A. Conv net notation and GPU implementation

For modern large-scale vision applications, it's necessary to implement conv nets efficiently for a GPU (or some other massively parallel computing architecture). We provide a very brief overview of the low-level efficiency issues which are relevant to K-FAC. We base our discussion on the Toronto Deep Learning ConvNet (TDLCN) package (Srivastava, 2015), whose convolution kernels we use in our experiments. Like many modern implementations, this implementation follows the approach of Chellapilla et al. (2006), which reduces the convolution operations to large matrix-vector products in order to exploit memory locality and efficient parallel BLAS operators. We describe the implementation explicitly, as it is important that our proposed algorithm be efficient using the same memory layout (shuffling operations are extremely expensive). As a bonus, these vectorized operations provide a convenient high-level notation which we will use throughout the paper.

The ordering of arrays in memory is significant, as it determines which operations can be performed efficiently without requiring (very expensive) transpose operations. The activations are stored as a $M \times |\mathcal{T}| \times J$ array $\tilde{\mathbf{A}}_{\ell-1}$, where M is the mini-batch size, $|\mathcal{T}|$ is the number of spatial locations, and J is the number of feature maps.[7] This can be interpreted as an $M|\mathcal{T}| \times J$ matrix. (We must assign orderings to $\mathcal{T}$ and Δ, but this choice is arbitrary.) Similarly, the weights are stored as an $I \times |\Delta| \times J$ array $\mathbf{W}_\ell$, which can be interpreted either as an $I \times |\Delta|J$ matrix or a $I|\Delta| \times J$ matrix without reshuffling elements in memory. We will almost always use the former interpretation, which we denote $\mathbf{W}_\ell$; the $I|\Delta| \times J$ matrix will be denoted $\breve{\mathbf{W}}_\ell$.

The naive implementation of convolution, while highly parallel in principle, suffers from poor memory locality. Instead, efficient implementations typically use what we will term the *expansion operator* and denote $[\![\cdot]\!]$. This operator extracts the patches surrounding each spatial location and flattens them into vectors. These vectors become the rows of a matrix. For instance, $[\![\tilde{\mathbf{A}}_{\ell-1}]\!]$ is a $M|\mathcal{T}| \times J|\Delta|$ matrix, defined as

$$[\![\tilde{\mathbf{A}}_{\ell-1}]\!]_{tM+m,\, j|\Delta|+\delta} = [\tilde{\mathbf{A}}_{\ell-1}]_{(t+\delta)M+m,\, j} = a_{j,t+\delta}^{(m)}, \tag{25}$$

for all entries such that $t + \delta \in \mathcal{T}$. All other entries are defined to be 0. Here, m indexes the data instance within the mini-batch.

In TDLCN, the forward pass is computed as

$$\tilde{\mathbf{A}}_\ell = \phi(\tilde{\mathbf{S}}_\ell) = \phi\left([\![\tilde{\mathbf{A}}_{\ell-1}]\!]\mathbf{W}_\ell^\top + \mathbf{1}\mathbf{b}_\ell^\top\right), \tag{26}$$

where ϕ is the nonlinearity, applied elementwise, $\mathbf{1}$ is a vector of ones, and $\mathbf{b}$ is the vector of biases. In backpropagation, the activation derivatives are computed as:

$$\mathcal{D}\tilde{\mathbf{A}}_{\ell-1} = [\![\mathcal{D}\tilde{\mathbf{S}}_\ell]\!]\breve{\mathbf{W}}_\ell. \tag{27}$$

Finally, the gradient for the weights is computed as

$$\mathcal{D}\mathbf{W}_\ell = \mathcal{D}\tilde{\mathbf{S}}_\ell^\top [\![\tilde{\mathbf{A}}_{\ell-1}]\!] \tag{28}$$

The matrix products are computed using the cuBLAS function `cublasSgemm`. In practice, the expanded matrix $[\![\tilde{\mathbf{A}}_{\ell-1}]\!]$ may be too large to store in memory. In this case, a subset of the rows of $[\![\tilde{\mathbf{A}}_{\ell-1}]\!]$ are computed and processed at a time.

We will also use the $|\mathcal{T}| \times J$ matrix $\mathbf{A}_{\ell-1}$ and the $|\mathcal{T}| \times I$ matrix $\mathbf{S}_\ell$ to denote the activations and pre-activations for a single training case. $\mathbf{A}_{\ell-1}$ and $\mathbf{S}_\ell$ can be substituted for $\tilde{\mathbf{A}}_{\ell-1}$ and $\tilde{\mathbf{S}}_\ell$ in Eqns. 26-28.

For fully connected networks, it is often convenient to append a homogeneous coordinate to the activations so that the biases can be folded into the weights (see Section 2.2). For convolutional layers, there is no obvious way to add extra activations such that the convolution operation simulates the effect of biases. However, we can achieve an analogous effect by adding a homogeneous coordinate (i.e. a column of all 1's) to the *expanded* activations. We will denote this $[\![\tilde{\mathbf{A}}_{\ell-1}]\!]_H$. Similarly, we can prepend the bias vector to the weights matrix: $\bar{\mathbf{W}}_\ell = (\mathbf{b}_\ell\ \mathbf{W}_\ell)$. The homogeneous coordinate is not typically used in conv net implementations, but it will be convenient for us notationally. For instance, the forward pass can be written as:

$$\tilde{\mathbf{A}}_\ell = \phi\left([\![\tilde{\mathbf{A}}_{\ell-1}]\!]_H \bar{\mathbf{W}}_\ell^\top\right) \tag{29}$$

Table 1 summarizes all of the conv net notation used in this paper.

B. Optimization methods

B.1. KFC as a preconditioner for SGD

The first optimization procedure we used in our experiments was a generic natural gradient descent approximation, where $\hat{\mathbf{F}}^{(\gamma)}$ was used to approximate $\mathbf{F}$. This procedure is like SGD with momentum, except that $\hat{\nabla}h$ is substituted for the Euclidean gradient. One can also view this as a preconditioned SGD method, where $\hat{\mathbf{F}}^{(\gamma)}$ is used as the preconditioner. To distinguish this optimization procedure from the KFC approximation itself, we refer to it as KFC-pre. Our procedure is perhaps more closely analogous to earlier Kronecker product-based natural gradient approximations (Heskes, 2000; Povey et al., 2015) than to K-FAC itself.

In addition, we used a variant of gradient clipping (Pascanu et al., 2013) to avoid instability. In particular, we clipped the approximate natural gradient update $\mathbf{v}$ so that

[7] The first index of the array is the least significant in memory.

j	input map index
J	number of input maps
i	output map index
I	number of output maps
$T_1 \times T_2$	feature map dimension
t	spatial location index
$\mathcal{T}$	set of spatial locations $= \{1, \ldots, T_1\} \times \{1, \ldots, T_2\}$
R	radius of filters
δ	spatial offset
Δ	set of spatial offsets (in a filter) $= \{-R, \ldots, R\} \times \{-R, \ldots, R\}$
$\delta = (\delta_1, \delta_2)$	explicit 2-D parameterization (δ_1 and δ_2 run from $-R$ to R)
$a_{j,t}$	input layer activations
$s_{i,t}$	output layer pre-activations
$\mathcal{D}s_{i,t}$	the loss derivative $\partial\mathcal{L}/\partial s_{i,t}$
ϕ	activation function (nonlinearity)
$w_{i,j,\delta}$	weights
b_i	biases
$M(j)$	mean activation
$\Omega(j, j', \delta)$	uncentered autocovariance of activations
$\Gamma(i, i', \delta)$	autocovariance of pre-activation derivatives
$\beta(\delta, \delta')$	function defined in Theorem 1

$\otimes$	Kronecker product
vec	Kronecker vector operator
ℓ	layer index
L	number of layers
M	size of a mini-batch
$\mathbf{A}_\ell$	activations for a data instance
$\tilde{\mathbf{A}}_\ell$	activations for a mini-batch
$[\![\mathbf{A}_\ell]\!]$	expanded activations
$[\![\mathbf{A}_\ell]\!]_H$	expanded activations with homogeneous coordinate
$\mathbf{S}_\ell$	pre-activations for a data instance
$\tilde{\mathbf{S}}_\ell$	pre-activations for a mini-batch
$\mathcal{D}\mathbf{S}_\ell$	the loss gradient $\nabla_{\mathbf{S}_\ell}\mathcal{L}$
$\boldsymbol{\theta}$	vector of trainable parameters
$\mathbf{W}_\ell$	weight matrix
$\mathbf{b}_\ell$	bias vector
$\bar{\mathbf{W}}_\ell$	combined parameters = $(\mathbf{b}_\ell \ \mathbf{W}_\ell)$
$\mathbf{F}$	exact Fisher matrix
$\hat{\mathbf{F}}$	approximate Fisher matrix
$\hat{\mathbf{F}}_\ell$	diagonal block of $\hat{\mathbf{F}}$ for layer ℓ
$\boldsymbol{\Omega}_\ell$	Kronecker factor for activations
$\boldsymbol{\Gamma}_\ell$	Kronecker factor for derivatives
λ	weight decay parameter
γ	damping parameter
$\hat{\mathbf{F}}^{(\gamma)}$	damped approximate Fisher matrix
$\boldsymbol{\Omega}_\ell^{(\gamma)}, \boldsymbol{\Gamma}_\ell^{(\gamma)}$	damped Kronecker factors

Table 1. Summary of convolutional network notation used in this paper. The left column focuses on a single convolution layer, which convolves its "input layer" activations with a set of filters to produce the pre-activations for the "output layer." Layer indices are omitted for clarity. The right column considers the network as a whole, and therefore includes explicit layer indices.

