0.0072	770 ± 0	500.8 ± 1.2
	K-FAC (Untuned)	$0.016\,84 \pm 0.000\,78$	$0.997\,62 \pm 0.000\,24$	0.4425 ± 0.0023	$0.879\,26 \pm 0.000\,46$	0.4256 ± 0.0031	200 ± 0	221.72 ± 0.17
	AdamQLR	0.0892 ± 0.0027	$0.974\,46 \pm 0.000\,80$	0.4034 ± 0.0054	$0.895\,95 \pm 0.000\,80$	0.3142 ± 0.0081	200 ± 0	77.03 ± 0.71
	AdamQLR (Untuned)	0.0876 ± 0.0029	$0.975\,05 \pm 0.000\,69$	0.3952 ± 0.0060	$0.897\,60 \pm 0.001\,00$	0.3076 ± 0.0089	200 ± 0	78.04 ± 0.39
CIFAR-10	SGD Minimal	0.2005 ± 0.0064	0.9261 ± 0.0035	0.8074 ± 0.0035	$0.800\,03 \pm 0.000\,83$	0.6069 ± 0.0099	$16\,200 \pm 0$	1846 ± 11
	SGD Full	0.3683 ± 0.0087	0.8633 ± 0.0027	0.5975 ± 0.0082	0.8065 ± 0.0024	0.229 ± 0.017	8136 ± 0	1058.6 ± 2.7
	Adam	0.173 ± 0.010	0.9344 ± 0.0065	0.7095 ± 0.0031	$0.823\,64 \pm 0.000\,55$	0.536 ± 0.013	$32\,400 \pm 0$	3480.1 ± 4.0
	Adam (Untuned)	0.140 ± 0.015	0.9475 ± 0.0044	0.6322 ± 0.0039	$0.842\,42 \pm 0.000\,70$	0.493 ± 0.019	$25\,344 \pm 0$	3310 ± 180
	K-FAC	0.0856 ± 0.0017	$0.943\,01 \pm 0.000\,91$	0.8087 ± 0.0024	$0.795\,93 \pm 0.000\,52$	0.7231 ± 0.0041	2088 ± 0	1601 ± 16
	K-FAC (Untuned)	0.0589 ± 0.0014	$0.982\,77 \pm 0.000\,61$	0.8068 ± 0.0020	$0.806\,61 \pm 0.000\,71$	0.7479 ± 0.0034	1080 ± 0	1227.0 ± 1.9
	AdamQLR	0.3202 ± 0.0040	0.8934 ± 0.0012	0.733 ± 0.011	0.7931 ± 0.0027	0.412 ± 0.015	1080 ± 0	463.73 ± 0.66
	AdamQLR (Untuned)	0.3213 ± 0.0035	$0.895\,07 \pm 0.000\,98$	0.706 ± 0.015	0.7986 ± 0.0032	0.385 ± 0.018	1080 ± 0	461.80 ± 0.42
Penn Treebank	SGD Minimal	4.413 ± 0.028	—	5.394 ± 0.098	—	0.98 ± 0.13	$51\,600 \pm 0$	$10\,090 \pm 190$
	SGD Full	5.177 ± 0.045	—	5.330 ± 0.063	—	0.15 ± 0.11	$53\,100 \pm 0$	$10\,030 \pm 140$
	Adam	4.310 ± 0.026	—	5.52 ± 0.11	—	1.21 ± 0.13	$37\,100 \pm 0$	9001 ± 14
	Adam (Untuned)	4.322 ± 0.029	—	5.64 ± 0.17	—	1.31 ± 0.20	$44\,200 \pm 0$	$21\,130 \pm 110$
	AdamQLR	4.418 ± 0.029	—	5.56 ± 0.13	—	1.15 ± 0.16	$29\,500 \pm 0$	$13\,930 \pm 190$
	AdamQLR (Untuned)	5.13 ± 0.33	—	5.46 ± 0.12	—	0.33 ± 0.46	$44\,200 \pm 0$	$19\,200 \pm 3000$

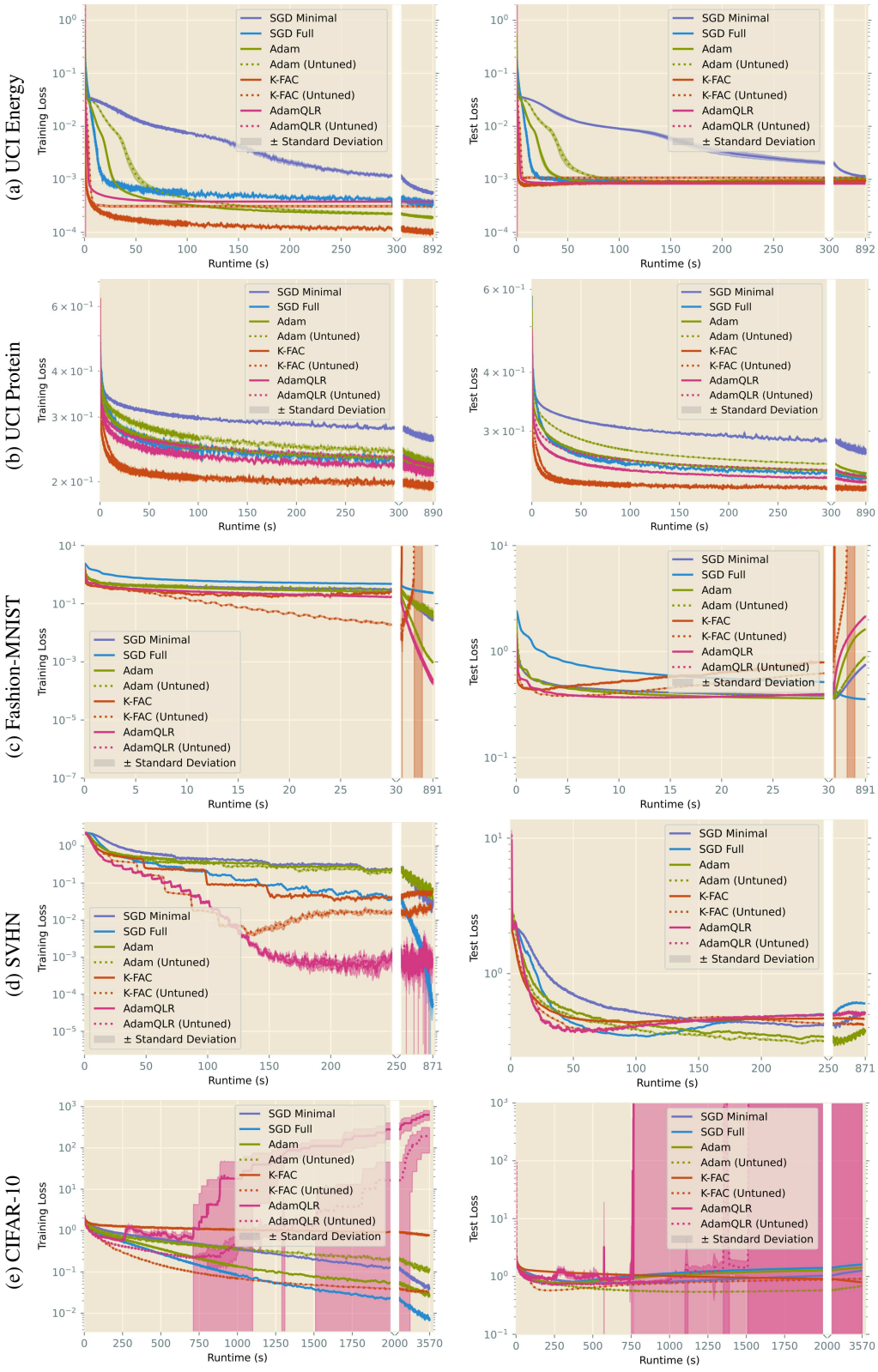


Figure 7: Median training (left) and test (right) loss trajectories, bootstrap-sampled over 50 repetitions per algorithm. Hyperparameters chosen by ASHA over 200 initialisations to minimise validation loss after a fixed runtime of 15 minutes. Note changes of scale on the time axes. See also our numerical comparison in Table 6.

Table 6: Numerical study of the results shown in Figure 7: final statistics after runtime-constrained training on our benchmark tasks, with hyperparameters optimised to minimise *validation* loss.

