Natural gradients and K-FAC

Matthew Johnson

June 15, 2017

1 Notation and problem setup

We want to predict a label or outcome y from features x. We postulate that there is a true underlying joint distribution

$$\pi(x, y) = \pi(x)\pi(y \,|\, x), \tag{1}$$

which we want to approximate as

$$\pi(x, y) \approx p_\theta(x, y) = \pi(x)p(y \,|\, x\,;\, \theta), \tag{2}$$

for some parametric family of densities $p(y \,|\, x\,;\, \theta)$ in which the support does not depend on the parameter $\theta \in \mathbb{R}^d$. For example, in a classification problem $p(y \,|\, x\,;\, \theta)$ is the likelihood of a categorical distribution, while in a regression problem $p(y \,|\, x\,;\, \theta)$ is often taken to be a Gaussian likelihood.

The population problem is to choose θ to minimize

$$\mathrm{KL}(\pi(x, y) \,\|\, p_\theta(x, y)) = \mathbb{E}_{\pi(x)} \mathrm{KL}(\pi(y \,|\, x) \,\|\, p(y \,|\, x\,;\, \theta)), \tag{3}$$

where $\mathrm{KL}(p(x) \,\|\, q(x)) = \mathbb{E}_{p(x)} \log \frac{p(x)}{q(x)}$. Given a fixed training set $\{(x_n, y_n)\}_{n=1}^N$, with $(x_n, y_n) \overset{\text{iid}}{\sim} \pi(x, y)$, the empirical distribution is

$$\hat{\pi}(x, y) = \frac{1}{N} \sum_{n=1}^N \delta(x - x_n)\delta(y - y_n), \tag{4}$$

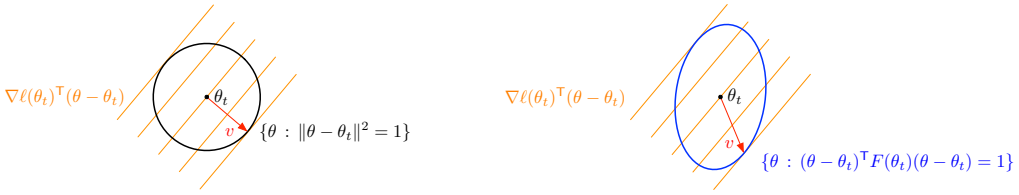
and the empirical problem is to choose θ to minimize

$$\mathrm{KL}(\hat{\pi}(x, y) \,\|\, \hat{\pi}(x)p(y \,|\, x\,;\, \theta)) = \mathbb{E}_{\hat{\pi}(x)} \mathrm{KL}(\hat{\pi}(y \,|\, x) \,\|\, p(y \,|\, x\,;\, \theta)) \tag{5}$$

$$= -\frac{1}{N} \sum_{n=1}^N \log p(y_n \,|\, x_n\,;\, \theta) + \text{const.}, \tag{6}$$

which is the maximum likelihood problem. For example, in a classification problem, the choice of a categorical likelihood for $p(y \,|\, x\,;\, \theta)$ leads to a cross-entropy loss to be minimized, while in a regression problem a Gaussian likelihood for $p(y \,|\, x\,;\, \theta)$ leads to a least-squares loss to be minimized.

For both the population and empirical problems, we denote the loss as $\ell(\theta)$, with the only meaningful difference being whether the expectations are taken over the population distribution or the empirical distribution.



(a) The Euclidean steepest descent direction. (b) The Fisher steepest descent direction.

Figure 1: Steepest descent directions in the Euclidean and Fisher metrics.

2 Gradient descent as Euclidean steepest descent

We can use stochastic gradient descent (SGD) to solve the population or empirical problems, using the iteration

$$\theta_{t+1} = \theta_t - \alpha_t \nabla\ell(\theta_t), \tag{7}$$

for a step size $\alpha_t > 0$, where we suppress notation to reflect that $\nabla\ell(\theta_t)$ is a sampled stochastic estimate of the loss gradient.[1] The update step direction is the steepest descent direction in the linearized approximation to the objective using the Euclidean metric:

$$-\frac{\nabla\ell(\theta_t)}{\|\nabla\ell(\theta_t)\|} = \arg\min_v \left\{\ell(\theta_t) + \nabla\ell(\theta_t)^\mathsf{T} v\right\} \tag{8}$$

$$\text{s.t.} \quad \|v\|^2 \le 1. \tag{9}$$

To see this, note that the linearized objective is extremized when v is collinear with $\nabla\ell(\theta_t)$. See Figure 1a for an illustration.