$\nu \triangleq \mathbf{v}^\top \mathbf{F} \mathbf{v} < 0.3$, where $\mathbf{F}$ is estimated using 1/4 of the training examples from the current mini-batch. One motivation for this heuristic is that ν approximates the KL divergence of the predictive distributions before and after the update, and one wouldn't like the predictive distributions to change too rapidly. The value ν can be computed using curvature-vector products (Schraudolph, 2002). The clipping was only triggered near the beginning of optimization, where the parameters (and hence also the curvature) tended to change rapidly.[8] Therefore, one can likely eliminate this step by initializing from a model partially trained using SGD.

Taking inspiration from Polyak averaging (Polyak & Juditsky, 1992; Swersky et al., 2010), we used an exponential moving average of the iterates. This helps to smooth out the variability caused by the mini-batch selection. The full optimization procedure is given in Algorithm 1.

B.2. Kronecker-factored approximate curvature

The central idea of K-FAC is the combination of approximations to the Fisher matrix described in Section 2.2. While one could potentially perform standard natural gradient descent using the approximate natural gradient $\hat{\nabla} h$, perhaps with a fixed learning rate and with fixed Tikhonov-style damping/reglarization, Martens & Grosse (2015) found that the most effective way to use $\hat{\nabla} h$ was within a robust 2nd-order optimization framework based on adaptively damped quadratic models, similar to the one employed in HF (Martens, 2010). In this section, we describe the K-FAC method in detail, while omitting certain aspects of the method which we do not use, such as the block tri-diagonal inverse approximation.

K-FAC uses a quadratic model of the objective to dynamically choose the step size α and momentum decay parameter μ at each step. This is done by taking $\mathbf{v} = \alpha\hat{\nabla} h + \mu \mathbf{v}_{prev}$ where $\mathbf{v}_{prev}$ is the update computed at the previous iteration, and minimizing the following quadratic model of the objective (over the current mini-batch):

$$M(\boldsymbol{\theta} + \mathbf{v}) = h(\boldsymbol{\theta}) + \nabla h^\top \mathbf{v} + \frac{1}{2}\mathbf{v}^\top(\mathbf{F} + r\mathbf{I})\mathbf{v}. \quad (30)$$

where we assume the h is the expected loss plus an ℓ_2-regularization term of the form $\frac{r}{2}\|\boldsymbol{\theta}\|^2$. Since $\mathbf{F}$ behaves like a curvature matrix, this quadratic function is similar to the second-order Taylor approximation to h. Note that here we use the *exact* $\mathbf{F}$ for the mini-batch, rather than the approximation $\hat{\mathbf{F}}$. Intuitively, one can think of $\mathbf{v}$ as being itself iteratively optimized at each step in order to better minimize M, or in other words, to more closely match the true

[8]This may be counterintuitive, since SGD applied to neural nets tends to take small steps early in training, at least for commonly used initializations. For SGD, this happens because the initial parameters, and hence also the initial curvature, are relatively small in magnitude. Our method, which corrects for the curvature, takes larger steps early in training, when the error signal is the largest.

Algorithm 1 Using KFC as a preconditioner for SGD

```
Require: initial network parameters $\boldsymbol{\theta}^{(0)}$
         weight decay penalty $\lambda$
         learning rate $\alpha$
         momentum parameter $\mu$ (suggested value: 0.9)
         parameter averaging timescale $\tau$ (suggested value: number of mini-batches in the dataset)
         damping parameter $\gamma$ (suggested value: $10^{-3}$, but this may require tuning)
         statistics update period $T_s$ (see Appendix B.3)
         inverse update period $T_f$ (see Appendix B.3)
         clipping parameter $C$ (suggested value: 0.3)
  $k \leftarrow 0$
  $\mathbf{p} \leftarrow \mathbf{0}$
  $\xi \leftarrow e^{-1/\tau}$
  $\bar{\boldsymbol{\theta}}^{(0)} \leftarrow \boldsymbol{\theta}^{(0)}$
  Estimate the factors $\{\boldsymbol{\Omega}_\ell\}_{\ell=0}^{L-1}$ and $\{\boldsymbol{\Gamma}_\ell\}_{\ell=1}^{L}$ on the full dataset using Eqn. 23
  Compute the inverses $\{[\boldsymbol{\Omega}_\ell^{(\gamma)}]^{-1}\}_{\ell=0}^{L-1}$ and $\{[\boldsymbol{\Gamma}_\ell^{(\gamma)}]^{-1}\}_{\ell=1}^{L}$ using Eqn. 21
  while stopping criterion not met do
    $k \leftarrow k + 1$
    Select a new mini-batch

    if $k \equiv 0 \pmod{T_s}$ then
      Update the factors $\{\boldsymbol{\Omega}_\ell\}_{\ell=0}^{L-1}$ and $\{\boldsymbol{\Gamma}_\ell\}_{\ell=1}^{L}$ using Eqn. 23
    end if
    if $k \equiv 0 \pmod{T_f}$ then
      Compute the inverses $\{[\boldsymbol{\Omega}_\ell^{(\gamma)}]^{-1}\}_{\ell=0}^{L-1}$ and $\{[\boldsymbol{\Gamma}_\ell^{(\gamma)}]^{-1}\}_{\ell=1}^{L}$ using Eqn. 21
    end if

    Compute $\nabla h$ using backpropagation
    Compute $\hat{\nabla} h = [\hat{\mathbf{F}}^{(\gamma)}]^{-1} \nabla h$ using Eqn. 22
    $\mathbf{v} \leftarrow -\alpha \hat{\nabla} h$

    {Clip the update if necessary}
    Estimate $\nu = \mathbf{v}^\top \mathbf{F} \mathbf{v} + \lambda \mathbf{v}^\top \mathbf{v}$ using a subset of the current mini-batch
    if $\nu > C$ then
      $\mathbf{v} \leftarrow \mathbf{v} / \sqrt{\nu / C}$
    end if

    $\mathbf{p}^{(k)} \leftarrow \mu \mathbf{p}^{(k-1)} + \mathbf{v}$ {Update momentum}
    $\boldsymbol{\theta}^{(k)} \leftarrow \boldsymbol{\theta}^{(k-1)} + \mathbf{p}^{(k)}$ {Update parameters}
    $\bar{\boldsymbol{\theta}}^{(k)} \leftarrow \xi \bar{\boldsymbol{\theta}}^{(k-1)} + (1 - \xi) \boldsymbol{\theta}^{(k)}$ {Parameter averaging}
  end while
  return Averaged parameter vector $\bar{\boldsymbol{\theta}}^{(k)}$
```

natural gradient (which is the exact minimum of M). Interestingly, in full batch mode, this method is equivalent to performing preconditioned conjugate gradient in the vicinity of a local optimum (where $\mathbf{F}$ remains approximately constant).

To see how this minimization over α and μ can be done efficiently, without computing the entire matrix $\mathbf{F}$, consider the general problem of minimizing M on the subspace spanned by arbitrary vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_R\}$. (In our case, $R = 2$, $\mathbf{v}_1 = \hat{\nabla} h$ and $\mathbf{v}_2 = \mathbf{v}_{prev}$.) The coefficients $\boldsymbol{\alpha}$ can be found by solving the linear system $\mathbf{C}\boldsymbol{\alpha} = -\mathbf{d}$, where $\mathbf{C}_{ij} = \mathbf{v}_i^\top \mathbf{F} \mathbf{v}_j$ and $\mathbf{d}_i = \nabla h^\top \mathbf{v}_i$. To compute the matrix $\mathbf{C}$, we compute each of the matrix-vector products $\mathbf{F}\mathbf{v}_j$ using automatic differentiation (Schraudolph, 2002).

Both the approximate natural gradient $\hat{\nabla} h$ and the update $\mathbf{v}$ (generated as described above) arise as the minimum, or approximate minimum, of a corresponding quadratic model. In the case of $\mathbf{v}$, this model is given by M and is designed to be a good local approximation to the objective h. Meanwhile, the quadratic model which is implicitly minimized when computing $\hat{\nabla} h$ is designed to approximate M (by approximating $\mathbf{F}$ with $\hat{\mathbf{F}}$).

Because these quadratic models are approximations, naively minimizing them over $\mathbb{R}^n$ can lead to poor results in both theory and practice. To help deal with this problem K-FAC employs an adaptive Tikhonov-style damping scheme applied to each of them (the details of which differ in either case).

To compensate for the inaccuracy of M as a model of h, K-FAC adds a Tikhonov regularization term $\frac{\lambda}{2}\|\mathbf{v}\|^2$ to M which encourages the update to remain small in magnitude, and thus more likely to remain in the region where M is a reasonable approximation to h. This amounts to replacing r with $r + \lambda$ in Eqn. 30. Note that this technique is formally equivalent to performing constrained minimization of M within some spherical region around $\mathbf{v} = 0$ (a "trust-region"). See for example Nocedal & Wright (2006).