Dataset	Algorithm	Training Loss	Training Accuracy	Test Loss	Test Accuracy	Generalisation Gap	Total Steps	Total Time (s)
UCI Energy	SGD Minimal	$0.000\,559 \pm 0.000\,017$	—	$0.001\,133 \pm 0.000\,033$	—	$0.000\,574 \pm 0.000\,049$	$67\,771 \pm 33$	891.225 ± 0.023
	SGD Full	$0.000\,384 \pm 0.000\,028$	—	$0.001\,000 \pm 0.000\,051$	—	$0.000\,616 \pm 0.000\,079$	$79\,380 \pm 67$	889.581 ± 0.054
	Adam	$0.000\,190\,3 \pm 0.000\,005\,2$	—	$0.001\,004 \pm 0.000\,043$	—	$0.000\,814 \pm 0.000\,049$	$38\,460 \pm 110$	889.118 ± 0.070
	Adam (Untuned)	$0.000\,184\,9 \pm 0.000\,006\,0$	—	$0.001\,053 \pm 0.000\,051$	—	$0.000\,868 \pm 0.000\,057$	$45\,570 \pm 220$	889.297 ± 0.053
	K-FAC	$0.000\,103\,4 \pm 0.000\,006\,9$	—	$0.000\,895 \pm 0.000\,040$	—	$0.000\,791 \pm 0.000\,047$	$67\,983 \pm 70$	887.048 ± 0.057
	K-FAC (Untuned)	$0.000\,311 \pm 0.000\,014$	—	$0.001\,071 \pm 0.000\,044$	—	$0.000\,760 \pm 0.000\,059$	1011 ± 28	63.7 ± 1.7
	AdamQLR	$0.000\,380\,8 \pm 0.000\,009\,0$	—	$0.000\,832 \pm 0.000\,028$	—	$0.000\,451 \pm 0.000\,037$	$36\,463 \pm 31$	887.179 ± 0.061
	AdamQLR (Untuned)	$1.9 \times 10^{29} \pm 2.9 \times 10^{28}$	—	$1.7 \times 10^{29} \pm 3.3 \times 10^{28}$	—	$-2.2 \times 10^{28} \pm 6.2 \times 10^{28}$	175.0 ± 2.8	10.63 ± 0.19
UCI Protein	SGD Minimal	0.2595 ± 0.0017	—	0.2696 ± 0.0024	—	0.0101 ± 0.0041	$13\,849 \pm 61$	887.052 ± 0.031
	SGD Full	0.2227 ± 0.0042	—	$0.238\,98 \pm 0.000\,80$	—	0.0163 ± 0.0050	$54\,190 \pm 120$	886.931 ± 0.032
	Adam	0.2211 ± 0.0014	—	$0.240\,47 \pm 0.000\,72$	—	0.0194 ± 0.0021	$24\,520 \pm 130$	886.767 ± 0.052
	Adam (Untuned)	0.2264 ± 0.0024	—	$0.244\,04 \pm 0.000\,60$	—	0.0176 ± 0.0030	$34\,000 \pm 220$	886.737 ± 0.054
	K-FAC	0.1951 ± 0.0020	—	$0.225\,97 \pm 0.000\,69$	—	0.0309 ± 0.0027	$22\,466 \pm 90$	884.757 ± 0.089
	K-FAC (Untuned)	0.1932 ± 0.0013	—	0.2251 ± 0.0012	—	0.0320 ± 0.0025	$12\,890 \pm 48$	884.557 ± 0.074
	AdamQLR	0.2147 ± 0.0026	—	$0.234\,18 \pm 0.000\,15$	—	0.0195 ± 0.0028	$38\,623 \pm 57$	884.319 ± 0.096
	AdamQLR (Untuned)	0.2218 ± 0.0010	—	$0.240\,02 \pm 0.000\,62$	—	0.0182 ± 0.0016	$14\,089 \pm 20$	886.631 ± 0.068
Fashion-MNIST	SGD Minimal	0.0282 ± 0.0013	0.9942 ± 0.0011	0.7486 ± 0.0029	$0.867\,06 \pm 0.000\,38$	0.7204 ± 0.0043	$44\,110 \pm 200$	886.24 ± 0.21
	SGD Full	0.2351 ± 0.0028	0.9195 ± 0.0011	$0.355\,19 \pm 0.000\,57$	$0.875\,04 \pm 0.000\,19$	0.1201 ± 0.0034	$20\,940 \pm 29$	886.380 ± 0.052
	Adam	$0.000\,996 \pm 0.000\,023$	1.0 ± 0	1.618 ± 0.013	$0.860\,59 \pm 0.000\,42$	1.617 ± 0.013	$12\,073 \pm 24$	886.212 ± 0.081
	Adam (Untuned)	0.0345 ± 0.0038	0.9936 ± 0.0092	0.9026 ± 0.0038	$0.867\,93 \pm 0.000\,47$	0.8681 ± 0.0076	$62\,801 \pm 88$	884.692 ± 0.087
	K-FAC	$3.9 \times 10^{9} \pm 3.3 \times 10^{9}$	0.045 ± 0.017	$3.2 \times 10^{9} \pm 2.7 \times 10^{9}$	0.10 ± 0	$-7.0 \times 10^{8} \pm 6.0 \times 10^{9}$	1888 ± 48	69.8 ± 1.7
	K-FAC (Untuned)	$4.0 \times 10^{13} \pm 1.9 \times 10^{14}$	0.084 ± 0.047	$5.0 \times 10^{13} \pm 2.2 \times 10^{14}$	0.10 ± 0	$1.0 \times 10^{13} \pm 4.1 \times 10^{14}$	5120 ± 240	425 ± 22
	AdamQLR	$0.000\,188 \pm 0.000\,025$	1.0 ± 0	2.137 ± 0.026	$0.857\,40 \pm 0.000\,43$	2.137 ± 0.026	$11\,907 \pm 47$	883.45 ± 0.14
	AdamQLR (Untuned)	$0.000\,231 \pm 0.000\,020$	1.0 ± 0	2.143 ± 0.022	$0.857\,80 \pm 0.000\,61$	2.142 ± 0.022	$11\,934 \pm 34$	886.09 ± 0.19
SVHN	SGD Minimal	0.0344 ± 0.0035	$0.989\,95 \pm 0.000\,80$	0.5214 ± 0.0015	$0.884\,78 \pm 0.000\,43$	0.4870 ± 0.0050	3551 ± 16	869.955 ± 0.059
	SGD Full	$0.000\,055 \pm 0.000\,015$	1.0 ± 0	0.6038 ± 0.0047	$0.913\,60 \pm 0.000\,42$	0.6037 ± 0.0047	3282 ± 12	867.930 ± 0.079
	Adam	0.0681 ± 0.0066	0.9788 ± 0.0025	0.3845 ± 0.0028	$0.910\,66 \pm 0.000\,67$	0.3164 ± 0.0094	3516 ± 13	868.24 ± 0.16
	Adam (Untuned)	0.0345 ± 0.0024	0.9887 ± 0.0011	0.3737 ± 0.0023	$0.918\,63 \pm 0.000\,42$	0.3392 ± 0.0047	3635 ± 25	867.911 ± 0.078
	K-FAC	0.0534 ± 0.0032	$0.992\,07 \pm 0.000\,83$	0.4658 ± 0.0011	$0.861\,04 \pm 0.000\,40$	0.4125 ± 0.0043	1296.0 ± 4.9	848.48 ± 0.14
	K-FAC (Untuned)	0.0212 ± 0.0020	$0.997\,17 \pm 0.000\,41$	0.4200 ± 0.0025	$0.880\,35 \pm 0.000\,58$	0.3988 ± 0.0045	770.0 ± 5.6	843.65 ± 0.33
	AdamQLR	$0.000\,73 \pm 0.000\,41$	$0.999\,82 \pm 0.000\,15$	0.5068 ± 0.0091	$0.918\,98 \pm 0.000\,84$	0.5061 ± 0.0095	2184.0 ± 7.8	860.16 ± 0.19
	AdamQLR (Untuned)	$0.001\,08 \pm 0.000\,53$	$0.999\,72 \pm 0.000\,23$	0.5068 ± 0.0082	$0.918\,54 \pm 0.000\,97$	0.5058 ± 0.0088	2168 ± 11	862.534 ± 0.076
CIFAR-10	SGD Minimal	0.0400 ± 0.0028	0.9868 ± 0.0010	1.2623 ± 0.0053	$0.777\,28 \pm 0.000\,36$	1.2222 ± 0.0082	$26\,157 \pm 70$	3567.73 ± 0.20
	SGD Full	$0.007\,43 \pm 0.000\,57$	$0.997\,52 \pm 0.000\,19$	1.6225 ± 0.0021	$0.784\,08 \pm 0.000\,71$	1.6151 ± 0.0027	$14\,950 \pm 29$	3564.70 ± 0.21
	Adam	0.0274 ± 0.0017	$0.989\,87 \pm 0.000\,36$	1.4217 ± 0.0048	0.7760 ± 0.0010	1.3943 ± 0.0064	$20\,855 \pm 28$	3565.16 ± 0.13
	Adam (Untuned)	0.1175 ± 0.0077	0.9555 ± 0.0042	0.6830 ± 0.0031	0.8434 ± 0.0010	0.566 ± 0.011	$31\,420 \pm 180$	3566.670 ± 0.059
	K-FAC	0.792 ± 0.015	0.7256 ± 0.0042	0.7967 ± 0.0032	$0.725\,17 \pm 0.000\,84$	0.005 ± 0.018	8609 ± 34	3548.41 ± 0.18
	K-FAC (Untuned)	0.0325 ± 0.0012	$0.991\,96 \pm 0.000\,37$	0.9141 ± 0.0044	$0.806\,56 \pm 0.000\,52$	0.8816 ± 0.0055	3097 ± 16	3541.10 ± 0.14
	AdamQLR	630 ± 170	0.1283 ± 0.0018	$1.4 \times 10^{12} \pm 2.7 \times 10^{12}$	0.10 ± 0	$1.4 \times 10^{12} \pm 2.7 \times 10^{12}$	4100 ± 760	1120 ± 210
	AdamQLR (Untuned)	190 ± 120	0.171 ± 0.043	$9.0 \times 10^{10} \pm 4.5 \times 10^{11}$	0.13 ± 0.12	$9.0 \times 10^{10} \pm 4.5 \times 10^{11}$	5700 ± 1000	2450 ± 450

