Optimizing Neural Networks with Kronecker-factored Approximate Curvature

James Martens JMARTENS@CS.TORONTO.EDU
Roger Grosse RGROSSE@CS.TORONTO.EDU
Department of Computer Science, University of Toronto

Abstract

We propose an efficient method for approximating natural gradient descent in neural networks which we call Kronecker-factored Approximate Curvature (K-FAC). K-FAC is based on an efficiently invertible approximation of a neural network's Fisher information matrix which is neither diagonal nor low-rank, and in some cases is completely non-sparse. It is derived by approximating various large blocks of the Fisher (corresponding to entire layers) as being the Kronecker product of two much smaller matrices. While only several times more expensive to compute than the plain stochastic gradient, the updates produced by K-FAC make *much* more progress optimizing the objective, which results in an algorithm that can be much faster than stochastic gradient descent with momentum in practice. And unlike some previously proposed approximate natural-gradient/Newton methods which use high-quality non-diagonal curvature matrices (such as Hessian-free optimization), K-FAC works very well in highly stochastic optimization regimes. This is because the cost of storing and inverting K-FAC's approximation to the curvature matrix does not depend on the amount of data used to estimate it, which is a feature typically associated only with diagonal or low-rank approximations to the curvature matrix.

1. Background and notation

1.1. Neural Networks

We begin by defining the basic notation for feed-forward neural networks which we will use throughout this paper.

A neural network transforms its input $a_0 = x$ to an output $f(x, \theta) = a_\ell$ through a series of ℓ layers, each of which consists of a bank of units/neurons. The units each receive as input a weighted sum of the outputs of units from the previous layer and compute their output via a nonlinear "activation" function. We denote by s_i the vector of these weighted sums for the i-th layer, and by a_i the vector of unit outputs (aka "activities"). The precise computation performed at each layer $i \in \{1, \dots, \ell\}$ is given as follows:

$$s_i = W_i \bar{a}_{i-1} \qquad a_i = \phi_i(s_i)$$

where ϕ_i is an element-wise nonlinear function, W_i is a weight matrix, and $\bar{a}_i$ is defined as the vector formed by appending to a_i an additional homogeneous coordinate with value 1. Note that we do not include explicit bias parameters here as these are captured implicitly through our use of homogeneous coordinates. In particular, the last column of each weight matrix W_i corresponds to what is usually thought of as the "bias vector".

We will define θ to be the vector consisting of all of the network's parameters concatenated together, i.e. $[\mathrm{vec}(W_1)^\top \mathrm{vec}(W_2)^\top \dots \mathrm{vec}(W_\ell)^\top]^\top$, where vec is the operator which vectorizes matrices by stacking their columns together.

We let $L(y, z)$ denote the loss function which measures the disagreement between a prediction z made by the network, and a target y. The training objective function $h(\theta)$ is the average (or expectation) of losses $L(y, f(x, \theta))$ with respect to a training distribution $\hat{Q}_{x,y}$ over input-target pairs (x, y). $h(\theta)$ is a proxy for the objective which we actually care about but don't have access to, which is the expectation of the loss taken with respect to the true data distribution $Q_{x,y}$.

We will assume that the loss is given by the negative log probability associated with a simple predictive distribution $R_{y|z}$ for y parameterized by z, i.e. that we have $L(y, z) = -\log r(y|z)$ where r is $R_{y|z}$'s density function. This is the case for both the standard least-squares and cross-entropy objective functions, where the predictive distributions are multivariate normal and multinomial, respectively.

We will let $P_{y|x}(\theta) = R_{y|f(x,\theta)}$ denote the conditional distribution defined by the neural network, as parameterized by θ, and $p(y|x, \theta) = r(y|f(x, \theta))$ its density function. Note that minimizing the objective function $h(\theta)$ can be seen as maximum likelihood learning of the model $P_{y|x}(\theta)$.

For convenience we will define the following additional no-

Proceedings of the 32^{nd} International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37.

tation:

$$\mathcal{D}v = \frac{\mathrm{d}L(y, f(x,\theta))}{\mathrm{d}v} = -\frac{\mathrm{d}\log p(y|x,\theta)}{\mathrm{d}v} \quad \text{and} \quad g_i = \mathcal{D}s_i$$

Algorithm 1 (in Appendix A) shows how to compute the gradient $\mathcal{D}\theta$ of the loss function of a neural network using standard backpropagation.

1.2. The Natural Gradient

Because our network defines a conditional model $P_{y|x}(\theta)$, it has an associated Fisher information matrix (which we will simply call "the Fisher") which is given by

$$F = \mathrm{E}\left[\frac{\mathrm{d}\log p(y|x,\theta)}{\mathrm{d}\theta}\frac{\mathrm{d}\log p(y|x,\theta)}{\mathrm{d}\theta}^\top\right] = \mathrm{E}[\mathcal{D}\theta\mathcal{D}\theta^\top]$$

Here, the expectation is taken with respect to the data distribution Q_x over inputs x, and the model's predictive distribution $P_{y|x}(\theta)$ over y. Since we usually don't have access to Q_x, and the above expectation would likely be intractable even if we did, we will instead compute F using the training distribution $\hat{Q}_x$ over inputs x.

The well-known natural gradient (Amari, 1998) is defined as $F^{-1}\nabla h(\theta)$. Motivated from the perspective of information geometry (Amari and Nagaoka, 2000), the natural gradient defines the direction in parameter space which gives the largest change in the objective per unit of change in the model, as measured by the KL-divergence. This is to be contrasted with the standard gradient, which can be defined as the direction in parameter space which gives the largest change in the objective per unit of change in the parameters, as measured by the standard Euclidean metric.

The natural gradient also has links to several classical ideas from optimization. It can be shown (Martens, 2014; Pascanu and Bengio, 2014) that the Fisher is equivalent to the Generalized Gauss-Newton matrix (GGN) (Schraudolph, 2002; Martens and Sutskever, 2012) in certain important cases, which is a well-known positive semi-definite approximation to the Hessian of the objective function.

The GGN has served as the curvature matrix of choice in Hessian-free optimization (HF) (Martens, 2010) and related methods, and so in light of its equivalence to the Fisher, these 2nd-order methods can be seen as approximate natural gradient methods. And perhaps more importantly from a practical perspective, natural gradient-based optimization methods can conversely be viewed as 2nd-order optimization methods, which as pointed out by Martens (2014)), brings to bare the vast wisdom that has accumulated about how to make such methods work well in both theory and practice (e.g Nocedal and Wright, 2006).