K-FAC uses the well-known Levenberg-Marquardt technique (Moré, 1978) to automatically adapt the damping parameter λ so that the damping is loosened or tightened depending on how accurately $M(\boldsymbol{\theta} + \mathbf{v})$ predicts the true decrease in the objective function after each step. This accuracy is measured by the so-called "reduction ratio", which is given by

$$\rho = \frac{h(\boldsymbol{\theta}) - h(\boldsymbol{\theta} + \mathbf{v})}{M(\boldsymbol{\theta}) - M(\boldsymbol{\theta} + \mathbf{v})}, \tag{31}$$

and should be close to 1 when the quadratic approximation is reasonably accurate around the given value of $\boldsymbol{\theta}$. The update rule for λ is as follows:

$$\lambda \leftarrow \begin{cases} \lambda \cdot \lambda_- & \text{if } \rho > 3/4 \\ \lambda & \text{if } 1/4 \leq \rho \leq 3/4 \\ \lambda \cdot \lambda_+ & \text{if } \rho < 1/4 \end{cases} \tag{32}$$

where λ_+ and λ_- are constants such that $\lambda_- < 1 < \lambda_+$.

To compensate for the inaccuracy of $\hat{\mathbf{F}}$, and encourage $\hat{\nabla} h$ to be smaller and more conservative, K-FAC similarly adds $\gamma \mathbf{I}$ to $\hat{\mathbf{F}}$ before inverting it. As discussed in Section 2.2, this can be done approximately by adding multiples of $\mathbf{I}$ to each of the Kronecker factors $\boldsymbol{\Psi}_\ell$ and $\boldsymbol{\Gamma}_\ell$ of $\hat{\mathbf{F}}_\ell$ before inverting them. Alternatively, an exact solution can be obtained by expanding out the eigendecomposition of each block $\hat{\mathbf{F}}_\ell$ of $\hat{\mathbf{F}}$, and using the following identity:

$$\begin{aligned} \left[\hat{\mathbf{F}}_\ell + \gamma \mathbf{I}\right]^{-1} &= \left[(\mathbf{Q}_{\boldsymbol{\Psi}} \otimes \mathbf{Q}_{\boldsymbol{\Gamma}})(\mathbf{D}_{\boldsymbol{\Psi}} \otimes \mathbf{D}_{\boldsymbol{\Gamma}})\left(\mathbf{Q}_{\boldsymbol{\Psi}}^\top \otimes \mathbf{Q}_{\boldsymbol{\Gamma}}^\top\right) + \gamma \mathbf{I}\right]^{-1} && (33) \\ &= \left[(\mathbf{Q}_{\boldsymbol{\Psi}} \otimes \mathbf{Q}_{\boldsymbol{\Gamma}})(\mathbf{D}_{\boldsymbol{\Psi}} \otimes \mathbf{D}_{\boldsymbol{\Gamma}} + \gamma \mathbf{I})\left(\mathbf{Q}_{\boldsymbol{\Psi}}^\top \otimes \mathbf{Q}_{\boldsymbol{\Gamma}}^\top\right)\right]^{-1} && (34) \\ &= (\mathbf{Q}_{\boldsymbol{\Psi}} \otimes \mathbf{Q}_{\boldsymbol{\Gamma}})(\mathbf{D}_{\boldsymbol{\Psi}} \otimes \mathbf{D}_{\boldsymbol{\Gamma}} + \gamma \mathbf{I})^{-1}\left(\mathbf{Q}_{\boldsymbol{\Psi}}^\top \otimes \mathbf{Q}_{\boldsymbol{\Gamma}}^\top\right), && (35) \end{aligned}$$

where $\boldsymbol{\Psi}_\ell = \mathbf{Q}_{\boldsymbol{\Psi}} \mathbf{D}_{\boldsymbol{\Psi}} \mathbf{Q}_{\boldsymbol{\Psi}}^\top$ and $\boldsymbol{\Gamma}_\ell = \mathbf{Q}_{\boldsymbol{\Gamma}} \mathbf{D}_{\boldsymbol{\Gamma}} \mathbf{Q}_{\boldsymbol{\Gamma}}^\top$ are the orthogonal eigendecompositions of $\boldsymbol{\Psi}_\ell$ and $\boldsymbol{\Gamma}_\ell$ (which are symmetric PSD). These manipulations are based on well-known properties of the Kronecker product which can be found in, e.g., Demmel (1997, sec. 6.3.3). Matrix-vector products $(\hat{\mathbf{F}} + \gamma \mathbf{I})^{-1} \nabla h$ can then be computed from the above identity using the following block-wise formulas:

$$\mathbf{V}_1 = \mathbf{Q}_{\boldsymbol{\Gamma}}^\top (\nabla_{\bar{\mathbf{W}}_\ell} h) \mathbf{Q}_{\boldsymbol{\Psi}} \tag{36}$$

$$\mathbf{V}_2 = \mathbf{V}_1 / (\mathbf{d}_{\boldsymbol{\Gamma}} \mathbf{d}_{\boldsymbol{\Psi}}^\top + \gamma) \tag{37}$$

$$(\hat{\mathbf{F}}_\ell + \gamma \mathbf{I})^{-1} \operatorname{vec}(\nabla_{\bar{\mathbf{W}}_\ell} h) = \operatorname{vec}\left(\mathbf{Q}_{\boldsymbol{\Gamma}} \mathbf{V}_2 \mathbf{Q}_{\boldsymbol{\Psi}}^\top\right), \tag{38}$$

where $\mathbf{d}_{\boldsymbol{\Gamma}}$ and $\mathbf{d}_{\boldsymbol{\Psi}}$ are the diagonals of $\mathbf{D}_{\boldsymbol{\Gamma}}$ and $\mathbf{D}_{\boldsymbol{\Psi}}$ and the division and addition in Eqn. 37 are both elementwise.

One benefit of this damping strategy is that it automatically accounts for the curvature contributed by both the quadratic damping term $\frac{\lambda}{2}\|\mathbf{v}\|^2$ and the weight decay penalty $\frac{r}{2}\|\boldsymbol{\theta}\|^2$ if these are used. Heuristically, one could even set $\gamma = \sqrt{\lambda + r}$, which can sometimes perform well. One should always choose γ at least this large. However, it may sometimes be advantageous to choose γ significantly larger, as $\hat{\mathbf{F}}$ might not be a good approximation to $\mathbf{F}$, and the damping may help reduce the impact of directions erroneously estimated to have low curvature. For consistency with Martens & Grosse (2015), we adopt their method of automatically adapting γ. In particular, each time we adapt γ, we compute $\hat{\nabla} h$ for three different values $\gamma_- < \gamma < \gamma_+$. We choose whichever of the three values results in the lowest value of $M(\boldsymbol{\theta} + \mathbf{v})$.

B.3. Efficient implementation

We based our implementation on the Toronto Deep Learning ConvNet (TDLCN) package (Srivastava, 2015), which is a Python wrapper around CUDA kernels. We needed to write a handful of additional kernels:

- a kernel for computing $\hat{\boldsymbol{\Omega}}_\ell$ (Eqn. 23)

- kernels which performed forward mode automatic differentiation for the max-pooling and response normalization layers

Most of the other operations for KFC could be performed on the GPU using kernels provided by TDLCN. The only exception is computing the inverses $\{[\mathbf{\Omega}_\ell^{(\gamma)}]^{-1}\}_{\ell=0}^{L-1}$ and $\{[\mathbf{\Gamma}_\ell^{(\gamma)}]^{-1}\}_{\ell=1}^{L}$, which was done on the CPU. (The forward mode kernels are only used in update clipping; as mentioned above, one can likely eliminate this step in practice by initializing from a partially trained model.)

KFC introduces several sources of overhead per iteration compared with SGD:

- Updating the factors $\{\mathbf{\Omega}_\ell\}_{\ell=0}^{L-1}$ and $\{\mathbf{\Gamma}_\ell\}_{\ell=1}^{L}$
- Computing the inverses $\{[\mathbf{\Omega}_\ell^{(\gamma)}]^{-1}\}_{\ell=0}^{L-1}$ and $\{[\mathbf{\Gamma}_\ell^{(\gamma)}]^{-1}\}_{\ell=1}^{L}$
- Computing the approximate natural gradient $\hat{\nabla} h = [\hat{\mathbf{F}}^{(\gamma)}]^{-1}\nabla h$
- Estimating $\nu = \mathbf{v}^\top \mathbf{F}\mathbf{v} + \lambda \mathbf{v}^\top \mathbf{v}$ (which is used for gradient clipping)

The overhead from the first two could be reduced by only periodically recomputing the factors and inverses, rather than doing so after every mini-batch. The cost of estimating $\mathbf{v}^\top \mathbf{F}\mathbf{v}$ can be reduced by using only a subset of the mini-batch. These shortcuts did not seem to hurt the per-epoch progress very much, as one can get away with using quite stale curvature information, and ν is only used for clipping and therefore doesn't need to be very accurate. The cost of computing $\hat{\nabla} h$ is unavoidable, but because it doesn't grow with the size of the mini-batch, its per-epoch cost can be made smaller by using larger mini-batches. (As we discuss further in Section 5.3, KFC can work well with large mini-batches.) These shortcuts introduce several additional hyperparameters, but fortunately these are easy to tune: we simply chose them such that the per-epoch cost of KFC was less than twice that of SGD. This requires only running a profiler for a few epochs, rather than measuring overall optimization performance.

