Complexity Issues in Natural Gradient Descent Method for Training Multilayer Perceptrons

Howard Hua Yang
Department of Computer Science, Oregon Graduate Institute, Portland, OR 97291, U.S.A.

Shun-ichi Amari
Laboratory for Information Synthesis, RIKEN Brain Science Institute, Saitama 351-01, Japan

The natural gradient descent method is applied to train an n-m-1 multilayer perceptron. Based on an efficient scheme to represent the Fisher information matrix for an n-m-1 stochastic multilayer perceptron, a new algorithm is proposed to calculate the natural gradient without inverting the Fisher information matrix explicitly. When the input dimension n is much larger than the number of hidden neurons m, the time complexity of computing the natural gradient is $O(n)$.

1 Introduction

Amari (1997, 1998) has shown that the natural gradient descent learning rule is statistically efficient. In principle, this learning rule can be used to train any adaptive system, but the complexity of this rule depends on the architecture of the learning machine. The main difficulty in implementing the rule is designing a fast algorithm to compute the natural gradient. For an n-m-1 multilayer perceptron, we show in this article that the natural gradient can be computed in $O(n)$ flops when $m \ll n$. Here, a flop is a floating-point operation: an add or a multiplication.

Orr and Leen (1997) used the curvature information (the Hessian matrix) in the nonlinear adaptive momentum scheme to optimize the convergence of the stochastic gradient descent. They have shown that the complexity of their algorithm is $O(n)$. However, their algorithm is different from the natural gradient descent. Instead of using the momentum scheme, we use the inverse of the Fisher information matrix to transform the stochastic gradient to optimize the learning dynamics. This method is based on statistical inference. A review of the learning of artificial neural networks is given by Yang, Murata, and Amari (1998).

The rest of this article is organized as follows. The stochastic multilayer perceptron is described in section 2. The natural gradient descent method using the inverse of the Fisher information matrix and its properties are discussed in section 3. A constructive procedure to compute the inverse of the Fisher information matrix is formulated in section 4. For a single-layer perceptron, we calculate an explicit expression of the natural gradient in section 4.1. It is obvious from this expression that the time complexity for computing the natural gradient for the single-layer perceptron is $O(n)$. Importantly, this is also true for a multilayer perceptron, but its proof is not straightforward. To prove this result, we analyze the structure of the Fisher information matrix in section 4.2. Based on this analysis, we discuss the time complexities of computing the inverse of the Fisher information matrix and the natural gradient in section 5 for the multilayer perceptron. We first discuss the simplest multilayer perceptron, a committee machine, in section 5.1, and then discuss the multilayer perceptron in section 5.2. The conclusions are summarized in section 6.

2 Stochastic Multilayer Perceptron

We consider the following stochastic multilayer perceptron model:

$$z = \sum_{i=1}^{m} a_i \varphi(\boldsymbol{w}_i^T \boldsymbol{x} + b_i) + \xi, \tag{2.1}$$

where $(\cdot)^T$ denotes the transpose, $\xi \sim N(0, \sigma^2)$ is the gaussian additive noise with the variance σ^2, and $\varphi(x)$ is a differentiable output function of hidden neurons. Assume that the multilayer network has an n-dimensional input, m hidden neurons, a one-dimensional output, and $m \le n$. Denote $\boldsymbol{a} = (a_1, \cdots, a_m)^T$ the weight vector of the output neuron, $\boldsymbol{w}_i = (w_{1i}, \cdots, w_{ni})^T$ the weight vector of the ith hidden neuron, and $\boldsymbol{b} = (b_1, \cdots, b_m)^T$ the vector of thresholds for the hidden neurons. Let $\boldsymbol{W} = [\boldsymbol{w}_1, \cdots, \boldsymbol{w}_m]$ be a matrix formed by column weight vectors $\boldsymbol{w}_i$; then equation 2.1 can be rewritten as

$$z = \boldsymbol{a}^T \varphi(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}) + \xi.$$

Here, the scalar function φ operates on each component of the vector $\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}$.

3 Fisher Information Matrix and the Natural Gradient Descent

The stochastic perceptron model enables us to derive learning algorithms for neural networks from some statistical inference methods and evaluate the performance of the algorithms systematically.

The joint probability density function (pdf) of the input and the output is

$$p(\boldsymbol{x}, z; \boldsymbol{W}, \boldsymbol{a}, \boldsymbol{b}) = p(z|\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{a}, \boldsymbol{b}) p(\boldsymbol{x}),$$

where $p(z|\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{a}, \boldsymbol{b})$ is the conditional pdf of the output z given the input

$\boldsymbol{x}$ and $p(\boldsymbol{x})$ is the pdf of the input $\boldsymbol{x}$. The loss function is defined as the negative log-likelihood function

$$L(\boldsymbol{x}, z; \boldsymbol{\theta}) = -\log p(\boldsymbol{x}, z; \boldsymbol{\theta}) = l(z|\boldsymbol{x}; \boldsymbol{\theta}) - \log p(\boldsymbol{x}),$$

where

$$\boldsymbol{\theta} = (\boldsymbol{w}_1^T, \cdots, \boldsymbol{w}_m^T, \boldsymbol{a}^T, \boldsymbol{b}^T)^T, \text{ and}$$
$$l(z|\boldsymbol{x}; \boldsymbol{\theta}) = -\log p(z|\boldsymbol{x}; \boldsymbol{\theta}) = \frac{1}{2\sigma^2}(z - \boldsymbol{a}^T\varphi(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b}))^2.$$

Since $p(\boldsymbol{x})$ does not depend on $\boldsymbol{\theta}$, minimizing the loss function $L(\boldsymbol{x}, z; \boldsymbol{\theta})$ is equivalent to minimizing $l(z|\boldsymbol{x}; \boldsymbol{\theta})$.