The Euclidean metric does not take into account the fact that small changes to θ in some directions may significantly alter the predictive distribution $p(y \mid x\,;\,\theta)$, while changes in other directions with the same Euclidean size may only have small effects on the predictive distribution. Not factoring this sensitivity into the metric used in the definition of steepest descent can lead to poor update directions and slow optimization. In other words, using the Euclidean steepest descent direction ignores a natural notion of curvature in distribution space, and instead treats the parameter space as flat.

3 The Fisher metric

We want a way to measure distances in the space of possible θ values that tracks how different the distribution indexed by θ is from the nearby distribution indexed by $\theta + \mathrm{d}\theta$. It suffices to come up with a local definition of distance, both because want to do local search (i.e. gradient-based optimization) and because the local definition of distance we develop can be extended to a global one.

To define such a local measure of distance, we start from the KL divergence,

$$\mathrm{KL}(\theta_1, \theta_2) \triangleq \mathrm{KL}(p_{\theta_1}(x, y) \,\|\, p_{\theta_2}(x, y)). \tag{10}$$

[1] See Section D for details on our derivative notation.

We Taylor-expand the KL divergence at a point θ up to its first nonzero term, which is the quadratic term:

$$\mathrm{KL}(\theta, \theta + \mathrm{d}\theta) \approx \mathrm{KL}(\theta, \theta) + \nabla_2 \mathrm{KL}(\theta, \theta)^\mathsf{T} \mathrm{d}\theta + \frac{1}{2} \mathrm{d}\theta^\mathsf{T} \nabla_{22}^2 \mathrm{KL}(\theta, \theta) \mathrm{d}\theta \tag{11}$$

$$= \frac{1}{2} \mathrm{d}\theta^\mathsf{T} \nabla_{22}^2 \mathrm{KL}(\theta, \theta) \mathrm{d}\theta \tag{12}$$

$$\triangleq \frac{1}{2} \mathrm{d}\theta^\mathsf{T} F(\theta) \mathrm{d}\theta, \tag{13}$$

where ∇_2 denotes differentiation with respect to the second argument, and $F(\theta)$ is the Fisher information matrix. That the first two terms in the Taylor expansion are zero and the Fisher information matrix is positive semidefinite follows from the fact that θ is a local minimizer of the smooth function $\theta' \mapsto \mathrm{KL}(\theta, \theta')$ with optimal value 0. For the remainder of the document, we assume that $F(\theta)$ is positive definite.

By using the Fisher information matrix $F(\theta)$ to assign a smoothly-varying positive definite matrix to each point θ, we say $F(\theta)$ defines a Riemannian metric on the space. This definition of the Fisher information matrix does not depend on the loss function being related to the model log likelihood, though it is particularly useful for optimizing such losses.

4 Natural gradients

We can now define a search direction as steepest descent in the Fisher metric,

$$\arg\min_v \left\{ \ell(\theta_t) + \nabla \ell(\theta_t)^\mathsf{T} v \right\} \tag{14}$$

$$\text{s.t. } v^\mathsf{T} F(\theta_t) v \leq 1. \tag{15}$$

By writing the Lagrangian for this problem and inspecting the first-order necessary conditions for optimality, we have

$$\nabla \ell(\theta_t) + 2\lambda F(\theta_t) v = 0, \tag{16}$$

for $\lambda \geq 0$, and hence the optimal v is a positive multiple of $-F(\theta_t)^{-1} \nabla \ell(\theta_t)$ (with unit Fisher norm). The natural gradient update step is then

$$\theta_{t+1} = \theta_t - \alpha_t F(\theta_t)^{-1} \nabla \ell(\theta_t). \tag{17}$$

See Figure 1b for an illustration.