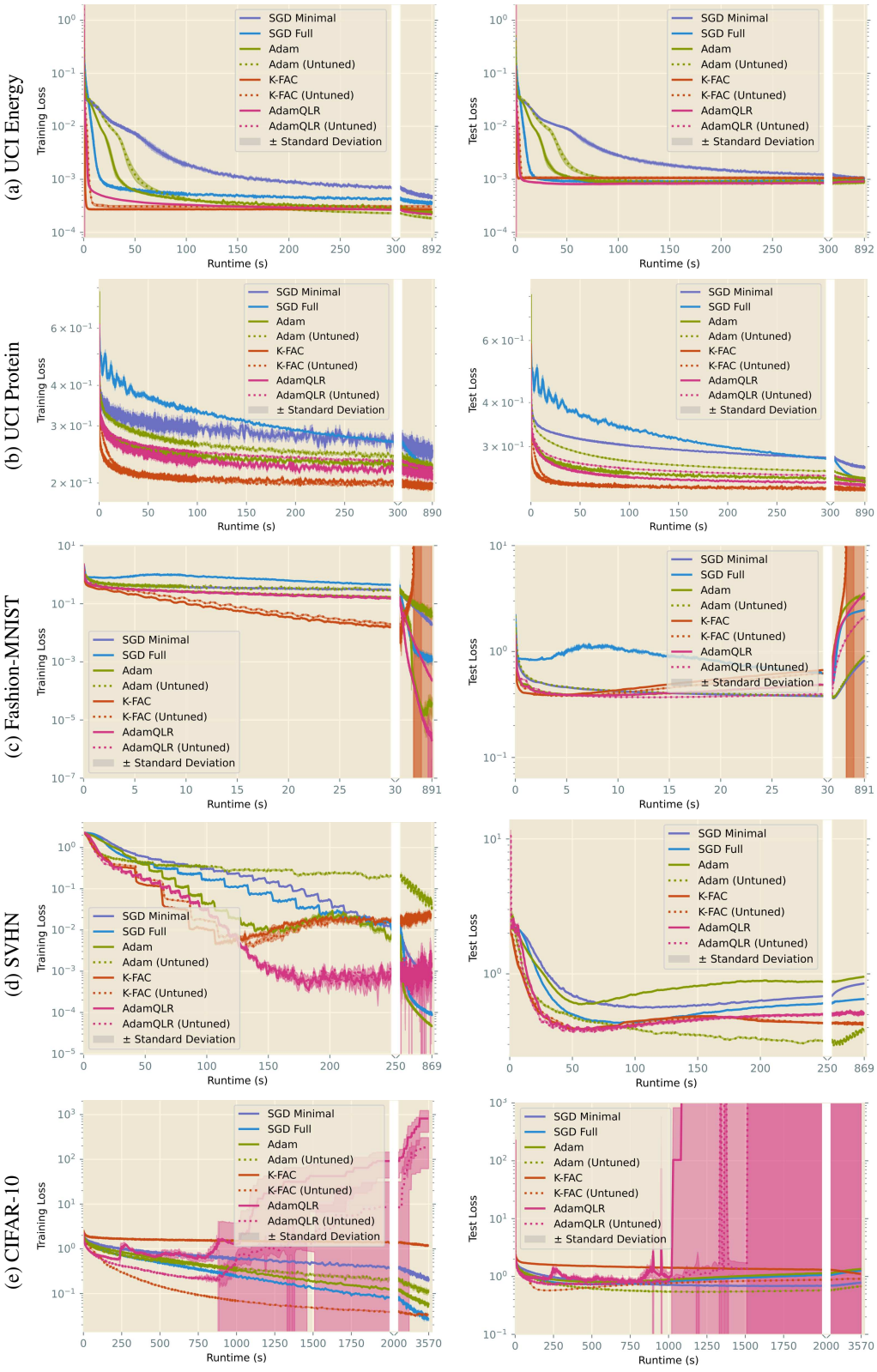


Figure 8: Median training (left) and test (right) loss trajectories, bootstrap-sampled over 50 repetitions per algorithm. Hyperparameters chosen by ASHA over 200 initialisations to minimise *training* loss after a fixed runtime of 15 minutes, characterising the naïve power of each algorithm. Note changes of scale on the time axes. See also our numerical comparison in Table 7

Table 7: Numerical study of the results shown in Figure 8: final statistics after runtime-constrained training on our benchmark tasks, with hyperparameters optimised to minimise *training* loss.

Dataset	Algorithm	Training Loss	Training Accuracy	Test Loss	Test Accuracy	Generalisation Gap	Total Steps	Total Time (s)
UCI Energy	SGD Minimal	0.000 458 ± 0.000 015	—	0.001 029 ± 0.000 028	—	0.000 571 ± 0.000 044	68 023 ± 29	891.103 ± 0.017
	SGD Full	0.000 355 ± 0.000 013	—	0.000 947 ± 0.000 052	—	0.000 592 ± 0.000 065	53 935 ± 25	889.305 ± 0.049
	Adam	0.000 232 ± 0.000 011	—	0.000 952 ± 0.000 035	—	0.000 720 ± 0.000 046	13 486.0 ± 6.3	888.054 ± 0.063
	Adam (Untuned)	0.000 185 3 ± 0.000 007 4	—	0.001 064 ± 0.000 051	—	0.000 879 ± 0.000 059	45 640 ± 190	889.305 ± 0.064
	K-FAC	0.000 271 0 ± 0.000 007 7	—	0.001 055 ± 0.000 032	—	0.000 784 ± 0.000 040	971 ± 25	23.47 ± 0.46
	K-FAC (Untuned)	0.000 308 ± 0.000 014	—	0.001 075 ± 0.000 043	—	0.000 767 ± 0.000 057	1018 ± 28	64.0 ± 2.0
	AdamQLR	0.000 218 3 ± 0.000 004 8	—	0.000 912 ± 0.000 025	—	0.000 693 ± 0.000 030	38 402 ± 63	889.021 ± 0.051
	AdamQLR (Untuned)	1.9×10^{29} ± 3.2×10^{28}	—	1.7×10^{29} ± 2.8×10^{28}	—	-2.3×10^{28} ± 6.0×10^{28}	175.0 ± 3.1	10.62 ± 0.19
UCI Protein	SGD Minimal	0.2535 ± 0.0091	—	0.262 85 ± 0.000 88	—	0.0094 ± 0.0099	75 471 ± 52	886.875 ± 0.094
	SGD Full	0.2214 ± 0.0014	—	0.241 21 ± 0.000 66	—	0.0198 ± 0.0020	13 813.0 ± 9.3	886.756 ± 0.057
	Adam	0.2209 ± 0.0013	—	0.237 47 ± 0.000 67	—	0.0166 ± 0.0020	39 106 ± 20	886.605 ± 0.056
	Adam (Untuned)	0.2266 ± 0.0030	—	0.244 19 ± 0.000 58	—	0.0176 ± 0.0036	34 000 ± 200	886.724 ± 0.047
	K-FAC	0.1994 ± 0.0018	—	0.2262 ± 0.0012	—	0.0268 ± 0.0030	22 381 ± 18	884.61 ± 0.11
	K-FAC (Untuned)	0.1927 ± 0.0013	—	0.2254 ± 0.0014	—	0.0327 ± 0.0027	12 907 ± 47	884.588 ± 0.083
	AdamQLR	0.2109 ± 0.0026	—	0.233 11 ± 0.000 55	—	0.0223 ± 0.0032	54 825 ± 37	886.587 ± 0.060
	AdamQLR (Untuned)	0.2215 ± 0.0010	—	0.239 81 ± 0.000 57	—	0.0183 ± 0.0016	14 088 ± 26	886.631 ± 0.061
Fashion-MNIST	SGD Minimal	0.019 21 ± 0.000 83	0.997 63 ± 0.000 52	0.8154 ± 0.0046	0.866 04 ± 0.000 59	0.7962 ± 0.0054	43 837 ± 51	881.95 ± 0.69
	SGD Full	0.001 19 ± 0.000 27	0.999 78 ± 0.000 14	2.492 ± 0.051	0.854 16 ± 0.000 54	2.490 ± 0.051	12 106.0 ± 5.3	886.384 ± 0.070
	Adam	0.000 034 ± 0.000 021	1.0 ± 0	3.26 ± 0.14	0.865 24 ± 0.000 48	3.26 ± 0.14	20 870 ± 46	885.83 ± 0.10
	Adam (Untuned)	0.0342 ± 0.0031	0.9948 ± 0.0086	0.9023 ± 0.0038	0.868 01 ± 0.000 36	0.8681 ± 0.0070	62 787 ± 88	884.72 ± 0.12
	K-FAC	9.0×10^{13} ± 2.8×10^{14}	0.038 ± 0.010	1.1 ± 3.9×10^{14}	0.10 ± 0	2.0×10^{13} ± 6.7×10^{14}	5130 ± 110	437 ± 13
	K-FAC (Untuned)	2.7×10^{13} ± 8.2×10^{13}	0.096 ± 0.061	1.3×10^{14} ± 4.4×10^{14}	0.10 ± 0	1.1×10^{14} ± 5.2×10^{14}	5170 ± 240	433 ± 24
	AdamQLR	0.000 001 9 ± 0.000 002 7	1.0 ± 0	3.500 ± 0.055	0.860 91 ± 0.000 87	3.500 ± 0.055	20 872 ± 47	885.711 ± 0.072
	AdamQLR (Untuned)	0.000 239 ± 0.000 025	1.0 ± 0	2.148 ± 0.024	0.857 97 ± 0.000 61	2.148 ± 0.024	11 937 ± 35	886.09 ± 0.22
SVHN	SGD Minimal	0.000 694 ± 0.000 019	1.0 ± 0	0.8492 ± 0.0058	0.831 71 ± 0.000 44	0.8485 ± 0.0058	3190.00 ± 0.28	867.716 ± 0.069
	SGD Full	0.000 089 5 ± 0.000 004 0	1.0 ± 0	0.6520 ± 0.0025	0.883 26 ± 0.000 38	0.6519 ± 0.0025	3479.00 ± 0.47	867.876 ± 0.089
	Adam	0.000 047 22 ± 0.000 000 95	1.0 ± 0	0.9528 ± 0.0053	0.844 17 ± 0.000 90	0.9528 ± 0.0053	3154.00 ± 0.51	865.862 ± 0.078
	Adam (Untuned)	0.0349 ± 0.0023	0.9888 ± 0.0014	0.3743 ± 0.0026	0.918 50 ± 0.000 45	0.3394 ± 0.0050	3641 ± 25	867.911 ± 0.068
	K-FAC	0.0254 ± 0.0021	0.997 02 ± 0.000 36	0.4344 ± 0.0026	0.8775 ± 0.0011	0.4090 ± 0.0047	786.00 ± 0.34	844.76 ± 0.18
	K-FAC (Untuned)	0.0208 ± 0.0015	0.997 21 ± 0.000 35	0.4202 ± 0.0032	0.880 47 ± 0.000 62	0.3994 ± 0.0047	770.0 ± 5.4	843.69 ± 0.37
	AdamQLR	0.0016 ± 0.0011	0.999 61 ± 0.000 18	0.5103 ± 0.0082	0.915 82 ± 0.000 69	0.5088 ± 0.0093	2180.0 ± 6.2	860.22 ± 0.15
	AdamQLR (Untuned)	0.000 99 ± 0.000 59	0.999 74 ± 0.000 21	0.5074 ± 0.0060	0.9185 ± 0.0010	0.5064 ± 0.0066	2167 ± 12	862.534 ± 0.092
CIFAR-10	SGD Minimal	0.1939 ± 0.0081	0.9262 ± 0.0057	0.7827 ± 0.0034	0.7961 ± 0.0010	0.589 ± 0.012	33 701 ± 21	3568.63 ± 0.10
	SGD Full	0.0264 ± 0.0014	0.992 45 ± 0.000 35	1.2644 ± 0.0064	0.792 74 ± 0.000 50	1.2380 ± 0.0078	26 676 ± 12	3566.73 ± 0.13
	Adam	0.0559 ± 0.0039	0.9817 ± 0.0019	1.3471 ± 0.0034	0.756 20 ± 0.000 57	1.2912 ± 0.0073	26 696 ± 11	3565.90 ± 0.11
	Adam (Untuned)	0.1192 ± 0.0058	0.9561 ± 0.0036	0.6822 ± 0.0036	0.8433 ± 0.0011	0.5630 ± 0.0094	31 390 ± 180	3566.676 ± 0.069
	K-FAC	1.180 ± 0.019	0.5787 ± 0.0082	1.1067 ± 0.0080	0.6116 ± 0.0020	−0.074 ± 0.027	9660.0 ± 1.7	3548.50 ± 0.13
	K-FAC (Untuned)	0.0327 ± 0.0016	0.992 03 ± 0.000 34	0.9137 ± 0.0043	0.806 52 ± 0.000 49	0.8810 ± 0.0059	3100 ± 14	3541.06 ± 0.17
	AdamQLR	820 ± 420	0.1268 ± 0.0028	7.0×10^{12} ± 1.7×10^{13}	0.10 ± 0	7.0×10^{12} ± 1.7×10^{13}	5130 ± 960	1390 ± 260
	AdamQLR (Untuned)	180 ± 120	0.174 ± 0.048	8.0×10^{10} ± 5.5×10^{11}	0.123 ± 0.088	8.0×10^{10} ± 5.5×10^{11}	5770 ± 970	2490 ± 420