Given the training set

$$D_T = \{(\boldsymbol{x}_t, z_t),\ t = 1, \cdots, T\},$$

minimizing the loss function $L(D_T; \boldsymbol{\theta}) = \prod_{t=1}^T L(\boldsymbol{x}_t, z_t; \boldsymbol{\theta})$ is equivalent to minimizing the training error

$$E_{\text{tr}}(\boldsymbol{\theta}, D_T) = \frac{1}{T}\sum_{t=1}^{T}(z_t - \boldsymbol{a}^T\varphi(\boldsymbol{W}^T\boldsymbol{x}_t + \boldsymbol{b}))^2.$$

Since $\frac{\partial L}{\partial \boldsymbol{\theta}} = \frac{\partial l}{\partial \boldsymbol{\theta}}$, the Fisher information matrix is defined by

$$\boldsymbol{G} = \boldsymbol{G}(\boldsymbol{\theta}) = E_{\boldsymbol{\theta}}\left[\frac{\partial L}{\partial \boldsymbol{\theta}}(\frac{\partial L}{\partial \boldsymbol{\theta}})^T\right] = E_{\boldsymbol{\theta}}\left[\frac{\partial l}{\partial \boldsymbol{\theta}}(\frac{\partial l}{\partial \boldsymbol{\theta}})^T\right], \tag{3.1}$$

where $E_{\boldsymbol{\theta}}[\cdot]$ denotes the expectation with respect to $p(\boldsymbol{x}, z; \boldsymbol{\theta})$.

Let Θ be a parameter set and $\mathcal{P} = \{p(\boldsymbol{x}, z; \boldsymbol{\theta}) : \ \boldsymbol{\theta} \in \Theta\}$ be a family of pdfs parameterized by $\boldsymbol{\theta} \in \Theta$. The Kullback-Leibler divergence between two pdfs in $\mathcal{P}$ is

$$D(\boldsymbol{\theta}, \boldsymbol{\theta}') = \int d\boldsymbol{x}dz\, p(\boldsymbol{x}, z; \boldsymbol{\theta}) \log \frac{p(\boldsymbol{x}, z; \boldsymbol{\theta})}{p(\boldsymbol{x}, z; \boldsymbol{\theta}')}.$$

Let $\boldsymbol{G}(\boldsymbol{\theta})$ be the Riemannian metric tensor for the Riemannian space Θ. The squared length of a small $d\boldsymbol{\theta}$ in Θ is

$$\|d\boldsymbol{\theta}\|^2_{\boldsymbol{G}(\boldsymbol{\theta})} = d\boldsymbol{\theta}^T\boldsymbol{G}(\boldsymbol{\theta})d\boldsymbol{\theta}.$$

The two spaces $\mathcal{P}$ and Θ are naturally related by the following equation:

$$D(\boldsymbol{\theta}, \boldsymbol{\theta}') = \frac{1}{2}d\boldsymbol{\theta}^T\boldsymbol{G}(\boldsymbol{\theta})d\boldsymbol{\theta} + O(\|d\boldsymbol{\theta}\|^3),$$

where $\boldsymbol{\theta}' = \boldsymbol{\theta} + d\boldsymbol{\theta}$ and $d\boldsymbol{\theta}$ is small.

Amari (1998) showed that in the Riemannian space Θ, for any differentiable loss function $F(\boldsymbol{\theta})$, the steepest descent direction is given by

$$-\boldsymbol{G}^{-1}(\boldsymbol{\theta})\frac{\partial F}{\partial \boldsymbol{\theta}}.$$

Based on this result, Amari (1997) proposed the following natural gradient descent algorithm:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\mu}{t}\boldsymbol{G}^{-1}(\boldsymbol{\theta})\frac{\partial F}{\partial \theta},$$

where μ is a learning rate. In particular, when negative log-likelihood function is chosen as the loss function, the above algorithm is known as the method of scoring in statistics. Amari (1998) showed that this algorithm gives a Fisher efficient online estimator; the asymptotic variance of $\boldsymbol{\theta}_t$ driven by equation 3.5 satisfies

$$E[(\boldsymbol{\theta}_t - \boldsymbol{\theta}^*)(\boldsymbol{\theta}_t - \boldsymbol{\theta}^*)^T \mid \boldsymbol{\theta}^*] \approx \frac{1}{t}\boldsymbol{G}^{-1}(\boldsymbol{\theta}^*), \tag{3.2}$$

which gives the mean square error

$$E[\|\boldsymbol{\theta}_t - \boldsymbol{\theta}^*\|^2 \mid \boldsymbol{\theta}^*] \approx \frac{1}{t}\mathrm{Tr}(\boldsymbol{G}^{-1}(\boldsymbol{\theta}^*)). \tag{3.3}$$

The above property is verified by the simulation results in Yang and Amari (1998).

Yang and Amari (1997a) have shown that $\boldsymbol{G}(\boldsymbol{\theta}) = \frac{1}{\sigma^2}\boldsymbol{A}(\boldsymbol{\theta})$ where $\boldsymbol{A}(\boldsymbol{\theta})$ does not depend on σ^2 (the variance of the additive noise).

