A. Derivation of the Block-Diagonal Hessian Recursion

The diagonal blocks of the pre-activation Hessian of a feedforward neural network can be related to each other via the recursion in (8). Starting from its definition in (6) we can derive the recursion:

$$\begin{aligned}
\mathcal{H}_\lambda &= \frac{\partial}{\partial h_b^\lambda}\frac{\partial E}{\partial h_a^\lambda} = \frac{\partial}{\partial h_b^\lambda}\sum_i \frac{\partial E}{\partial h_i^{\lambda+1}}\frac{\partial h_i^{\lambda+1}}{\partial h_a^\lambda} = \sum_i \frac{\partial}{\partial h_b^\lambda}\left(\frac{\partial E}{\partial h_i^{\lambda+1}}\frac{\partial h_i^{\lambda+1}}{\partial a_a^\lambda}\frac{\partial a_a^\lambda}{\partial h_a^\lambda}\right) \\
&= \sum_i W_{i,a}^{\lambda+1}\frac{\partial}{\partial h_b^\lambda}\left(\frac{\partial E}{\partial h_i^{\lambda+1}}\frac{\partial a_a^\lambda}{\partial h_a^\lambda}\right) = \sum_i W_{i,a}^{\lambda+1}\left(\frac{\partial a_a^\lambda}{\partial h_a^\lambda}\frac{\partial^2 E}{\partial h_b^\lambda \partial h_i^{\lambda+1}} + \frac{\partial E}{\partial h_i^{\lambda+1}}\frac{\partial^2 a_a^\lambda}{\partial h_a^\lambda \partial h_b^\lambda}\right) \\
&= \sum_i W_{i,a}^{\lambda+1}\left(\frac{\partial a_a^\lambda}{\partial h_a^\lambda}\sum_j \frac{\partial^2 E}{\partial h_j^{\lambda+1}\partial h_i^{\lambda+1}}\frac{\partial h_j^{\lambda+1}}{\partial h_b^\lambda} + \frac{\partial E}{\partial h_i^{\lambda+1}}\delta_{a,b}\frac{\partial^2 a_a^\lambda}{\partial {h_a^\lambda}^2}\right) \\
&= \delta_{a,b}\frac{\partial^2 a_a^\lambda}{\partial {h_a^\lambda}^2}\left(\sum_i W_{i,a}^{\lambda+1}\frac{\partial E}{\partial h_i^{\lambda+1}}\right) + \sum_{i,j} W_{i,a}^{\lambda+1}\frac{\partial a_a^\lambda}{\partial h_a^\lambda}\frac{\partial^2 E}{\partial h_j^{\lambda+1}\partial h_i^{\lambda+1}} W_{j,b}^{\lambda+1}\frac{\partial a_b^\lambda}{\partial h_b^\lambda} \\
&= \delta_{a,b}\frac{\partial^2 a_a^\lambda}{\partial {h_a^\lambda}^2}\frac{\partial E}{\partial a_a^\lambda} + \sum_{i,j} W_{i,a}^{\lambda+1}\frac{\partial a_a^\lambda}{\partial h_a^\lambda}\frac{\partial^2 E}{\partial h_j^{\lambda+1}\partial h_i^{\lambda+1}}\frac{\partial a_b^\lambda}{\partial h_b^\lambda} W_{j,b}^{\lambda+1}
\end{aligned} \tag{28}$$

Hence the pre-activation Hessian can be written in matrix notation as

$$\mathcal{H}_\lambda = B_\lambda W_{\lambda+1}^\mathsf{T} \mathcal{H}_{\lambda+1} W_{\lambda+1} B_\lambda + D_\lambda \tag{29}$$

where we define the diagonal matrices

$$[B_\lambda]_{a,a'} = \delta_{a,a'}\frac{\partial a_a^\lambda}{\partial h_a^\lambda} = \delta_{a,a'} f'(h_a^\lambda) \tag{30}$$

$$[D_\lambda]_{a,a'} = \delta_{a,a'}\frac{\partial^2 a_a^\lambda}{\partial {h_a^\lambda}^2}\frac{\partial E}{\partial x_a^\lambda} = \delta_{a,a'} f''(h_a^\lambda)\frac{\partial E}{\partial x_a^\lambda} \tag{31}$$

f' and f'' are the first and second derivatives of the transfer function f respectively.

Note that this is a special case of a more general recursion for calculating a Hessian (Gower & Gower, 2016).

B. Implementation Details

Second-order optimisation methods are based on finding some positive semi-definite quadratic approximation to the function of interest around the current value of the parameters. For the rest of the appendix we define $\hat{f}$ to be a local quadratic approximation to f given a positive semi-definite curvature matrix C:

$$f(\theta + \delta) \approx f(\theta) + \delta^\mathsf{T}\nabla_\theta f + \frac{1}{2}\delta^\mathsf{T} C\delta = \hat{f}(\delta; C) \tag{32}$$

The curvature matrix depends on the specific optimisation method and will often be only an estimate. For notational simplicity, the dependence of $\hat{f}$ on θ is omitted. Setting C to the true Hessian matrix of f would make $\hat{f}$ the exact second-order Taylor expansion of the function around θ. However, when f is a nonlinear function, the Hessian can be indefinite, which leads to an ill-conditioned quadratic approximation $\hat{f}$. For this reason, C is usually chosen to be positive-semi definite by construction, such as the Gauss-Newton or the Fisher matrix. In the experiments discussed in the paper, C can be either the full Gauss-Newton matrix $\bar{G}$, obtained from running Conjugate Gradient as in (Martens, 2010), or a block diagonal approximation to it, denoted by $\widetilde{G}$. The analysis below is independent of whether this approximation is based on KFLR, KFRA, KFAC or if it is the exact block-diagonal part of $\bar{G}$, hence there will be no reference to a specific approximation.

Damping plays an important role in second-order optimisation methods. It can improve the numerical stability of the quadratic approximation and can also be related to trust region methods. Hence, we will introduce two damping coefficients - one for $\bar{G}$ and one for $\widetilde{G}$. In practice, an additional weight decay term is often applied to neural network models. As a result and following the presentation in (Martens & Grosse, 2015), a total of three extra parameters are introduced:

- A L_2 regularisation on θ with weight $\frac{\eta}{2}$, which implies an additive diagonal term to both $\bar{G}$ and $\widetilde{G}$
- A damping parameter τ added to the full Gauss-Newton matrix $\bar{G}$
- A separate damping parameter γ added to the approximation matrix $\widetilde{G}$

Subsequently, for notational convenience, we define $\bar{C} = \bar{G} + (\tau + \eta)I$, which is the curvature matrix obtained when using the full Gauss-Newton matrix in the quadratic approximation (32). Similarly, $\widetilde{C} = \widetilde{G} + (\gamma + \eta)I$ is the curvature matrix obtained when using any of the block-diagonal approximations.

