Hindawi Publishing Corporation
Advances in Artificial Neural Systems
Volume 2011, Article ID 407497, 9 pages
doi:10.1155/2011/407497

*Research Article*

# A Simplified Natural Gradient Learning Algorithm

**Michael R. Bastian, Jacob H. Gunther, and Todd K. Moon**

*Department of Electrical and Computer Engineering, Utah State University, Logan, UT 84322, USA*

Correspondence should be addressed to Michael R. Bastian, ulthanash@msn.com

Received 2 February 2011; Accepted 12 June 2011

Academic Editor: Shantanu Chakrabartty



Adaptive natural gradient learning avoids singularities in the parameter space of multilayer perceptrons. However, it requires a larger number of additional parameters than ordinary backpropagation in the form of the Fisher information matrix. This article describes a new approach to natural gradient learning that uses a smaller Fisher information matrix. It also uses a prior distribution on the neural network parameters and an annealed learning rate. While this new approach is computationally simpler, its performance is comparable to that of Adaptive natural gradient learning.

## 1. Introduction

Amari et al. developed the adaptive natural gradient learning (ANGL) algorithm for multilayer perceptrons [1–3]. It is similar to Newton's method in which the gradient of the error function is multiplied by the inverse of the Hessian matrix of the error function. However, Amari's algorithm replaces the inverse of the Hessian matrix with the inverse of the Fisher information matrix. The simplified natural gradient learning (SNGL) algorithm introduced in this paper uses a new formulation of the Fisher information matrix. SNGL is based on the backpropagation algorithm [4]. In addition, the SNGL algorithm also uses regularization [5] to penalize solutions with large connection weights. This regularization also prevents the Fisher information matrix from approaching a singular matrix. The learning rate is also annealed [6] to improve the general performance of SNGL.

The remainder of this section will review the natural gradient and discuss the details of Amari's ANGL algorithm in [2, 7, 8]. The next section will introduce the SNGL algorithm and the theory behind it. The third section will describe two improvements used in the SNGL algorithm. These improvements are regularization and annealed learning rates. The following section will show the experimental results and compare the performance of these two algorithms with that of ordinary backpropagation. The final section will conclude the ideas discussed in this paper.

*1.1. Mathematical Preliminaries and Notation.* The natural gradient learning algorithm is "natural" in the sense that it follows the manifold of the underlying parameter space of multilayer perceptrons. This parameter space has Riemannian geometry. Calculating the direction of steepest descent in a Riemannian manifold is more complicated than it is in a Euclidean space.

The parameter space of multilayer perceptrons is an affine space in which probability distributions are the points and random variables are vectors [9, 10]. Given a probability density function $p$ of a random variable $x$, the translation operation is $p + x = p\exp(x)$.

*1.1.1. Exponential Families.* An exponential family is a set of distributions with probability density functions of the form

$$p(x \mid \theta^1, \ldots, \theta^r) = c(x)\exp\left(\sum_{i=1}^{r}\theta^i\eta_i(x) - A(\theta^1, \ldots, \theta^r)\right). \tag{1}$$

Thus, the distributions in the exponential family are parameterized by $\{\theta^1, \ldots, \theta^r\}$, and these parameters are the canonical coordinates of the distribution. The number $A(\theta^1, \ldots, \theta^r)$ is chosen such that $\int p(x \mid \theta^1, \ldots, \theta^r)dx = 1$. Given an open set $U$ in the event space of the random variable $X$,

the probability that the result of the random variable $X$ lies within $U$ is

$$P[X \in U] = \int_U p(x \mid \theta^1, \ldots, \theta^r)\, dx. \tag{2}$$

The sufficient statistics for the exponential family are the $r$ random variables $\eta_1, \ldots, \eta_r$ in (1). These statistics can be used to estimate the coordinates $\theta^1, \ldots, \theta^r$ from the values of samples of $x$.

The log-likelihood function of a probability distribution is the logarithm of its probability density function $\log p$. The score functions are the derivatives of the log-likelihood function with respect to the coordinates $\theta^1, \ldots, \theta^r$. The score for parameter $i$ is

$$\mathbf{s}^i = \frac{\partial \log p(x \mid \theta^1, \ldots, \theta^r)}{\partial \theta^i} = \eta_i(x) - \frac{\partial A}{\partial \theta^i}. \tag{3}$$

The Fisher information of two score functions is the expected value of their product. For the score functions $\mathbf{s}^i$ and $\mathbf{s}^j$, the Fisher information is

$$g_p^{ij} = E_p\left\{\mathbf{s}^i \mathbf{s}^j\right\}, \tag{4}$$

where $E_p$ is the expectation operator for the distribution with probability density function $p$. The Fisher information matrix is a real symmetric $r \times r$ matrix $\mathbf{G}_p = (g_p^{ij})$ whose elements are the Fisher information for each pair of score functions.

*1.1.2. Direction of Steepest Descent.* When finding the minimum of a function $f$ in a Euclidean space, a descent direction [11] is a unit norm tangent vector $\mathbf{v}_p$ such that

$$\left\langle \mathbf{v}_p, \nabla f(p) \right\rangle \triangleq \sum_{i=1}^{n} v_i \left. \frac{\partial f}{\partial x_i} \right|_p < 0. \tag{5}$$

The minimum value of (5) is $-\alpha \|\nabla f(p)\|^2$ where $\alpha$ is an arbitrary scalar and $\|\nabla f(p)\|^2 = \sum_{i=1}^{n} (\partial f/\partial x_i \mid_p)^2$ which occurs when $\mathbf{v}_p = -\nabla f(p)/\|\nabla f(p)\|$. Thus, in a Euclidean space, the direction of steepest descent is the unit tangent vector $-\nabla f(p)/\|\nabla f(p)\|$.

Finding the direction of steepest descent turns out to be harder than it first appears to be. In a Euclidean geometry, it is relatively straightforward. In a Riemannian geometry, the coordinate system is curved, and a gradient descent algorithm should follow the curvature of the manifold. The direction of steepest descent should be compensated for this curvature.

In a Riemannian space, a metric $g_p$ exists at every point $p$. If the tangent vectors $\mathbf{v}_p^1, \ldots, \mathbf{v}_p^n$ form a basis for the tangent space $T_p$ at $p$, then $g_p^{ij} = \langle \mathbf{v}_p^i, \mathbf{v}_p^j \rangle$ is the Riemannian metric where $\langle \cdot, \cdot \rangle$ is an inner product defined on the tangent space $T_p$.