Observe that the inverses $\{[\mathbf{\Omega}_\ell^{(\gamma)}]^{-1}\}_{\ell=0}^{L-1}$ and $\{[\mathbf{\Gamma}_\ell^{(\gamma)}]^{-1}\}_{\ell=1}^{L}$ are computed on the CPU, while all of the other heavy computation is GPU-bound. In principle, since KFC works fine with stale curvature information, the inverses could be computed asychronously while the algorithm is running, thereby making their cost almost free. We did not exploit this in our experiments, however.

C. Relationship with other algorithms

Other neural net optimization methods have been proposed which attempt to correct for various statistics of the activations or gradients. Perhaps the most commonly used are algorithms which attempt to adapt learning rates for individual parameters based on the variance of the gradients (LeCun et al., 1998; Duchi et al., 2011; Tieleman & Hinton, 2012; Zeiler, 2013; Kingma & Ba, 2015). These can be thought of as diagonal approximations to the curvature.

Another class of approaches attempts to reparameterize a network such that its activations have zero mean and unit variance, with the goals of preventing covariate shift and improving the conditioning of the curvature (Cho et al., 2013; Vatanen et al., 2013; Ioffe & Szegedy, 2015). Centering can be viewed as an approximation to natural gradient where the Fisher matrix is approximated with a directed Gaussian graphical model (Grosse & Salakhutdinov, 2015). As discussed in Section 4.1, KFC is invariant to re-centering of activations, so it ought to automatically enjoy the optimization benefits of centering. However, batch normalization (Ioffe & Szegedy, 2015) includes some effects not automatically captured by KFC. First, the normalization is done separately for each mini-batch rather than averaged across mini-batches; this introduces stochasticity into the computations which may serve as a regularizer. Second, it discourages large covariate shifts in the pre-activations, which may help to avoid dead units. Since batch normalization is better regarded as a modification to the architecture than an optimization algorithm, it can be combined with KFC; we investigated this in our experiments.

Projected Natural Gradient (PRONG; Desjardins et al., 2015) goes a step further than centering methods by fully whitening the activations in each layer. In the case of fully connected layers, the activations are transformed to have zero mean and unit covariance. For convolutional layers, they apply a linear transformation that whitens the activations *across feature maps*. While PRONG includes clever heuristics for updating the statistics, it's instructive to consider an idealized version of the method which has access to the exact statistics. We can interpret this idealized PRONG in our own framework as arising from following two additional approximations:

- **Spatially uncorrelated activations (SUA).** The activations at any two distinct spatial locations are uncorrelated, *i.e.* $\mathrm{Cov}(a_{j,t}, a_{j',t'}) = 0$ for $t \neq t'$. Also assuming **SH**, the correlations can then be written as $\mathrm{Cov}(a_{j,t}, a_{j',t}) = \Sigma(j, j')$.
- **White derivatives (WD).** Pre-activation derivatives are uncorrelated and have spherical covariance, i.e. $\Gamma(i, i', \delta) \propto \mathbb{1}_{i=i'}\mathbb{1}_{\delta=0}$. We can assume WLOG that the proportionality constant is 1, since any scalar factor can be absorbed into the learning rate.

Theorem 4. *Combining approximations* **IAD**, **SH**, **SUA**, *and* **WD** *results in the following approximation to the entries of the Fisher matrix:*

$$\mathbb{E}\left[\mathcal{D}w_{i,j,\delta}\mathcal{D}w_{i',j',\delta'}\right] = \beta(\delta,\delta')\,\tilde{\Omega}(j,j',\delta'-\delta)\,\mathbb{1}_{i=i'}, \tag{39}$$

where $\mathbb{1}$ is the indicator function and $\tilde{\Omega}(j, j', \delta) \triangleq \Sigma(j, j')\mathbb{1}_{\delta=0} + M(j)M(j')$ is the uncentered autocovariance function. (β is defined in Theorem 1. Formulas for the remaining entries are given in Appendix E.) If the $\beta(\delta, \delta')$ term is dropped, the resulting approximate natural gradient descent update rule is equivalent to idealized PRONG, up to rescaling.

As we later discuss in Section 5.1, assumption **WD** appears to hold up well empirically, while **SUA** appears to lose a lot of information. Observe, for instance, that the input images are themselves treated as a layer of activations. Assumption **SUA** amounts to modeling each channel of an image as white noise, corresponding to a flat power spectrum. Images have a well-characterized $1/f^p$ power spectrum with $p \approx 2$ (Simoncelli & Olshausen, 2001), which implies that the curvature may be much larger in directions corresponding to low-frequency Fourier components than in directions corresponding to high-frequency components.

D. Experiments

D.1. Evaluating the probabilistic modeling assumptions

In order to analyze the reasonableness of our spatially uncorrelated derivatives (**SUD**) assumption, we investigated the autocorrelation functions for networks trained on CIFAR-10 and SVHN, each with 50 epochs of SGD. (These models were trained long enough to achieve good test error, but not long enough to overfit.) Derivatives were sampled from the model's distribution as described in Section 2.2. Figure 3(a) shows the autocorrelation functions of the pre-activation gradients for three (arbitrary) feature maps in all of the convolution layers of both networks. Figure 3(b) shows the correlations between derivatives for different feature maps in the same spatial position. Evidently, the derivatives are very weakly correlated, both spatially and cross-map, although there are some modest cross-map correlations in the first layers of both models, as well as modest spatial correlations in the top convolution layer of the CIFAR-10 network. This suggests that **SUD** is a good approximation for these networks.

Interestingly, the lack of correlations between derivatives appears to be a result of max-pooling. Max-pooling has a well-known sparsifying effect on the derivatives, as any derivative is zero unless the corresponding activation achieves the maximum within its pooling group. Since neighboring locations are unlikely to simultaneously achieve the maximum, max-pooling weakens the spatial correlations. To test this hypothesis, we trained networks equivalent to those described above, except that the max-pooling layers were replaced with average pooling. The spatial autocorrelations and cross-map correlations are shown in Figure 3(c, d). Replacing max-pooling with average pooling dramatically strengthens both sets of correlations.

In contrast with the derivatives, the activations have very strong correlations, both spatially and cross-map, as shown in Figure 4. This suggests the spatially uncorrelated activations **(SUA)** assumption implicitly made by some algorithms could be problematic, despite appearing superficially analogous to **SUD**.

D.2. Comparison with batch normalization

Batch normalization (BN Ioffe & Szegedy, 2015) has recently had much success at training a variety of neural network architectures. It has been motivated both in terms of optimization benefits (because it reduces covariate shift) and regularization benefits (because it adds stochasticity to the updates). However, BN is best regarded not as an optimization algorithm, but as a modification to the network architecture, and it can be used in conjunction with algorithms other than SGD. We modified the original CIFAR-10 architecture to use batch normalization in each layer. Since the parameters of a batch normalized network would have a different scale from those of an ordinary network, we disabled the ℓ_2 regularization term so that both networks would be optimized to the same objective function. While our own (inefficient) implementation of batch normalization incurred substantial computational overhead, we believe an efficient implementation ought to have very little overhead; therefore, we simulated an efficient implementation by reusing the timing data from the non-batch-normalized networks. Learning rates were tuned separately for all four conditions (similarly to the rest of our experiments).

Training curves are shown in Figure 5. All of the methods achieved worse test error than the original network as a result of ℓ_2 regularization being eliminated. However, the BN networks reached a lower test error than the non-BN networks before they started overfitting, consistent with the stochastic regularization interpretation of BN.[9] For both the BN and non-BN architectures, KFC-pre optimized both the training and test error and NLL considerably faster than SGD. Furthermore, it appeared not to lose the regularization benefit of BN. This suggests that KFC-pre and BN can be combined synergistically.

[9]Interestingly, the BN networks were slower to optimize the training error than their non-BN counterparts. We speculate that this is because (1) the SGD baseline, being carefully tuned, didn't exhibit the pathologies that BN is meant to correct for (i.e. dead units and extreme covariate shift), and (2) the regularization effects of BN made it harder to overfit.

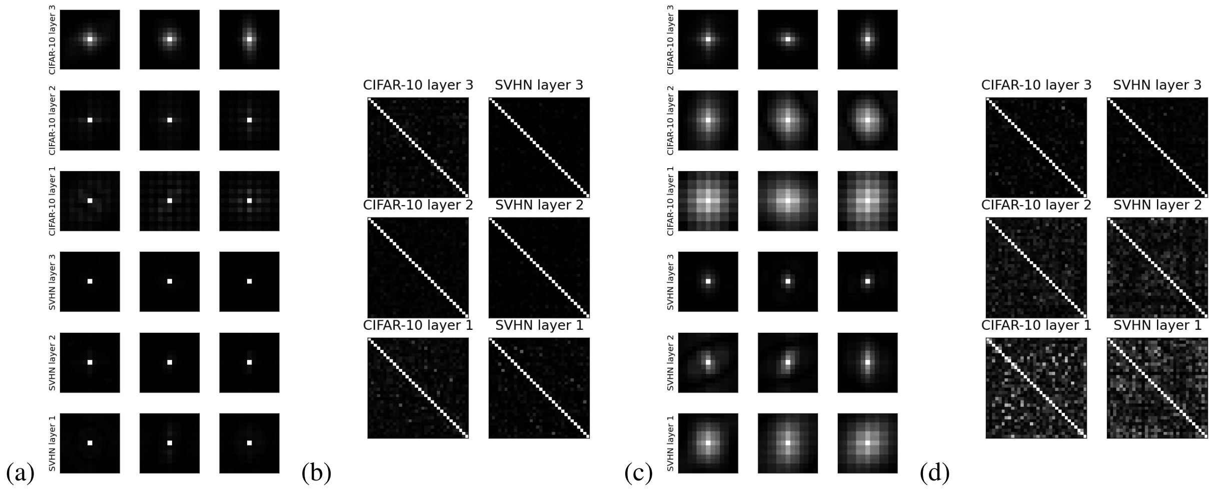

Figure 3. Visualization of the absolute values of the correlations between the pre-activation derivatives for all of the convolution layers of CIFAR-10 and SVHN networks trained with SGD. **(a)** Autocorrelation functions of the derivatives of three feature maps from each layer. **(b)** Cross-map correlations for a single spatial position. **(c, d)** Same as (a) and (b), except that the networks use average pooling rather than max-pooling.