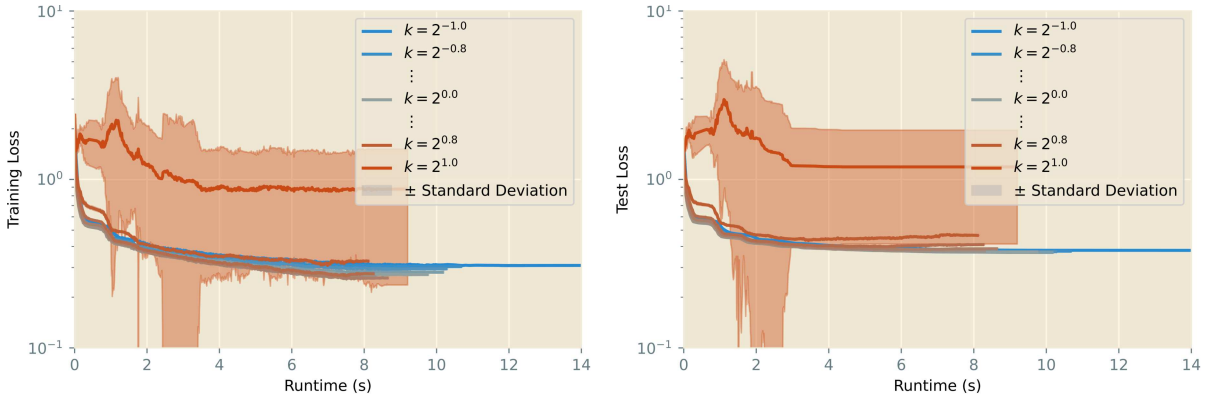


Figure 9: Sensitivity studies over learning rate, which is scaled by a variety of constant factors k for our Fashion-MNIST trial from Section 4.4.

losses are obtained for k slightly larger than unity. This makes sense given our use of damping: if stability can be achieved without damping for any given update, then the damping will serve only to downsize our proposed update step, so we should expect the best results to be obtained by slightly increasing it again. However, test loss appears generally less sensitive to k, with the lowest value obtained for $k = 1$: this would also be expected under damping, since we would hope the damping would increase generalisation performance. In aggregate, these results confirm our approach accurately selects the correct learning rate to use for any given optimisation step.

B.2.2. Initial Damping

Next, we consider the initial value λ_0 assigned to our Levenberg-Marquardt damping term λ, testing values in $\{10^{-8.0}, 10^{-7.5}, 10^{-7.0}, \cdots, 10^{0.0}\}$. Here, we seek to quantify the trade-off between damping's stabilising effect and its tendency to worsen training loss. Figure 10 presents our results.

With the exception of the very smallest values, we see our performance is largely insensitive to λ_0. This matches our empirical observation that damping becomes most important for larger-scale problems than our Fashion-MNIST setting, and thus has minimal effect here. However, given its substantial importance in these more complex experiments, it is reassuring that the inclusion of damping does not dramatically worsen performance when its influence is not required.

B.2.3. Batch Size

In Figure 11, we consider each batch size available to ASHA in Section 4.4 ($\{50, 100, 200, 400, 800, 1\,600, 3\,200\}$) to investigate the effect of this hyperparameter on our algorithm.

Since the optimal batch size selected by ASHA for *AdamQLR* was generally large (3 200 in this case), it is perhaps unsurprising that we see divergence from smaller batches. This also matches our intuition: unlike classical first-order methods, *AdamQLR* uses each batch to (implicitly) construct a full curvature matrix for the optimisation surface, which magnifies the importance of having a low-bias sample of the training data. Empirically, we found the computational benefits of fewer batches outweighed the increased cost of computing each batch, so this preference for larger batch sizes aligns with our desire to minimise runtime. Thus, our results show a clear trend that larger batch sizes give greater training and generalisation performance.

B.2.4. Damping Stepping Factor

Finally, we explore the effect of different stepping factors by setting ω_{inc} to values in $\{2^{0.0}, 2^{0.2}, 2^{0.4}, \cdots, 2^{2.0}\}$, then choosing a symmetric $\omega_{\text{dec}} = \frac{1}{\omega_{\text{inc}}}$. Our results are plotted in Figure 12.

The impact of different damping stepping factors becomes most apparent when damping plays a key role in stabilising the optimiser, which does not happen in this Fashion-MNIST test case. However, the plots match our subjective observation that the behaviour at the very start of training is critical to defining the optimisation trajectory, with a high variance at around 2 s of runtime indicating an increased sensitivity here. Moreover, the results reinforce our intuition that the exact factor by which the damping λ is modified is not crucially important, so long as AdamQLR is capable of making rapid adjustments over successive optimisation iterations when this becomes necessary.

B.2.5. Adam (Batch Size and Learning Rate) and K-FAC (Batch Size and Initial Damping

To reinforce our conclusions about the robustness of AdamQLR, we repeat our sensitivity study on batch size applied to *Adam*, and perform an additional study over learning rates, setting α to values in $\{10^{-6.00}, 10^{-5.67}, 10^{-5.33}, \cdots, 10^{0.00}\}$. Our results are plotted in Figures 13 and 14, respectively.

Similarly, we examine the robustness of *K-FAC* by means of repeated sensitivity studies over batch size and initial damping. These results are plotted in Figures 15 and 16, respectively.

We see Adam is substantially more sensitive to its learning rate than AdamQLR and K-FAC are to any of their hyperparameters, while AdamQLR strikes a middle ground between Adam and K-FAC in its sensitivity to batch size. Despite AdamQLR's near-invariance to initial damping, this hyperparameter has a sizeable impact on K-FAC. These results support the conclusion that AdamQLR is at least as robust as — and for some particularly important hyperparameters is more robust than — Adam and K-FAC.

B.3. Ablation Studies

In addition to the algorithms plotted in Section 4, we conduct additional experiments to study the impact of different components of *AdamQLR* on its overall performance. Specifically, we examine the effects of Levenberg-Marquardt damping and the choice of curvature matrix used to construct our quadratic model. We use the same experimental configuration as in Section 4, including hyperparameter tuning with ASHA, and plot bootstrapped average trends over 50 repetitions of the best hyperparameters found.

The new results presented in this section were computed on a mix of the Consumer Desktop and Local Cluster, using one GPU at a time. Since the GPUs are of the same model, we expect any performance difference due to this discrepancy to be minor.

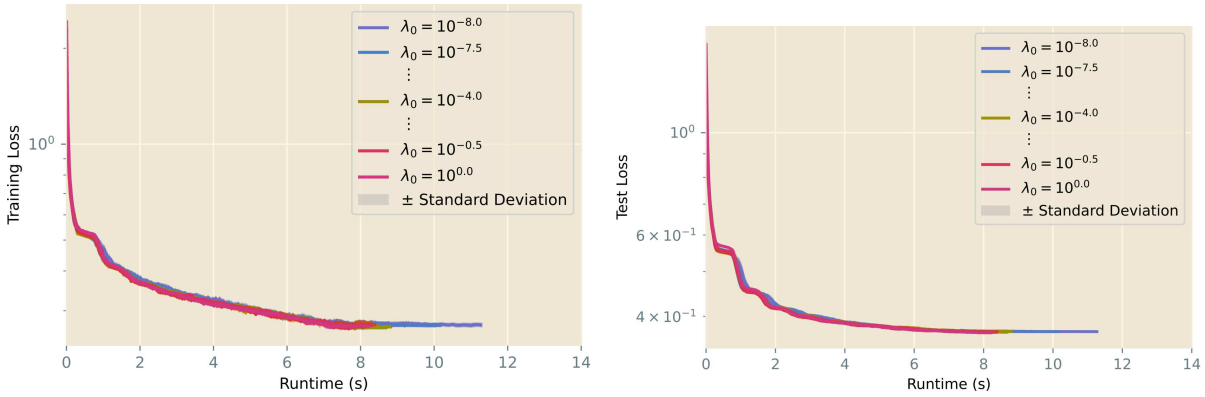


Figure 10: Sensitivity studies over initial damping value λ_0 for our Fashion-MNIST trial from Section 4.4.