Given a tangent vector $\mathbf{w}_p = \sum_{i=1}^{n} \alpha_i \mathbf{v}_p^i$ and the gradient $\nabla f(p) = \sum_{i=1}^{n} \beta_i \mathbf{v}_p^i$, their inner product will be

$$\left\langle \mathbf{w}_p, \nabla f(p) \right\rangle = \sum_{i=1}^{n} \sum_{j=1}^{n} g_p^{ij} \alpha_i \beta_j. \tag{6}$$

If $\mathbf{w}_p$ is the steepest descent direction, then, according to the analysis above, $\langle \mathbf{w}_p, \nabla f(p) \rangle = -\alpha \|\nabla f(p)\|^2$. This occurs when $\mathbf{w}_p = -\mathbf{G}_p^{-1} \nabla f(p)$ where $\mathbf{G}_p = (g_p^{ij})$ is the matrix whose elements are equal to the inner products of the basis vectors. In other words, the coordinate system matters when doing gradient descent, and it may change at each point on the manifold. The natural gradient learning algorithm uses this "natural" descent direction at each step.

*1.1.3. A Probability Model for Multilayer Perceptrons.* In order for this analysis to be applied to neural networks, there must be a probability density function associated with a neural network. This section describes an exponential family in which each member of the family has a probability density function of network output errors. This exponential family will then be used to determine the direction of steepest descent and the natural gradient to be used in a learning algorithm.

Let the vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$, respectively, be the inputs and outputs of a training set. A convenient model for training a multilayer perceptron is

$$\mathbf{y}_i = \phi(\mathbf{x}_i) + \epsilon_i, \tag{7}$$

where $\phi(\mathbf{x}_i)$ represents the output of the multilayer perceptron for training input $i$, and each $\epsilon_i \sim \mathcal{N}(0, \boldsymbol{\Sigma})$ represents the error on each training pair. A probability distribution can then be defined with the density function

$$p(\mathbf{y}_i \mid \mathbf{x}_i) = \frac{1}{\sqrt{(2\pi)^m}} \exp\left(-\frac{1}{2} \left|\left|\mathbf{y}_i - \phi(\mathbf{x}_i)\right|\right|^2\right), \tag{8}$$

where $m$ is the dimension of the vector space to which $\mathbf{y}_1, \ldots, \mathbf{y}_n$ belong. In this model we assume that each error vector is independent and identically distributed so that the joint density of all the training patterns is

$$p(\mathbf{y}_1, \ldots, \mathbf{y}_n \mid \mathbf{x}_1, \ldots, \mathbf{x}_n) = \prod_{i=1}^{n} p(\mathbf{y}_i \mid \mathbf{x}_i). \tag{9}$$

With the training set and the network architecture (e.g., number of hidden nodes, number of hidden layers, transfer functions) specified, the probability distributions in (8) can be points in a manifold [9]. The network parameters are the coordinates of the points. This is called a neuromanifold [10]. Random variables are the tangent vectors at the point $p$. If $\mathbf{v}$ is a random variable, then the point $p + \mathbf{v}$ is (by definition) the distribution with the density function $p \exp(\mathbf{v})$. The score functions in (3) of the network parameters form a basis for the tangent space at a point [12]. If the inner product of this tangent space is defined to be $\langle \mathbf{v}, \mathbf{w} \rangle = E_p\{\mathbf{v}\mathbf{w}\}$, where $E_p$ is the expectation operator with density function $p$, then the Riemannian metric is the Fisher information matrix because the Fisher information between two score functions is $g^{ij} = E_p\{\mathbf{s}^i \mathbf{s}^j\}$ [9, 10, 12]. These definitions are illustrated in the following example.

*1.1.4. Natural Gradient with a Single Neuron.* This is an example of a neuromanifold. Consider a neural network that

consists of a single neuron that is described by a weight vector $\mathbf{w}$ of dimension $d$ and a bias $b$. This example will show how the Fisher information matrix is determined for a neural network. It also shows some of the pitfalls that are encountered when computing its inverse.

The log-likelihood of a distribution $p$ in a neuromanifold is

$$\ell_p = -\frac{m}{2}\log(2\pi) - \frac{1}{2}\sum_{i=1}^{n}||\mathbf{y}_i - \phi(\mathbf{x}_i)||^2. \tag{10}$$

If the network is a single neuron where $\mathbf{x}$ is a $d$-dimensional input vector,

$$\phi(\mathbf{x}) = f(\mathbf{w}\cdot\mathbf{x} + b), \tag{11}$$

and $f$ is a sigmoid function such as tanh or the logistic function, then the score functions are

$$\begin{aligned}\frac{\partial \ell_p}{\partial \mathbf{w}} &= \sum_{i=1}^{n} f'(\mathbf{w}\cdot\mathbf{x}_i + b)[y_i - f(\mathbf{w}\cdot\mathbf{x}_i + b)]\mathbf{x}_i,\\ \frac{\partial \ell_p}{\partial b} &= \sum_{i=1}^{n} f'(\mathbf{w}\cdot\mathbf{x}_i + b)[y_i - f(\mathbf{w}\cdot\mathbf{x}_i + b)].\end{aligned} \tag{12}$$

The gradient of $\ell_p$ is

$$\nabla\ell_p = \begin{bmatrix}\dfrac{\partial \ell_p}{\partial \mathbf{w}}\\ \dfrac{\partial \ell_p}{\partial b}\end{bmatrix}. \tag{13}$$

The Fisher information matrix is $\mathbf{G}_p = E_p\{\nabla\ell_p(\nabla\ell_p)^T\}$. In this case, the straightforward calculation yields

$$\mathbf{G}_p = \sum_{i=1}^{n}[f'(\mathbf{w}\cdot\mathbf{x}_i + b)]^2\begin{bmatrix}\mathbf{x}_i\mathbf{x}_i^T & \mathbf{x}_i\\ \mathbf{x}_i^T & 1\end{bmatrix}, \tag{14}$$

assuming that $E_p\{[y_i - f(\mathbf{w}\cdot\mathbf{x}_i + b)]^2\} = 1$.

One property of sigmoid functions is that their derivatives are small outside of the region near the origin. Therefore, as the norm of $\mathbf{w}$ or the bias $b$ increases, the derivative of the output tends toward zero. Since the derivative is squared in (14), the Fisher information matrix $\mathbf{G}_p$ could become nearly singular.