Interpreting this iteration as a (stochastic) quasi-Newton method, we call $F(\theta_t)$ the preconditioner matrix. A variant is to use the damped matrix $F(\theta_t) + \beta_t I$ for some $\beta_t > 0$ as a preconditioner, which ensures the preconditioner is positive definite and interpolates between the gradient and natural gradient search directions.

5 Natural gradient descent and Newton directions

The natural gradient iteration (17) can be compared with an iteration based on Newton directions,

$$\theta_{t+1} = \theta_t - \alpha_t H(\theta_t)^{-1} \nabla \ell(\theta_t), \tag{18}$$

where $H(\theta_t) = \nabla^2 \ell(\theta_t)$ is the Hessian matrix of the loss.

First, because $F(\theta)$ is a positive definite matrix, the natural gradient search direction is always a descent direction in the sense that for a small enough step size the update must on average decrease the value of a smooth objective when not at a stationary point. The Newton direction does not have this property, and as a result Newton methods are attracted to stationary points $\nabla \ell(\theta) = 0$ of any signature (i.e. also saddle points and local optima). Like gradient descent, natural gradient descent is only attracted to local minima.

Second, as we show in the following sections, $F(\theta)$ can often be estimated from samples, or accessed implicitly via matrix-vector products, more efficiently than the Hessian $H(\theta)$.

Finally, as we show in Section A, for a large class of common models the Fisher information matrix can be related explicitly to the Hessian matrix, and indeed can be interpreted as a positive definite approximation to the Hessian.

6 Approximate natural gradient methods

For models with many parameters, like neural networks, computing the exact natural gradient direction for each update can be computationally expensive. That is, when the dimension d of the parameter vector $\theta \in \mathbb{R}^d$ is large, evaluating $F(\theta)^{-1} \nabla \ell(\theta)$ may be prohibitively expensive, since $F(\theta)$ is dense in general and direct factor-solve methods have time complexity that scales like d^3. Indeed, for models with millions of parameters even forming an explicit version of $F(\theta)$ in memory may be too expensive. As a result, for large models we must turn to approximate natural gradient methods which avoid these expensive operations.

There are two main approximation strategies. The first is to rely on implicit access to $F(\theta)$, evaluating instead only Fisher-vector products $v \mapsto F(\theta)v$ as a subroutine in a truncated conjugate gradient (CG) method or other Krylov subspace method for approximating $F(\theta)^{-1} \nabla \ell(\theta)$. These Fisher-vector products can be evaluated efficiently using a combination of forward- and reverse-mode automatic differentiation by following the formula in Section A, so that each evaluation of $v \mapsto F(\theta)v$ costs about twice as much as an evaluation of the objective for typical models. However, this per-update cost is substantial relative to standard stochastic gradient updates; we might require tens or hundreds of evaluations of $v \mapsto F(\theta)v$ just to form the approximate update direction, depending on how clustered the eigenvalues of $F(\theta)$ are. For a similar cost, we could compute many SGD updates. As a result, SGD is often preferred to methods based solely on implicit access to $F(\theta)$.

The second main approximation strategy is to maintain an explicit approximation to the Fisher information matrix, $\widetilde{F}(\theta) \approx F(\theta)$, and to compute and cache $\widetilde{F}(\theta)^{-1}$ across parameter update steps. These explicit approximations involve tradeoffs: simpler approximations are easier to store and invert but do not accurately represent the full Fisher matrix.

The K-FAC algorithm uses an explicit Fisher information matrix approximation that is adapted to neural network training. It uses block diagonal and Kronecker-factorized matrix structures, with each block corresponding to the parameters of a neural network layer, to make $\widetilde{F}(\theta)^{-1}$ efficient to store, invert, and update. Further, the structure of the approximation can be interpreted as making certain reasonable statistical modeling assumptions based on the neural network architecture being trained. We focus on K-FAC for the remainder of the document, starting with the statistical characterization of the Fisher information matrix used to motivate it.