B.1. Inverting the Approximate Curvature Matrix

The Gauss-Newton method calculates its step direction by multiplying the gradient with the inverse of the curvature matrix, in this case $\widetilde{C}$. This gives the unmodified step direction:

$$\widetilde{\delta} = \widetilde{C}^{-1}\nabla_\theta f \tag{33}$$

Since $\widetilde{C}$ is a block diagonal matrix (each block corresponds to the parameters of a single layer) the problem naturally factorises to solving L independent linear systems:

$$\widetilde{\delta}_\lambda = \widetilde{C}_\lambda^{-1}\nabla_{W_\lambda} f \tag{34}$$

For all of the approximate methods under consideration – KFLR, KFRA and KFAC – the diagonal blocks $\widetilde{G}_\lambda$ have a Kronecker factored form $\widetilde{Q}_\lambda \otimes \widetilde{\mathcal{G}}_\lambda$, where $\widetilde{Q}_\lambda = \mathbb{E}\left[Q_\lambda\right]$ and $\widetilde{\mathcal{G}}_\lambda$ denotes the approximation to $\mathbb{E}\left[\mathcal{G}_\lambda\right]$ obtained from the method of choice. Setting $k = (\eta + \gamma)$ implies:

$$\widetilde{C}_\lambda = \widetilde{Q}_\lambda \otimes \widetilde{\mathcal{G}}_\lambda + kI \otimes I \tag{35}$$

The exact calculation of (34) given the structural form of $\widetilde{C}_\lambda$ requires the eigen decomposition of both matrices $\widetilde{Q}_\lambda$ and $\widetilde{\mathcal{G}}_\lambda$, see (Martens & Grosse, 2015). However, the well known Kronecker identity $(A \otimes B)^{-1}\text{vec}\left(V\right) = A^{-1}VB^{-1}$ motivates the following approximation:

$$\widetilde{Q}_\lambda \otimes \widetilde{\mathcal{G}}_\lambda + kI \otimes I \approx \left(\widetilde{Q}_\lambda + \omega\sqrt{k}I\right) \otimes \left(\widetilde{\mathcal{G}}_\lambda + \omega^{-1}\sqrt{k}I\right) \tag{36}$$

The optimal setting of ω can be found analytically by bounding the norm of the approximation's residual, namely:

$$\begin{aligned} R(\omega) &= \widetilde{Q}_\lambda \otimes \widetilde{\mathcal{G}}_\lambda + kI \otimes I - \left(\widetilde{Q}_\lambda + \omega\sqrt{k}I\right) \otimes \left(\widetilde{\mathcal{G}}_\lambda + \omega^{-1}\sqrt{k}I\right) \\ &= -\omega^{-1}\sqrt{k}\widetilde{Q}_\lambda \otimes I - \omega\sqrt{k}\widetilde{\mathcal{G}}_\lambda \otimes I \\ ||R(\pi)|| &\leq \omega^{-1}\sqrt{k}||\widetilde{Q}_\lambda \otimes I|| + \omega\sqrt{k}||\widetilde{\mathcal{G}}_\lambda \otimes I|| \end{aligned} \tag{37}$$

Minimising the right hand side with respect to ω gives the solution

$$\omega = \sqrt{\frac{||\widetilde{Q}_\lambda \otimes I||}{||I \otimes \widetilde{\mathcal{G}}_\lambda||}} \tag{38}$$

The choice of the norm is arbitrary, but for comparative purposes with previous work on KFAC, we use the trace norm in all of our experiments. Importantly, this approach is computationally cheaper as it requires solving only two linear systems per layer, compared to an eigen decomposition and four matrix-matrix multiplications for the exact calculation. Alternatively, one can consider this approach as a special form of damping for Kronecker factored matrices.

B.2. Choosing the Step Size

The approximate block diagonal Gauss-Newton update can be significantly different from the full Gauss-Newton update. It is therefore important, in practice, to choose an appropriate step size, especially in cases where the curvature matrices are estimated from mini-batches rather than the full dataset. The step size is calculated based on the work in (Martens & Grosse, 2015), using the quadratic approximation $\hat{f}(\delta; \bar{C})$ from (32), induced by the full Gauss-Newton matrix. Given the initial step direction $\widetilde{\delta}$ from (33) a line search is performed along that direction and the resulting optimal step size is used for the final update.

$$\alpha_* = \arg\min_\alpha \hat{f}(\alpha\widetilde{\delta}; \bar{C}) = \arg\min_\alpha f(\theta) + \alpha\widetilde{\delta}^\mathsf{T}\nabla f + \frac{1}{2}\alpha^2\widetilde{\delta}^\mathsf{T}\bar{C}\widetilde{\delta} \tag{39}$$

This can be readily solved as

$$\delta_* = \alpha_*\widetilde{\delta} = -\frac{\widetilde{\delta}^\mathsf{T}\nabla f}{\widetilde{\delta}^\mathsf{T}\bar{C}\widetilde{\delta}}\widetilde{\delta} \tag{40}$$

where

$$\widetilde{\delta}^\mathsf{T}\bar{C}\widetilde{\delta} = \widetilde{\delta}^\mathsf{T}\bar{G}\widetilde{\delta} + (\tau + \eta)\widetilde{\delta}^\mathsf{T}\widetilde{\delta} \tag{41}$$

The term $\widetilde{\delta}^\mathsf{T}\bar{G}\delta$ can be calculated efficiently (without explicitly evaluating $\bar{G}$) using the R-operator (Pearlmutter, 1994). The final update of the approximate GN method is δ_*. Notably, the damping parameter γ affects the resulting update direction $\widetilde{\delta}$, whilst τ affects only the step size along that direction.