The matrix $\mathbf{G}_p$ will also be singular unless there are $d$ linearly independent input vectors. For example, if the input vectors of the training set were three-dimensional vectors, then there must be three linearly independent vectors in order to successfully invert $\mathbf{G}_p$. Care must be taken when computing the inverse of $\mathbf{G}_p$. Calculating $\mathbf{G}_p$ becomes much more complicated with the complex network architectures of multilayer perceptrons.

*1.2. Adaptive Natural Gradient.* Multilayer perceptrons can have many parameters. If the dimension of the tangent space $T_p$ is $m$, then $\mathbf{G}_p$ will be an $m \times m$ matrix. Inverting the matrix has computational complexity $O(m^3)$. In the calculation of the natural gradient, the inverse of the Fisher information matrix needs to be known at each step of the algorithm. However, the Fisher information matrix itself does not need to be calculated directly for each step.

Amari et al. developed an algorithm called adaptive natural gradient learning (ANGL) [2]. It uses the matrix inversion lemma [12] to perform a rank 1 update on the inverse of the Fisher information matrix at each step. The parameters are arranged and indexed as a column vector. The score function $\mathbf{s}_k = \nabla\ell_p$ then has the same form and is used in the updated formula

$$\mathbf{G}_{k+1}^{-1} = (1+\delta)\mathbf{G}_k^{-1} - \delta\frac{\mathbf{G}_k^{-1}\mathbf{s}_k\mathbf{s}_k^T\mathbf{G}_k^{-1}}{1 + \mathbf{s}_k^T\mathbf{G}_k^{-1}\mathbf{s}_k}. \tag{15}$$

The variable $k$ is an integer representing the current iteration number of the learning algorithm. At each value of $k$, new values of the network parameters are used to compute the training error vectors. These errors are used to calculate the changes to the network parameters. The inverse of the Fisher information matrix is first initialized as $\mathbf{G}_0^{-1} = \delta^{-1}\mathbf{I}$.

As mentioned in the previous paragraph, the ANGL algorithm stacks all the network parameters into one tall column vector. This is typically done when using a nonlinear least squares algorithm like Levenberg-Marquardt [11]. For networks with multiple layers, this leads to tall vectors and a large Fisher information matrix. Amari avoids inverting the matrix by using the matrix inversion lemma. However, the matrix is still usually large enough to be an issue with memory.

The ANGL algorithm performs well, but it needs sufficient memory to store a large matrix and is sensitive to initial conditions and learning rates.

## 2. A Simplified Natural Gradient Learning Algorithm

This section describes the learning rule used in the SNGL algorithm. First, the directional derivative for a single-layer perceptron is determined. Then the Fisher information matrix for this network is described in detail. These two results are then generalized to find the directional derivative and the Fisher information matrix for a multilayer perceptron. Finally, the section will conclude with a discussion of the computational complexity of SNGL compared to that of ANGL.

The derivation and analysis of the simplified natural gradient learning algorithm begin with the observation that the neural network is parameterized by $(\mathbf{A}, \mathbf{b})$ in the affine transformation $\mathbf{Ax} + \mathbf{b}$. There is one affine transformation for each layer of the network. The formula in (11) was for a single neuron that has only one output. In general, the affine transformations in each layer of a multilayer perceptron can have more than one output. The objective of the algorithm is to minimize the sum of squares of the error vector norms. These norms are calculated with the standard $L_2$ norm of a Cartesian coordinate system. However, that does not mean that the matrix parameters need to be column-ordered into tall vectors. While the homomorphism as done in [2] preserves the vector space operations such as addition and

multiplication by a scalar, it does not preserve multiplying the matrix by a vector. Therefore, it makes sense that the score function of a matrix parameter should itself be a matrix. The Frobenius norm of a $m \times n$ Matrix is $\sqrt{\text{tr}(\mathbf{A}^T\mathbf{A})} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2}$. The inner product associated with this norm is $\langle \mathbf{A}, \mathbf{B} \rangle = \text{tr}(\mathbf{A}^T\mathbf{B})$. The following analysis shows how the derivative of the norm of an error vector can be equivalent to an inner product between matrices.

*2.1. The Directional Derivative.* Assume that the network is a single-layer perceptron with the vector-valued sigmoidal transfer function $\mathbf{F}(\cdot)$. Thus, the log-likelihood function $\ell$ with $n$ training examples that depends on the parameters $\mathbf{A}$ and $\mathbf{b}$ is

$$\ell_\phi = -\frac{1}{2}\sum_{i=1}^{n} ||\mathbf{y}_i - \mathbf{F}(\mathbf{A}\mathbf{x}_i + \mathbf{b})||^2. \tag{16}$$

Furthermore, assume that if the affine transformation $\psi(\mathbf{x}) = \mathbf{C}\mathbf{x} + \mathbf{d}$ is added to $\phi(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$ as $(\phi + \psi)(x) = (\mathbf{A} + \mathbf{C})\mathbf{x} + (\mathbf{b} + \mathbf{d})$, then the minimum point, according to (16), is reached. Thus, $\mathbf{C}$ and $\mathbf{d}$ represent a descent direction.

Let $\gamma(t, \mathbf{x}) = \phi(\mathbf{x}) + t\psi(\mathbf{x}) = (\mathbf{A} + t\mathbf{C})\mathbf{x} + \mathbf{b} + t\mathbf{d}$ be a curve through the manifold for $0 \le t \le 1$. The affine transformation $\gamma(0, \mathbf{x})$ is the initial point, and $\gamma(1, \mathbf{x})$ is the final point. The derivative of this curve with respect to $t$ is $\dot{\gamma}(t, \mathbf{x}) = \mathbf{C}\mathbf{x} + \mathbf{d}$ and is constant with respect to $t$. The log-likelihood along this curve is

$$\ell_\gamma(t) = -\frac{1}{2}\sum_{i=1}^{n} ||\mathbf{y}_i - \mathbf{F}(\gamma(t, \mathbf{x}_i))||^2. \tag{17}$$