For some good recent discussion and analysis of the natural gradient, see Arnold et al. (2011); Martens (2014); Pascanu and Bengio (2014).

2. A block-wise Kronecker-factored Fisher approximation

The main computational challenge associated with using the natural gradient is computing F^{-1} (or its product with ∇h). For large networks, with potentially millions of parameters, computing this inverse naively is computationally impractical. In this section we develop an initial approximation of F which will be a key ingredient in deriving our efficiently computable approximation to F^{-1} and the natural gradient.

Note that $\mathcal{D}\theta = [d_1^\top\ d_2^\top\ \cdots\ d_\ell^\top]^\top$ where $d_i = \mathrm{vec}(\mathcal{D}W_i)$ and so $F = \mathrm{E}[\mathcal{D}\theta\mathcal{D}\theta^\top]$ can be viewed as an ℓ by ℓ block matrix, with the (i,j)-th block $F_{i,j}$ given by $F_{i,j} = \mathrm{E}\left[d_i d_j^\top\right]$.

Noting that $\mathcal{D}W_i = g_i\bar{a}_{i-1}^\top$ and that $\mathrm{vec}(uv^\top) = v \otimes u$ we have $d_i = \mathrm{vec}(g_i\bar{a}_{i-1}^\top) = \bar{a}_{i-1} \otimes g_i$, and thus we can rewrite $F_{i,j}$ as

$$\begin{aligned} F_{i,j} &= \mathrm{E}\left[d_i d_j^\top\right] = \mathrm{E}\left[(\bar{a}_{i-1} \otimes g_i)(\bar{a}_{j-1} \otimes g_j)^\top\right] \\ &= \mathrm{E}\left[(\bar{a}_{i-1} \otimes g_i)(\bar{a}_{j-1}^\top \otimes g_j^\top)\right] = \mathrm{E}\left[\bar{a}_{i-1}\bar{a}_{j-1}^\top \otimes g_i g_j^\top\right] \end{aligned}$$

where $A \otimes B$ denotes the Kronecker product between A and B.

Our initial approximation $\tilde{F}$ to F will be defined by the following block-wise approximation:

$$\begin{aligned} F_{i,j} &= \mathrm{E}\left[\bar{a}_{i-1}\bar{a}_{j-1}^\top \otimes g_i g_j^\top\right] \approx \mathrm{E}\left[\bar{a}_{i-1}\bar{a}_{j-1}^\top\right] \otimes \mathrm{E}\left[g_i g_j^\top\right] \\ &= \bar{A}_{i-1,j-1} \otimes G_{i,j} = \tilde{F}_{i,j} \end{aligned} \tag{1}$$

where $\bar{A}_{i,j} = \mathrm{E}\left[\bar{a}_i\bar{a}_j^\top\right]$ and $G_{i,j} = \mathrm{E}\left[g_i g_j^\top\right]$.

This gives

$$\tilde{F} = \begin{bmatrix} \bar{A}_{0,0} \otimes G_{1,1} & \bar{A}_{0,1} \otimes G_{1,2} & \cdots & \bar{A}_{0,\ell-1} \otimes G_{1,\ell} \\ \bar{A}_{1,0} \otimes G_{2,1} & \bar{A}_{1,1} \otimes G_{2,2} & \cdots & \bar{A}_{1,\ell-1} \otimes G_{2,\ell} \\ \vdots & \vdots & \ddots & \vdots \\ \bar{A}_{\ell-1,0} \otimes G_{\ell,1} & \bar{A}_{\ell-1,1} \otimes G_{\ell,2} & \cdots & \bar{A}_{\ell-1,\ell-1} \otimes G_{\ell,\ell} \end{bmatrix}$$

which has the form of what is known as a Khatri-Rao product in multivariate statistics.

The expectation of a Kronecker product is, in general, not equal to the Kronecker product of expectations, and so this is indeed a major approximation to make, and one which likely won't become exact under any realistic set of assumptions, or as a limiting case in some kind of asymptotic analysis. Nevertheless, it seems to be fairly accurate in practice, and is able to successfully capture the "coarse structure" of the Fisher, as demonstrated in Figure 1 for an example network.

As we will see in later sections, this approximation leads to significant computational savings in terms of storage and inversion, which we will be able to leverage in order to design an efficient algorithm for computing an approximation to the natural gradient.

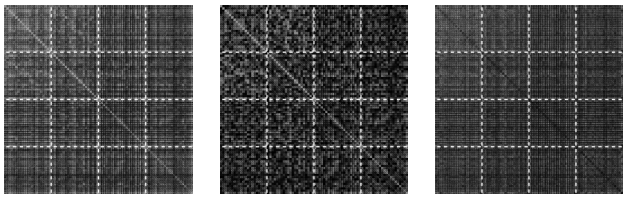

Figure 1. A comparison of the exact Fisher F and our block-wise Kronecker-factored approximation $\tilde{F}$, for the middle 4 layers of a standard deep neural network partially trained to classify a 16x16 down-scaled version of MNIST. The network was trained with 7 iterations of K-FAC in batch mode, achieving 5% error (the error reached 0% after 22 iterations) . The network architecture is 256-20-20-20-20-20-10 and uses standard tanh units. On the **left** is the exact Fisher F, in the **middle** is our approximation $\tilde{F}$, and on the **right** is the difference of these. The dashed lines delineate the blocks.

2.1. Interpretations of this approximation

Consider an arbitrary pair of weights $[W_i]_{k_1,k_2}$ and $[W_j]_{k_3,k_4}$ from the network, where $[\cdot]_{i,j}$ denotes the value of the (i,j)-th entry. We have that the corresponding derivatives of these weights are given by $\mathcal{D}[W_i]_{k_1,k_2} = \bar{a}^{(1)}g^{(1)}$ and $\mathcal{D}[W_j]_{k_3,k_4} = \bar{a}^{(2)}g^{(2)}$, where we denote for convenience $\bar{a}^{(1)} = [\bar{a}_{i-1}]_{k_1}$, $\bar{a}^{(2)} = [\bar{a}_{j-1}]_{k_3}$, $g^{(1)} = [g_i]_{k_2}$, and $g^{(2)} = [g_j]_{k_4}$.