Define

$$l_1(z|\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{2}(z - \boldsymbol{a}^T\varphi(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b}))^2$$

and

$$\widetilde{\nabla} l_1(z|\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{A}^{-1}(\boldsymbol{\theta})\frac{\partial l_1}{\partial \boldsymbol{\theta}}(z|\boldsymbol{x};\boldsymbol{\theta}). \tag{3.4}$$

To minimize the training error, we propose the following natural gradient descent algorithm:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\mu}{t}\widetilde{\nabla} l_1(z_t|\boldsymbol{x}_t;\boldsymbol{\theta}_t), \tag{3.5}$$

where the natural gradient $\widetilde{\nabla} l_1(z_t|\boldsymbol{x}_t;\boldsymbol{\theta}_t)$ does not depend on σ^2.

To implement this algorithm, we need to compute the natural gradient $\widetilde{\nabla} l_1(z_t|\boldsymbol{x}_t;\boldsymbol{\theta}_t)$ in each iteration. If we compute the inverse of $\boldsymbol{A}(\boldsymbol{\theta}_t)$ first, then the natural gradient, the time complexity is $O(n^3)$.

We proposed a method (Yang & Amari, 1997b) based on the conjugate gradient method to compute the natural gradient without inverting the Fisher information matrix. The idea is to solve the following linear equation,

$$A(\boldsymbol{\theta}_t)\boldsymbol{y} = \frac{\partial l_1}{\partial \boldsymbol{\theta}}(z_t|\boldsymbol{x}_t; \boldsymbol{\theta}_t),$$

for $\boldsymbol{y} = \widetilde{\nabla} l_1(z_t|\boldsymbol{x}_t; \boldsymbol{\theta}_t)$ without inverting the matrix $A(\boldsymbol{\theta}_t)$. Since the matrix $A(\boldsymbol{\theta})$ is $(n+2)m \times (n+2)m$, it will take at most $(n+2)m$ steps by the conjugate gradient method to compute $\widetilde{\nabla} l_1(z_t|\boldsymbol{x}_t; \boldsymbol{\theta}_t)$ and each step needs $O(n)$ flops to compute a matrix vector product. The number of flops needed to compute the natural gradient by the conjugate gradient method is $O(n^2)$ when $m \ll n$ and $O(n^3)$ when $m = O(n)$. Using the conjugate gradient method to compute the natural gradient is also useful for other probability families when the method of scoring is used to find the maximum likelihood estimates (MLEs). However, the conjugate gradient method is still slow in computing the natural gradient for training the stochastic perceptron. We shall explore the structure of the Fisher information matrix of the stochastic perceptron and develop an algorithm that computes the natural gradient in just $O(n)$ flops.

4 Computing the Natural Gradient for a Stochastic Perceptron

To find an analytic form of the Fisher information matrix, we assume a white gaussian input $\boldsymbol{x} \sim N(\mathbf{0}, \boldsymbol{I})$ with an identity matrix $\boldsymbol{I}$ as the covariance matrix of the input. Yang and Amari (1997a) proposed an efficient scheme to represent the Fisher information matrix for the multilayer perceptron. Due to this scheme, when $m \ll n$, the storage space needed for the Fisher information matrix is $O(n)$ units rather than $O(n^2)$ units, where a unit is the memory space to keep a parameter or a variable. Based on this scheme, we found a constructive procedure to compute the inverse of the Fisher information matrix with $O(n^2)$ flops. This procedure can be improved to compute the natural gradient with only $O(n)$ flops.

4.1 Single-Layer Perceptron. We shall give some explicit expressions of the Fisher information matrix and the natural gradient for a one-layer stochastic perceptron.

Since $z = \varphi(\boldsymbol{w}^T\boldsymbol{x} + b) + \xi$, we have

$$\frac{\partial l}{\partial \boldsymbol{w}} = -\frac{\xi}{\sigma^2}\varphi'(\boldsymbol{w}^T\boldsymbol{x} + b)\boldsymbol{x},$$
$$\frac{\partial l}{\partial b} = -\frac{\xi}{\sigma^2}\varphi'(\boldsymbol{w}^T\boldsymbol{x} + b).$$

The Fisher information matrix is

$$G(\boldsymbol{\theta}) = \frac{1}{\sigma^2}A(\boldsymbol{\theta}) = \frac{1}{\sigma^2}\begin{bmatrix} A_{11} & \boldsymbol{a}_{12} \\ \boldsymbol{a}_{21} & a_{22} \end{bmatrix},$$

where $\boldsymbol{\theta} = (\boldsymbol{w}^T, b)^T$,

$$\begin{aligned}
\boldsymbol{A}_{11} &= E[(\varphi'(\boldsymbol{w}^T\boldsymbol{x}+b))^2\boldsymbol{x}\boldsymbol{x}^T],\\
\boldsymbol{a}_{12} &= \boldsymbol{a}_{21}^T = E[(\varphi'(\boldsymbol{w}^T\boldsymbol{x}+b))^2\boldsymbol{x}],\\
a_{22} &= E[(\varphi'(\boldsymbol{w}^T\boldsymbol{x}+b))^2].
\end{aligned}$$