B.3. Adaptive Damping

B.3.1. τ

In order to be able to correctly adapt τ to the current curvature of the objective we use a Levenberg-Marquardt heuristic based on the reduction ratio ρ defined as

$$\rho = \frac{f(\theta + \delta_*) - f(\theta)}{\hat{f}(\delta_*; \bar{C}) - \hat{f}(0; \bar{C})} \tag{42}$$

This quantity measures how well the quadratic approximation matches the true function. When $\rho < 1$ it means that the true function f has a lower value at $\theta + \delta_*$ (and thus the quadratic underestimates the curvature), while in the other case the quadratic overestimates the curvature. The Levenberg-Marquardt method introduces the parameter $\omega_\tau < 1$. When $\rho > 0.75$ the τ parameter is multiplied by ω_τ, when $\rho < 0.75$ it is divided by ω_τ. In order for this to not introduce a significant computational overhead (as it requires an additional evaluation of the function – $f(\theta + \delta_*)$) we adapt τ only every T_τ iterations. For all experiments we used $\omega_\tau = 0.95^{T_\tau}$ and $T_\tau = 5$.

B.3.2. γ

The role of γ is to regularise the approximation of the quadratic function $\hat{f}(\delta, \widetilde{C})$ induced by the approximate Gauss-Newton to that induced by the full Gauss-Newton $\hat{f}(\delta, \bar{C})$. This is in contrast to τ, which regularises the quality of the latter approximation to the true function. γ can be related to how well the approximate curvature matrix $\widetilde{C}$ reflects the full Gauss-Newton matrix. The main reason for having two parameters is because we have two levels of approximations, and each parameter independently affects each one of them:

$$f(\theta + \delta) \overset{\tau}{\approx} \hat{f}(\delta; \bar{C}) \overset{\gamma}{\approx} \hat{f}(\delta; \widetilde{C}) \tag{43}$$

The parameter γ is updated greedily. Every T_γ iterations the algorithm computes the update δ_* for each of $\{\omega_\gamma\gamma, \gamma, \omega_\gamma^{-1}\gamma, \}$ and some scaling factor $\omega_\gamma < 1$. From these three values the one that minimises $\hat{f}(\delta; \bar{C})$ is selected. Similar to the previous section, we use $\omega_\gamma = 0.95^{T_\gamma}$ and $T_\gamma = 20$ across all experiments.

B.4. Parameter Averaging

Compared with stochastic first-order methods (for example stochastic gradient descent), stochastic second-order methods do not exhibit any implicit averaging. To address this, Martens & Grosse (2015) introduce a separate value $\widehat{\theta}_t$ which tracks the moving average of the parameter values θ_t used for training:

$$\widehat{\theta}_t = \beta_t \widehat{\theta}_{t-1} + (1 - \beta_t)\theta_t \tag{44}$$

Importantly, $\widehat{\theta}_t$ has no effect or overhead on training as it is not used for the updates on θ_t. The extra parameter β_t is chosen such that in the initial stage when t is small, $\widehat{\theta}_t$ is the exact average of the first t parameter values of θ:

$$\beta_t = \min(0.95, 1 - 1/t) \tag{45}$$

Another factor playing an important role in stochastic second-order methods is the mini-batch size m. In Martens & Grosse (2015), the authors concluded that because of the high signal to noise ratio that arises close to the minimum, in practice one should use increasingly larger batch sizes for KFAC as the optimisation proceeds. However, our work does not focus on this aspect and all of the experiments are conducted using a fixed batch size of 1000.

C. The Fisher Matrix and KFAC

C.1. The Fisher Matrix

For a general probability distribution $p_\theta(x)$ parametrised by θ, the Fisher matrix can be expressed in two equivalent forms (Martens, 2014):

$$\begin{aligned}\bar{F} &= \mathbb{E}\left[\nabla_\theta \log p_\theta(x) \nabla_\theta \log p_\theta(x)^\mathsf{T}\right]_{p_\theta(x)} \\ &= -\mathbb{E}\left[\nabla\nabla \log p_\theta(x)\right]_{p_\theta(x)}\end{aligned} \tag{46}$$

By construction the Fisher matrix is positive semi-definite. Using the Fisher matrix in place of the Hessian to form the parameter update $\bar{F}^{-1}g$ is known as Natural Gradient (Amari, 1998).

In the neural network setting, the model specifies a conditional distribution $p_\theta(y|x)$, and the Fisher matrix is given by

$$\begin{aligned}\bar{F} &= \mathbb{E}\left[\nabla_\theta \log p_\theta(x, y) \nabla_\theta \log p_\theta(x, y)^\mathsf{T}\right]_{p_\theta(x,y)} \\ &= \mathbb{E}\left[\nabla_\theta \log p_\theta(y|x) \nabla_\theta \log p_\theta(y|x)^\mathsf{T}\right]_{p_\theta(x,y)}\end{aligned} \tag{47}$$

Using the chain rule $\nabla_\theta \log p_\theta(y|x) = {J_\theta^{h_L}}^\mathsf{T} \nabla_{h_L} \log p_\theta(y|x)$ and defining

$$\mathcal{F}_L \equiv \nabla_{h_L} \log p_\theta(y|x) \nabla_{h_L} \log p_\theta(y|x)^\mathsf{T} \tag{48}$$

the Fisher can be calculated as:

$$\begin{aligned}\bar{F} &= \mathbb{E}\left[{J_\theta^{h_L}}^\mathsf{T} \nabla_{h_L} \log p_\theta(y|x) \nabla_{h_L} \log p_\theta(y|x)^\mathsf{T} J_\theta^{h_L}\right]_{p_\theta(x,y)} \\ &= \mathbb{E}\left[{J_\theta^{h_L}}^\mathsf{T} \mathcal{F}_L J_\theta^{h_L}\right]_{p_\theta(x,y)}\end{aligned} \tag{49}$$

The equation is reminiscent of (12) and in Appendix C.2 we discuss the conditions under which the Fisher and the Gauss-Newton matrix are indeed equivalent.