The directional derivative of the log-likelihood with respect to $t$ is

$$\begin{aligned} \frac{d\ell_\gamma}{dt} &= \sum_{i=1}^{n} \langle \mathbf{y}_i - \mathbf{F}(\gamma(t, \mathbf{x}_i)), \mathbf{F}'(\gamma(t, \mathbf{x}_i))\dot{\gamma}(t, \mathbf{x}_i) \rangle \\ &= \sum_{i=1}^{n} \left\langle [\mathbf{F}'(\gamma(t, \mathbf{x}_i))]^T [\mathbf{y}_i - \mathbf{F}(\gamma(t, \mathbf{x}_i))], \dot{\gamma}(t, \mathbf{x}_i) \right\rangle, \end{aligned} \tag{18}$$

where $\mathbf{F}'(\cdot)$ is the Jacobian matrix of $\mathbf{F}(\cdot)$. Here, we have isolated $\dot{\gamma}(t, \mathbf{x})$ by using the adjoint of $\mathbf{F}'(\cdot)$ in the second argument of the inner product. Given that $\gamma(0, \mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$ and $\dot{\gamma}(0, \mathbf{x}) = \mathbf{C}\mathbf{x} + \mathbf{d}$, the directional derivative of the log-likelihood function at $t = 0$ is

$$\left.\frac{d\ell_\gamma}{dt}\right|_{t=0} = \sum_{i=1}^{n} \left\langle [\mathbf{F}'(\mathbf{A}\mathbf{x}_i + \mathbf{b})]^T [\mathbf{y}_i - \mathbf{F}(\mathbf{A}\mathbf{x}_i + \mathbf{b})], \mathbf{C}\mathbf{x}_i + \mathbf{d} \right\rangle. \tag{19}$$

Let $\mathbf{z}_i = \mathbf{F}'(\mathbf{A}\mathbf{x}_i + \mathbf{b})^T [\mathbf{y}_i - \mathbf{F}(\mathbf{A}\mathbf{x}_i + \mathbf{b})]$. Then, by using the linearity property of the inner product, (19) can be rewritten as

$$\left.\frac{d\ell_\gamma}{dt}\right|_{t=0} = \sum_{i=1}^{n} \langle \mathbf{z}_i, \mathbf{C}\mathbf{x}_i \rangle + \sum_{i=1}^{n} \langle \mathbf{z}_i, \mathbf{d} \rangle. \tag{20}$$

The matrix $\mathbf{C}$ can be isolated because the matrix inner product $\langle \mathbf{A}, \mathbf{B} \rangle = \text{tr}(\mathbf{A}^T\mathbf{B})$ is being used on the right-hand side. Thus, $\langle \mathbf{z}_i, \mathbf{C}\mathbf{x}_i \rangle = \langle \mathbf{z}_i\mathbf{x}_i^T, \mathbf{C} \rangle$.

Since $\ell_\gamma = \ell_\phi \circ \gamma(t, \mathbf{x})$, then, by the chain rule, $d\ell_\gamma/dt = \langle \nabla\ell_\phi, \dot{\gamma}(t, \mathbf{x}) \rangle$. Thus, $\langle \nabla\ell_\phi, \dot{\gamma}(t, \mathbf{x}) \rangle = \sum_{i=1}^{n} \langle \partial\ell_\phi/\partial\mathbf{A}, \mathbf{C} \rangle + \langle \partial\ell_\phi/\partial\mathbf{b}, \mathbf{d} \rangle = \sum_{i=1}^{n} \langle \mathbf{z}_i\mathbf{x}_i^T, \mathbf{C} \rangle + \langle \mathbf{z}_i, \mathbf{d} \rangle$ and the components of the score function are

$$\begin{aligned} \frac{\partial\ell_\phi}{\partial\mathbf{A}} &= \sum_{i=1}^{n} [\mathbf{F}'(\mathbf{A}\mathbf{x}_i + \mathbf{b})]^T [\mathbf{y}_i - \mathbf{F}(\mathbf{A}\mathbf{x}_i + \mathbf{b})]\mathbf{x}_i^T, \\ \frac{\partial\ell_\phi}{\partial\mathbf{b}} &= \sum_{i=1}^{n} [\mathbf{F}'(\mathbf{A}\mathbf{x}_i + \mathbf{b})]^T [\mathbf{y}_i - \mathbf{F}(\mathbf{A}\mathbf{x}_i + \mathbf{b})]. \end{aligned} \tag{21}$$

The components in (21) would point in the direction of steepest descent if the space of affine transformations was a Euclidean space. To find the direction of steepest descent, then (21) must be corrected for the curvature of the parameter space using the Riemannian metric $\mathbf{G}$.

*2.2. The Fisher Information Matrix.* The Fisher information matrix is the Riemannian metric in this manifold [1]. For the affine transformation $\phi(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$, the Fisher information matrix is

$$\mathbf{G}_\phi = E_p\left\{\nabla\ell_\phi \nabla\ell_\phi^T\right\}. \tag{22}$$

The score function with the components in (21) can be arranged in an augmented matrix form as

$$\nabla\ell_\phi = \begin{bmatrix} \dfrac{\partial\ell_\phi}{\partial\mathbf{A}} & \dfrac{\partial\ell_\phi}{\partial\mathbf{b}} \end{bmatrix}. \tag{23}$$

Thus, in terms of these components, the Fisher information matrix is

$$\begin{aligned} \mathbf{G}_\phi &= E_p\left\{ \begin{bmatrix} \dfrac{\partial\ell_\phi}{\partial\mathbf{A}} & \dfrac{\partial\ell_\phi}{\partial\mathbf{b}} \end{bmatrix} \begin{bmatrix} \left(\dfrac{\partial\ell_\phi}{\partial\mathbf{A}}\right)^T \\ \left(\dfrac{\partial\ell_\phi}{\partial\mathbf{b}}\right)^T \end{bmatrix} \right\} \\ &= E_p\left\{ \frac{\partial\ell_\phi}{\partial\mathbf{A}}\left(\frac{\partial\ell_\phi}{\partial\mathbf{A}}\right)^T + \frac{\partial\ell_\phi}{\partial\mathbf{b}}\left(\frac{\partial\ell_\phi}{\partial\mathbf{b}}\right)^T \right\}. \end{aligned} \tag{24}$$

The partial derivative $\partial\ell_\phi/\partial\mathbf{A}$ in (21) is a sum of $n$ rank 1 matrices. Therefore, the outer product of $\partial\ell_\phi/\partial\mathbf{A}$ with itself is the sum

$$\mathbf{R}_{ij} = [\mathbf{F}'(\mathbf{A}\mathbf{x}_i + \mathbf{b})]^T \epsilon_i \mathbf{x}_i^T \mathbf{x}_j \epsilon_j^T \mathbf{F}'\left(\mathbf{A}\mathbf{x}_j + \mathbf{b}\right), \tag{25}$$

where $\epsilon_i = \mathbf{y}_i - \mathbf{F}(\mathbf{A}\mathbf{x}_i + \mathbf{b})$. When $i \neq j$ then $E_p\{\mathbf{R}_{ij}\} = 0$ because $E_p\{\epsilon_i\epsilon_j\} = E_p\{\epsilon_i\}E_p\{\epsilon_j\} = 0 \cdot 0 = 0$. When $i = j$ then

$$E_p\{\mathbf{R}_{ii}\} = \|\mathbf{x}_i\|^2 [\mathbf{F}'(\mathbf{A}\mathbf{x}_i + \mathbf{b})]^T \boldsymbol{\Sigma} \mathbf{F}'(\mathbf{A}\mathbf{x}_i + \mathbf{b}). \tag{26}$$