Let $w = \|\boldsymbol{w}\|$ and $\boldsymbol{u}_1 = \boldsymbol{w}/w$ and extend $\boldsymbol{u}_1$ to an orthogonal basis $\{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n\}$ for $\Re^n$. The gaussian input $\boldsymbol{x}$ is decomposed as $\boldsymbol{x} = \sum_i x_i \boldsymbol{u}_i$; we have $x_i \sim N(0,1)$ and

$$E[x_i x_j] = E[\boldsymbol{u}_i^T \boldsymbol{x}\boldsymbol{x}^T \boldsymbol{u}_j] = \boldsymbol{u}_i^T \boldsymbol{u}_j = \delta_{ij}, \tag{4.1}$$

since $\boldsymbol{x} \sim N(\boldsymbol{0}, \boldsymbol{I})$. The outer product $\boldsymbol{x}\boldsymbol{x}^T$ can be written as

$$\boldsymbol{x}\boldsymbol{x}^T = x_1^2 \boldsymbol{u}_1\boldsymbol{u}_1^T + \sum_{j=2}^n x_j x_1 (\boldsymbol{u}_j \boldsymbol{u}_1^T + \boldsymbol{u}_1 \boldsymbol{u}_j^T) + \sum_{j,k=2}^n x_j x_k \boldsymbol{u}_j \boldsymbol{u}_k^T. \tag{4.2}$$

Noticing $\boldsymbol{w}^T\boldsymbol{x} = wx_1$ and applying equations 4.1 and 4.2 we have,

$$\begin{aligned}
\boldsymbol{A}_{11} &= E[(\varphi'(\boldsymbol{w}^T\boldsymbol{x}+b))^2\boldsymbol{x}\boldsymbol{x}^T]\\
&= E[(\varphi'(wx_1+b))^2x_1^2]\boldsymbol{u}_1\boldsymbol{u}_1^T\\
&\quad + E[(\varphi'(wx_1+b))^2x_1]\sum_{j=2}^n E[x_j](\boldsymbol{u}_j\boldsymbol{u}_1^T + \boldsymbol{u}_1\boldsymbol{u}_j^T)\\
&\quad + E[(\varphi'(wx_1+b))^2]\sum_{j,k=2}^n E[x_jx_k]\boldsymbol{u}_j\boldsymbol{u}_k^T\\
&= E[(\varphi'(wx_1+b))^2x_1^2]\boldsymbol{u}_1\boldsymbol{u}_1^T + E[(\varphi'(wx_1+b))^2]\sum_{j=2}^n \boldsymbol{u}_j\boldsymbol{u}_j^T.
\end{aligned}$$

Let us define three integrals:

$$d_0(w,b) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}(\varphi'(wx+b))^2 e^{-\frac{x^2}{2}}dx > 0, \tag{4.3}$$

$$d_1(w,b) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}(\varphi'(wx+b))^2 x e^{-\frac{x^2}{2}}dx, \tag{4.4}$$

$$d_2(w,b) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}(\varphi'(wx+b))^2 x^2 e^{-\frac{x^2}{2}}dx > 0. \tag{4.5}$$

Then we have

$$\begin{aligned}
\boldsymbol{A}_{11} &= d_2(w,b)\boldsymbol{u}_1\boldsymbol{u}_1^T + d_0(w,b)\sum_{j=2}^n \boldsymbol{u}_j\boldsymbol{u}_j^T,\\
\boldsymbol{a}_{12} &= E[(\varphi'(wx_1+b))^2x_1]\boldsymbol{u}_1 = d_1(w,b)\boldsymbol{u}_1, \quad \text{and}\\
a_{22} &= d_0(w,b).
\end{aligned}$$

$\boldsymbol{A}_{11}$ can be written as

$$\boldsymbol{A}_{11} = d_0(w,b)\boldsymbol{I} + (d_2(w,b) - d_0(w,b))\boldsymbol{u}_1\boldsymbol{u}_1^T,$$

since $\{\boldsymbol{u}_i\}$ are orthogonal and

$$\sum_{k=2}^{n} \boldsymbol{u}_i\boldsymbol{u}_i^T = \boldsymbol{I} - \boldsymbol{u}_1\boldsymbol{u}_1^T.$$

Summarizing previous calculations for the blocks in $\boldsymbol{G}(\boldsymbol{\theta})$, we have

$$\boldsymbol{A}_{11} = d_0(w,b)\boldsymbol{I} + (d_2(w,b) - d_0(w,b))\boldsymbol{w}\boldsymbol{w}^T/w^2, \tag{4.6}$$

$$\boldsymbol{a}_{12} = \boldsymbol{a}_{21}^T = d_1(w,b)\boldsymbol{w}/w, \quad a_{22} = d_0(w,b). \tag{4.7}$$

Saad and Solla (1995) used the scaled error function

$$\varphi(x) = \mathrm{erf}(\frac{x}{\sqrt{2}}) = \frac{2}{\sqrt{\pi}}\int_0^{\frac{x}{\sqrt{2}}} e^{-t^2}dt$$

as the sigmoid function for the hidden units in order to obtain an analytic expression of the generalization error. If we choose $\varphi(x)$ as the sigmoid function for the hidden neurons, we obtain the following close forms for the integrals 4.3 through 4.5:

$$d_0(w,b) = \frac{\sqrt{2}}{\pi\sqrt{w^2+0.5}}\exp\{-b^2 + \frac{b^2w^2}{w^2+0.5}\},$$

$$d_2(w,b) = d_0(w,b)\frac{1}{w^2+0.5}(\frac{1}{2} + \frac{w^2b^2}{w^2+0.5}),$$

$$d_1(w,b) = -\frac{\sqrt{2}wb}{\pi(w^2+0.5)^{3/2}}\exp\{-b^2 + \frac{b^2w^2}{w^2+0.5}\}.$$

Even for the single-layer perceptron, the size of the Fisher information matrix is $(n+1)\times(n+1)$. However, the Fisher information matrix can be generated online with $O(n^2)$ flops by equations 4.6 and 4.7. If one wants to trace the Fisher information matrix, one need only $O(n)$ units to store $\boldsymbol{w}$ and b in each iteration.