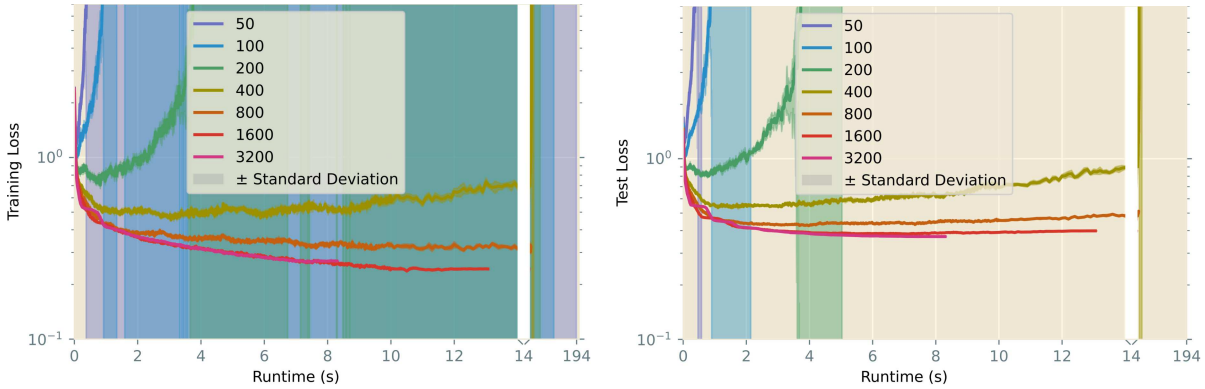


Figure 11: Sensitivity studies over batch size for our Fashion-MNIST trial from Section 4.4.

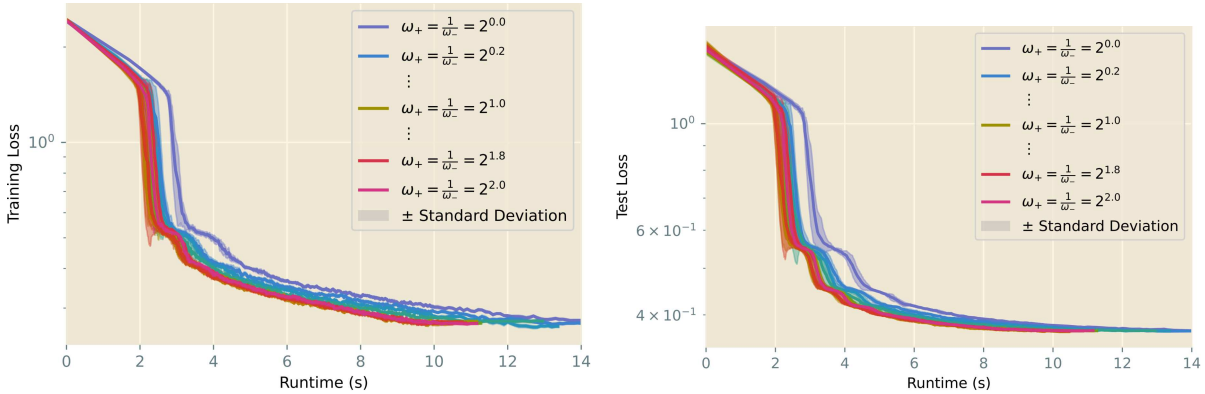


Figure 12: Sensitivity studies over damping stepping factor for our Fashion-MNIST trial from Section 4.4.

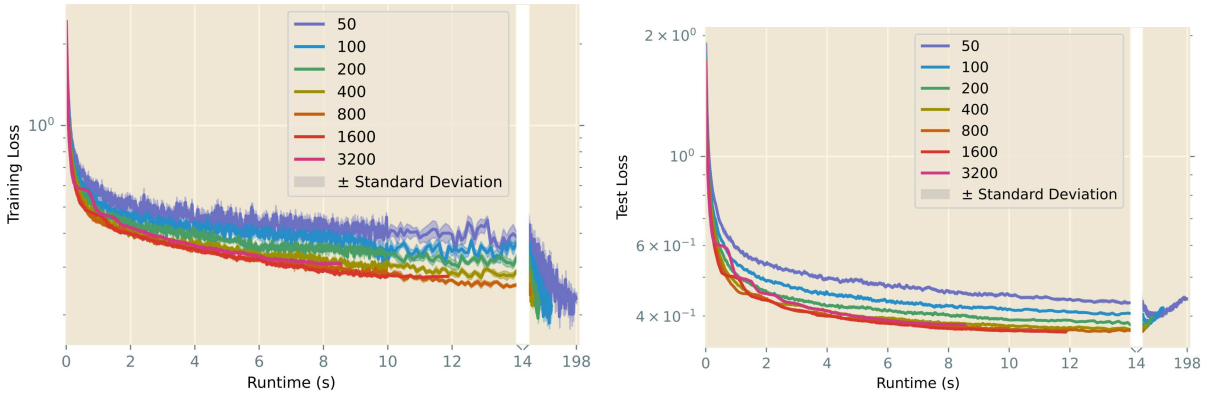


Figure 13: Sensitivity studies over batch size for our Fashion-MNIST trial from Section 4.4, considering now the *Adam* setting.

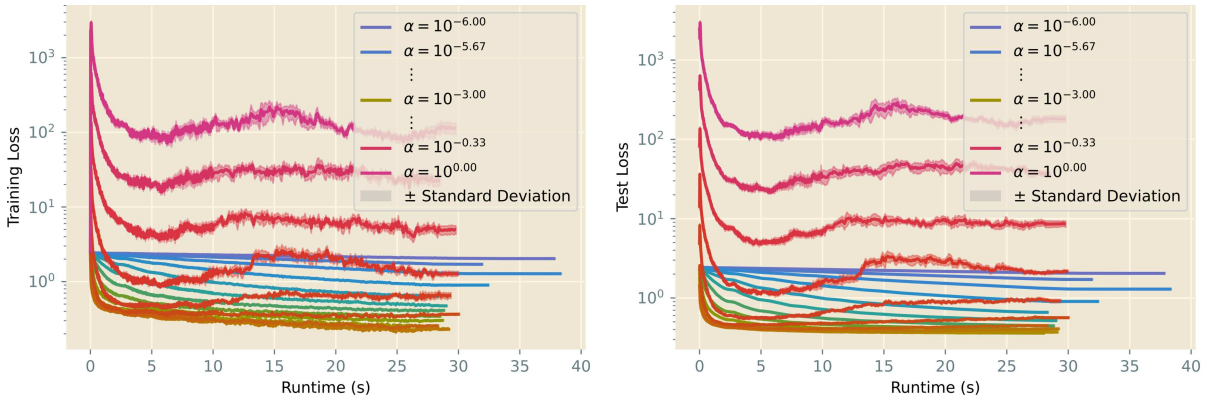


Figure 14: Sensitivity studies over learning rates for our Fashion-MNIST trial from Section 4.4, considering now the *Adam* setting.

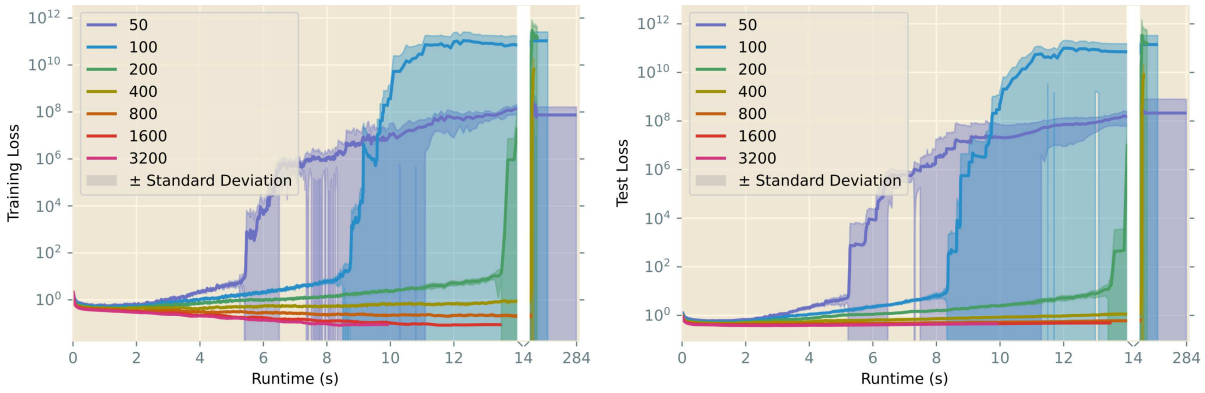


Figure 15: Sensitivity studies over batch size for our Fashion-MNIST trial from Section 4.4, considering now the *K-FAC* setting.

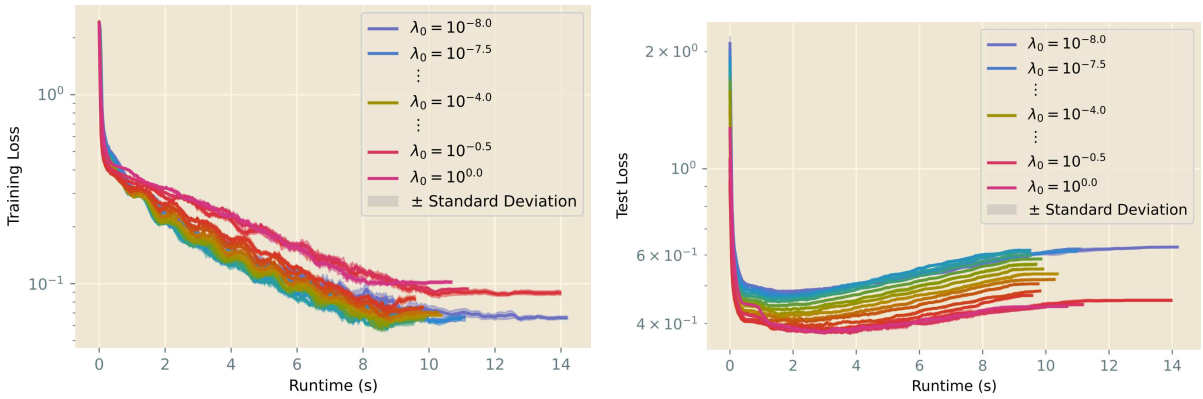


Figure 16: Sensitivity studies over initial damping for our Fashion-MNIST trial from Section 4.4, considering now the *K-FAC* setting.