The approximation given by eqn. 1 is equivalent to making the following approximation for each pair of weights:

$$\begin{aligned}\mathrm{E}\left[\mathcal{D}[W_i]_{k_1,k_2}\mathcal{D}[W_j]_{k_3,k_4}\right] &= \mathrm{E}\left[(\bar{a}^{(1)}g^{(1)})(\bar{a}^{(2)}g^{(2)})\right] \\ &= \mathrm{E}\left[\bar{a}^{(1)}\bar{a}^{(2)}\ g^{(1)}g^{(2)}\right] \approx \mathrm{E}\left[\bar{a}^{(1)}\bar{a}^{(2)}\right]\mathrm{E}\left[g^{(1)}g^{(2)}\right] \quad (2)\end{aligned}$$

And thus one way to interpret the approximation in eqn. 1 is that we are assuming statistical independence between products $\bar{a}^{(1)}\bar{a}^{(2)}$ of unit activities and products $g^{(1)}g^{(2)}$ of unit input derivatives.

Another more detailed interpretation of the approximation emerges by considering the following expression for the approximation error $\mathrm{E}\left[\bar{a}^{(1)}\bar{a}^{(2)}\ g^{(1)}g^{(2)}\right] - \mathrm{E}\left[\bar{a}^{(1)}\bar{a}^{(2)}\right]\mathrm{E}\left[g^{(1)}g^{(2)}\right]$ (which is derived in the appendix):

$$\begin{aligned}\kappa(\bar{a}^{(1)},\bar{a}^{(2)},g^{(1)},g^{(2)}) + \mathrm{E}[\bar{a}^{(1)}]\kappa(\bar{a}^{(2)},g^{(1)},g^{(2)}) \quad (3)\\ + \mathrm{E}[\bar{a}^{(2)}]\kappa(\bar{a}^{(1)},g^{(1)},g^{(2)})\end{aligned}$$

Here $\kappa(\cdot)$ denotes the cumulant of its arguments. Cumulants are a natural generalization of the concept of mean and variance to higher orders, and indeed 1st-order cumulants are means and 2nd-order cumulants are covariances. Intuitively, cumulants of order k measure the degree to which the interaction between variables is intrinsically of order k, as opposed to arising from many lower-order interactions.

A basic upper bound for the approximation error is

$$\begin{aligned}|\kappa(\bar{a}^{(1)},\bar{a}^{(2)},g^{(1)},g^{(2)})| + |\,\mathrm{E}[\bar{a}^{(1)}]||\kappa(\bar{a}^{(2)},g^{(1)},g^{(2)})| \\ + |\,\mathrm{E}[\bar{a}^{(2)}]||\kappa(\bar{a}^{(1)},g^{(1)},g^{(2)})| \quad (4)\end{aligned}$$

which will be small if all of the higher-order cumulants are small (i.e. those of order 3 or higher). Note that in principle this upper bound may be loose due to possible cancellations between the terms in eqn. 3.

Because higher-order cumulants are zero for variables jointly distributed according to a multivariate Gaussian, it follows that this upper bound on the approximation error will be small insofar as the joint distribution over $\bar{a}^{(1)}$, $\bar{a}^{(2)}$, $g^{(1)}$, and $g^{(2)}$ is well approximated by a multivariate Gaussian. And while we are not aware of an argument for why this should be the case in practice, it does seem to be the case that for the example network from Figure 1, the size of the error is well predicted by the size of the higher-order cumulants. In particular, the total approximation error, summed over all pairs of weights in the middle 4 layers, is 2894.4, and is of roughly the same size as the corresponding upper bound (4134.6), whose size is tied to that of the higher order cumulants (due to the impossibility of cancellations in eqn. 4).

3. Additional approximations to $\tilde{F}$ and inverse computations

To the best of our knowledge there is no efficient general method for inverting a Khatri-Rao product like $\tilde{F}$. Thus, we must make further approximations if we hope to obtain an efficiently computable approximation of the inverse Fisher.

In the following subsections we argue that the inverse of $\tilde{F}$ can be reasonably approximated as having one of two special structures, either of which make it efficiently computable. The second of these will be slightly less restrictive than the first (and hence a better approximation) at the cost of some additional complexity. We will then show how matrix-vector products with these approximate inverses can be efficiently computed, which will thus give an efficient algorithm for computing an approximation to the natural gradient.

3.1. Structured inverses and the connection to linear regression

Suppose we are given a multivariate distribution whose associated covariance matrix is Σ.

Define the matrix B so that for $i \neq j$, $[B]_{i,j}$ is the coefficient on the j-th variable in the optimal linear predictor of the i-th variable from all the other variables, and for $i = j$, $[B]_{i,j} = 0$. Then define the matrix D to be the diagonal matrix where $[D]_{i,i}$ is the variance of the error associated with such a predictor of the i-th variable.

Pourahmadi (2011) showed that the inverse covariance ma-

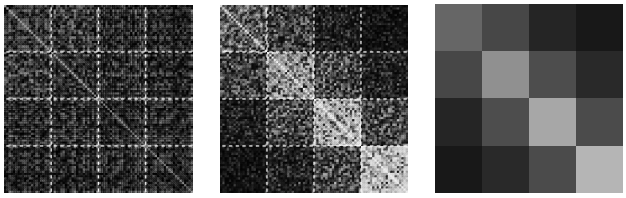

Figure 2. A comparison of our block-wise Kronecker-factored approximation $\tilde{F}$, and its inverse, using the example neural network from Figure 1. On the **left** is $\tilde{F}$, in the **middle** is its exact inverse, and on the **right** is a 4x4 matrix containing the averages of the absolute values of the entries in each block of the inverse.

trix can be expressed as $\Sigma^{-1} = D^{-1}(I - B)$.

Intuitively, this result says that each row of the inverse covariance Σ^{-1} is given by the coefficients of the optimal linear predictor of the i-th variable from the others, up to a scaling factor. So if the j-th variable is much less "useful" than the other variables for predicting the i-th variable, we can expect that the (i, j)-th entry of the inverse covariance will be relatively small.

Note that "usefulness" is a subtle property as we have informally defined it. In particular, it is not equivalent to the degree of correlation between the j-th and i-th variables, or any such simple measure. As a simple example, consider the case where the j-th variable is equal to the k-th variable plus independent Gaussian noise. Since any linear predictor can achieve a lower variance simply by shifting weight from the j-th variable to the k-th variable, we have that the j-th variable is not useful (and its coefficient will thus be zero) in the task of predicting the i-th variable for any setting of i other than $i = j$ or $i = k$.

Noting that the Fisher F is a covariance matrix over $\mathcal{D}\theta$ w.r.t. the model's distribution (because $\mathrm{E}[\mathcal{D}\theta] = 0$ by Lemma 4), we can thus apply the above analysis to the distribution over $\mathcal{D}\theta$ to gain insight into the approximate structure of F^{-1}, and by extension its approximation $\tilde{F}^{-1}$.