To compute the inverse of $\boldsymbol{A}(\boldsymbol{\theta})$, we need the following well-known inverse formula of a four-block matrix:

Lemma 1. $$\begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{21} & \boldsymbol{B}_{22} \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{B}^{11} & \boldsymbol{B}^{12} \\ \boldsymbol{B}^{21} & \boldsymbol{B}^{22} \end{bmatrix}$$

provided $|\boldsymbol{B}_{11}| \neq 0$ *and* $|\boldsymbol{B}_{22} - \boldsymbol{B}_{21}\boldsymbol{B}_{11}^{-1}\boldsymbol{B}_{12}| \neq 0$. *Here,*
$\boldsymbol{B}^{11} = \boldsymbol{B}_{11}^{-1} + \boldsymbol{B}_{11}^{-1}\boldsymbol{B}_{12}\boldsymbol{B}_{22,1}^{-1}\boldsymbol{B}_{21}\boldsymbol{B}_{11}^{-1}$,
$\boldsymbol{B}_{22,1} = \boldsymbol{B}_{22} - \boldsymbol{B}_{21}\boldsymbol{B}_{11}^{-1}\boldsymbol{B}_{12}$,
$\boldsymbol{B}^{22} = \boldsymbol{B}_{22,1}^{-1}$,
and $\boldsymbol{B}^{12} = (\boldsymbol{B}^{21})^T = -\boldsymbol{B}_{11}^{-1}\boldsymbol{B}_{12}\boldsymbol{B}_{22,1}^{-1}$.

Applying the above inverse formula of a four-block matrix, we obtain

$$A^{-1}(\boldsymbol{\theta}) = \begin{bmatrix} \boldsymbol{A}^{11} & \boldsymbol{a}^{12} \\ \boldsymbol{a}^{21} & a^{22} \end{bmatrix}$$

where

$$\boldsymbol{A}^{11} = \frac{1}{d_0}\boldsymbol{I} + \left(\frac{1}{d_2} - \frac{1}{d_0}\right)\frac{\boldsymbol{w}\boldsymbol{w}^T}{w^2} + \frac{d_1^2}{d_2(d_0d_2 - d_1^2)}\frac{\boldsymbol{w}\boldsymbol{w}^T}{w^2}, \tag{4.8}$$

$$\boldsymbol{a}^{12} = (\boldsymbol{a}^{21})^T = -\frac{d_1d_2}{d_2(d_0d_2 - d_1^2)}\boldsymbol{w}, \tag{4.9}$$

$$a^{22} = \frac{d_2}{d_0d_2 - d_1^2}. \tag{4.10}$$

By using equations 4.8 and 4.9, the time complexity for computing $\boldsymbol{A}^{-1}(\boldsymbol{\theta})$ is $O(n^2)$. If we compute the inverse of the Fisher information matrix and the function $l_1(z|\boldsymbol{x};\boldsymbol{\theta})$ in two separate steps and then multiply them together, the time complexity for computing the natural gradient is $O(n^2)$. Instead of computing the natural gradient in this way, we compute it in a single step by applying equations 4.8 and 4.9:

$$\begin{aligned} \widetilde{\nabla} l_1(z|\boldsymbol{x};\boldsymbol{\theta}) &= -h(\boldsymbol{x}, z; \boldsymbol{w}, b) \\ &\times \begin{bmatrix} \frac{\boldsymbol{x}}{d_0} + (\frac{1}{d_2} - \frac{1}{d_0} + \frac{d_1^2}{d_2(d_0d_2 - d_1^2)})\frac{\boldsymbol{w}^T\boldsymbol{x}}{w^2}\boldsymbol{w} - \frac{d_1d_2}{d_2(d_0d_2 - d_1^2)}\boldsymbol{w} \\ -\frac{d_1d_2\boldsymbol{w}^T\boldsymbol{x}}{d_2(d_0d_2 - d_1^2)} + \frac{d_2}{d_0d_2 - d_1^2} \end{bmatrix}, \end{aligned} \tag{4.11}$$

where $h(\boldsymbol{x}, z; \boldsymbol{w}, b) = (z - \varphi(\boldsymbol{w}^T\boldsymbol{x} + b))\varphi'(\boldsymbol{w}^T\boldsymbol{x} + b)$. It needs only $O(n)$ flops to compute the natural gradient $\widetilde{\nabla} l_1(z|\boldsymbol{x};\boldsymbol{\theta})$ by equation 4.11.

4.2 Multilayer Perceptron. The main difficulty in implementing the natural gradient descent algorithm (see equation 3.5) is to compute the natural gradient online. The number of parameters in the stochastic perceptron defined by equation 2.1 is $m(n+2)$, so the size of the Fisher information matrix is $m(n+2) \times m(n+2)$. In this article, we assume $m \ll n$. To compute the inverse of an $n \times n$ matrix, it generally needs $O(n^3)$ flops by commonly used methods such as the lower and upper triangular factorization together with the triangular matrix inverse (Stewart, 1973) .