7 The Fisher information matrix as a covariance

To develop approximate natural gradient methods that rely on explicit estimates of the Fisher information matrix, there is a useful alternative characterization of the Fisher information matrix as a covariance of certain random gradients. That is, it is the covariance matrix of gradients of the model's log predictive density under random targets sampled from the model's predictive distribution. In symbols, denoting $\xi(\theta) = \log p(y \,|\, x\,;\, \theta)$, we have

$$F(\theta) = \mathbb{E}_{p_\theta(x,y)}\big[\nabla\xi(\theta)\nabla\xi(\theta)^\mathsf{T}\big], \quad \mathbb{E}_{p_\theta(x,y)}\big[\nabla\xi(\theta)\big] = 0. \tag{19}$$

To derive this result, we can first show that for every x, for random targets y sampled from the model distribution $p(y \,|\, x\,;\, \theta)$, the expectation of $\nabla\xi(\theta)$ is zero,

$$\mathbb{E}_{p(y \,|\, x\,;\, \theta)}\left[\nabla\xi(\theta)\right] = 0. \tag{20}$$

This fact follows from the normalization of the probability density $p(y \,|\, x\,;\, \theta)$, and an assumption that differentiation and integration may be exchanged:

$$\mathbb{E}_{p(y \,|\, x\,;\, \theta)}\nabla_\theta \log p(y \,|\, x\,;\, \theta) = \int \nabla_\theta p(y \,|\, x\,;\, \theta)\, \mathrm{d}y = \nabla_\theta \int p(y \,|\, x\,;\, \theta)\, \mathrm{d}y = \nabla_\theta 1 = 0. \tag{21}$$

Next, to show that the covariance of $\nabla\xi(\theta)$ under $p_\theta(x, y) = \pi(x)p_\theta(y \,|\, x\,;\, \theta)$ is the Fisher information matrix, we apply ∇_θ to both sides of $\mathbb{E}_{p_\theta(x,y)}\nabla\xi(\theta) = 0$ to yield

$$\mathbb{E}_{p_\theta(x,y)}\left[\nabla\xi(\theta)\nabla\xi(\theta)^\mathsf{T}\right] + \mathbb{E}_{p_\theta(x,y)}\left[\nabla^2\xi(\theta)\right] = 0, \tag{22}$$

and observe that $\mathbb{E}_{p_\theta(x,y)}\left[\nabla^2\xi(\theta)\right] = -\nabla^2_{22}\,\mathrm{KL}(\theta, \theta)$.

As a result, we can estimate the Fisher information matrix from Monte Carlo samples of $x \sim \pi(x)$ and the model's predictive distribution $y \,|\, x \sim p(y \,|\, x\,;\, \theta)$ using only first-order derivative information, rather than requiring any Hessian estimates.

8 Kronecker-factored approximate curvature (K-FAC)

To develop the K-FAC algorithm, we first introduce some notation for neural networks. For simplicity, we restrict our attention to the case of feed-forward networks comprising only fully-connected layers with no parameter sharing across layers. Let the model predictive log density have the form

$$\log p(y \,|\, x\,;\, \theta) = f(y, g(x, \theta)), \tag{23}$$

where g is the network prediction function. For example, in a classification problem, $g(x, \theta)$ may be the logits produced for the features x, while $f(y, g(x, \theta))$ is the (negative) cross-entropy loss against the target label y. We decompose g into layers, writing $g(x, \theta) = s_K$, where for $k = 1, 2, \ldots, K$ we have

$$s_k = W_k a_k + b_k, \tag{24}$$

$$a_{k+1} = \sigma(s_k), \tag{25}$$

taking $a_1 = x$ and $\theta = \{(W_k, b_k)\}_{k=1}^K$. To simplify notation, we collect the weights and biases by writing

$$\bar{a}_k = \begin{bmatrix} a_k \\ 1 \end{bmatrix}, \qquad \overline{W}_k = \begin{bmatrix} W_k & b_k \end{bmatrix}, \qquad s_k = \overline{W}_k \bar{a}_k. \tag{26}$$

For each layer k, we call $\bar{a}_k$ the input activations (in homogeneous coordinates), s_k the output pre-activations, and $\overline{W}_k$ the parameters.