Consider the derivative $\mathcal{D}W_i$ of the loss with respect to the weights W_i of layer i. Intuitively, if we are trying to predict one of the entries of $\mathcal{D}W_i$ from the other entries of $\mathcal{D}\theta$, those entries also in $\mathcal{D}W_i$ will likely be the most useful in this regard. Thus, it stands to reason that the largest entries of $\tilde{F}^{-1}$ will be those on the diagonal blocks, so that $\tilde{F}^{-1}$ will be well approximated as block-diagonal, with each block corresponding to a different $\mathcal{D}W_i$.

Beyond the other entries of $\mathcal{D}W_i$, it is the entries of $\mathcal{D}W_{i+1}$ and $\mathcal{D}W_{i-1}$ (i.e. those associated with adjacent layers) that will arguably be the most useful in predicting a given entry of $\mathcal{D}W_i$. This is because the true process for computing the loss gradient only uses information from the layer below (during the forward pass) and from the layer above (during the backwards pass). Thus, approximating $\tilde{F}^{-1}$ as block-tridiagonal seems like a reasonable and milder alternative than taking it to be block-diagonal. Indeed, this approximation would be exact if the distribution over $\mathcal{D}\theta$ were given by a directed graphical model which generated each of the $\mathcal{D}W_i$'s, one layer at a time, from either $\mathcal{D}W_{i+1}$ or $\mathcal{D}W_{i-1}$. Or equivalently, if $\mathcal{D}W_i$ were distributed according to an undirected Gaussian graphical model with binary potentials only between entries in the same or adjacent layers. Both of these models are depicted in Figure 3.

Now while in reality the $\mathcal{D}W_i$'s are generated using information from adjacent layers according to a process that is *neither linear nor Gaussian*, it nonetheless stands to reason that their joint statistics might be reasonably approximated by such a model. In fact, the idea of approximating the distribution over loss gradients with a directed graphical model forms the basis of the recent FANG method of Grosse and Salakhutdinov (2015).

Figure 2 examines the extent to which the inverse Fisher is well approximated as block-diagonal or block-tridiagonal for an example network.

In the following two subsections we show how both the block-diagonal and block-tridiagonal approximations to $\tilde{F}^{-1}$ give rise to computationally efficient methods for computing matrix-vector products with it. And in Appendix C we present two figures (Figures 5 and 6) which examine the quality of these approximations for an example network.

3.2. Approximating $\tilde{F}^{-1}$ as block-diagonal

Approximating $\tilde{F}^{-1}$ as block-diagonal is equivalent to approximating $\tilde{F}$ as block-diagonal. A natural choice for such an approximation $\breve{F}$ of $\tilde{F}$, is to take the block-diagonal of $\breve{F}$ to be that of $\tilde{F}$. This gives the matrix

$$\breve{F} = \mathrm{diag}\left(\bar{A}_{0,0} \otimes G_{1,1}, \bar{A}_{1,1} \otimes G_{2,2}, \ \ldots, \ \bar{A}_{\ell-1,\ell-1} \otimes G_{\ell,\ell}\right)$$

Using the identity $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$ we can easily compute the inverse of $\breve{F}$ as

$$\breve{F}^{-1} = \mathrm{diag}\left(\bar{A}_{0,0}^{-1} \otimes G_{1,1}^{-1}, \ \ldots, \ \bar{A}_{\ell-1,\ell-1}^{-1} \otimes G_{\ell,\ell}^{-1}\right)$$

Thus, computing $\breve{F}^{-1}$ amounts to computing the inverses of 2ℓ smaller matrices.

Then to compute $u = \breve{F}^{-1}v$, we can make use of the well-known identity $(A \otimes B)\,\mathrm{vec}(X) = \mathrm{vec}(BXA^\top)$ to get

$$U_i = G_{i,i}^{-1} V_i \bar{A}_{i-1,i-1}^{-1}$$

where v maps to $(V_1, V_2, \ldots, V_\ell)$ and u maps to $(U_1, U_2, \ldots, U_\ell)$ in an analogous way to how θ maps to $(W_1, W_2, \ldots, W_\ell)$.

3.3. Approximating $\tilde{F}^{-1}$ as block-tridiagonal

Note that unlike in the above block-diagonal case, approximating $\tilde{F}^{-1}$ as block-tridiagonal is *not* equivalent to approximating $\tilde{F}$ as block-tridiagonal. Thus we require a

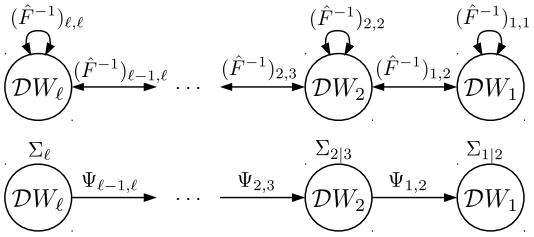

Figure 3. A diagram depicting the UGGM corresponding to $\hat{F}^{-1}$ and its equivalent DGGM. The UGGM's edges are labeled with the corresponding weights of the model (these are distinct from the network's weights). Here, $(\hat{F}^{-1})_{i,j}$ denotes the (i,j)-th block of $\hat{F}^{-1}$. The DGGM's edges are labeled with the matrices that specify the linear mapping from the source node to the conditional mean of the destination node (whose conditional covariance is given by its label).

more sophisticated approach to deal with such an approximation. We develop such an approach in this subsection.

To start, we will define $\hat{F}$ to be the matrix which agrees with $\tilde{F}$ on the tridiagonal blocks, and which satisfies the property that $\hat{F}^{-1}$ is block-tridiagonal. Note that this definition implies certain values for the off-tridiagonal blocks of $\hat{F}$ which will differ from those of $\tilde{F}$ insofar as $\tilde{F}^{-1}$ is not actually block-tridiagonal.

To establish that such a matrix $\hat{F}$ is well defined and can be inverted efficiently, we first observe that assuming that $\hat{F}^{-1}$ is block-tridiagonal is equivalent to assuming that it is the precision matrix of an undirected Gaussian graphical model (UGGM) over $\mathcal{D}\theta$ (as depicted in Figure 3), whose density function is proportional to $\exp(-\mathcal{D}\theta^\top \hat{F}^{-1}\mathcal{D}\theta)$. As this graphical model has a tree structure, there is an equivalent *directed* graphical model with the same distribution and the same (undirected) graphical structure (e.g. Bishop, 2006), where the directionality of the edges is given by a directed acyclic graph (DAG). Moreover, this equivalent directed model will also be linear/Gaussian, and hence a directed Gaussian Graphical model (DGGM).