In the previous section, we showed some procedures for computing the Fisher information matrix and the natural gradient for a one-layer perceptron. Following similar procedures, Yang and Amari (1997a) found a scheme to represent the Fisher information matrix for multilayer perceptrons. Based on this scheme, we gave a method that requires $O(n^2)$ flops to compute the inverse of the Fisher information matrix. In this article, we adapt this method to compute the natural gradient $\widetilde{\nabla} l_1(z|\boldsymbol{x};\boldsymbol{\theta})$. We shall show that the adapted method requires only $O(n)$ flops to compute the natural gradient.

We first briefly describe this scheme to represent the Fisher information matrix and then introduce the method to compute the natural gradient.

We need some notations for block matrix. An $m \times n$ block matrix $\boldsymbol{X}$ is denoted by $\boldsymbol{X} = [\boldsymbol{X}_{ij}]_{[m\times n]}$ or $\boldsymbol{X} = [\boldsymbol{X}_{ij}]_{i=1,\cdots,m,j=1,\cdots,n}$.

Let

$$\boldsymbol{A}(\boldsymbol{\theta}) = [\boldsymbol{A}_{ij}]_{[(m+2)\times(m+2)]}$$

be a partition of $\boldsymbol{A}(\boldsymbol{\theta})$ corresponding to the partition of

$$\boldsymbol{\theta} = (\boldsymbol{w}_1^T, \cdots, \boldsymbol{w}_m^T, \boldsymbol{a}^T, \boldsymbol{b}^T)^T.$$

Define

$$\begin{aligned} \boldsymbol{u}_i &= \boldsymbol{w}_i/\|\boldsymbol{w}_i\|, i = 1, \cdots, m, \\ \boldsymbol{U}_1 &= [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m], \text{ and} \\ [\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m] &= \boldsymbol{U}_1(\boldsymbol{U}_1^T\boldsymbol{U}_1)^{-1}. \end{aligned}$$

Let $r_{ij} = \boldsymbol{u}_i^T\boldsymbol{u}_j$, $\boldsymbol{R}_1 = (r_{ij})_{m\times m}$ and $\boldsymbol{R}_1^{-1} = (r^{ij})_{m\times m}$. Define $x_i' = \boldsymbol{u}_i^T\boldsymbol{x}$ for $i = 1, \cdots, m$; then

$$(x_1', \cdots, x_m') \sim N(\boldsymbol{0}, \boldsymbol{R}_1).$$

The structure of the blocks in $\boldsymbol{A}(\boldsymbol{\theta})$ is elucidated by the following two lemmas proved in Yang and Amari (1997a), which gives an efficient scheme to represent the Fisher information matrix for the stochastic multilayer perceptron.

Lemma 2. *For $1 \le i, j \le m$, and $a_i \ne 0$,*

$$\boldsymbol{A}_{ij} = a_i a_j (c_{ij}\Omega_0 + \sum_{l,k=1}^{m} c_{ij}^{lk}\boldsymbol{u}_l\boldsymbol{v}_k^T) \tag{4.12}$$

where

$$\Omega_0 = \sum_{k=m+1}^{n} \boldsymbol{u}_k\boldsymbol{u}_k^T = \boldsymbol{I} - \sum_{k=1}^{m}\boldsymbol{u}_k\boldsymbol{v}_k^T, \tag{4.13}$$

$$c_{ij} = E[\varphi'(w_i x_i' + b_i)\varphi'(w_j x_j' + b_j)], \tag{4.14}$$

$$c_{ij}^{lk} = E\left[\varphi'(w_i x_i' + b_i)\varphi'(w_j x_j' + b_j)\left(\sum_{s=1}^{m} r^{ls}x_s'\right)x_k'\right]. \tag{4.15}$$

Lemma 3. *For $1 \le i \le m$,*

$$\boldsymbol{A}_{i,m+1} = \boldsymbol{A}^T_{m+1,i} = \left(\sum_{k=1}^{m} c^k_{i1} \boldsymbol{v}_k, \cdots, \sum_{k=1}^{m} c^k_{im} \boldsymbol{v}_k \right), \tag{4.16}$$

where

$$c^k_{ij} = E[\varphi'(w_i x'_i + b_i)\varphi(w_j x'_j + b_j) x'_k], \quad 1 \le i, j, k \le m. \tag{4.17}$$

$\boldsymbol{A}_{i,m+2}$ has the same structure as $\boldsymbol{A}_{i,m+1}$:

$$\boldsymbol{A}_{i,m+2} = \boldsymbol{A}^T_{m+2,i} = \left(\sum_{k=1}^{m} \tilde{c}^k_{i1} \boldsymbol{v}_k, \cdots, \sum_{k=1}^{m} \tilde{c}^k_{im} \boldsymbol{v}_k \right), \tag{4.18}$$

where

$$\tilde{c}^k_{ij} = a_i a_j E[\varphi'(w_i x'_i + b_i)\varphi'(w_j x'_j + b_j) x'_k], \quad 1 \le i, j, k \le m.$$

$$\boldsymbol{A}_{m+1,m+1} = (b_{ij})_{m\times m} \tag{4.19}$$

with $b_{ij} = E[\varphi(w_i x'_i + b_i)\varphi(w_j x'_j + b_j)]$ is a function of b_i, b_j, w_i, w_j and r_{ij}.

$$\boldsymbol{A}_{m+1,m+2} = \boldsymbol{A}^T_{m+2,m+1} = (\tilde{b}_{ij})_{m\times m}, \tag{4.20}$$

with $\tilde{b}_{ij} = a_j E[\varphi(w_i x'_i + b_i)\varphi'(w_j x'_j + b_j)]$.

$$\boldsymbol{A}_{m+2,m+2} = (b'_{ij})_{m\times m} \tag{4.21}$$

with $b'_{ij} = a_i a_j E[\varphi'(w_i x'_i + b_i)\varphi'(w_j x'_j + b_j)]$.