B.3.1. Levenberg-Marquardt Damping

Appropriate damping is viewed as a necessity in many second-order algorithms in order to defend against degenerate parameter updates, and Figure 17 examines its inclusion in *AdamQLR*. We consider vanilla *Adam* alongside two versions of *AdamQLR*: one which includes damping, and another which excludes it, and perform hyperparameter optimisation as before on each algorithm.

On Fashion-MNIST, we see minimal effect from the inclusion of damping, as the problem does not suffer greatly from degenerate parameter updates. Thus, especially when the internal model of objective space performs well and damping is pushed to very low values, the damping makes a proportionally very small difference to the updates we take. As such, while we do benefit slightly from damping here, the advantage is very slight.

On CIFAR-10, however, we see more dramatic differences from the inclusion of damping, though we note the difference in horizontal scale is likely due to different optimal batch sizes chosen by ASHA. Adjusting for this factor, we see the undamped version of AdamQLR is substantially less stable than the standard damped setting. This result is intuitive — since the model is larger and is substantially more overparameterised than in the Fashion-MNIST case, there are likely to be more parameters to which the output of our network is insensitive, corresponding to low-curvature directions of optimisation space. These low-curvature directions correspond to small eigenvalues of the curvature matrix, so a naïve curvature-based approach would take very large steps in these directions. Because the problem is inherently non-convex and non-quadratic, such large steps would not be well-motivated, and we would suffer a resulting penalty in our rapidly-excursing loss. Indeed, during our development of AdamQLR, we observed damping to play an important role in avoiding the destabilisation of training. Further, damping clearly stabilises the algorithm here to allow for more aggressive optimisation over time. This evidence justifies our use of damping in the default AdamQLR approach.

B.3.2. Curvature Matrix

As discussed in Appendix C, there is good reason to motivate both the Hessian and the Fisher matrices as curvatures to use to select the learning rate α at each update step. To explore their relative merits, we consider two versions of *AdamQLR*: one which uses Hessian curvature to compute a learning rate and update damping, and another which uses Fisher curvature for the same purposes. The performance of hyperparameter-optimised versions of each setting is compared alongside vanilla *Adam* in Figure 18.

On Fashion-MNIST, we see a slight advantage for Fisher curvature compared to the Hessian curvature, both of which generalise very slightly better than vanilla *Adam*. While the generalisation benefit does not transfer to CIFAR-10, we still see a small speed advantage for Fisher-based curvature. Again, we note that different optimal batch sizes are likely responsible for most of the horizontal scaling difference. The similarity of these results, combined with the subjectively greater stability of the Fisher version of *AdamQLR* in our development process, justify our use of the Fisher curvature as the default in our algorithm. While Fisher-vector products are more intricate than Hessian-vector products, requiring a rederived component for each loss function, a relatively small number of different loss functions see regular use in practice, so we accept this additional burden.

B.3.3. Alternative *Adam (Tuned ϵ)* Setting

While the Adam optimiser is often invoked without tuning the ϵ hyperparameter, its interpretation as a form of 'damping' for Adam's 'curvature estimate' invites the alternative *Adam (Tuned ϵ)* setting, in which our hyperparameter optimisation approach also selects ϵ in the logarithmic range

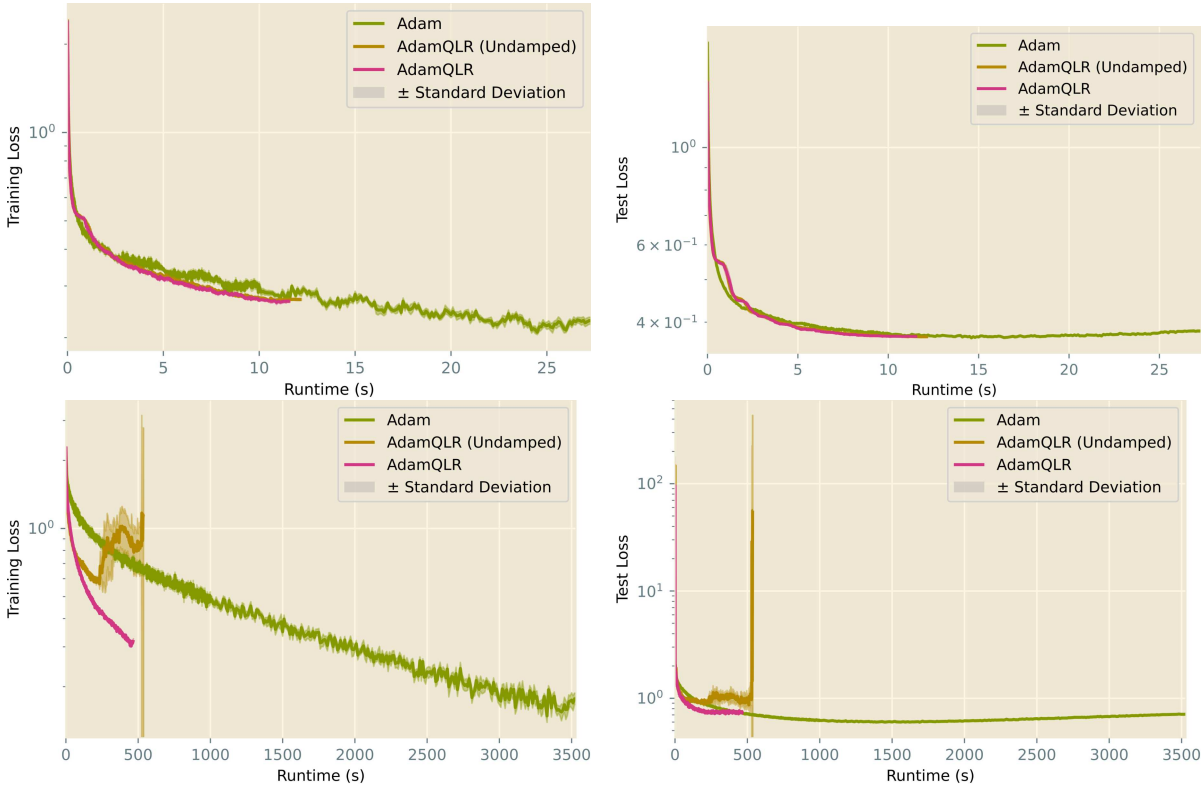


Figure 17: Evolution of Levelberg-Marquardt damping, as measured by Training (left) and Test (right) loss on Fashion-MNIST (top) and CIFAR-10 (bottom)

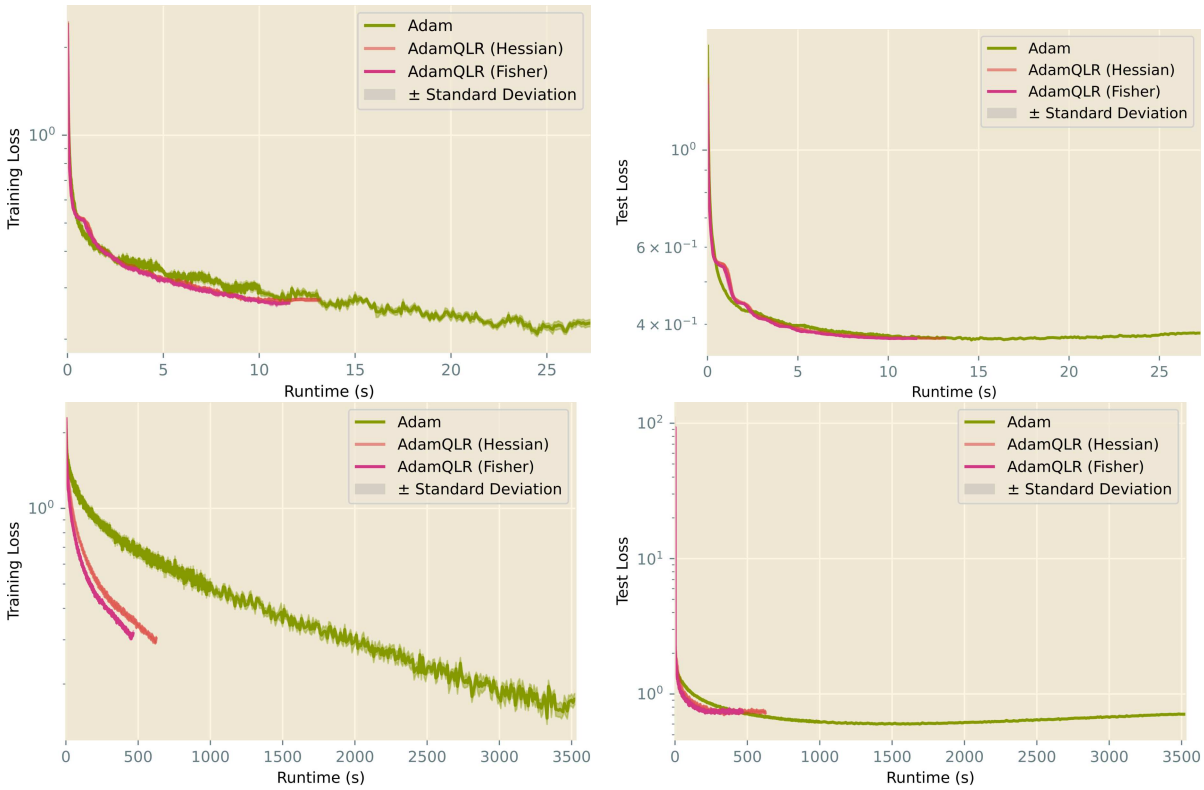


Figure 18: Evaluation of the choice of curvature matrix for the learning rate and damping calculations in *AdamQLR*

$[10^{-8}, 10^{0}]$. Our results in Figure 19 demonstrate that this implementation is generally interchangeable with the *Adam* baseline of Section 4 in terms of performance.