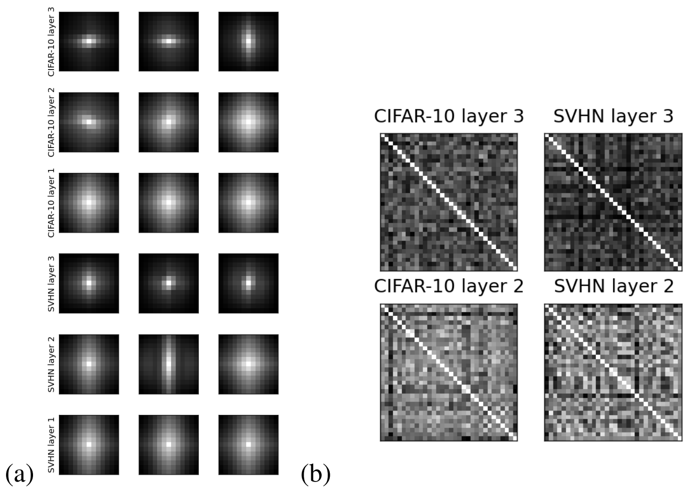

Figure 4. Visualization of the uncentered correlations Ω between activations in all of the convolution layers of the CIFAR-10 and SVHN networks. **(a)** Spatial autocorrelation functions of three feature maps in each layer. **(b)** Correlations of the activations at a given spatial location. The activations have much stronger correlations than the backpropagated derivatives.

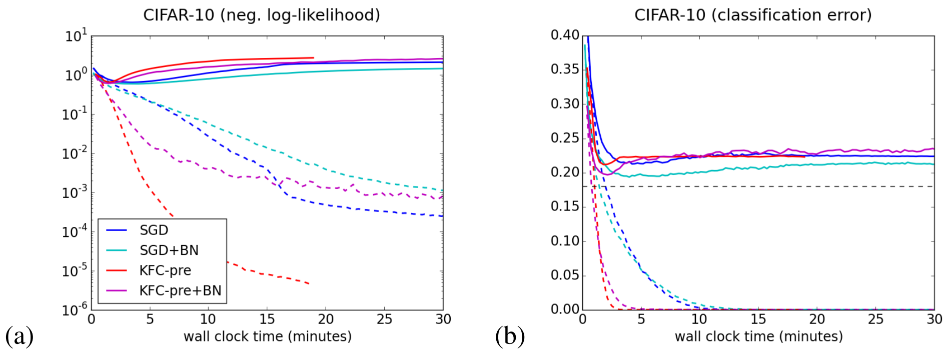

Figure 5. Optimization performance of KFC-pre and SGD on a CIFAR-10 network, with and without batch normalization (BN). **(a)** Negative log-likelihood, on a log scale. **(b)** Classification error. **Solid lines** represent test error and **dashed lines** represent training error. The **horizontal dashed line** represents the previously reported test error for the same architecture. The KFC-pre training curve is cut off because the algorithm became unstable when the training NLL reached 4×10^{-6}.

E. Proofs of theorems

E.1. Proofs for Section 3

Lemma 1. *Under approximation* **IAD***,*

$$\begin{aligned}
\mathbb{E}\left[\mathcal{D}w_{i,j,\delta}\mathcal{D}w_{i',j',\delta'}\right] &= \sum_{t\in\mathcal{T}}\sum_{t'\in\mathcal{T}} \mathbb{E}\left[a_{j,t+\delta}a_{j',t'+\delta'}\right]\mathbb{E}\left[\mathcal{D}s_{i,t}\mathcal{D}s_{i',t'}\right] && (40)\\
\mathbb{E}\left[\mathcal{D}w_{i,j,\delta}\mathcal{D}b_{i'}\right] &= \sum_{t\in\mathcal{T}}\sum_{t'\in\mathcal{T}} \mathbb{E}\left[a_{j,t+\delta}\right]\mathbb{E}\left[\mathcal{D}s_{i,t}\mathcal{D}s_{i',t'}\right] && (41)\\
\mathbb{E}\left[\mathcal{D}b_{i}\mathcal{D}b_{i'}\right] &= |\mathcal{T}|\,\mathbb{E}\left[\mathcal{D}s_{i,t}\mathcal{D}s_{i',t'}\right] && (42)
\end{aligned}$$

Proof. We prove the first equality; the rest are analogous.

$$\begin{aligned}
\mathbb{E}[\mathcal{D}w_{i,j,\delta}\mathcal{D}w_{i',j',\delta'}] &= \mathbb{E}\left[\left(\sum_{t\in\mathcal{T}} a_{j,t+\delta}\mathcal{D}s_{i,t}\right)\left(\sum_{t'\in\mathcal{T}} a_{j',t'+\delta'}\mathcal{D}s_{i',t'}\right)\right] && (43)\\
&= \mathbb{E}\left[\sum_{t\in\mathcal{T}}\sum_{t'\in\mathcal{T}} a_{j,t+\delta}\mathcal{D}s_{i,t}a_{j',t'+\delta'}\mathcal{D}s_{i',t'}\right] && (44)\\
&= \sum_{t\in\mathcal{T}}\sum_{t'\in\mathcal{T}} \mathbb{E}\left[a_{j,t+\delta}\mathcal{D}s_{i,t}a_{j',t'+\delta'}\mathcal{D}s_{i',t'}\right] && (45)\\
&= \sum_{t\in\mathcal{T}}\sum_{t'\in\mathcal{T}} \mathbb{E}\left[a_{j,t+\delta}a_{j',t'+\delta'}\right]\mathbb{E}\left[\mathcal{D}s_{i,t}\mathcal{D}s_{i',t'}\right] && (46)
\end{aligned}$$

Assumption **IAD** is used in the final line. □

Theorem 1. *Combining approximations* **IAD***,* **SH***, and* **SUD** *yields the following factorization:*

$$\begin{aligned}
\mathbb{E}\left[\mathcal{D}w_{i,j,\delta}\mathcal{D}w_{i',j',\delta'}\right] &= \beta(\delta,\delta')\,\Omega(j,j',\delta'-\delta)\,\Gamma(i,i',0),\\
\mathbb{E}\left[\mathcal{D}w_{i,j,\delta}\mathcal{D}b_{i'}\right] &= \beta(\delta)\,M(j)\,\Gamma(i,i',0)\\
\mathbb{E}\left[\mathcal{D}b_{i}\mathcal{D}b_{i'}\right] &= |\mathcal{T}|\,\Gamma(i,i',0) && (47)
\end{aligned}$$

where

$$\begin{aligned}
\beta(\delta) &\triangleq (T_1-|\delta_1|)\;(T_2-|\delta_2|)\\
\beta(\delta,\delta') &\triangleq (T_1-\max(\delta_1,\delta_1',0)+\min(\delta_1,\delta_1',0))\cdot(T_2-\max(\delta_2,\delta_2',0)+\min(\delta_2,\delta_2',0)) && (48)
\end{aligned}$$

Proof.

$$\begin{aligned}
\mathbb{E}[\mathcal{D}w_{i,j,\delta}\mathcal{D}w_{i',j',\delta'}] &= \sum_{t\in\mathcal{T}}\sum_{t'\in\mathcal{T}} \mathbb{E}\left[a_{j,t+\delta}a_{j',t'+\delta'}\right]\mathbb{E}\left[\mathcal{D}s_{i,t}\mathcal{D}s_{i',t'}\right] & (49)\\
&= \sum_{t\in\mathcal{T}}\sum_{t'\in\mathcal{T}} \Omega(j,j',t'+\delta'-t-\delta)\,\mathbb{1}_{\substack{t+\delta\in\mathcal{T}\\ t'+\delta'\in\mathcal{T}}}\,\Gamma(i,i',t'-t) & (50)\\
&= \sum_{t\in\mathcal{T}} \Omega(j,j',\delta'-\delta)\,\mathbb{1}_{\substack{t+\delta\in\mathcal{T}\\ t+\delta'\in\mathcal{T}}}\,\Gamma(i,i',0) & (51)\\
&= \left|\{t\in\mathcal{T}: t+\delta\in\mathcal{T}, t+\delta'\in\mathcal{T}\}\right|\,\Omega(j,j',\delta'-\delta)\,\Gamma(i,i',0) & (52)\\
&= \beta(\delta,\delta')\,\Omega(j,j',\delta'-\delta)\,\Gamma(i,i',0) & (53)
\end{aligned}$$