C.2. Equivalence between the Gauss-Newton and Fisher Matrices

The expected Gauss-Newton matrix is given by

$$\bar{G} = \mathbb{E}\left[{J_\theta^{h_L}}^\mathsf{T} \mathcal{H}_L J_\theta^{h_L}\right]_{p(x,y)} = \mathbb{E}\left[{J_\theta^{h_L}}^\mathsf{T} \mathbb{E}\left[\mathcal{H}_L\right]_{p(y|x)} J_\theta^{h_L}\right]_{p(x)} \tag{50}$$

Using that $\mathbb{E}[\mathcal{F}_L] = \mathbb{E}[\mathcal{H}_L]$ which follows from (46) and the fact that the Jacobian $J_\theta^{h_L}$ is independent of y, the Fisher matrix can be expressed as:

$$\bar{F} = \mathbb{E}\left[J_\theta^{h_L}{}^\intercal \mathcal{F}_L J_\theta^{h_L}\right]_{p_\theta(x,y)} = \mathbb{E}\left[J_\theta^{h_L}{}^\intercal \mathbb{E}[\mathcal{F}_L]_{p_\theta(y|x)} J_\theta^{h_L}\right]_{p(x)} = \mathbb{E}\left[J_\theta^{h_L}{}^\intercal \mathbb{E}[\mathcal{H}_L]_{p_\theta(y|x)} J_\theta^{h_L}\right]_{p(x)} \tag{51}$$

Hence the Fisher and Gauss-Newton matrices matrices are equivalent when $\mathbb{E}[\mathcal{H}_L]_{p(y|x)} = \mathbb{E}[\mathcal{H}_L]_{p_\theta(y|x)}$. Since the model distribution $p_\theta(y|x)$ and the true data distribution $p(y|x)$ are not equal, a sufficient condition for the expectations to be equal is $\mathcal{H}_L$ being independent of y. Although this might appear restrictive, if h_L parametrises the natural parameters of an exponential family distribution this independence holds (Martens, 2014). To show this, consider

$$\log p(y|x,\theta) = \log h(y) + T(y)^\intercal \eta(x,\theta) - \log Z(x,\theta) = \log h(y) + T(y)^\intercal h_L - \log Z(h_L) \tag{52}$$

where h is the base measure, T is the sufficient statistic, Z is the partition function and η are the natural parameters. Taking the gradient of the log-likelihood with respect to h_L

$$\nabla_{h_L} \log p(y|x,\theta) = T(y) - \nabla_{h_L} \log Z(h_L) \tag{53}$$

Assuming that the objective is the negative log-likelihood as in Section 2.1 and differentiating again

$$\mathcal{H}_L = \nabla\nabla_{h_L} \log Z(h_L) \tag{54}$$

which is indeed independent of y. This demonstrates that in many practical supervised learning problems in Machine Learning, the Gauss-Newton and Natural Gradient methods are equivalent.

The parameter update for these approaches is then given by computing $\bar{G}^{-1}g$ or $\bar{F}^{-1}g$, where g is the gradient of the objective with respect to all parameters. However, the size of the linear systems is prohibitively large in the case of neural networks, thus it is computationally infeasible to solve them exactly. As shown in (Schraudolph, 2002), matrix-vector products with $\bar{G}$ can be computed efficiently using the R-operator (Pearlmutter, 1994). The method does not need to compute $\bar{G}$ explicitly, at the cost of approximately twice the computation time of a standard backward pass. This makes it suitable for iterative methods such as conjugate gradient for approximately solving the linear system. The resulting method is called 'Hessian-free', with promising results in deep feedforward and recurrent neural networks (Martens, 2010; Martens & Sutskever, 2011). Nevertheless, the convergence of the conjugate gradient step may require many iterations by itself, which can significantly increase the computation overhead compared to standard methods. As a result, this approach can have worse performance per clock time compared to a well-tuned first-order method (Sutskever et al., 2013). This motivates the usage of approximate Gauss-Newton methods instead.

C.3. The Fisher Approximation to $\mathbb{E}[\mathcal{G}_\lambda]$ and KFAC

The key idea in this approach is to use the fact that the Fisher matrix is an expectation of the outer product of gradients and that it is equal to the Gauss-Newton matrix (Section C.2). This is independent of whether the Gauss-Newton matrix is with respect to W_λ or h_λ, so we can write the pre-activation Gauss-Newton as

$$\begin{aligned}
\mathbb{E}[\mathcal{G}_\lambda]_{p(x,y)} &= \mathbb{E}\left[J_{h_\lambda}^{h_L}{}^\intercal \mathcal{H}_L J_{h_\lambda}^{h_L}\right]_{p(x)} && (55)\\
&= \mathbb{E}\left[J_{h_\lambda}^{h_L}{}^\intercal \mathbb{E}[\mathcal{F}_L]_{p_\theta(y|x)} J_{h_\lambda}^{h_L}\right]_{p(x)} && (56)\\
&= \mathbb{E}\left[J_{h_\lambda}^{h_L}{}^\intercal \mathcal{F}_L J_{h_\lambda}^{h_L}\right]_{p_\theta(x,y)} && (57)\\
&= \mathbb{E}\left[J_{h_\lambda}^{h_L}{}^\intercal \nabla_{h_L} \log p_\theta(y|x) \nabla_{h_L} \log p_\theta(y|x)^\intercal J_{h_\lambda}^{h_L}\right]_{p_\theta(x,y)} && (58)\\
&= \mathbb{E}\left[\nabla_{h_\lambda} \log p_\theta(y|x) \nabla_{h_\lambda} \log p_\theta(y|x)^\intercal\right]_{p_\theta(x,y)} && (59)
\end{aligned}$$

where the first equality follows from (54) and the second one from (51) in the supplement.

We stress here that the resulting expectation is over the model distribution $p_\theta(x, y)$ and not the data distribution $p(x, y)$. In order to approximate (59) the method proceeds by taking Monte Carlo samples of the gradients from the model conditional distribution $p_\theta(y|x)$.