We can decompose the Fisher information matrix according to the layer structure of g. Recall from Section 7 that the Fisher information matrix can be expressed as a covariance matrix of random gradients,

$$F(\theta) = \mathbb{E}_{p_\theta(x,y)}\big[(\nabla_\theta \log p(y \,|\, x\,;\,\theta))(\nabla_\theta \log p(y \,|\, x\,;\,\theta))^\mathsf{T}\big]. \tag{27}$$

For each layer k, recall that the gradient of the model log predictive density with respect to the weight matrix for layer k as can be written as

$$\nabla_{\overline{W}_k} \log p(y \,|\, x\,;\,\theta) = \bar{a}_k g_k^\mathsf{T}, \qquad g_k = \nabla_{s_k} \log p(y \,|\, x\,;\,\theta), \tag{28}$$

where g_k is the gradient of $\log p(y \,|\, x\,;\,\theta)$ with respect to the output pre-activations for layer k. We can write the vectorized (flattened) gradient as

$$\operatorname{vec}\big(\nabla_{\overline{W}_k} \log p(y \,|\, x\,;\,\theta)\big) = \bar{a}_k \otimes g_k, \tag{29}$$

where $\otimes$ denotes the Kronecker product, defined on matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{s\times r}$ as

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix} \in \mathbb{R}^{ms\times nr}. \tag{30}$$

As a result, if we form a block partition of the Fisher information matrix $F(\theta)$ corresponding to the vectorized parameters of each layer, we can write each block as

$$\begin{aligned} F_{ij}(\theta) &= \mathbb{E}_{p_\theta(x,y)}\big[\operatorname{vec}\big(\nabla_{\overline{W}_i} \log p(y \,|\, x\,;\,\theta)\big) \operatorname{vec}\big(\nabla_{\overline{W}_j} \log p(y \,|\, x\,;\,\theta)\big)^\mathsf{T}\big] && (31)\\ &= \mathbb{E}_{p_\theta(x,y)}\big[\operatorname{vec}(\bar{a}_i g_i^\mathsf{T}) \operatorname{vec}(\bar{a}_j g_j^\mathsf{T})^\mathsf{T}\big] && (32)\\ &= \mathbb{E}_{p_\theta(x,y)}\big[(\bar{a}_i \otimes g_i)(\bar{a}_j^\mathsf{T} \otimes g_j^\mathsf{T})\big] && (33)\\ &= \mathbb{E}_{p_\theta(x,y)}\big[\bar{a}_i\bar{a}_j^\mathsf{T} \otimes g_i g_j^\mathsf{T}\big], \qquad i,j = 1,2,\ldots,K. && (34) \end{aligned}$$

The K-FAC approximation $\widetilde{F}(\theta)$ to the Fisher information matrix $F(\theta)$ is derived by applying two statistical assumptions to the expression in (34). The first assumption is that the statistics of activations and pre-activation gradients are independent across layers, so that $\widetilde{F}_{ij}(\theta) = 0$ for $i \neq j$ and hence $\widetilde{F}(\theta)$ is a block-diagonal matrix. The second assumption is that the activations and pre-activation gradients are independent, so that

$$\begin{aligned} \widetilde{F}_{kk}(\theta) &= \mathbb{E}_{p_\theta(x,y)}\left[\bar{a}_i\bar{a}_j^\mathsf{T} \otimes g_i g_j^\mathsf{T}\right] && (35)\\ &= \mathbb{E}_{\pi(x)}\left[\bar{a}_i\bar{a}_j^\mathsf{T}\right] \otimes \mathbb{E}_{p_\theta(x,y)}\left[g_i g_j^\mathsf{T}\right] && (36)\\ &= A_k \otimes G_k. && (37) \end{aligned}$$

As a result, the K-FAC approximation to the Fisher information matrix is block-diagonal with each diagonal block, corresponding to the parameters of a layer, a Kronecker product of two smaller matrices.

It is informative to estimate the sizes of the matrices involved in a simple example. For a network that has $K = 10$ layers, with the inputs and outputs to each layer having 1000 units so that each $\overline{W}_k$ is approximately a 1000×1000 matrix, we have a total of

about $d = 10 \cdot 10^6$ parameters. While the exact Fisher information matrix $F(\theta)$ is a dense $d \times d$ matrix in general with up to 10^{14} nonzero entries, the nonzero elements of the K-FAC approximation $\widetilde{F}(\theta)$ can be stored as 10 digaonal blocks. Each diagonal block is of size $10^6 \times 10^6$, but is factorized into a Kronecker product of two 1000×1000 matrices, thus requiring a total of only $2 \cdot 10^7$ nonzero entries to be stored.