However, since $\boldsymbol{\Sigma}$ is not known, it must be estimated. The unbiased estimator is

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{n-1}\sum_{i=1}^{n} \epsilon_i \epsilon_i^T. \tag{27}$$

The Fisher information of the weight matrix $\mathbf{A}$ is estimated as

$$\hat{\mathbf{G}}_{\mathbf{A}} = \frac{1}{n-1}\sum_{i=1}^{n}\|\mathbf{x}_i\|^2[\mathbf{F}'(\mathbf{A}\mathbf{x}_i+\mathbf{b})]^T\epsilon_i\epsilon_i^T\mathbf{F}'(\mathbf{A}\mathbf{x}_i+\mathbf{b}), \quad (28)$$

and the Fisher information of the bias vector $\mathbf{b}$ is estimated as

$$\hat{\mathbf{G}}_{\mathbf{b}} = \frac{1}{n-1}\sum_{i=1}^{n}[\mathbf{F}'(\mathbf{A}\mathbf{x}_i+\mathbf{b})]^T\epsilon_i\epsilon_i^T\mathbf{F}'(\mathbf{A}\mathbf{x}_i+\mathbf{b}). \quad (29)$$

The Fisher information matrix can be estimated as the sum of these two matrices. Combining like terms gives

$$\hat{\mathbf{G}}_{\phi} = \frac{1}{n-1}\sum_{i=1}^{n}\left(\|\mathbf{x}_i\|^2+1\right)[\mathbf{F}'(\phi(\mathbf{x}_i))]^T\epsilon_i\epsilon_i^T\mathbf{F}'(\phi(\mathbf{x}_i)). \quad (30)$$

The direction of steepest descent in (19) is now represented as

$$\mathbf{C} = \hat{\mathbf{G}}_{\phi}^{-1}\frac{\partial\ell_{\phi}}{\partial\mathbf{A}}, \qquad \mathbf{d} = \hat{\mathbf{G}}_{\phi}^{-1}\frac{\partial\ell_{\phi}}{\partial\mathbf{b}}. \quad (31)$$

These equations are for a perceptron with only one layer. Please note that by using the matrix forms, the Fisher information matrix is only a $d \times d$ matrix. If we had stacked the weight matrix and bias vector into a single column vector, then the Fisher information matrix would be $d(d+1)\times d(d+1)$ matrix.

*2.3. Extension to Multilayer Perceptrons.* A multilayer perceptron has $r$ layers each with its own nonlinear sigmoidal function $\mathbf{F}_k(\cdot)$ and affine transformation $\phi_k(\mathbf{x}) = \mathbf{A}_k\mathbf{x}+\mathbf{b}_k$. Each layer $k$ feeds into the one above it with index $k+1$. Thus, the output of layer $k$ is $\mathbf{F}_k\circ\phi_k\circ\mathbf{F}_{k-1}\circ\phi_{k-1}\circ\cdots\circ\mathbf{F}_1\circ\phi_1(\mathbf{x})$. The input layer is defined to be the identity transformation $\phi_0(\mathbf{x}) = \mathbf{x}$ and $\mathbf{F}_0(\mathbf{x}) = \mathbf{x}$. For convenience, let the vector $\mathbf{z}_i^k$ be the output of layer $k$ when given input $\mathbf{x}_i$. The output $\mathbf{z}_i$ can be written recursively,

$$\mathbf{z}_i^k = \mathbf{F}_k\left(\phi_k\left(\mathbf{z}_i^{k-1}\right)\right) \quad (32)$$

with $\mathbf{z}_i^0 = \mathbf{x}_i$.

Two vector spaces $V$ and $W$ can be joined together by means of a direct sum $V\oplus W$ in which stacking a vector $v\in V$ on top of a vector $w\in W$ results in the vector

$$\begin{bmatrix} v \\ w \end{bmatrix} \in V\oplus W. \quad (33)$$

The analog to the direct sum with matrices is to create a block matrix. Given two matrices $A$ and $B$, their direct sum is

$$A\oplus B = \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix}. \quad (34)$$

The direct sum of the score functions of the layers in a multilayer perceptron is used as the analog for stacking the layers. The Fisher information matrix is then a block-diagonal matrix where each square matrix along the main diagonal is the Fisher information matrix for a specific layer.

The score functions for layer $k$ are

$$\begin{aligned} \frac{\partial\ell_{\phi_k}}{\partial\mathbf{A}_k} &= \sum_{i=1}^{n}\mathbf{B}_i^k\left(\mathbf{y}_i-\mathbf{z}_i^r\right)\left(\mathbf{z}_i^{k-1}\right)^T, \\ \frac{\partial\ell_{\phi_k}}{\partial\mathbf{b}_k} &= \sum_{i=1}^{n}\mathbf{B}_i^k\left(\mathbf{y}_i-\mathbf{z}_i^r\right), \end{aligned} \quad (35)$$

where $\mathbf{B}_i^k$ is the backpropagation matrix that is determined with the recurrence relation

$$\begin{aligned} \mathbf{B}_i^r &= \mathbf{F}_r'\left(\phi_r\left(\mathbf{z}_i^{r-1}\right)\right)^T, \\ \mathbf{B}_i^k &= \mathbf{F}_k'\left(\phi_k\left(\mathbf{z}_i^{k-1}\right)\right)^T\mathbf{A}_{k+1}^T\mathbf{B}_i^{k+1}, \end{aligned} \quad (36)$$

where $r$ is the number of layers in the network. Thus, for the output layer, the layer index is $r$ and the backpropagation matrix is the Jacobian matrix of the output layer's transfer function.