The KFAC approximation presented in (Martens & Grosse, 2015) is analogous to the above approach, but it is derived in a different way. The authors directly focus on the parameter Fisher matrix. Using the fact that $J^{h_\lambda}_{W_\lambda} = a^\mathsf{T}_{\lambda-1} \otimes I$ and ${J^{h_\lambda}_{W_\lambda}}^\mathsf{T} v = a_{\lambda-1} \otimes v$, the blocks of the Fisher matrix become:

$$\left[\bar{F}\right]_{\lambda,\beta} = \mathbb{E}\left[\nabla_{W_\lambda} \log p_\theta(y|x) \nabla_{W_\beta} \log p_\theta(y|x)^\mathsf{T}\right]_{p_\theta(x,y)} \tag{60}$$

$$= \mathbb{E}\left[\left(a_{\lambda-1} \otimes \nabla_{h_\lambda} \log p_\theta(y|x)\right)\left(a_{\beta-1} \otimes \nabla_{h_\beta} \log p_\theta(y|x)\right)^\mathsf{T}\right]_{p_\theta(x,y)} \tag{61}$$

$$= \mathbb{E}\left[\left(a_{\lambda-1} a^\mathsf{T}_{\beta-1}\right) \otimes \left(\nabla_{h_\lambda} \log p_\theta(y|x) \nabla_{h_\beta} \log p_\theta(y|x)^\mathsf{T}\right)\right]_{p_\theta(x,y)} \tag{62}$$

This equation is equivalent to our result in (16) [13]. In (Martens & Grosse, 2015) the authors similarly approximate the expectation of the Kronecker products by the product of the individual expectations, which makes the second term equal to the pre-activation Gauss-Newton as in (59).

C.4. Differences between KFAC and KFRA

It is useful to understand the computational complexity of both KFAC and KFRA and the implications of the approximations. In order to not make any additional assumptions about the underlying hardware or mode (serial or parallel) of execution, we denote with $O_{mm}(m, n, p)$ the complexity of a matrix matrix multiplication of an $m \times n$ and $n \times p$ matrices and with $O_{el}(m, n)$ the complexity of an elementwise multiplication of two matrices of size $m \times n$.

KFRA We need to backpropagate the matrix $\mathbb{E}\left[\mathcal{G}_\lambda\right]$ of size $D_\lambda \times D_\lambda$, where D_λ is the dimensionality of the layer. For each layer, this requires two matrix-matrix multiplications with W_λ and single element wise multiplication (this is due to A_λ being diagonal, which allows for such a simplification). The overall complexity of the procedure is $2O_{mm}(D_\lambda, D_\lambda, D_{\lambda-1}) + O_{el}(D_{\lambda-1}, D_{\lambda-1})$.

KFAC We need to draw samples from $p_\theta(y|x)$ for each datapoint x and backprogate the corresponding gradients through the network (this is in addition to the gradients of the objective function). This requires backpropagating a matrix of size $D_{\lambda-1} \times NS$, where S denotes the number of samples taken per datapoint. Per layer, the method requires also two matrix-matrix multiplications (one with W_λ and the outer product of C^s_λ) and a single element wise multiplication. The overall complexity of the procedure is $O_{mm}(NS, D_\lambda, D_{\lambda-1}) + O_{el}(NS, D_{\lambda-1}) + O_{mm}(D_{\lambda-1}, NS, D_{\lambda-1})$.

There are several observations which we deem important. Firstly, if $N = 1$, KFRA is no longer an approximate method, but computes the exact $\mathcal{G}_\lambda$ matrix. Secondly, if $S = 1$ and $D_\lambda \sim N$ then the two methods have similar computational complexity. If we assume that the complexity scales linearly in S, in the extreme case of $S = N$ and $D_\lambda \sim N$, it is possible to execute KFRA independently for each datapoint providing the exact value $\mathcal{G}_\lambda$ for the same computational cost, while KFAC would nevertheless still be an approximation.

D. The Rank of the Monte Carlo Gauss-Newton

Using the definition of the sample Gauss-Newton matrix in (12) we can infer that its rank is bounded by the rank of $\mathcal{H}_L$:

$$G \equiv {J^{h_L}_\theta}^\mathsf{T} \mathcal{H}_L J^{h_L}_\theta \Rightarrow \mathbf{rank}(G) \leq \mathbf{rank}(\mathcal{H}_L) \tag{63}$$

This does not provide any bound on the rank of the "true" Gauss-Newton matrix, which is an expectation of the above. However, for any practical algorithm which approximates the expectations via N Monte Carlo samples as:

$$\bar{G} = \mathbb{E}\left[G\right] \approx \frac{1}{N} \sum_n G_n \tag{64}$$

[13]Under the condition that the Fisher and Gauss-Newton matrices are equivalent, see Section C.2

it provides a bound on the rank of the resulting matrix. Using the sub-additive property of the rank operator, it follows that $\mathbf{rank}(\frac{1}{N}\sum_n G_n) \leq \mathbf{rank}(\mathcal{H}_L)N$. Similarly, the approximate Fisher matrix computed in KFAC will have a rank bounded by NS, where S is the number of samples taken per data point (usually one). This provides an explanation for the results in Section 5.2 for binary classification, since the last layer output in this problem is a scalar, thus its rank is 1. Hence, both the Gauss-Newton and the Fisher for a mini-batch have a rank bounded by the mini-batch size N. This leads to the conclusion that in such a situation the curvature information provided from a Monte-Carlo estimate is not sufficient to render the approximate Gauss-Newton methods competitive against well-tuned first order methods, although we observe that in the initial stages they are still better. In some of the more recent works on KFAC the authors use momentum terms in conjunction with the second-order updates or do not rescale by the full Gauss-Newton. This leaves space for exploration and more in depth research on developing techniques that can robustly and consistently improve the performance of second-order methods for models with a small number of outputs and small batch sizes.

E. Absence of Smooth Local Maxima for Piecewise Linear Transfer Functions

In order to show that the Hessian of a neural network with piecewise linear transfer functions can have no differentiable strict local maxima, we first establish that all of its diagonal blocks are positive semi-definite.

Lemma 1. *Consider a neural network as defined in (1). If the second derivative of all transfer functions f_λ for $1 \leq \lambda \leq L$ is zero where defined, and if the Hessian of the error function w.r.t. the outputs of the network is positive semi-definite, then all blocks*

$$H_\lambda = \frac{\partial^2 E}{\partial vec\,(W_\lambda) \partial vec\,(W_\lambda)} \tag{65}$$

on the diagonal of the Hessian — corresponding to the weights W_λ of single layer — are positive semi-definite.