B.3.4. K-FAC Adaptive Heuristics

In addition to extending Adam with heuristics from K-FAC, we briefly study the impact of *removing* these heuristics from K-FAC. To this end, we introduce a *K-FAC (Unadaptive)* baseline, which replaces the adaptive damping, learning rate and momentum of the *K-FAC* setting with fixed tuned values. The results of this additional comparison are shown in Figure 20, which shows the *Unadaptive* variant makes much slower progress than vanilla *K-FAC*, and in some cases the lack of adaptivity completely destabilises K-FAC. This gives convincing evidence that the adaptive heuristics make a significant difference to *K-FAC*, and thus reinforces the central hypothesis of this work.

We also note that correctly implementing K-FAC is an extremely subtle and complex task, with many libraries excluding some aspects of the algorithm presented by Martens & Grosse (2015). For this reason, we feel many results in the literature are not entirely fair comparisons between K-FAC and other methods, which may lead to contradictory conclusions about the effectiveness of K-FAC.

B.3.5. Learning Rate Schedules

Many ML training approaches use learning rates which follow a predefined schedule in order to improve performance. We have chosen not to focus on learning rate scheduling in this work, partly due to the myriad schedules available, and partly because this can easily be incorporated into any method we consider, including AdamQLR. In support of this position, we introduce a new *Baydin SGD* baseline, which combines the adaptive learning rate method of Baydin et al. (2018) with our *SGD Minimal* setting, and compare to this method in Figure 21.

These results show the adaptive learning rate — even when its initialisation and hyper-learning rate are tuned — does not perform significantly differently to the tuned fixed learning rate of *SGD Minimal*. This concurs with the general belief that well-optimised learning rates tend to outperform adaptive learning rate methods in practice.

C. Curvature Matrices: Hessian and Fisher

In this section we discuss in more detail the two main candidates for the curvature matrix $\mathbf{C}$ in our algorithm. Recall from Section 3 that throughout we consider an arbitrary function $f(\boldsymbol{\theta})$ representing the loss function of some network parameterised by $\boldsymbol{\theta}$.

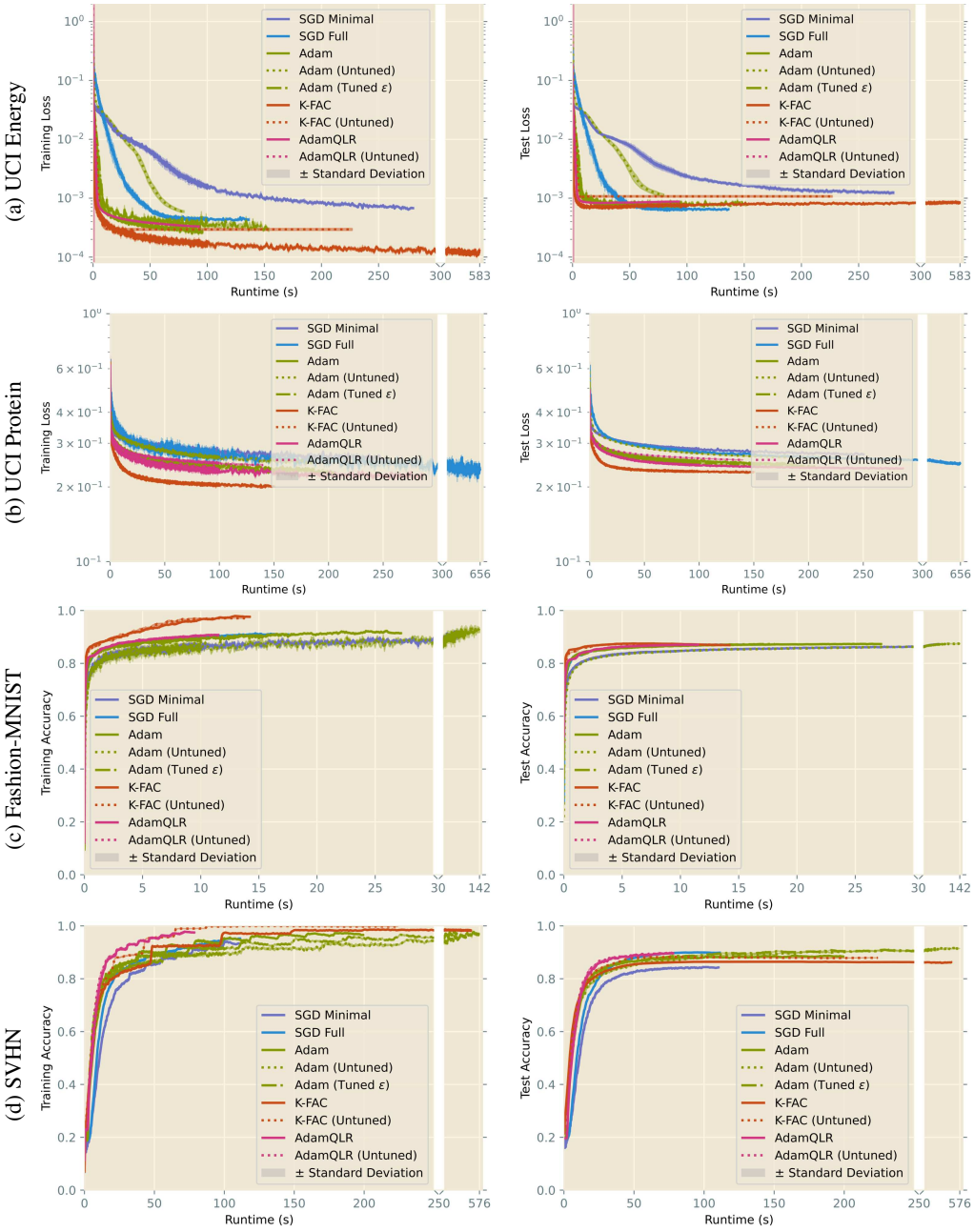


Figure 19: Reprise of Figure 2, with additional results on an *Adam (Tuned ϵ)* setting extending *Adam* by additionally tuning the ϵ hyperparameter.

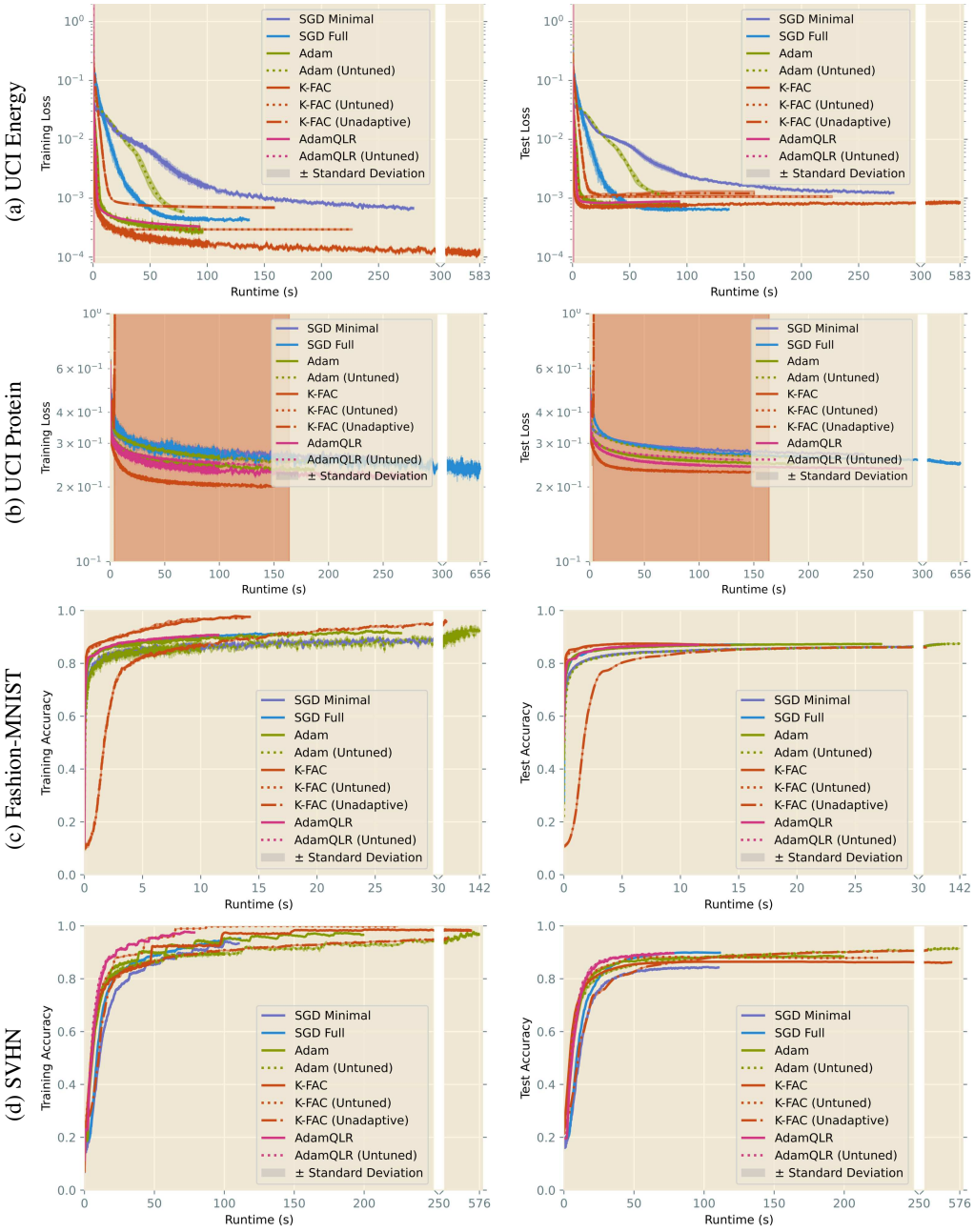


Figure 20: Reprise of Figure 2, with additional results on a *K-FAC (Unadaptive)* setting replacing adaptive heuristics with fixed tuned values.