Assume $\varphi(x) = \text{erf}(x/\sqrt{2})$ where $\text{erf}(u)$ is the error function. It is not difficult to obtain analytic expressions for c_{ij}, b'_{ij}, c^{lk}_{ij}, and $\tilde{c}^k_{ij}$. When $b_i = 0, i = 1, \cdots, m$, applying the multivariate gaussian average method in Saad and Solla (1995), we can obtain analytic expressions for these coefficients: c^k_{ij}, b_{ij}, and $\tilde{b}_{ij}$. But when $b_i \ne 0$, the analytic expressions for these coefficients seemingly do not exist. These coefficients are m-dimensional integrals, which can be approximated offline by some numerical integral methods.

Recalling that n is the input dimension and m is the number of hidden neurons, in this article, we assume $m \ll n$. Using lemmas 2 and 3, instead of using $O(n^2)$ units to store the Fisher information matrix, we only need $O(n)$ units to store the vectors $\boldsymbol{u}_k$ and $\boldsymbol{v}_k$. $O(n^2)$ flops are needed to construct the Fisher information matrix from these vectors.

5 Complexity Issues

We shall discuss the time complexity to compute $\boldsymbol{A}(\boldsymbol{\theta})$, $\boldsymbol{A}^{-1}(\boldsymbol{\theta})$ and the natural gradient for the multilayer perceptron. By lemmas 2 and 3, to compute the blocks in $\boldsymbol{A}(\boldsymbol{\theta})$, we need to compute $\boldsymbol{u}_k$ and $\boldsymbol{v}_k$, $k = 1, \cdots, m$. It needs $O(n)$ flops to compute $\boldsymbol{u}_k$ and $\boldsymbol{v}_k$ by their definitions. The time complexity to compute $\boldsymbol{A}(\boldsymbol{\theta})$ is $O(n^2)$ since each outer product $\boldsymbol{u}_k\boldsymbol{v}_k^T$ needs $O(n^2)$ flops. It is shown in Yang and Amari (1997a) that about $O(n^2)$ flops are needed to compute the inverse of the Fisher information matrix. If we compute the Fisher information matrix first and then compute the natural gradient, this will take $O(n^2)$ flops in each iteration to implement the natural gradient descent. This is in the same order as the conjugate gradient method proposed by Yang and Amari (1997b). In the rest of this article, we show that the time complexity of computing the natural gradient for the multilayer perceptron can be significantly improved by using the representation scheme in lemmas 2 and 3 and by computing the natural gradient without inverting the Fisher information matrix. We shall show that we can compute the natural gradient directly with only $O(n)$ flops. To show this, we reexamine the process of computing the inverse of the Fisher information matrix.

We need the following notations:

$$Gl(m, \Re) = \{\boldsymbol{A} \in \Re^{m\times m} : \det(\boldsymbol{A}) \neq 0\}.$$

$$\mathcal{M} = \left\{ a_0\Omega_0 + \sum_{i,j=1}^{m} a_{ij}\boldsymbol{u}_i\boldsymbol{v}_j^T : a_0 \neq 0,\ \boldsymbol{A} = (a_{ij}) \in Gl(m, \Re) \right\}.$$

$$\overline{\mathcal{M}} = \left\{ a_0\Omega_0 + \sum_{i,j=1}^{m} a_{ij}\boldsymbol{u}_i\boldsymbol{v}_j^T \right\} \quad \text{(the closure of } \mathcal{M}\text{)}.$$

$$\widetilde{\mathcal{M}} = \{[a_0, \boldsymbol{A}] : a_0 \neq 0,\ \boldsymbol{A} \in Gl(m, \Re)\}.$$

$$\overline{\widetilde{\mathcal{M}}} = \{[a_0, \boldsymbol{A}] : a_0 \in \Re,\ \boldsymbol{A} \in \Re^{m\times m}\}(\text{ the closure of } \widetilde{\mathcal{M}}\ ).$$

$\widetilde{\mathcal{M}}$ is a subset of $\overline{\widetilde{\mathcal{M}}}$. An element in $\overline{\widetilde{\mathcal{M}}}$ is denoted by $[a_0, \boldsymbol{A}]$ or $[a_0, (a_{ij})]$. Define a mapping $\psi :\ \overline{\widetilde{\mathcal{M}}} \to \Re^{n\times n}$,

$$\psi(\widetilde{A}) = a_0\Omega_0 + \sum_{i,j=1}^{m} a_{ij}\boldsymbol{u}_i\boldsymbol{v}_j^T \in \overline{\mathcal{M}}, \quad \text{for } \widetilde{A} = [a_0, (a_{ij})] \in \overline{\widetilde{\mathcal{M}}}.$$

For $\widetilde{A} = [a_0, \boldsymbol{A}]$ and $\widetilde{B} = [b_0, \boldsymbol{B}] \in \overline{\widetilde{\mathcal{M}}}$, we define multiplication, summation, and scaling in $\overline{\widetilde{\mathcal{M}}}$ by

$$\begin{aligned} \widetilde{A} \star \widetilde{B} &= [a_0 b_0, \boldsymbol{AB}], \\ \widetilde{A} + \widetilde{B} &= [a_0 + b_0, \boldsymbol{A} + \boldsymbol{B}], \\ a[a_0, \boldsymbol{A}] &= [aa_0, a\boldsymbol{A}]. \end{aligned}$$

It is easy to verify that $\widetilde{\mathcal{M}}$ is a group. For $\widetilde{A} = [a_0, \boldsymbol{A}] \in \widetilde{\mathcal{M}}$, its inverse element is $\widetilde{A}^{-1} = [\frac{1}{a_0}, \boldsymbol{A}^{-1}]$.