Proof. By the definition in (10) $D^\lambda_{i,j} = \delta_{i,j} f''(h^\lambda_{i,j})$. From the assumption of the lemma, $f'' = 0$ for all layers, hence $\forall\lambda \quad D_\lambda = 0$. Using the recursive equation (8) we can analyze the quadratic form $v^\mathsf{T}\mathcal{H}_\lambda v$:

$$\begin{aligned} \mathcal{H}_\lambda &= B_\lambda W_{\lambda+1}^\mathsf{T} \mathcal{H}_{\lambda+1} W_{\lambda+1} B_\lambda + D_\lambda \\ &= \left(W_{\lambda+1} B_\lambda\right)^\mathsf{T} \mathcal{H}_{\lambda+1} \left(W_{\lambda+1} B_\lambda\right) \end{aligned} \tag{66}$$

where we used the fact that by definition B_λ is a diagonal matrix, thus it is symmetric and $B_\lambda = B_\lambda^\mathsf{T}$. Defining

$$\tilde{v} = W_{\lambda+1} B_\lambda v \tag{67}$$

yields

$$\begin{aligned} v^\mathsf{T}\mathcal{H}_\lambda v &= \left(W_{\lambda+1} B_\lambda v\right)^\mathsf{T} \mathcal{H}_{\lambda+1} \left(W_{\lambda+1} B_\lambda v\right) \\ &= \tilde{v}^\mathsf{T} \mathcal{H}_{\lambda+1} \tilde{v} \end{aligned} \tag{68}$$

hence

$$\mathcal{H}_{\lambda+1} \geq 0 \Rightarrow \mathcal{H}_\lambda \geq 0 \tag{69}$$

It follows by induction that if $\mathcal{H}_L$ is positive semi-definite, all of the pre-activation Hessian matrices are positive semi-definite as well.

Using the proof that the blocks H_λ can be written as a Kronecker product in (7), we can analyze the quadratic form of the Hessian block diagonals:

$$\begin{aligned} \text{vec}\,(V)^\mathsf{T} H_\lambda \text{vec}\,(V) &= \text{vec}\,(V)^\mathsf{T} \left[\left(a_\lambda a_\lambda^\mathsf{T}\right) \otimes \mathcal{H}_\lambda\right] \text{vec}\,(V) \\ &= \text{vec}\,(V)^\mathsf{T} \text{vec}\left(\mathcal{H}_\lambda V a_\lambda a_\lambda^\mathsf{T}\right) \\ &= \text{trace}\left(V^\mathsf{T} \mathcal{H}_\lambda V a_\lambda a_\lambda^\mathsf{T}\right) \\ &= \text{trace}\left(a_\lambda^\mathsf{T} V^\mathsf{T} \mathcal{H}_\lambda V a_\lambda\right) \\ &= \left(V a_\lambda\right)^\mathsf{T} \mathcal{H}_\lambda \left(V a_\lambda\right) \end{aligned} \tag{70}$$

$$\mathcal{H}_\lambda \geq 0 \Rightarrow H_\lambda \geq 0$$

The second line follows from the well known identity $(A \otimes B)\text{vec}(V) = \text{vec}\left(BVA^{\mathsf{T}}\right)$. Similarly, the third line follows from the fact that $\text{vec}(A)^{\mathsf{T}}\text{vec}(B) = \mathrm{trace}\left(A^{\mathsf{T}}B\right)$. The fourth line uses the fact that $\mathrm{trace}(AB) = \mathrm{trace}(BA)$ when the product AB is a square matrix. This concludes the proof of the lemma.

□

This lemma has two implications:

- If we fix the weights of all layers but one, the error function becomes *locally* convex, wherever the second derivatives of all transfer functions in the network are defined.
- The error function can have no differentiable strict local maxima.

We formalise the proof of the second proposition below:

Lemma 2. *Under the same assumptions as in Lemma 1, the overall objective function E has no differentiable strict local maxima with respect to the parameters θ of the network.*

Proof. For a point to be a strict local maximum, all eigenvalues of the Hessian at this location would need to be simultaneously negative. However, as the trace of a matrix is equal to the sum of the eigenvalues it is sufficient to prove that $\mathrm{trace}(H) \geq 0$.

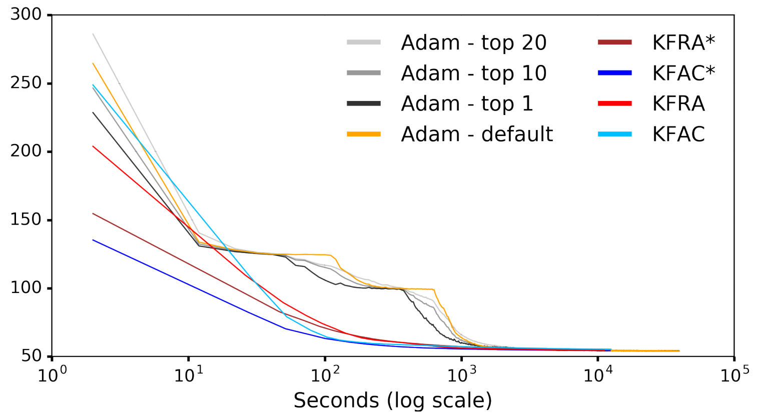
The trace of the full Hessian matrix is equal to the sum of the diagonal elements, so it is also equal to the sum of the traces of the diagonal blocks. Under the assumptions in Lemma 1, we showed that all of the diagonal blocks are positive semi-definite, hence their traces are non-negative. It immediately follows that:

$$\mathrm{trace}(H) = \sum_{\lambda=1}^{L} \mathrm{trace}(H_\lambda) \geq 0 \tag{71}$$

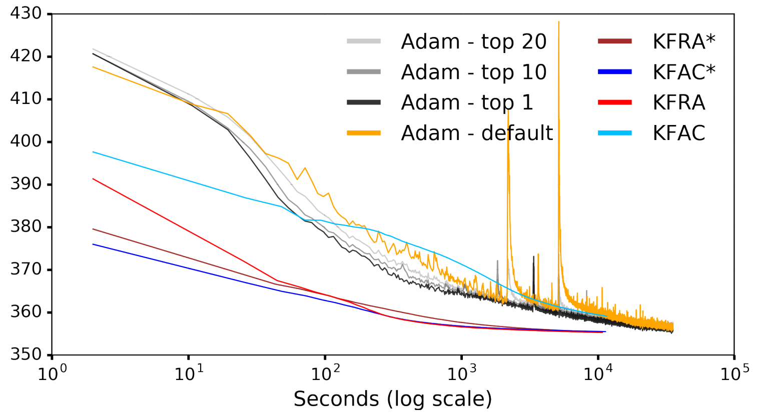
Therefore, it is impossible for all eigenvalues of the Hessian to be simultaneously negative. As a corollary it follows that all strict local maxima must lie in the non-differentiable boundary points of the nonlinear transfer functions. □