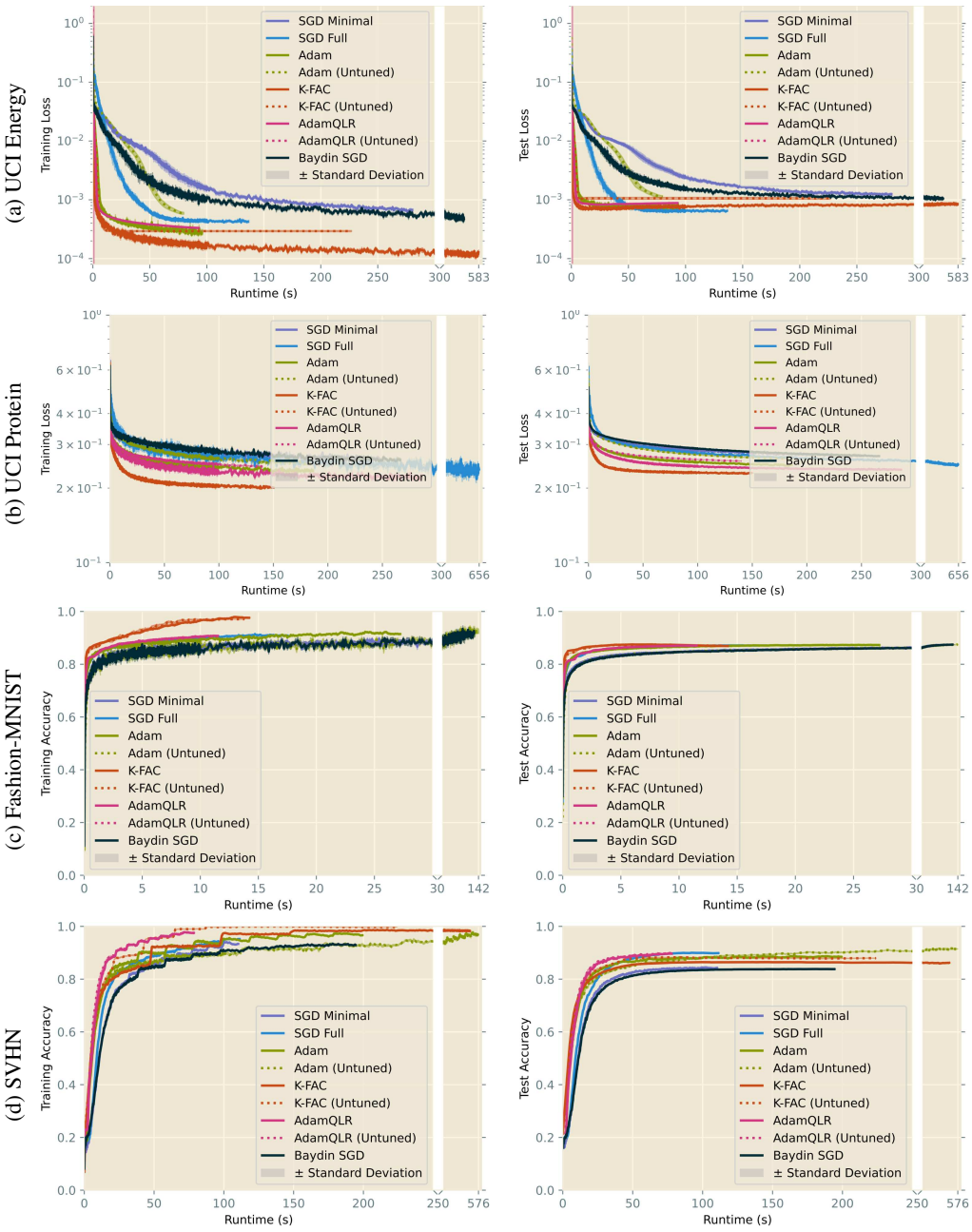


Figure 21: Reprise of Figure 2, with additional results on a *Baydin SGD* setting examining the effect of a simple adaptive learning rate strategy.

C.1. Hessian Matrix

In this setting, the Hessian curvature matrix follows naturally from the definition of the objective function. A first derivative with respect to $\boldsymbol{\theta}$ yields the gradient vector $\mathbf{g} = (\nabla_{\boldsymbol{\theta}} f)(\boldsymbol{\theta})$, and repeating the derivative yields the Hessian $\mathbf{H} = (\nabla_{\boldsymbol{\theta}} (\nabla_{\boldsymbol{\theta}} f)^{\mathsf{T}})(\boldsymbol{\theta})$.

C.2. Fisher Information Matrix

To draw a connection with the Fisher matrix, we must restate our problem in a probabilistic form. We shall separate the loss function from the neural network, naming the latter $\mathbf{w}_{\boldsymbol{\theta}}(\cdot)$, and consider input-output data pairs $(\mathbf{x}, \mathbf{y})$. Let the input data have some ground truth distribution $p(\mathbf{x})$, and suppose we choose to interpret the output of the network as a probabilistic relationship, such that $\mathbf{w}_{\boldsymbol{\theta}}(\mathbf{x}) = \log p(\mathbf{y}|\mathbf{x})$.

For this model $\mathbf{w}$, the *Fisher Information Matrix* (FIM, or "the Fisher") is defined as:

$$\mathbf{F} = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})} \mathbb{E}_{\mathbf{y}\sim p(\mathbf{y}|\mathbf{x})} \left[\frac{\partial \log p(\mathbf{y}|\mathbf{x})}{\partial \boldsymbol{\theta}} \frac{\partial \log p(\mathbf{y}|\mathbf{x})}{\partial \boldsymbol{\theta}}^{\mathsf{T}} \right]. \tag{4}$$

In its exact form, the Fisher bears many favourable properties for use in optimisation: it is positive semi-definite by construction (so represents a convex space), it is amenable to efficient computation in the form of a matrix=vector product, and provides a parameterisation-independent view of the problem (as in the Natural Gradient Descent (Amari, 1998) family of methods).

Since $\frac{\partial \log p(\mathbf{y}|\mathbf{x})}{\partial \boldsymbol{\theta}}$ is the Jacobian of the network output $\mathbf{w}_{\boldsymbol{\theta}}$ with respect to the parameters $\boldsymbol{\theta}$, the outer product of derivatives is readily available as part of our standard training regime. Although $p(\mathbf{x})$ is unknown, in the mini-batched training setting it is commonly approximated by the empirical distribution $\widehat{p}(\mathbf{x})$ implied by our training dataset. It is important to stress that the expectation of $\mathbf{y}$ is taken with respect to the output distribution of the network, *not* with respect to any ground-truth or empirical distribution $\widehat{p}(\mathbf{y}|\mathbf{x})$ given by the training data. However, some previous work uses the latter distribution as an approximation, resulting in the *empirical* Fisher matrix, which is known to be inferior to the true Fisher.

C.3. Gauss-Newton Matrix

We briefly note the Gauss-Newton matrix, which is also commonly used in second-order optimisation algorithms. We chose not to focus on this matrix because it is motivated as a lossy approximation to the Hessian, which seemed undesirable when we could access the exact Hessian and Fisher matrices through Jacobian-vector products. While it has been used in a derivation of K-FAC, we note also the equivalence of the Fisher and Generalised Gauss-Newton matrices when applying negative log-likelihood loss to the natural parameters of an exponential family (Botev et al., 2017), and to our knowledge there is only a marginal performance difference between Fisher- and Gauss-Newton-based K-FAC.

C.4. Adam and Fisher Matrix

While Adam is described by its authors as representing an approximation to the Fisher matrix (Kingma & Ba, 2015), we seek here to make the connection more explicit.

The matrix computed inside the expectation of Equation 4 has as its diagonal the elementwise square of $\frac{\partial \log p(\mathbf{y}|\mathbf{x})}{\partial \boldsymbol{\theta}}$. This is connected to the quantity $\mathbf{g}_t = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_{t-1})$ computed by Adam; by the chain rule, $\mathbf{g}_t$ is precisely the product of $\frac{\partial \log p(\mathbf{y}|\mathbf{x})}{\partial \boldsymbol{\theta}}$ and the derivative of the loss function with respect to the model output. Neglecting the effect of the latter allows us to view Adam's second-moment buffer $\widehat{\mathbf{v}}_t$ as an approximation to the diagonal of the outer product in Equation 4.

Further, because $\mathbf{g}_t$ is averaged over a mini-batch of input data, we are automatically taking approximate expectations over $\widehat{p}(\mathbf{x})$ and $\widehat{p}(\mathbf{y}|\mathbf{x})$. The approximation arises because the underlying Fisher matrix is not constant, so the contributions from each mini-batch relate to different underlying curvatures. However, the argument motivates the idea that Adam develops an approximation to the diagonal of the empirical Fisher matrix in its buffer $\widehat{\mathbf{v}}_t$.

From this perspective, Adam's elementwise division by the reciprocal of $\widehat{\mathbf{v}}_t$ is simply multiplication by the inverse (approximate) empirical Fisher, and we may interpret ϵ as a fixed damping term. This picture is slightly corrupted by the square root of $\widehat{\mathbf{v}}_t$ being the quantity actually used by Adam; this operation brings the eigenvalues of the approximate empirical Fisher closer to one, in particular increasing problematic near-zero eigenvalues to more stable values, thus justifying Kingma & Ba's statement that the square root permits more "conservative" preconditioning.