Next we will show how the parameters of such a DGGM corresponding to $\hat{F}$ can be efficiently recovered from the tridiagonal blocks of $\hat{F}$, so that $\hat{F}$ is uniquely determined by these blocks (and hence well-defined). We will assume here that the direction of the edges is from the higher layers to the lower ones. Note that a different choice for these directions would yield a superficially different algorithm for computing the inverse of $\hat{F}$ that would nonetheless yield the same output.

For each i, we will denote the conditional covariance matrix of $\text{vec}(\mathcal{D}W_i)$ on $\text{vec}(\mathcal{D}W_{i+1})$ by $\Sigma_{i|i+1}$ and the linear coefficients from $\text{vec}(\mathcal{D}W_{i+1})$ to $\text{vec}(\mathcal{D}W_i)$ by the matrix $\Psi_{i,i+1}$, so that the conditional distributions defining the model are $\text{vec}(\mathcal{D}W_i) \sim \mathcal{N}\left(\Psi_{i,i+1}\text{vec}(\mathcal{D}W_{i+1}),\ \Sigma_{i|i+1}\right)$ and $\text{vec}(\mathcal{D}W_\ell) \sim \mathcal{N}\left(\vec{0},\ \Sigma_\ell\right)$

Since Σ_ℓ is just the covariance of $\text{vec}(\mathcal{D}W_\ell)$, it is given simply by $\tilde{F}_{\ell,\ell}$. And for $i \leq \ell - 1$, we can see that $\Psi_{i,i+1}$ is given by $\Psi_{i,i+1} = \hat{F}_{i,i+1}\hat{F}^{-1}_{i+1,i+1}$. where

$$\Psi^{\bar{A}}_{i-1,i} = \bar{A}_{i-1,i}\bar{A}^{-1}_{i,i} \qquad \text{and} \qquad \Psi^{G}_{i,i+1} = G_{i,i+1}G^{-1}_{i+1,i+1}$$

The conditional covariance $\Sigma_{i|i+1}$ is thus given by

$$\begin{aligned}\Sigma_{i|i+1} = \tilde{F}_{i,i} - \Psi_{i,i+1}\tilde{F}_{i+1,i+1}\Psi^\top_{i,i+1} &= \bar{A}_{i-1,i-1} \otimes G_{i,i} \\ &\quad - \Psi^{\bar{A}}_{i-1,i}\bar{A}_{i,i}\Psi^{\bar{A}\top}_{i-1,i} \otimes \Psi^{G}_{i,i+1}G_{i+1,i+1}\Psi^{G\top}_{i,i+1}\end{aligned}$$

Following the work of Grosse and Salakhutdinov (2015), we use the block generalization of well-known "Cholesky" decomposition of the precision matrix of DGGMs (Pourahmadi, 1999), which gives

$$\begin{aligned}\hat{F}^{-1} &= \Xi^\top \Lambda \Xi \\ \text{where} \quad \Lambda &= \text{diag}\left(\Sigma^{-1}_{1|2},\ \ldots,\ \Sigma^{-1}_{\ell-1|\ell}, \Sigma^{-1}_{\ell}\right) \\ \text{and} \quad \Xi &= \begin{bmatrix} I & -\Psi_{1,2} & & & \\ & I & -\Psi_{2,3} & & \\ & & I & \ddots & \\ & & & \ddots & -\Psi_{\ell-1,\ell} \\ & & & & I \end{bmatrix}\end{aligned}$$

Thus, matrix-vector multiplication with $\hat{F}^{-1}$ amounts to performing matrix-vector multiplication by Ξ, followed by Λ, and then by $\Xi^\top$.

As in the block-diagonal case considered in the previous subsection, matrix-vector products with Ξ (and $\Xi^\top$) can be efficiently computed by using the well-known property $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$. In particular, $u = \Xi^\top v$ can be computed as

$$U_i = V_i - \Psi^{G\top}_{i-1,i}V_{i-1}\Psi^{\bar{A}}_{i-2,i-1} \qquad \text{and} \qquad U_1 = V_1$$

and similarly $u = \Xi v$ can be computed as

$$U_i = V_i - \Psi^{G}_{i,i+1}V_{i+1}\Psi^{\bar{A}\top}_{i-1,i} \qquad \text{and} \qquad U_\ell = V_\ell$$

where the U_i's and V_i's are defined in terms of u and v as in the previous subsection.

Multiplying a vector v by Λ amounts to multiplying each $\text{vec}(V_i)$ by the corresponding $\Sigma^{-1}_{i|i+1}$. This is slightly tricky because $\Sigma_{i|i+1}$ is the difference of Kronecker products, so we cannot use the straightforward identity $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$. Fortunately, there are efficient techniques for inverting such matrices which we discuss in detail in Appendix I.

4. Invariance Properties and the Relationship to Whitening and Centering

When computed with the exact Fisher, the natural gradient specifies a direction in the space of predictive distributions

which is invariant to the specific way that the model is parameterized. This invariance means that the smooth path through distribution space produced by following the natural gradient with infinitesimally small steps will be similarly invariant.

For a practical natural gradient based optimization method which takes large discrete steps in the direction of the natural gradient, this invariance of the optimization path will only hold approximately. As shown by Martens (2014), the approximation error will go to zero as the effects of damping diminish and the reparameterizing function ζ tends to a locally linear function. Note that the latter will happen as ζ becomes smoother, or the local region containing the update shrinks to zero.

Because K-FAC uses an approximation of the natural gradient, these invariance results are not applicable in our case. Fortunately, as was shown by Martens (2014), one can establish invariance of an update direction with respect to a given reparameterization of the model by verifying certain simple properties of the curvature matrix C used to compute the update. We will use this result to show that, under the assumption that damping is absent (or negligible in its affect), K-FAC is invariant to a broad and natural class of affine transformations of the network.

This class of transformations is given by the following modified network definition (c.f. the def. in Section 1.1):

$$s_i^\dagger = W_i \bar{a}_{i-1}^\dagger \qquad \bar{a}_i^\dagger = \Omega_i \bar{\phi}_i(\Phi_i s_i^\dagger)$$

where $\bar{\phi}_i$ is the function that computes ϕ_i and then appends a homogeneous coordinate (with value 1), Ω_i and Φ_i are arbitrary invertible matrices of the appropriate sizes (except that we assume $\Omega_\ell = I$), $\bar{a}_0^\dagger = \Omega_0 \bar{a}_0$, and where the network's output is given by $f^\dagger(x, \theta) = a_\ell^\dagger$. Note that because Ω_i multiplies $\bar{\phi}_i(\Phi_i s_i^\dagger)$, it can implement arbitrary translations of the unit activities $\phi_i(\Phi_i s_i^\dagger)$ in addition to arbitrary linear transformations.