The Kronecker factorization structure makes the K-FAC approximation $\widetilde{F}(\theta)$ easy to work with not only for memory efficiency but also for numerical operation efficiency, since we can use the formulas

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}, \qquad (A \otimes B)\,\mathrm{vec}(C) = \mathrm{vec}(ACB^\mathsf{T}), \tag{38}$$

to compute and apply explicit inverses.

Paramter sharing across layers can complicate the approximation because the partial derivative expression (28) no longer applies. In particular, the expression for the gradient with respect to a shared weight matrix can be the sum of many terms, destroying the rank-one gradient structure that led to the Kronecker-rank-one structure used in the Fisher approximation. Convolutional layers also share weights within a layer, but by applying additional assumptions (namely spatial homogeneity of the activation statistics and spatially uncorrelated derivatives) an efficient K-FAC approximation for convolutions can be derived. In general, each weight sharing pattern requires its own approximation strategy.

A The Fisher and generalized Gauss-Newton matrices

For many models, especially neural network models, the loss $\ell(\theta)$ can be written as

$$\ell(\theta) = \mathbb{E}_{\pi(x,y)} f(g(\theta)), \tag{39}$$

where $g : \mathbb{R}^d \to \mathbb{R}^m$ may map the value of the parameters to the prediction for a particular input x (where x is suppressed from the notation) and $f : \mathbb{R}^m \to \mathbb{R}^1$ may map the prediction to a loss on a particular target y (also suppressed from the notation). Using this decomposition we can expand the Hessian of the loss function as

$$H(\theta) = \nabla^2 \ell(\theta) = \mathbb{E}_{\pi(x,y)} \left[\nabla g(\theta) \nabla^2 f(g(\theta)) \nabla g(\theta)^\mathsf{T} + \nabla^2 g(\theta) \nabla f(g(\theta)) \right], \tag{40}$$

where the second term yields a matrix by contracting a three-index tensor against a one-index tensor.

For models in which the loss is the average negative log likelihood of the model, we can relate this expression for the Hessian $H(\theta)$ to the Fisher information matrix $F(\theta)$. In particular, consider models where we can write the loss as

$$\ell(\theta) = \mathbb{E}_{\pi(x,y)} f(g(\theta)) = -\mathbb{E}_{\pi(x,y)} \log p(y \,|\, x\,;\, \theta). \tag{41}$$

Using the chain rule the Fisher information matrix can be written as

$$F(\theta) = \mathbb{E}_{p_\theta(x,y)} \left[\nabla g(\theta) \nabla^2 f(g(\theta)) \nabla g(\theta)^\mathsf{T} \right]. \tag{42}$$

Comparing this expression for $F(\theta)$ to the expression for $H(\theta)$ in (40), we see that the Fisher information matrix differs from the Hessian in two ways: first, the integrand for the Fisher information matrix only contains one of the two terms in the Hessian integrand, indeed the term that is always positive semidefinite, and second the expectation over the

targets $y \,|\, x$ is taken with respect to the model predictive distribution for the Fisher and the data conditional distribution for the Hessian. The expression for $F(\theta)$ in (42) also makes clear how we can efficiently estimate $v \mapsto F(\theta)v$ using automatic differentiation, since using samples $(x, y) \sim p_\theta(x, y)$ we can write

$$F(\theta)v = (J^\mathsf{T}(M(Jv))), \qquad J^\mathsf{T} = \nabla g(\theta), \qquad M = \nabla^2 f(g(\theta)), \tag{43}$$

which can be evaluated using a Jacobian-vector product for g (forward-mode), a small Hessian-vector product for f, and a vector-Jacobian product for g (reverse-mode).