Lines 49, 50, and 51 use Lemma 1 and assumptions **SH**, and **SUD**, respectively. In Line 50, the indicator function (denoted $\mathbb{1}$) arises because the activations are defined to be zero outside the set of spatial locations. The remaining formulas can be derived analogously. □

Theorem 2. *Under assumption* **SH**,

$$\begin{aligned}
\boldsymbol{\Omega}_\ell &= \mathbb{E}\left[[\![\mathbf{A}_\ell]\!]_H^\top [\![\mathbf{A}_\ell]\!]_H\right] & (54)\\
\boldsymbol{\Gamma}_\ell &= \frac{1}{|\mathcal{T}|}\mathbb{E}\left[\mathcal{D}\mathbf{S}_\ell^\top \mathcal{D}\mathbf{S}_\ell\right] & (55)
\end{aligned}$$

Proof. In this proof, all activations and pre-activations are taken to be in layer ℓ. The expected entries are given by:

$$\begin{aligned}
\mathbb{E}\left[[\![\mathbf{A}_\ell]\!]_H^\top [\![\mathbf{A}_\ell]\!]_H\right]_{j|\Delta|+\delta,\, j'|\Delta|+\delta} &= \mathbb{E}\left[\sum_{t\in\mathcal{T}} a_{j,t+\delta}a_{j',t+\delta'}\right] & (56)\\
&= \sum_{t\in\mathcal{T}} \mathbb{E}\left[a_{j,t+\delta}a_{j',t+\delta'}\right] & (57)\\
&= \sum_{t\in\mathcal{T}} \Omega(j,j',\delta'-\delta)\,\mathbb{1}_{\substack{t+\delta\in\mathcal{T}\\ t+\delta'\in\mathcal{T}}} & (58)\\
&= \left|\{t\in\mathcal{T}: t+\delta\in\mathcal{T}, t+\delta'\in\mathcal{T}\}\right|\,\Omega(j,j',\delta'-\delta) & (59)\\
&= \beta(\delta,\delta')\,\Omega(j,j',\delta'-\delta) & (60)\\
&= [\boldsymbol{\Omega}_\ell]_{j|\Delta|+\delta,\, j'|\Delta|+\delta'} & (61)
\end{aligned}$$

SH is used in Line 58. Similarly,

$$\begin{aligned}
\mathbb{E}\left[[\![\mathbf{A}_\ell]\!]_H^\top [\![\mathbf{A}_\ell]\!]_H\right]_{0,\, j|\Delta|+\delta} &= \mathbb{E}\left[\sum_{t\in\mathcal{T}} a_{j,t+\delta}\right] & (62)\\
&= \beta(\delta)\,M(j) & (63)\\
&= [\boldsymbol{\Omega}_\ell]_{0,\, j|\Delta|+\delta} & (64)\\
\left[[\![\mathbf{A}_\ell]\!]_H^\top [\![\mathbf{A}_\ell]\!]_H\right]_{0,\,0} &= |\mathcal{T}| & (65)\\
&= [\boldsymbol{\Omega}_\ell]_{0,\,0} & (66)\\
\mathbb{E}\left[\mathcal{D}\mathbf{S}_\ell^\top \mathcal{D}\mathbf{S}_\ell\right]_{i,i'} &= \mathbb{E}\left[\sum_{t\in\mathcal{T}} \mathcal{D}s_{i,t}\mathcal{D}s_{i',t}\right] & (67)\\
&= |\mathcal{T}|\,\Gamma(i,i',0) & (68)\\
&= |\mathcal{T}|\;[\boldsymbol{\Gamma}_\ell]_{i,\,i'} & (69)
\end{aligned}$$

□

E.2. Proofs for Section 4

Preliminaries and notation. In discussing invariances, it will be convenient to add homogeneous coordinates to various matrices:

$$[\mathbf{A}_\ell]_H \triangleq \begin{pmatrix} \mathbf{1} & \mathbf{A}_\ell \end{pmatrix} \tag{70}$$

$$[\mathbf{S}_\ell]_H \triangleq \begin{pmatrix} \mathbf{1} & \mathbf{S}_\ell \end{pmatrix} \tag{71}$$

$$[\bar{\mathbf{W}}_\ell]_H \triangleq \begin{pmatrix} 1 & \\ \mathbf{b}_\ell & \mathbf{W}_\ell \end{pmatrix} \tag{72}$$

We also define the activation function ϕ to ignore the homogeneous column, so that

$$[\mathbf{A}_\ell]_H = \phi([\mathbf{S}_\ell]_H) = \phi([\![\mathbf{A}_{\ell-1}]\!][\bar{\mathbf{W}}_\ell]_H). \tag{73}$$

Using the homogeneous coordinate notation, we can write the effect of the affine transformations on the pre-activations and activations:

$$\begin{aligned} [\mathbf{S}_\ell^\dagger \mathbf{U}_\ell + \mathbf{1}\mathbf{c}_\ell^\top]_H &= [\mathbf{S}_\ell^\dagger]_H [\mathbf{U}_\ell]_H \\ [\mathbf{A}_\ell \mathbf{V}_\ell + \mathbf{1}\mathbf{d}_\ell^\top]_H &= [\mathbf{A}_\ell]_H [\mathbf{V}_\ell]_H, \end{aligned} \tag{74}$$

where

$$[\mathbf{U}_\ell]_H \triangleq \begin{pmatrix} 1 & \mathbf{c}_\ell^\top \\ & \mathbf{U}_\ell \end{pmatrix} \tag{75}$$

$$[\mathbf{V}_\ell]_H \triangleq \begin{pmatrix} 1 & \mathbf{d}_\ell^\top \\ & \mathbf{V}_\ell \end{pmatrix}. \tag{76}$$

The inverse transformations are represented as

$$[\mathbf{U}_\ell]_H^{-1} \triangleq \begin{pmatrix} 1 & -\mathbf{c}_\ell^\top \mathbf{U}_\ell^{-1} \\ & \mathbf{U}_\ell^{-1} \end{pmatrix} \tag{77}$$

$$[\mathbf{V}_\ell]_H^{-1} \triangleq \begin{pmatrix} 1 & -\mathbf{d}_\ell^\top \mathbf{V}_\ell^{-1} \\ & \mathbf{V}_\ell^{-1} \end{pmatrix}. \tag{78}$$

We can also determine the effect of the affine transformation on the *expanded* activations:

$$[\![\mathbf{A}_\ell \mathbf{V}_\ell + \mathbf{1}\mathbf{d}_\ell^\top]\!]_H = [\![\mathbf{A}_\ell]\!]_H [\![\mathbf{V}_\ell]\!]_H, \tag{79}$$

where

$$[\![\mathbf{V}_\ell]\!]_H \triangleq \begin{pmatrix} 1 & \mathbf{d}_\ell^\top \otimes \mathbf{1}^\top \\ & \mathbf{V}_\ell \otimes \mathbf{I} \end{pmatrix}, \tag{80}$$

with inverse

$$[\![\mathbf{V}_\ell]\!]_H^{-1} = \begin{pmatrix} 1 & -\mathbf{d}_\ell^\top \mathbf{V}_\ell^{-1} \otimes \mathbf{1}^\top \\ & \mathbf{V}_\ell^{-1} \otimes \mathbf{I} \end{pmatrix}. \tag{81}$$

Note that $[\![\mathbf{V}_\ell]\!]_H$ is simply a suggestive notation, rather than an application of the expansion operator $[\![\cdot]\!]$.

Lemma 2. *Let $\mathcal{N}$, $\boldsymbol{\theta}$, $\{\phi_\ell\}_{\ell=0}^L$, and $\{\phi_\ell^\dagger\}_{\ell=0}^L$ be given as in Theorem 3. The network $\mathcal{N}^\dagger$ with activations functions $\{\phi_\ell^\dagger\}_{\ell=0}^L$ and parameters defined by*

$$[\bar{\mathbf{W}}_\ell^\dagger]_H \triangleq [\mathbf{U}_\ell]_H^{-\top} [\bar{\mathbf{W}}_\ell]_H [\![\mathbf{V}_{\ell-1}]\!]_H^{-\top}, \tag{82}$$

compute the same function as $\mathcal{N}$.

Remark. The definition of $\phi_\ell^\dagger$ (Eqn. 24) can be written in homogeneous coordinates as

$$[\mathbf{A}_\ell^\dagger]_H = \phi_\ell^\dagger([\mathbf{S}_\ell^\dagger]_H) = \phi_\ell([\mathbf{S}_\ell^\dagger]_H [\mathbf{U}_\ell]_H)[\mathbf{V}_\ell]_H. \tag{83}$$

Eqn. 82 can be expressed equivalently without homogeneous coordinates as

$$\bar{\mathbf{W}}_\ell^\dagger \triangleq \mathbf{U}_\ell^{-\top} \left(\bar{\mathbf{W}}_\ell - \mathbf{c}_\ell \mathbf{e}^\top\right) [\![\mathbf{V}_{\ell-1}]\!]_H^{-\top}, \tag{84}$$

where $\mathbf{e} = (1\ 0\ \cdots\ 0)^\top$.