Here, and going forward, we will add a "†" superscript to any network-dependent quantity in order to denote the analogous version of it computed by the transformed network. Note that under this identification, the loss derivative formulas for the transformed network are analogous to those of the original network, and so our various Fisher approximations are still well defined.

The following theorem describes the main technical result of this section.

Theorem 1. *There exists an invertible linear function $\theta = \zeta(\theta^\dagger)$ so that $f^\dagger(x, \theta^\dagger) = f(x, \theta) = f(x, \zeta(\theta^\dagger))$, and thus the transformed network can be viewed as a reparameterization of the original network by $\theta^\dagger$. Moreover, additively updating θ by $\delta = -\alpha \breve{F}^{-1} \nabla h$ or $\delta = -\alpha \hat{F}^{-1} \nabla h$ in the original network is equivalent to additively updating $\theta^\dagger$ by $\delta^\dagger = -\alpha \breve{F}^{\dagger -1} \nabla h^\dagger$ or $\delta^\dagger = -\alpha \hat{F}^{\dagger -1} \nabla h^\dagger$ (resp.) in the transformed network, in the sense that $\zeta(\theta^\dagger + \delta^\dagger) = \theta + \delta$.*

This immediately implies the following corollary which characterizes the invariance of a basic version of K-FAC to the given class of network transformations.

Corollary 2. *The optimization path taken by K-FAC (using either of our Fisher approximations $\breve{F}$ or $\hat{F}$) through the space of predictive distributions is the same for the default network as it is for the transformed network (where the Ω_i's and Φ_i's remain fixed). This assumes the use of an equivalent initialization ($\theta_0 = \zeta(\theta_0^\dagger)$), and a basic version of K-FAC where damping is absent or negligible in effect, momentum is not used, and where the learning rates are chosen in a way that is independent of the network's parameterization.*

While this corollary assumes that the Ω_i's and Φ_i's are fixed, if we relax this assumption so that they are allowed to vary smoothly with θ, then ζ will be a smooth function of θ, and so as discussed in Martens (2014), invariance of the optimization path will hold approximately in a way that depends on the smoothness of ζ (which measures how quickly the Ω_i's and Φ_i's change) and the size of the update. Moreover, invariance will hold exactly in the limit as the learning rate goes to 0.

Note that the network transformations can be interpreted as replacing the network's nonlinearity $\bar{\phi}_i(s_i)$ at each layer i with a "transformed" version $\Omega_i \bar{\phi}_i(\Phi_i s_i)$. So since the well-known logistic sigmoid and tanh functions are related to each other by such a transformation, an immediate consequence of Corollary 2 is that K-FAC is invariant to the choice of logistic sigmoid vs. tanh activation functions (provided that equivalent initializations are used and that the effect of damping is negligible, etc.). Also note that because the network inputs are also transformed by Ω_0, K-FAC is thus invariant to arbitrary affine transformations of the input, which includes many popular training data preprocessing techniques.

In the case where we use the block-diagonal approximation $\breve{F}$ and compute updates without damping, Theorem 1 affords us an additional elegant interpretation of what K-FAC is doing. In particular, the updates produced by K-FAC end up being equivalent to those produced by *standard gradient descent* using a network which is transformed so that the unit activities and the unit-gradients are both centered and whitened (with respect to the model's distribution). This is stated formally in the following corollary.

Corollary 3. *Additively updating θ by $-\alpha \breve{F}^{-1} \nabla h$ in the original network is equivalent to additively updating $\theta^\dagger$ by the gradient descent update $-\alpha \nabla h^\dagger$ in a transformed version of the network where the unit activities $a_i^\dagger$ and the unit-gradients $g_i^\dagger$ are both centered and whitened with respect to the model's distribution.*

5. Additional details

Due to the constraints of space we have left certain details about how to implement K-FAC in practice to the appendix.

Appendix D describes how we compute online estimates of the quantities required by our inverse Fisher approximation over a large "window" of previously processed mini-batches (which makes K-FAC very different from methods like HF or KSD, which base their estimates of the curvature on a single mini-batch). Appendix E describes how we use our approximate Fisher to obtain a practical and robust optimization algorithm which requires very little manual tuning, through the careful application of various theoretically well-founded "damping" techniques that are standard in the optimization literature. Note that damping techniques compensate both for the local quadratic approximation being implicitly made to the objective, and for our further approximation of the Fisher, and are non-optional for essentially any 2nd-order method like K-FAC to work properly, as is well established by both theory and practice within the optimization literature (Nocedal and Wright, 2006). Appendix F describes a simple and effective way of adding a type of "momentum" to K-FAC, which we have found works very well in practice. Appendix G describes the computational costs associated with K-FAC, and various ways to reduce them to the point where each update is at most only several times more expensive to compute than the stochastic gradient. Finally, Appendix H gives complete high-level pseudocode for K-FAC.

6. Related Work

Centering methods work by either modifying the gradient (Schraudolph, 1998) or dynamically reparameterizing the network itself (Raiko et al., 2012; Vatanen et al., 2013; Wiesler et al., 2014), so that various unit-wise scalar quantities like the activities (the a_i's) and local derivatives (the $\phi_i'(s_i)$'s) are 0 on average (i.e. "centered"), as they appear in the formula for the gradient. Typically, these methods require the introduction of additional "skip" connections (which bypass the nonlinearities of a given layer) in order to preserve the expressive power/efficiency of the network after these transformations are applied.

It is argued by Raiko et al. (2012) that the application of the centering transformation makes the Fisher of the resulting network closer to a diagonal matrix, and thus makes its gradient more closely resemble its natural gradient. However, this argument uses the strong approximating assumption that the correlations between various network-dependent quantities, such as the activities of different units within a given layer, are zero. In our notation, this would be like assuming that the $G_{i,i}$'s are diagonal, and that the $\bar{A}_{i,i}$'s are rank-1 plus a diagonal term. Indeed, using such an approximation within the block-diagonal version of K-FAC would yield an algorithm similar to standard centering, although without the need for skip connections (and hence similar to the version of centering proposed by Wiesler et al. (2014)).

As shown in Corollary 3, K-FAC can also be interpreted as using the gradient of a transformed network as its update direction, although one in which the g_i's and a_i's are both centered and *whitened* (with respect to the model's distribution). Intuitively, it is this whitening which accounts for the correlations between activities (or back-propagated gradients) within a given layer.