For exponential family models, the Hessian and the Fisher can be even more closely compared. Consider the case where $p(y \,|\, x\,;\, \theta)$ is a regular exponential family with $g(\theta)$ its natural parameter. In that case, we can write

$$p(y \,|\, x\,;\, \theta) = \exp\left\{\langle g(\theta),\, t(y)\rangle - \mathcal{A}(g(\theta))\right\}. \tag{44}$$

Here we have that

$$\nabla^2 f(g(\theta)) = -\nabla^2 \log p(y \,|\, x\,;\, \theta) = \nabla^2 \mathcal{A}(g(\theta)), \tag{45}$$

which does not depend on y. As a result, for exponential family models we can write

$$H(\theta) = F(\theta) + \mathbb{E}_{\pi(x,y)}\left[\nabla^2 g(\theta) \nabla f(g(\theta))\right], \tag{46}$$

$$F(\theta) = \mathbb{E}_{\pi(x)}\left[\nabla g(\theta) \nabla^2 \mathcal{A}(g(\theta)) \nabla g(\theta)^\mathsf{T}\right], \tag{47}$$

and hence the comparison between the Fisher information matrix and the Hessian matrix of the loss is more direct. In particular, we can interpret $F(\theta)$ as a positive definite approximation to the Hessian, formed by dropping a term that might be indefinite. Note also that for these models we can evaluate $v \mapsto F(\theta)v$ without drawing Monte Carlo samples for y, and instead only requiring samples of $x \sim \pi(x)$.

Finally, we note that when f is a convex function, the expression

$$F(\theta) = \mathbb{E}_{p_\theta(x,y)}\left[\nabla g(\theta) \nabla^2 f(g(\theta)) \nabla g(\theta)\right] \tag{48}$$

shows that the Fisher information matrix is a generalized Gauss-Newton matrix, and in the special case where f is the squared Euclidean norm, it is a classical Gauss-Newton matrix. For exponential family models, f is always a convex function.

B Update invariance to reparameterization

Natural gradients are invariant to smooth reparameterizations of the model predictive density $\log p(y \,|\, x\,;\, \theta)$, and natural gradient updates are invariant to linear (or affine) transformations of θ. That is, consider defining another parameterization $\theta = A\theta'$ for some invertible matrix A.[2] Denoting the loss and Fisher information matrix in the new parameterization as $\ell'(\theta')$ and $F'(\theta')$, respectively, using the chain rule we have

$$\nabla \ell'(\theta') = A^\mathsf{T} \nabla \ell(A\theta'), \qquad F'(\theta') = A^\mathsf{T} F(A\theta') A, \tag{49}$$

[2] In general we could take $A = A(\theta')$ to be a smooth function of θ'. Here we keep the notation simple.

and hence the update direction in the new parameterization is

$$F'(\theta')^{-1}\nabla\ell'(\theta') = A^{-1}F(A\theta')^{-1}A^{-\mathsf{T}}A^{\mathsf{T}}\nabla\ell(A\theta') \tag{50}$$

$$= A^{-1}F(A\theta')^{-1}\nabla\ell(A\theta'). \tag{51}$$

As a result, the update steps agree in the two parameterizations, in the sense that

$$\theta - \alpha F(\theta)^{-1}\nabla\ell(\theta) = A\theta' - \alpha F(A\theta')^{-1}\nabla\ell(A\theta') = A\left(\theta' - \alpha F'(\theta')^{-1}\nabla\ell'(\theta')\right). \tag{52}$$

One way to interpret the invariance of natural gradients to smooth reparameterization is that natural gradients are intrinsically defined on the manifold of predictive densities and hence are consistent no matter what parameterization, or coordinate system, we work with.

Because it is based on an approximation to the Fisher information matrix, the K-FAC update is invariant to a smaller set of reparameterizations. In particular, it is invariant to affine transformation of the activations within each layer. The extension to convolutional layers, not discussed here, is invariant to spatially-homogeneous pointwise affine transformations of the activations.

C Metric and pseudo-metric pullback more generally

In Section 3 we took a pseudo-metric, in that case KL divergence on predictive distributions indexed by θ, and pulled it back to define a Riemannian metric on our parameter space θ. This construction applies to more than just KL divergence, and we can use it more generally to define metrics and corresponding steepest-descent updates. For example, if we have smooth functions $g : \mathbb{R}^d \to \mathcal{S}$ and $f : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ satisfying

1. $f(a, b) \geq 0 \quad \forall a, b \in \mathcal{S}$,
2. $f(a, b) = 0$ if and only if $a = b$,

then taking $h(x, y) = f(g(x), g(y))$ we can define a Riemannian metric on $\mathbb{R}^d$ via

$$M(x) = \partial g(x)^{\mathsf{T}}\partial_{22}^2 f(g(x), g(x))\partial g(x). \tag{53}$$