Let $\epsilon_i = \mathbf{y}_i - \mathbf{z}_i^r$ be the error for pattern $i$. The estimate of the Fisher information matrix for layer $k$ is

$$\hat{\mathbf{G}}_{\phi_k} = \frac{1}{n-1}\sum_{i=1}^{n}\left(\left\|\mathbf{z}_i^{k-1}\right\|^2+1\right)\mathbf{B}_i^k\epsilon_i\epsilon_i^T\left(\mathbf{B}_i^k\right)^T. \quad (37)$$

Thus, the updates for the parameters in layer $k$ are

$$\mathbf{C}_k = \hat{\mathbf{G}}_{\phi_k}^{-1}\frac{\partial\ell_{\phi_k}}{\partial\mathbf{A}_k}, \qquad \mathbf{d}_k = \hat{\mathbf{G}}_{\phi_k}^{-1}\frac{\partial\ell_{\phi_k}}{\partial\mathbf{b}_k}. \quad (38)$$

*2.4. Computational Complexity.* The new formulation of the Fisher information matrix is significantly smaller than what was used by Amari et al. [2]. Suppose that a two-layer perceptron has $m$ inputs, $n$ outputs, and $p$ hidden nodes. The weight matrices have dimensions $(m+1)\times p$ and $(p+1)\times n$. The addition of 1 to both $m$ and $p$ is due to the extra bias vectors of dimensions $p$ and $n$. If the parameters are column ordered into a single vector, then that vector has dimension $(m+1)p+(p+1)n$. In the ANGL algorithm's Fisher information matrix, there are $((m+1)p+(p+1)n)^2$ elements. The SNGL algorithm's block-diagonal Fisher information matrix has two blocks on the main diagonal that together have $p^2+n^2$ elements. Clearly,

$$p^2+n^2 < (m+1)^2p^2+2(m+1)p(p+1)n+(p+1)^2n^2, \quad (39)$$

if $m$, $n$, and $p$ are positive integers.

It is also less computationally intensive to invert the block matrix used in the SNGL algorithm since the matrix inversion algorithms would have complexity $O(p^3+n^3)$. The complexity to update Amari's recursion in (15) is $O((m+1)^2p^2+2(m+1)p(p+1)n+(p+1)^2n^2)$. This takes into account that ANGL exploits the matrix inversion lemma. Table 1 shows the computational complexity in both space

Table 1: Complexity comparison between the SNGL and ANGL algorithms on other problems.

| Algorithm | SNGL | | ANGL | Ratio | |
|---|---|---|---|---|---|
| Complexity dimension | Space | Time | Both | Space | Time |
| Exclusive OR | 5 | 9 | 81 | 6% | 11% |
| Fisher's Iris data | 25 | 91 | 1,225 | 2% | 7% |
| Mackey-Glass time series | 101 | 1,001 | 3,721 | 3% | 27% |

and time of the ANGL algorithm and the SNGL algorithm for three problems described by Park et al. [3]. The last two columns labelled "Ratio" in Table 1 are the ratios of the space and time complexities of the SNGL algorithm compared to that of the ANGL written as percentages.

*2.5. SNGL Improvements.* There are two more elements of the simplified natural gradient learning algorithm. The first is the regularization of the gradient descent algorithm by adding a prior distribution to the probability density function of the network errors [5]. The second is annealing the learning rate of the algorithm [6]. Neither has any significant impact on the computational complexity. The complexity of the regularization is $O(mn + np)$ for a two-layer network. The learning rate is recalculated once per learning epoch.

*2.5.1. Regularization of Network Parameters.* The goal of regularization is to penalize larger connection weights. This helps to discourage overtraining. The penalty of the weights is governed by the hyperparameter $\alpha$. For a neuron with parameters $\mathbf{w}$ and $b$, the probability density function of these parameters in the prior distribution is

$$p_\alpha(\mathbf{w}, b) = \sqrt{\left(\frac{\alpha}{2\pi}\right)^{q+1}} \exp\left(-\frac{\alpha}{2}\left(\|\mathbf{w}\|^2 + b^2\right)\right), \quad (40)$$

where $q$ is the dimension of the vector space to which $\mathbf{w}$ belongs. Thus, the weight vector is distributed normally as $\mathbf{w} \sim \mathcal{N}(0, (1/\alpha)\mathbf{I})$ and the bias as $b \sim \mathcal{N}(0, 1/\alpha)$. The log-likelihood function is

$$\ell_\alpha(\mathbf{w}, b) = \frac{q+1}{2}\left[\log\alpha - \log(2\pi)\right] - \frac{\alpha}{2}\left(\|\mathbf{w}\|^2 + b^2\right). \quad (41)$$

The score functions of $\mathbf{w}$ and $b$ with respect to $\ell_\alpha$ are, respectively

$$\frac{\partial \ell_\alpha}{\partial \mathbf{w}} = -\alpha\mathbf{w}, \qquad \frac{\partial \ell_\alpha}{\partial b} = -\alpha b. \quad (42)$$

The probability density function of this prior distribution can be added to the probability density function of the network error distribution. Then the updates for the parameters in a multilayer perceptron are

$$\begin{aligned} \frac{\partial \ell_{\phi_k,\alpha}}{\partial \mathbf{A}_k} &= \sum_{i=1}^{n} \mathbf{B}_i^k \left(\mathbf{y}_i - \mathbf{z}_i^r\right)\left(\mathbf{z}_i^{k-1}\right)^T - \alpha\mathbf{A}_k, \\ \frac{\partial \ell_{\phi_k,\alpha}}{\partial \mathbf{b}_k} &= \sum_{i=1}^{n} \mathbf{B}_i^k \left(\mathbf{y}_i - \mathbf{z}_i^r\right) - \alpha\mathbf{b}_k. \end{aligned} \quad (43)$$

When the algorithm is overtraining, the parameters are still increasing. Subtracting $\alpha$ times the current parameter from the update effectively counteracts the update when the norm of that parameter is large enough. The algorithm will eventually reach a steady state where the parameters are longer increasing.

Since $E_{p_\alpha}\{\mathbf{w}\} = 0$, $E_{p_\alpha}\{b\} = 0$, $E_{p_\alpha}\{\mathbf{w}\mathbf{w}^T\} = (1/\alpha)\mathbf{I}$, and $E_{p_\alpha}\{b^2\} = 1/\alpha$, the Fisher information matrix for the regularized algorithm is

$$\hat{\mathbf{G}}_{\phi_k,\alpha} = \frac{1}{n-1}\sum_{i=1}^{n}\left(\left\|\mathbf{z}_i^{k-1}\right\|^2 + 1\right)\mathbf{B}_i^k \epsilon_i \epsilon_i^T \left(\mathbf{B}_i^k\right)^T + \alpha\mathbf{I}. \quad (44)$$

Adding $\alpha\mathbf{I}$ to the Fisher information matrix helps the algorithm avoid inverting an almost singular matrix. Thus, if all the errors are zero, then the Fisher information matrix will be $\alpha\mathbf{I}$. The effect will be multiplying all the weight update elements by $1/\alpha$. This is the largest value the inverse will reach during the SNGL algorithm's execution.