The work most closely related to ours is that of Heskes (2000), who proposed an approximation of the Fisher of feed-forward neural networks similar to our Kronecker-factored block-diagonal approximation $\breve{F}$ from Section 3.2, and used it to derive an efficient approximate natural-gradient based optimization method by exploiting the identity $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$. K-FAC differs from Heskes' method in several important ways which turn out to be crucial to it working well in practice.

In Heskes' method, update damping is accomplished using a basic factored Tikhonov technique where γI is added to each $G_{i,i}$ and $\bar{A}_{i,i}$ for a fixed parameter $\gamma > 0$ which is set by hand. By contrast, K-FAC uses a factored Tikhonov technique where γ adapted dynamically as described in Appendix E.6, combined with a re-scaling technique based on a local quadratic model computed using the exact Fisher (see Appendix E.4). Note that the adaptation of γ is important since what constitutes a good or even merely acceptable value of γ will change significantly over the course of optimization. And the use of our re-scaling technique, or something similar to it, is also crucial as we have observed empirically that basic Tikhonov damping is incapable of producing high quality updates by itself, even when γ is chosen optimally at each iteration (see Figure 7 of Appendix E.4).

Also, while Heskes' method computes the $G_{i,i}$'s exactly, K-FAC uses a stochastic approximation which scales efficiently to neural networks with much higher-dimensional outputs (see Appendix D). Other advances we have introduced include the more accurate block-tridiagonal approximation to the inverse Fisher, a parameter-free type of momentum (see Appendix F), online estimation of the $G_{i,i}$ and $\bar{A}_{i,i}$ matrices, and various improvements in computational efficiency (see Appendix G). We have found that each of these additional elements is important in order for K-FAC to work as well as it does in various settings.

For a discussion of more related work, see Appendix L.

7. Experiments

To investigate the practical performance of K-FAC we applied it to the 3 deep-autoencoder optimization problems from Hinton and Salakhutdinov (2006), which use the "MNIST", "CURVES", and "FACES" datasets respectively (see Hinton and Salakhutdinov (2006) for a complete description of the network architectures and datasets). Due to their high difficulty, performance on these problems has become a standard benchmark for neural network optimization methods (e.g. Martens, 2010; Vinyals and Povey, 2012; Sutskever et al., 2013).

As our baseline we used the version of SGD with momentum based on Nesterov's Accelerated Gradient (Nesterov, 1983) described in Sutskever et al. (2013), which was calibrated to work well on these particular deep autoencoder problems. For each problem we followed the prescription given by Sutskever et al. (2013) for determining the learning rate, and the increasing schedule for the decay parameter μ. We did not compare to methods based on diagonal approximations of the curvature matrix, as in our experience such methods tend not perform as well on these kinds of optimization problems as the baseline does (which is consistent with the findings of Zeiler (2013)).

Our implementation of K-FAC used most of the efficiency improvements described in Appendix G, except that all "tasks" were computed serially (and thus with better engineering and more hardware, a faster implementation could likely be obtained). Both K-FAC and the baseline were implemented using vectorized MATLAB code accelerated with the GPU package Jacket. All tests were performed on a single computer with a 4.4 Ghz Intel CPU and an NVidia GTX 580 GPU with 3GB of memory. Each method used the same initial parameter setting, which was generated using the "sparse initialization" technique from Martens (2010) (which was also used by Sutskever et al. (2013)).

To help mitigate the detrimental effect that the noise in the stochastic gradient has on the convergence of the baseline (and to a lesser extent K-FAC) we used a exponentially decayed iterate averaging approach based on Polyak averaging (e.g. Swersky et al., 2010). In particular, at each iteration we took the "averaged" parameter estimate to be the previous such estimate, multiplied by ξ, plus the new iterate produced by the optimizer, multiplied by $1 - \xi$, for $\xi = 0.99$.

To be consistent with the numbers given in previous papers we report the reconstruction error instead of the actual objective function value (although these are almost perfectly correlated in our experience). And we report the error on the training set as opposed to the test set, as we are chiefly interested in optimization speed and not the generalization capabilities of the networks themselves.

In our main experiment we evaluated the performance of our implementation of K-FAC versus the baseline on all 3 deep autoencoder problems, where we used an exponentially increasing schedule for m within K-FAC (which we explain and provide empirical justification for in Appendix M), and a fixed setting of m within the baseline and momentum-less K-FAC (which was chosen from a small range of candidates to give the best overall per-second rate of progress).

The results from this experiment are plotted in Figure 4, with additional information about per-iteration rates of progress plotted in Figure 9 of Appendix M. For each problem K-FAC had a *per-iteration* rate of progress which was orders of magnitude higher than that of the baseline's, provided that momentum was used, which translated into an overall much higher *per-second* rate of progress, despite the higher cost of K-FAC's iterations (due mostly to the much larger mini-batch sizes used).

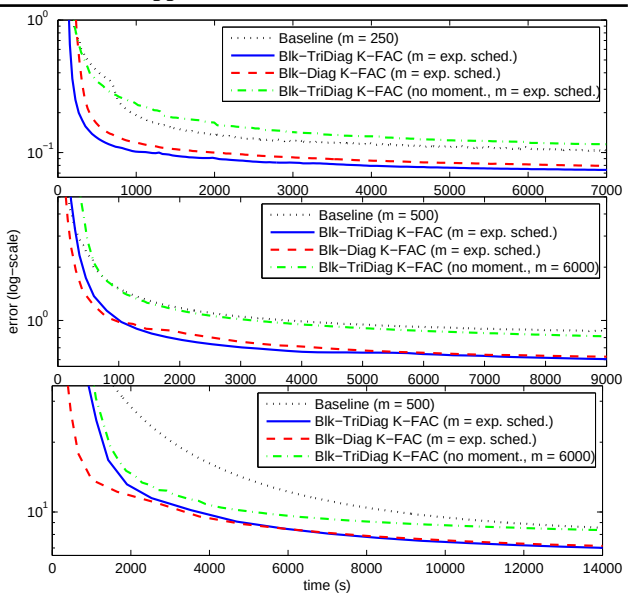

Figure 4. Results from our main experiment showing training error versus computation time on the CURVES (**top**), MNIST (**middle**), and FACES (**bottom**) deep autoencoder problems. Here, "Blk-TriDiag K-FAC" is the block-tridiagonal version of K-FAC, and "Blk-Diag K-FAC" is the block-diagonal version. "No moment." indicates that momentum was not used.

The importance of using some form of momentum on these problems is emphasized in these experiments by the fact that without the momentum technique developed in Appendix F, K-FAC wasn't significantly faster than the baseline (which itself used a strong form of momentum). These results echo those of Sutskever et al. (2013), who found that without momentum, SGD was orders of magnitude slower on these particular problems.