As before, this corresponds to Taylor-expanding $h(x, x)$ to second order, and the facts that $h(x, x) = 0$, $\partial_2 h(x, x) = 0$, and $M(x) \succeq 0$ follow from the fact that x is a local minimum of the smooth function $y \mapsto f(g(x), g(y))$. Alternatively, we can see that the second term is zero in the Hessian expression

$$\partial_{22}^2 h(x, x) = \partial g(x)^{\mathsf{T}}\partial_{22}^2 f(g(x), g(x))\partial g(x) + \partial_2 f(g(x), g(x))\partial^2 g(x), \tag{54}$$

since $\partial_2 f(g(x), g(x)) = 0$ follows from the fact that $g(x)$ is a local minimum of the smooth function $a \mapsto f(g(x), a)$.

D Derivative notation

If $f : \mathbb{R}^n \to \mathbb{R}^m$ is a continuously differentiable function, we denote the partial derivative as $\partial f : \mathbb{R}^n \to \mathbb{R}^n \to \mathbb{R}^m$, so that $\partial f(x)$ can be identified with the $m \times n$ Jacobian matrix

$$\partial f(x) = \frac{\partial f(x)}{\partial x} = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_m(x)}{\partial x_1} & \cdots & \frac{\partial f_m(x)}{\partial x_n} \end{bmatrix}, \tag{55}$$

where f_i is the ith coordinate function of f for $i = 1, 2, \ldots, m$. The gradient matrix of f, denoted $\nabla f(x)$, is the transpose of the Jacobian matrix, so that $\nabla f(x) = \partial f(x)^{\mathsf{T}}$. That is, $\nabla f(x)$ is the $n \times m$ matrix in which the ith column is the gradient $\nabla f_i(x)$ of f_i,

$$\nabla f(x) = \begin{bmatrix} \nabla f_1(x) & \cdots & \nabla f_m(x) \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \cdots & \frac{\partial f_m(x)}{\partial x_1} \\ \vdots & & \vdots \\ \frac{\partial f_1(x)}{\partial x_n} & \cdots & \frac{\partial f_m(x)}{\partial x_n} \end{bmatrix} \tag{56}$$

If $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable with continuously differentiable partial derivatives, then we define the Hessian matrix of f evaluated at x, denoted $\nabla^2 f(x)$ or $\partial^2 f(x)$, to be the symmetric matrix in which the ijth entry is the function $\partial^2 f(x)/\partial x_i \partial x_j$.

Finally, if $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is a function of (x, y) with $x \in R^n$ and $y \in \mathbb{R}^m$, we write

$$\nabla_1 f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x,y)}{\partial x_m} \end{bmatrix}, \qquad \partial_1 f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x_1} & \cdots & \frac{\partial f(x,y)}{\partial x_m} \end{bmatrix}, \tag{57}$$

$$\nabla_2 f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial y_1} \\ \vdots \\ \frac{\partial f(x,y)}{\partial y_n} \end{bmatrix}, \qquad \partial_2 f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial y_1} & \cdots & \frac{\partial f(x,y)}{\partial y_n} \end{bmatrix} \tag{58}$$

$$\nabla^2_{11} f(x,y) = \partial^2_{11} f(x,y) = \left(\frac{\partial^2 f(x,y)}{\partial x_i \partial x_j} \right), \quad \nabla^2_{22} f(x,y) = \partial^2_{22} f(x,y) = \left(\frac{\partial^2 f(x,y)}{\partial y_i \partial y_j} \right), \tag{59}$$

$$\nabla^2_{12} f(x,y) = \partial^2_{12} f(x,y)^{\mathsf{T}} = \left(\frac{\partial^2 f(x,y)}{\partial x_i \partial y_j} \right). \tag{60}$$

We occasionally also use variable subscripts on ∇ when the notation is not ambiguous, following the usual convention that

$$\nabla_x f(x,y) = \nabla_1 f(x,y), \qquad \nabla_y f(x,y) = \nabla_2 f(x,y). \tag{61}$$