In [13] Park adds regularization to the ANGL algorithm with an Akaike Information Criterion.

*2.5.2. Annealed Learning Rates.* Annealing the learning rate [6] is a technique in which a learning algorithm will slowly decay its learning rate after a specific epoch during the execution of the algorithm. Let the learning rate of the SNGL algorithm be $\eta_t$, where $t$ is the current learning epoch. It is defined as

$$\eta_t = \min\left(\eta_0, \frac{\tau}{t}\right), \quad (45)$$

where $\eta_0$ is the initial learning rate and $\tau$ is a factor that determines which learning epoch begins the annealing process. The epoch in which the decay begins is $t_0 = \lceil \tau/\eta_0 \rceil$.

## 3. Experimental Results

The exclusive OR function was the first Boolean function to be learned by a multilayer perceptron that was not linearly separable [4]. Encoding data with a low density parity check (LDPC) code [14] could be considered the exclusive OR problem's big brother. An LDPC is computed by multiplying (0,1)-vectors with a (0,1)-matrix over $GF(2)$ to get a longer (0,1)-vector. Remember that in $GF(2)$ addition is the XOR operation and multiplication is the AND operation. Formally, a low density parity check code is a code in which extra bits that are parity checks are added to the message.

A matrix is used to represent the transformation of the bits over $GF(2)$. The encoding function is the matrix

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (46)$$

This code has a rate of 1/2 because the code words are twice as long as the message words.

The training set consists of 32 training patterns. The inputs are the 32 possible messages, and the outputs are the 32 10-bit code words corresponding to each message. Thus, the network has 5 input units and 10 output units. The multilayer perceptron will also have 10 hidden units. During the training the bits are changed according to a bit error rate of 0.01. This will help the network deal with bit errors.

First, the performance of the three algorithms will be compared. This will be followed by a comparison of their effective learning rates. Finally, an experiment in which the effective learning rate of the SNGL and ANGL algorithms when these learning rates are used in the ordinary gradient learning (OGL) algorithm as described in [4] will be discussed.

*3.1. Performance Comparison.* Figure 1 shows the sum-squared error of the three algorithms when learning this LDPC encoding function. As can be seen, the performance of the SNGL algorithm is comparable to that of the OGL algorithm, but it reaches convergence in less than 1% of the time.

Table 2 lists the parameters used in all three algorithms and some of the performance metrics. The ANGL algorithm achieves a much lower MSSE. Using the prior probability by adding the current parameters multiplied by the hyperparameter $\alpha$ caused the components of the Fisher information matrix to approach infinity. Therefore, it was omitted, and the weights of the network were allowed to grow without constraint. Thus, the ANGL overtrained the network significantly.

Table 2 may at first seem to be an unfair comparison. However, each of the algorithms were run with and without the regularization and annealed learning rates. The algorithm that performed the best is listed above. This was done to conserve space and simplify the reports. Thus, OGL performs better without an annealed learning rate on this problem. There were several difficulties implementing ANGL with the regularization. It did not perform well with regularization.

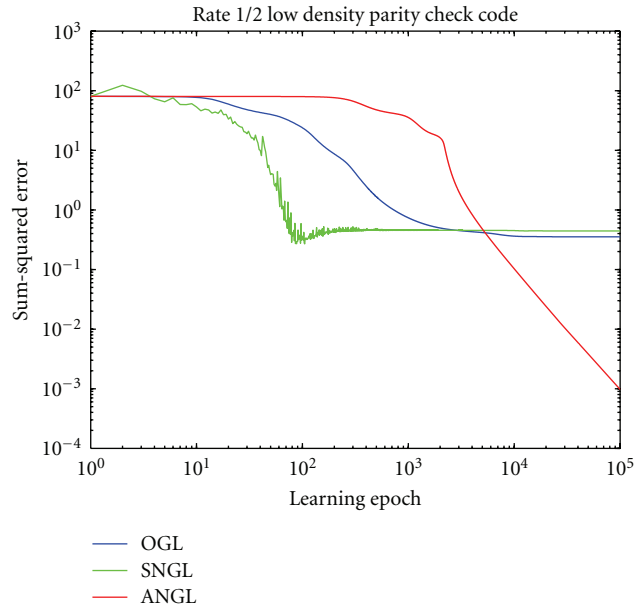


Figure 1: Sum-squared error of the OGL, SNGL, and ANGL algorithms when learning the low density parity check (LDPC).

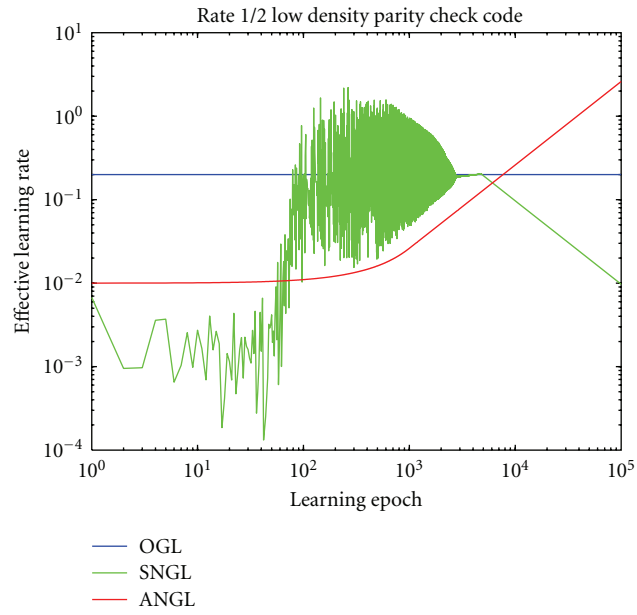


Figure 2: Effective learning rate of the OGL, SNGL, and ANGL algorithms when learning the LDPC encoding function.