While our results suggest that the block-diagonal version is probably the better option overall due to its greater simplicity (and comparable per-second progress rate), the situation may be different given a more efficient implementation of K-FAC where the expensive SVDs required by the tri-diagonal version are computed approximately and/or in parallel with the other tasks, or perhaps even while the network is being optimized.

Our results also suggest that K-FAC may be much better suited than the SGD baseline for a massively distributed implementation, since it would require far fewer synchronization steps (by virtue of the fact that it performs far fewer iterations).

Acknowledgments

We gratefully acknowledge support from Google, NSERC, and the University of Toronto.

References

S. Amari and H. Nagaoka. *Methods of Information Geometry*, volume 191 of *Translations of Mathematical monographs*. Oxford University Press, 2000.

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

Ludovic Arnold, Anne Auger, Nikolaus Hansen, and Yann Ollivier. Information-geometric optimization algorithms: A unifying picture via invariance principles. 2011, arXiv:1106.3708.

Sue Becker and Yann LeCun. Improving the Convergence of Back-Propagation Learning with Second Order Methods. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 29–37, 1989.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2006.

King-wah Eric Chu. The solution of the matrix equations $AXB - CXD = E$ and $(YA - DZ, YC - BZ) = (E, F)$. *Linear Algebra and its Applications*, 93(0):93 – 105, 1987.

Christian Darken and John E. Moody. Note on learning rate schedules for stochastic optimization. In *Advances in Neural Information Processing Systems*, pages 832–838, 1990.

Michael P. Friedlander and Mark W. Schmidt. Hybrid deterministic-stochastic methods for data fitting. *SIAM J. Scientific Computing*, 34(3), 2012.

Judith D. Gardiner, Alan J. Laub, James J. Amato, and Cleve B. Moler. Solution of the sylvester matrix equation $AXB^T + CXD^T = E$. *ACM Trans. Math. Softw.*, 18(2):223–231, June 1992.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of AISTATS 2010*, volume 9, pages 249–256, may 2010.

Roger Grosse and Ruslan Salakhutdinov. Scaling up natural gradient by factorizing fisher information. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015.

Tom Heskes. On "natural" learning and pruning in multilayered perceptrons. *Neural Computation*, 12(4):881–901, 2000.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, July 2006.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

Ryan Kiros. Training neural networks with stochastic Hessian-free optimization. In *International Conference on Learning Representations (ICLR)*, 2013.

Nicolas Le Roux, Pierre-antoine Manzagol, and Yoshua Bengio. Topmoumoute online natural gradient algorithm. In *Advances in Neural Information Processing Systems 20*, pages 849–856. MIT Press, 2008.

Y. LeCun, L. Bottou, G. Orr, and K. Müller. Efficient backprop. *Neural networks: Tricks of the trade*, pages 546–546, 1998.

J. Martens. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010.

J. Martens. New insights and perspectives on the natural gradient method, 2014, arXiv:1411.7717.

J. Martens and I. Sutskever. Training deep and recurrent networks with Hessian-free optimization. In *Neural Networks: Tricks of the Trade*, pages 479–535. 2012.

J. Martens, I. Sutskever, and K. Swersky. Estimating the Hessian by backpropagating curvature. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, 2012.

J.J. Moré. The Levenberg-Marquardt algorithm: implementation and theory. *Numerical analysis*, pages 105–116, 1978.

Yurii Nesterov. A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/\sqrt{k})$. *Soviet Mathematics Doklady*, 27:372–376, 1983.

Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer, 2. ed. edition, 2006.

V. Pan and R. Schreiber. An improved newton iteration for the generalized inverse of a matrix, with applications. *SIAM Journal on Scientific and Statistical Computing*, 12(5):1109–1130, 1991.

H. Park, S.-I. Amari, and K. Fukumizu. Adaptive natural gradient learning algorithms for various stochastic models. *Neural Networks*, 13(7):755–764, September 2000.

Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. In *International Conference on Learning Representations*, 2014.

D. Plaut, S. Nowlan, and G. E. Hinton. Experiments on learning by back propagation. Technical Report CMU-CS-86-126, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1986.

B.T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1 – 17, 1964. ISSN 0041-5553.

M. Pourahmadi. Joint mean-covariance models with applications to longitudinal data: unconstrained parameterisation. *Biometrika*, 86(3):677–690, 1999.

M. Pourahmadi. Covariance Estimation: The GLM and Regularization Perspectives. *Statistical Science*, 26(3): 369–387, August 2011.

Daniel Povey, Xiaohui Zhang, and Sanjeev Khudanpur. Parallel training of DNNs with natural gradient and parameter averaging. In *International Conference on Learning Representations: Workshop track*, 2015.

Tapani Raiko, Harri Valpola, and Yann LeCun. Deep learning made easier by linear transformations in perceptrons. In *AISTATS*, volume 22 of *JMLR Proceedings*, pages 924–932, 2012.

Silvia Scarpetta, Magnus Rattray, and David Saad. Matrix momentum for practical natural gradient learning. *Journal of Physics A: Mathematical and General*, 32(22): 4047, 1999.

Tom Schaul, Sixin Zhang, and Yann LeCun. No More Pesky Learning Rates. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013.

Nicol N. Schraudolph. Centering neural network gradient factors. In Genevieve B. Orr and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pages 207–226. Springer Verlag, Berlin, 1998.

Nicol N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14, 2002.

V. Simoncini. Computational methods for linear matrix equations. 2014.

R.A. Smith. Matrix equation $XA + BX = C$. *SIAM J. Appl. Math.*, 16(1):198 – 201, 1968.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013.

K. Swersky, Bo Chen, B. Marlin, and N. de Freitas. A tutorial on stochastic approximation algorithms for training restricted boltzmann machines and deep belief nets. In *Information Theory and Applications Workshop (ITA), 2010*, pages 1–10, Jan 2010.

Tommi Vatanen, Tapani Raiko, Harri Valpola, and Yann LeCun. Pushing stochastic gradient towards second-order methods – backpropagation learning with transformations in nonlinearities. 2013, arXiv:1301.3476.

O. Vinyals and D. Povey. Krylov subspace descent for deep learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.

Simon Wiesler, Alexander Richard, Ralf Schlüter, and Hermann Ney. Mean-normalized stochastic gradient for large-scale deep learning. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 180–184, 2014.

Matthew D. Zeiler. ADADELTA: An adaptive learning rate method. 2013, arXiv:1212.5701.