Proof. We will show inductively the following relationship between the activations in each layer of the two networks:

$$[\mathbf{A}_\ell^\dagger]_H = [\mathbf{A}_\ell]_H [\mathbf{V}_\ell]_H. \tag{85}$$

(By our assumption that the top layer inputs are not transformed, i.e. $[\mathbf{V}_L]_H = \mathbf{I}$, this would imply that $[\mathbf{A}_L^\dagger]_H = [\mathbf{A}_L]_H$, and hence that the networks compute the same function.) For the first layer, Eqn. 85 is true by definition. For the inductive step, assume Eqn. 85 holds for layer $\ell - 1$. From Eqn 79, this is equivalent to

$$[\![\mathbf{A}_{\ell-1}^\dagger]\!]_H = [\![\mathbf{A}_{\ell-1}]\!]_H [\![\mathbf{V}_{\ell-1}]\!]_H. \tag{86}$$

We then derive the activations in the following layer:

$$\begin{aligned}
[\mathbf{A}_\ell^\dagger]_H &= \phi_\ell^\dagger \left([\mathbf{S}_\ell^\dagger]_H \right) && (87)\\
&= \phi_\ell \left([\mathbf{S}_\ell^\dagger]_H \, [\mathbf{U}_\ell]_H \right) \, [\mathbf{V}_\ell]_H && (88)\\
&= \phi_\ell \left([\![\mathbf{A}_{\ell-1}^\dagger]\!]_H \, [\bar{\mathbf{W}}_\ell^\dagger]_H^\top \, [\mathbf{U}_\ell]_H \right) \, [\mathbf{V}_\ell]_H && (89)\\
&= \phi_\ell \left([\![\mathbf{A}_{\ell-1}]\!]_H \, [\![\mathbf{V}_{\ell-1}]\!]_H \, [\bar{\mathbf{W}}_\ell^\dagger]_H^\top \, [\mathbf{U}_\ell]_H \right) \, [\mathbf{V}_\ell]_H && (90)\\
&= \phi_\ell \left([\![\mathbf{A}_{\ell-1}]\!]_H \, [\![\mathbf{V}_{\ell-1}]\!]_H \, [\![\mathbf{V}_{\ell-1}]\!]_H^{-1} \, [\bar{\mathbf{W}}_\ell]_H^\top \, [\mathbf{U}_\ell]_H^{-1} \, [\mathbf{U}_\ell]_H \right) \, [\mathbf{V}_\ell]_H && (91)\\
&= \phi_\ell \left([\![\mathbf{A}_{\ell-1}]\!]_H \, [\bar{\mathbf{W}}_\ell]_H^\top \right) \, [\mathbf{V}_\ell]_H && (92)\\
&= [\mathbf{A}_\ell]_H \, [\mathbf{V}_\ell]_H && (93)
\end{aligned}$$

Lines 89 and 93 are from Eqn. 73. This proves the inductive hypothesis for layer ℓ, so we have shown that both networks compute the same function. □

Lemma 3. *Suppose the parameters are transformed according to Lemma 2, and the parameters are updated according to*

$$[\bar{\mathbf{W}}_\ell^\dagger]^{(k+1)} \leftarrow [\bar{\mathbf{W}}_\ell^\dagger]^{(k)} - \alpha \mathbf{P}_\ell^\dagger (\nabla_{\bar{\mathbf{W}}_\ell^\dagger} h) \mathbf{R}_\ell^\dagger, \tag{94}$$

for matrices $\mathbf{P}_\ell$ and $\mathbf{R}_\ell$. This is equivalent to applying the following update to the original network:

$$[\bar{\mathbf{W}}_\ell]^{(k+1)} \leftarrow [\bar{\mathbf{W}}_\ell]^{(k+1)} - \alpha \mathbf{P}_\ell (\nabla_{\bar{\mathbf{W}}_\ell} h) \mathbf{R}_\ell, \tag{95}$$

with

$$\begin{aligned}
\mathbf{P}_\ell &= \mathbf{U}_\ell^\top \mathbf{P}_\ell^\dagger \mathbf{U}_\ell && (96)\\
\mathbf{R}_\ell &= [\![\mathbf{V}_{\ell-1}]\!]_H \mathbf{R}_\ell^\dagger [\![\mathbf{V}_{\ell-1}]\!]_H^\top. && (97)
\end{aligned}$$

Proof. This is a special case of Lemma 5 from Martens & Grosse (2015). □

Theorem 3. *Let $\mathcal{N}$ be a network with parameter vector $\boldsymbol{\theta}$ and activation functions $\{\phi_\ell\}_{\ell=0}^L$. Given activation functions $\{\phi_\ell^\dagger\}_{\ell=0}^L$ defined as in Eqn. 24, there exists a parameter vector $\boldsymbol{\theta}^\dagger$ such that a network $\mathcal{N}^\dagger$ with parameters $\boldsymbol{\theta}^\dagger$ and activation functions $\{\phi_\ell^\dagger\}_{\ell=0}^L$ computes the same function as $\mathcal{N}$. The KFC updates on $\mathcal{N}$ and $\mathcal{N}^\dagger$ are equivalent, in that the resulting networks compute the same function.*

Proof. Lemma 2 gives the desired $\boldsymbol{\theta}^\dagger$. We now prove equivalence of the KFC updates. The Kronecker factors for $\mathcal{N}^\dagger$ are given by:

$$\Omega_\ell^\dagger = \mathbb{E}\left[[\![\mathbf{A}_\ell^\dagger]\!]_H^\top [\![\mathbf{A}_\ell^\dagger]\!]_H\right] \tag{98}$$
$$= \mathbb{E}\left[[\![\mathbf{V}_\ell]\!]_H^\top [\![\mathbf{A}_\ell]\!]_H^\top [\![\mathbf{A}_\ell]\!]_H [\![\mathbf{V}_\ell]\!]_H\right] \tag{99}$$
$$= [\![\mathbf{V}_\ell]\!]_H^\top \mathbb{E}\left[[\![\mathbf{A}_\ell]\!]_H^\top [\![\mathbf{A}_\ell]\!]_H\right] [\![\mathbf{V}_\ell]\!]_H \tag{100}$$
$$= [\![\mathbf{V}_\ell]\!]_H^\top \Omega_\ell [\![\mathbf{V}_\ell]\!]_H \tag{101}$$
$$\Gamma_\ell^\dagger = \frac{1}{|\mathcal{T}|}\mathbb{E}\left[(\mathcal{D}\mathbf{S}_\ell^\dagger)^\top \mathcal{D}\mathbf{S}_\ell^\dagger\right] \tag{102}$$
$$= \frac{1}{|\mathcal{T}|}\mathbb{E}\left[\mathbf{U}_\ell(\mathcal{D}\mathbf{S}_\ell^\dagger)^\top \mathcal{D}\mathbf{S}_\ell^\dagger \mathbf{U}_\ell^\top\right] \tag{103}$$
$$= \frac{1}{|\mathcal{T}|}\mathbf{U}_\ell\mathbb{E}\left[(\mathcal{D}\mathbf{S}_\ell^\dagger)^\top \mathcal{D}\mathbf{S}_\ell^\dagger\right] \mathbf{U}_\ell^\top \tag{104}$$
$$= \mathbf{U}_\ell \Gamma_\ell \mathbf{U}_\ell^\top \tag{105}$$

The approximate natural gradient update, ignoring momentum, clipping, and damping, is given by $\boldsymbol{\theta}^{(k+1)} \leftarrow \boldsymbol{\theta}^{(k)} - \alpha\hat{\mathbf{F}}^{-1}\nabla_{\boldsymbol{\theta}} h$. For each layer of $\mathcal{N}^\dagger$,

$$[\bar{\mathbf{W}}_\ell^\dagger]^{(k+1)} \leftarrow [\bar{\mathbf{W}}_\ell^\dagger]^{(k)} - \alpha(\Gamma_\ell^\dagger)^{-1}(\nabla_{\bar{\mathbf{W}}_\ell^\dagger} h)(\Omega_{\ell-1}^\dagger)^{-1} \tag{106}$$

We apply Lemma 3 with $\mathbf{P}_\ell^\dagger = (\Gamma_\ell^\dagger)^{-1}$ and $\mathbf{R}_\ell^\dagger = (\Omega_{\ell-1}^\dagger)^{-1}$. This gives us

$$\mathbf{P}_\ell = \mathbf{U}_\ell^\top(\Gamma_\ell^\dagger)^{-1}\mathbf{U}_\ell \tag{107}$$
$$= \Gamma_\ell^{-1} \tag{108}$$
$$\mathbf{R}_\ell = [\![\mathbf{V}_{\ell-1}]\!]_H(\Omega_{\ell-1}^\dagger)^{-1}[\![\mathbf{V}_{\ell-1}]\!]_H^\top \tag{109}$$
$$= \Omega_{\ell-1}^{-1}, \tag{110}$$

with the corresponding update

$$[\bar{\mathbf{W}}_\ell]^{(k+1)} \leftarrow [\bar{\mathbf{W}}_\ell]^{(k)} - \alpha\Gamma_\ell^{-1}(\nabla_{\bar{\mathbf{W}}_\ell} h)\Omega_{\ell-1}^{-1}. \tag{111}$$

But this is the same as the KFC update for the original network. Therefore, the two updates are equivalent, in that the resulting networks compute the same function. □

Theorem 4. *Combining approximations* **IAD**, **SH**, **SUA**, *and* **WD** *results in the following approximation to the entries of the Fisher matrix:*

$$\mathbb{E}\left[\mathcal{D}w_{i,j,\delta}\mathcal{D}w_{i',j',\delta'}\right] = \beta(\delta,\delta')\,\tilde{\Omega}(j,j',\delta'-\delta)\,\mathbb{1}_{i=i'} \tag{112}$$
$$\mathbb{E}\left[\mathcal{D}w_{i,j,\delta}\mathcal{D}b_{i'}\right] = \beta(\delta)\,M(j)\,\mathbb{1}_{i=i'} \tag{113}$$
$$\mathbb{E}\left[\mathcal{D}b_i\mathcal{D}b_{i'}\right] = |\mathcal{T}|\,\mathbb{1}_{i=i'} \tag{114}$$

where $\mathbb{1}$ is the indicator function and $\tilde{\Omega}(j,j',\delta) = \Sigma(j,j')\mathbb{1}_{\delta=0} + M(j)M(j')$ is the uncentered autocovariance function. (β is defined in Theorem 1.) If the β and $|\mathcal{T}|$ terms are dropped, the resulting approximate natural gradient descent update rule is equivalent to idealized PRONG, up to rescaling.