*3.2. The Effective Learning Rate.* The effective learning rate is defined to be the product of the baseline learning rate and the smallest eigenvalue of the matrix chosen to represent the inverse of the Fisher information matrix. For OGL the effective learning rate is just the learning rate chosen for the algorithm. For ANGL it is the learning rate multiplied by the smallest eigenvalue of the estimated Fisher information matrix. For SNGL it is the inverse of the largest eigenvalue of the Fisher information matrix multiplied by the learning

Table 2: LDPC experiment results.

| LDPC | OGL | SNGL | ANGL |
|---|---|---|---|
| Hidden units | 10 | 10 | 10 |
| Initial learning rate | 0.2 | 0.2 | 0.01 |
| Annealed learning rate | No | Yes | No |
| Prior probability on parameters | Yes | Yes | No |
| Parameter of prior distribution | 0.001 | 0.001 | N/A |
| Fisher information matrix size | N/A | $10 \times 10 \times 2$ | $170 \times 170$ |
| Adaptation rate | N/A | N/A | $1/t$ |
| Number of extra parameters | 0 | 200 | 28,900 |
| Final learning rate | 0.2 | 0.0097 | 2.6015 |
| Minimum sum-squared error | 0.3540 | 0.2691 | 0.00098 |
| Final learning epoch | 24,852 | 105 | 100,000 |

rate. Figure 2 shows the effective learning rates for these three algorithms. It is a simplified number to give the reader a sense of the step size used by the algorithm during each learning epoch.

The effective learning rate for the SNGL algorithm is small for the first 100 learning epochs. Between the hundredth and the first thousandth epoch, the learning rate becomes quite large, for a step size, becoming larger than one for a few epochs. When the algorithm reaches convergence, then the learning rate begins to decrease due to annealing. The LDPC function is more complicated than XOR, and this is reflected in the number of learning epochs required to reach the three phases of learning for SNGL.

*3.3. Substituting Learning Rates.* One might argue that SNGL and ANGL perform better than OGL because they both use higher learning rates. However, as Figure 2 shows, both of these algorithms have smaller learning rates at first, and the learning rates do not increase until progress is made towards good parameter values.

Figure 3 shows the performance of OGL with the effective learning rates of ANGL and SNGL in addition to OGL with a flat learning rate along with ANGL and SNGL. The plot shows that using the effective learning rate of ANGL improves the performance of OGL. However, the performance of OGL does not improve with the SNGL algorithm's effective learning rate. For this problem, the Fisher information matrix used in SNGL is a block-diagonal matrix with two 10-by-10 blocks on the main diagonal. In the ANGL algorithm, it is a 61-by-61 matrix. These matrices are positive definite. The eigenvalues represent how much information is associated with each direction. The eigenvectors associated with these eigenvalues are these directions. Using the maximum eigenvalue to determine an effective learning rate is good for demonstration purposes. However, it will not improve performance of the algorithm because the directional information is missing.

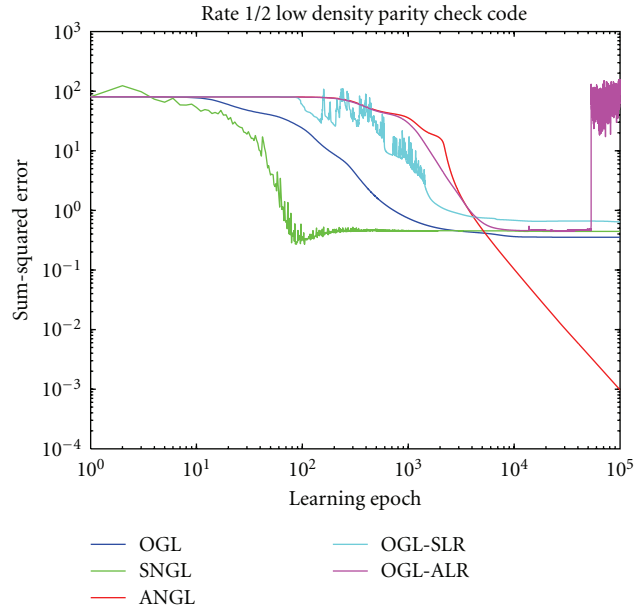


Figure 3: Sum-squared error of the OGL, SNGL, and ANGL algorithms compared to the OGL algorithm with the effective LR of the ANGL and SNGL algorithms.

## 4. Conclusion

This paper developed a simplified natural gradient learning algorithm. Using an algebraic justification to form the gradient of a multilayer perceptron as a block matrix, a smaller block-diagonal Fisher information matrix was discovered. For the three problems studied in the experiments, the Fisher information matrix was much smaller than that in ANGL. The minimum sum-squared error performance of this algorithm is comparable to Amari et al. [2] but uses less memory and computational resources.

## References

[1] S. I. Amari, "Natural gradient works efficiently in learning," *Neural Computation*, vol. 10, no. 2, pp. 251–276, 1998.

[2] S. I. Amari, H. Park, and K. Fukumizu, "Adaptive method of realizing natural gradient learning for multilayer perceptrons," *Neural Computation*, vol. 12, no. 6, pp. 1399–1409, 2000.

[3] H. Park, S. I. Amari, and K. Fukumizu, "Adaptive natural gradient learning algorithms for various stochastic models," *Neural Networks*, vol. 13, no. 7, pp. 755–764, 2000.

[4] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing*, MIT Press, Cambridge, Mass, USA, 1986.

[5] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, New York, NY, USA, 2003.

[6] S. Bös and S. Amari, "Annealed online learning in multilayer neural networks," in *On-line Learning in Neural Networks*, D. Saad, Ed., Cambridge University Press, New York, NY, USA, 1999.

[7] W. Wan, "Implementing online natural gradient learning: Problems and solutions," *IEEE Transactions on Neural Networks*, vol. 17, no. 2, pp. 317–329, 2006.

[8] S. Fiori, "Learning by natural gradient on noncompact matrix-type pseudo-riemannian manifolds," *IEEE Transactions on Neural Networks*, vol. 21, no. 5, pp. 841–852, 2010.

[9] M. K. Murray and J. W. Rice, *Differential Geometry and Statistics*, vol. 48 of *Monographs on Statistics and Applied Probability*, Chapman & Hall/CRC, London, UK, 1993.

[10] S. Amari and H. Nagaoka, *Methods of Information Geometry*, vol. 191 of *Translations of Mathematical Monographs*, Oxford University Press, New York, NY, USA, 2000.

[11] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer Series in Operations Research, Springer, New York, NY, USA, 1999.

[12] T. K. Moon and W. C. Stirling, *Mathematical Methods and Algorithmsfor Signal Processing*, Prentice Hall, Upper Saddle River, NJ, USA, 1999.

[13] H. Park, N. Murata, and S. I. Amari, "Improving generalization performance of natural gradient learning using optimized regularization by NIC," *Neural Computation*, vol. 16, no. 2, pp. 355–382, 2004.

[14] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*, Wiley-Interscience, New York, NY, USA, 2005.