F. Additional Figures

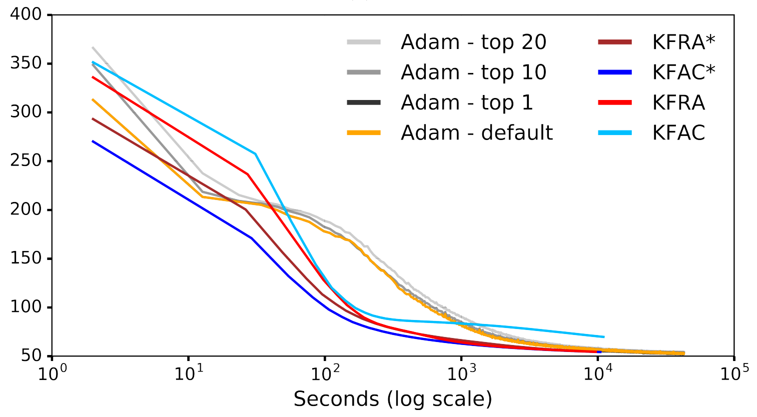
F.1. CPU Benchmarks



(a) CURVES



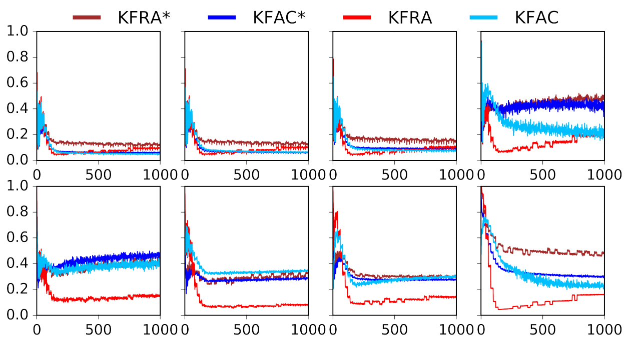
(b) FACES



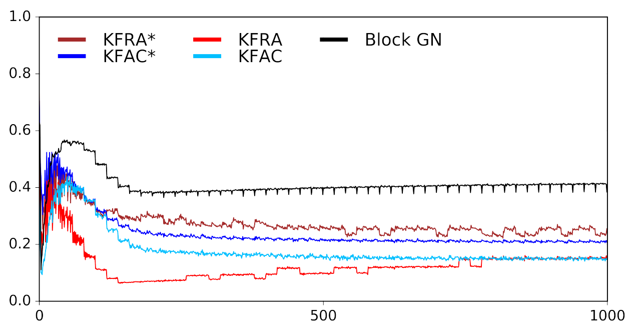
(c) MNIST

Figure 4. Optimisation performance on the CPU. These timings are obtained with a previous implementation in Arrayfire, different to the one used for the figures in the main text. For the second-order methods, the asterisk indicates the use of the approximate inversion as described in Section B.1. The error function on all three datasets is binary cross-entropy.

F.2. Comparison of the Alignment of the Approximate Updates with the Gauss-Newton Update

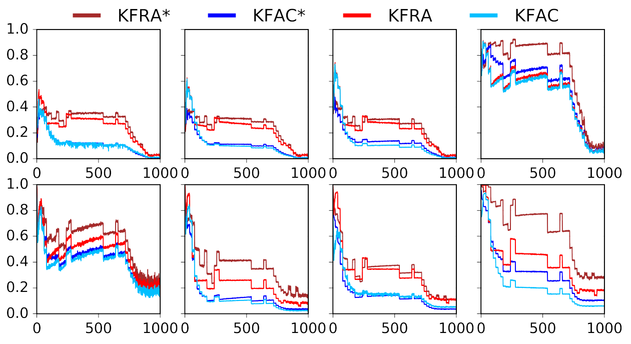


(a) Block-diagonal Gauss-Newton

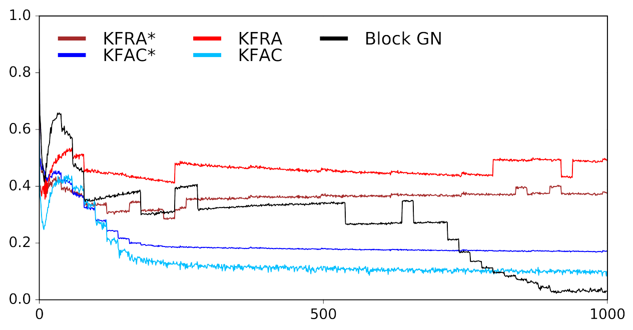


(b) Full Gauss-Newton

Figure 5. CURVES: Cosine similarity between the update vector per layer, given by the corresponding approximate method, $\widetilde{\delta}_\lambda$ with that for the block-diagonal GN (a) and the full vector with that from the full GN matrix (b). The optimal value is 1.0. The * indicates approximate inversion in (36). The x-axis is the number of iterations. Layers one to four are in the top; five to eight in the bottom row. The trajectory of parameters we follow is the one generated by KFRA*.



(a) Block-diagonal Gauss-Newton



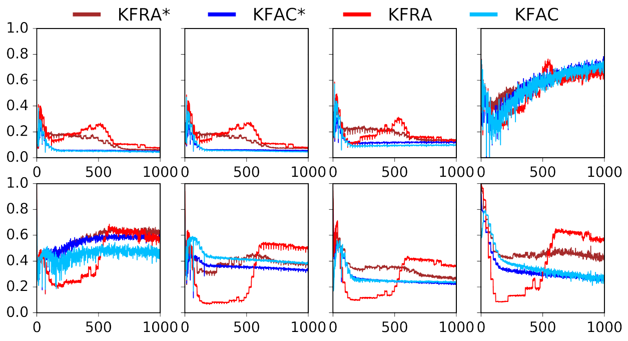
(b) Full Gauss-Newton

Figure 6. FACES: Cosine similarity between the update vector per layer, given by the corresponding approximate method, $\widetilde{\delta}_\lambda$ with that for the block-diagonal GN (a) and the full vector with that from the full GN matrix (b). The optimal value is 1.0. The * indicates approximate inversion in (36). The x-axis is the number of iterations. Layers one to four are in the top; five to eight in the bottom row. The trajectory of parameters we follow is the one generated by KFRA*.