Proof. We first compute the second moments of the activations and derivatives, under assumptions **SH**, **SUA**, and **WD**:

$$\mathbb{E}\left[a_{j,t}a_{j',t'}\right] = \mathrm{Cov}(a_{j,t},a_{j',t'}) + \mathbb{E}[a_{j,t}]\mathbb{E}[a_{j',t'}] \tag{115}$$
$$= \Sigma(j,j')\mathbb{1}_{\delta=0} + M(j)M(j') \tag{116}$$
$$\triangleq \tilde{\Omega}(j,j',\delta) \tag{117}$$
$$\mathbb{E}\left[\mathcal{D}s_{i,t}\mathcal{D}s_{i',t'}\right] = \mathbb{1}_{i=i'}\mathbb{1}_{\delta=\delta'}. \tag{118}$$

for any $t, t' \in \mathcal{T}$. We now compute

$$\begin{aligned}
\mathbb{E}\left[\mathcal{D}w_{i,j,\delta}\mathcal{D}w_{i,j,\delta}\right] &= \sum_{t\in\mathcal{T}}\sum_{t'\in\mathcal{T}} \mathbb{E}\left[a_{j,t+\delta}a_{j',t'+\delta'}\right]\mathbb{E}\left[\mathcal{D}s_{i,t}\mathcal{D}s_{i',t'}\right] && (119)\\
&= \sum_{t\in\mathcal{T}}\sum_{t'\in\mathcal{T}} \tilde{\Omega}(j,j',t'+\delta'-t-\delta)\,\mathbb{1}_{\substack{t+\delta\in\mathcal{T}\\ t'+\delta'\in\mathcal{T}}}\mathbb{1}_{i=i'}\mathbb{1}_{t=t'} && (120)\\
&= \sum_{t\in\mathcal{T}} \tilde{\Omega}(j,j',\delta'-\delta)\,\mathbb{1}_{\substack{t+\delta\in\mathcal{T}\\ t+\delta'\in\mathcal{T}}}\mathbb{1}_{i=i'} && (121)\\
&= \left|\{t\in\mathcal{T} : t+\delta\in\mathcal{T}, t+\delta'\in\mathcal{T}\}\right|\tilde{\Omega}(j,j',\delta'-\delta)\,\mathbb{1}_{i=i'} && (122)\\
&= \beta(\delta,\delta')\,\tilde{\Omega}(j,j',\delta'-\delta)\,\mathbb{1}_{i=i'} && (123)
\end{aligned}$$

Line 119 is from Lemma 1. The other formulas are derived analogously.

This can be written in matrix form as

$$\hat{\mathbf{F}} = \tilde{\boldsymbol{\Omega}} \otimes \mathbf{I} \tag{124}$$

$$\tilde{\boldsymbol{\Omega}} \triangleq \begin{pmatrix} 1 & \boldsymbol{\mu}^\top \otimes \mathbf{1}^\top \\ \boldsymbol{\mu}\otimes\mathbf{1} & \boldsymbol{\Sigma}\otimes\mathbf{I} + \boldsymbol{\mu}\boldsymbol{\mu}^\top \otimes \mathbf{1}\mathbf{1}^\top \end{pmatrix} \tag{125}$$

It is convenient to compute block Cholesky decompositions:

$$\begin{aligned}
\tilde{\boldsymbol{\Omega}} &= \begin{pmatrix} 1 & \\ \boldsymbol{\mu}\otimes\mathbf{1} & \mathbf{B}\otimes\mathbf{I} \end{pmatrix}\begin{pmatrix} 1 & \boldsymbol{\mu}^\top\otimes\mathbf{1}^\top \\ & \mathbf{B}^\top\otimes\mathbf{I} \end{pmatrix} && (126)\\
&\triangleq \mathbf{L}\mathbf{L}^\top && (127)\\
\tilde{\boldsymbol{\Omega}}^{-1} &= \mathbf{L}^{-\top}\mathbf{L}^{-1} && (128)\\
&= \begin{pmatrix} 1 & -\boldsymbol{\mu}^\top\mathbf{B}^{-\top}\otimes\mathbf{1}^\top \\ & \mathbf{B}^{-\top}\otimes\mathbf{I} \end{pmatrix}\begin{pmatrix} 1 & \\ -\mathbf{B}^{-1}\boldsymbol{\mu}\otimes\mathbf{1} & \mathbf{B}^{-1}\otimes\mathbf{I} \end{pmatrix}, && (129)
\end{aligned}$$

where $\mathbf{B}$ is some square root matrix, i.e. $\mathbf{B}\mathbf{B}^\top = \boldsymbol{\Sigma}$ (not necessarily lower triangular).

Now consider PRONG. In the original algorithm, the network is periodically reparameterized such that the activations are white. In our idealized version of the algorithm, we assume this is done after every update. For convenience, we assume that the network is converted to the white parameterizaton immediately before computing the SGD update, and then converted back to its original parameterization immediately afterward. In other words, we apply an affine transformation (Eqn. 24) which whitens the activations:

$$\begin{aligned}
\mathbf{A}_\ell^\dagger = \phi_\ell^\dagger(\mathbf{S}_\ell^\dagger) &= \left(\phi_\ell(\mathbf{S}_\ell^\dagger) - \mathbf{1}\boldsymbol{\mu}^\top\right)\mathbf{B}^{-1} && (130)\\
&= \phi_\ell(\mathbf{S}_\ell^\dagger)\mathbf{B}^{-1} - \mathbf{1}\boldsymbol{\mu}^\top\mathbf{B}^{-1}, && (131)
\end{aligned}$$

where $\mathbf{B}$ is a square root matrix of $\boldsymbol{\Sigma}$, as defined above. This is an instance of Eqn. 24 with $\mathbf{U}_\ell = \mathbf{I}$, $\mathbf{c}_\ell = \mathbf{0}$, $\mathbf{V}_\ell = \mathbf{B}^{-1}$, and $\mathbf{d}_\ell = -\mathbf{B}^{-1}\boldsymbol{\mu}$. The transformed weights which compute the same function as the original network according to Lemma 2 are $\bar{\mathbf{W}}_\ell^\dagger = \bar{\mathbf{W}}_\ell\,[\![\mathbf{B}^{-1}]\!]_H^{-\top}$, where

$$[\![\mathbf{B}^{-1}]\!]_H \triangleq \begin{pmatrix} 1 & -\boldsymbol{\mu}^\top\mathbf{B}^{-\top}\otimes\mathbf{1}^\top \\ & \mathbf{B}^{-1}\otimes\mathbf{I} \end{pmatrix}, \tag{132}$$

is defined according to Eqn. 80. But observe that $[\![\mathbf{B}^{-1}]\!]_H = \mathbf{L}^{-\top}$, where $\mathbf{L}$ is the Cholesky factor of $\tilde{\boldsymbol{\Omega}}$ (Eqn. 129). Therefore, we have

$$\bar{\mathbf{W}}_\ell^\dagger = \bar{\mathbf{W}}_\ell\,\mathbf{L}. \tag{133}$$

We apply Lemma 3 with $\mathbf{P}_\ell^\dagger = \mathbf{I}$ and $\mathbf{R}_\ell^\dagger = \mathbf{I}$. This gives us the update in the original coordinate system:

$$\begin{aligned}
\bar{\mathbf{W}}_\ell^{(k+1)} &\leftarrow \bar{\mathbf{W}}_\ell^{(k)} - \alpha(\nabla_{\bar{\mathbf{W}}_\ell} h)\,\mathbf{L}^{-\top}\mathbf{L}^{-1} && (134)\\
&= \bar{\mathbf{W}}_\ell^{(k)} - \alpha(\nabla_{\bar{\mathbf{W}}_\ell} h)\,\tilde{\boldsymbol{\Omega}}^{-1}. && (135)
\end{aligned}$$

This is equivalent to the approximate natural gradient update where the Fisher block is approximated as $\tilde{\boldsymbol{\Omega}}\otimes\mathbf{I}$. This is the same approximate Fisher block we derived given the assumptions of the theorem (Eqn. 124). □