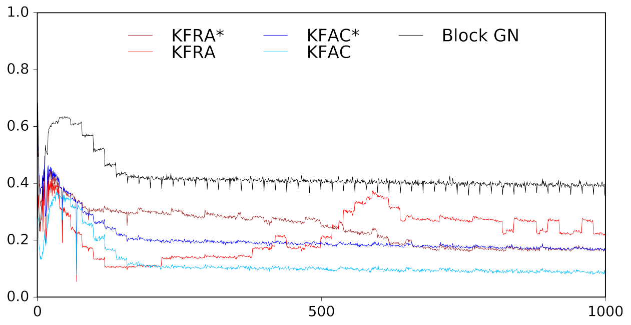
To gain insight into the quality of the approximations that are made in the second-order methods under consideration, we compare how well the KFAC and KFRA parameter updates $\widetilde{\delta}$ are aligned with updates obtained from using the regularised block diagonal GN and the full GN matrix. Additionally we check how using the approximate inversion of the Kronecker factored curvature matrices discussed in Appendix B impacts the alignment.

In order to find the updates for the full GN method we use conjugate gradients and the R-operator and solve the linear system $\bar{G}\delta = \nabla_\theta f$ as in (Martens, 2010). For the block diagonal GN method we use the same strategy, however the method is applied independently for each separate layer of the network, see Appendix B.

We compared the different approaches for batch sizes of 250, 500 and 1000. However, the results did not differ significantly. We therefore show results only for a batch size of 1000. In Figures 5 to 7, subfigure 6a plots the cosine similarity between the update vector $\widetilde{\delta}_\lambda$ for a specific layer, given by the corresponding approximate method, and the update vector when using the block diagonal GN matrix on CURVES, FACES and MNIST. Throughout the optimisation, compared to KFAC, the KFRA update has better alignment with the exact GN update. Subfigure 6b shows the same similarity for the whole update vector $\widetilde{\delta}$, however in comparison with the update vector given by the full GN update. Additionally, we also show the similarity between the update vector of the block diagonal GN and the full GN approach in those plots. There is a decay in the alignment between the block-diagonal and full GN updates towards the end of training on FACES, however this is most likely just due to the conjugate gradients being badly conditioned and is not observed on the other two datasets.



(a) Block-diagonal Gauss-Newton



(b) Full Gauss-Newton

Figure 7. MNIST: Cosine similarity between the update vector per layer, given by the corresponding approximate method, $\widetilde{\delta}_\lambda$ with that for the block-diagonal GN (a) and the full vector with that from the full GN matrix (b). The optimal value is 1.0. The * indicates approximate inversion in (36). The x-axis is the number of iterations. Layers one to four are in the top; five to eight in the bottom row. The trajectory of parameters we follow is the one generated by KFRA*.

After observing that KFRA generally outperforms KFAC, it is not surprising to see that its updates are beter aligned with both the block diagonal and the full GN matrix.

Considering that (for exponential family models) both methods differ only in how they approximate the expected GN matrix, gaining a better understanding of the impact of the separate approximations on the optimisation performance could lead to an improved algorithm.

G. Algorithm for a Single Backward Pass

Algorithm 1 Algorithm for KFRA parameter updates excluding heuristic updates for η and γ

```
**Input:** minibatch $X$, weight matrices $W_{1:L}$, transfer functions $f_{1:L}$, true outputs $Y$, parameters $\eta$ and $\gamma$
*- Forward Pass -*
$a_0 = X$
**for** $\lambda = 1$ **to** L **do**
    $h_\lambda = W_\lambda a_{\lambda-1}$
    $a_\lambda = f_\lambda(h_\lambda)$
**end for**
*- Derivative and Hessian of the objective -*
$d_L = \frac{\partial E}{\partial h_L}\Big|_{h_L}$
$\widetilde{\mathcal{G}}_{L\lambda} = \mathbb{E}\left[\mathcal{H}_L\right]\Big|_{h_L}$
*- Backward pass -*
**for** $\lambda = L$ **to** $1$ **do**
    *- Update for $W_\lambda$ -*
    $g_\lambda = \frac{1}{N} d_\lambda a_{\lambda-1}^\mathsf{T} + \eta W_\lambda$
    $\widetilde{Q}_\lambda = \frac{1}{N} a_{\lambda-1} a_{\lambda-1}^\mathsf{T}$
    $\omega = \sqrt{\frac{Tr(\widetilde{Q}_\lambda) * dim(\widetilde{\mathcal{G}}_\lambda)}{Tr(\widetilde{\mathcal{G}}_\lambda) * dim(\widetilde{Q}_\lambda)}}$
    $k = \sqrt{\gamma + \eta}$
    $\widetilde{\delta}_\lambda = (\widetilde{Q}_\lambda + \omega k)^{-1} g_\lambda (\widetilde{\mathcal{G}}_\lambda + \omega^{-1} k)^{-1}$
    **if** $\lambda > 1$ **then**
        *- Propagate gradient and approximate pre-activation Gauss-Newton -*
        $A_{\lambda-1} = f'(h_{\lambda-1})$
        $d_{\lambda-1} = W_\lambda^\mathsf{T} d_\lambda \odot A_{\lambda-1}$
        $\widetilde{\mathcal{G}}_{\lambda-1} = (W_\lambda^\mathsf{T} \widetilde{\mathcal{G}}_\lambda W_\lambda) \odot \left(\frac{1}{N} A_{\lambda-1} A_{\lambda-1}^\mathsf{T}\right)$
    **end if**
**end for**
$v = J_\theta^{h_L} \widetilde{\delta}$        (using the R-op from (Pearlmutter, 1994))
$\widetilde{\delta}^\mathsf{T} \bar{G} \widetilde{\delta} = v^\mathsf{T} \mathcal{H}_L v$
$\widetilde{\delta}^\mathsf{T} \bar{C} \widetilde{\delta} = \widetilde{\delta}^\mathsf{T} \bar{G} \widetilde{\delta} + (\tau + \eta) ||\widetilde{\delta}||_2^2$
$\alpha_* = -\frac{\widetilde{\delta}^\mathsf{T} \nabla f}{\widetilde{\delta}^\mathsf{T} \bar{C} \widetilde{\delta}}$
$\delta_* = \alpha_* \widetilde{\delta}$
**for** $\lambda = 1$ **to** $L$ **do**
    $W_\lambda = W_\lambda + \delta_{*\lambda}$
**end for**
```