Note that $\mathcal{M}$ is the image set of $\widetilde{\mathcal{M}}$ under the mapping ψ. It is not difficult to show that $\psi : \widetilde{\mathcal{M}} \to \mathcal{M}$ is an isomorphism, meaning that ψ is one-to-one and $\psi(\widetilde{A})\psi(\widetilde{B}) = \psi(\widetilde{A} \star \widetilde{B})$ for $\widetilde{A}$ and $\widetilde{B} \in \widetilde{\mathcal{M}}$. This implies that $\mathcal{M}$ is a subgroup of $Gl(n, \Re)$ and for $\boldsymbol{C} = a_0\Omega_0 + \sum_{i,j=1}^{m} a_{ij}\boldsymbol{u}_i\boldsymbol{v}_j^T \in \mathcal{M}$,

$$\boldsymbol{C}^{-1} = \frac{1}{a_0}\Omega_0 + \sum_{i,j=1}^{m} a^{ij}\boldsymbol{u}_i\boldsymbol{v}_j^T \tag{5.1}$$

where the matrix (a^{ij}) is the inverse of the matrix $(a_{ij})^{-1}$.

The following lemma, proved in Yang and Amari (1997a), gives the relations between $\mathcal{M}$ and $\overline{\mathcal{M}}$.

Lemma 4.

$$\mathcal{M} = \overline{\mathcal{M}} \bigcap Gl(n, \Re). \tag{5.2}$$

We shall apply equation 5.1 and lemma 4 to study the structure of $\boldsymbol{A}^{-1}(\boldsymbol{\theta})$.

5.1 Committee Machine. Let us first consider a committee machine that is a multilayer perceptron with $a_i = 1$ and $b_i = 0$ for $i = 1, \cdots, m$.

Repeatedly applying the inverse formula of the four-block matrix, formula 5.1, and lemma 4, we know that $\boldsymbol{A}^{-1}(\boldsymbol{\theta})$ has the same structure as $\boldsymbol{A}(\boldsymbol{\theta})$: $\boldsymbol{A}^{-1}(\boldsymbol{\theta}) = [\boldsymbol{A}^{ij}]_{i=1,\cdots,m, j=1,\cdots,m}$, where every $\boldsymbol{A}^{ij} \in \overline{\mathcal{M}}$. In this process, all matrix multiplications, summations, and scalings are carried out in the space $\overline{\widetilde{\mathcal{M}}}$, and all matrix inversions are made in the group $\widetilde{\mathcal{M}}$. Therefore, the time cost for computing the representations of $\boldsymbol{A}^{ij}$ in $\overline{\widetilde{\mathcal{M}}}$ is $O(m^4)$, which does not depend on n. Only when each block $\boldsymbol{A}^{ij}$ is realized in the matrix space $\Re^{n\times n}$, the time complexity of computing $\boldsymbol{A}^{-1}(\boldsymbol{\theta})$ is $O(n^2)$ since realizing each $\boldsymbol{u}_k\boldsymbol{v}_k^T$ needs n^2 flops of computations. We shall avoid computing the outer product $\boldsymbol{u}_k\boldsymbol{v}_k^T$ in computing the natural gradient $\widetilde{\nabla} l_1$.

Let

$$\tilde{\nabla} l_1 = \begin{bmatrix} (\tilde{\nabla} l_1)_{\boldsymbol{w}_1} \\ \vdots \\ (\tilde{\nabla} l_1)_{\boldsymbol{w}_m} \end{bmatrix}$$

be a partition of $\tilde{\nabla} l_1$ corresponding to the partition of $\boldsymbol{\theta} = (\boldsymbol{w}_1^T, \cdots, \boldsymbol{w}_m^T)^T$. Here each $(\tilde{\nabla} l_1)_{\boldsymbol{w}_i}$ is an $n \times 1$ vector.

From the block representation of $\boldsymbol{A}^{-1}(\boldsymbol{\theta})$, we have

$$(\tilde{\nabla} l_1)_{\boldsymbol{w}_i} = \sum_{j=1}^{m} \boldsymbol{A}^{ij} \frac{\partial l_1}{\partial \boldsymbol{w}_j}.$$

Noticing $\boldsymbol{A}^{ij} \in \overline{\mathcal{M}}$, we have

$$\boldsymbol{A}^{ij} = b_{ij}\Omega_0 + \sum_{k,l=1}^{m} b_{ij}^{kl} \boldsymbol{u}_k \boldsymbol{v}_l^T,$$

where the coefficients b_{ij} and b_{ij}^{kl} are determined in the process of computing $\boldsymbol{A}^{-1}(\boldsymbol{\theta})$. From the above expression of $\boldsymbol{A}^{ij}$, we obtain

$$(\tilde{\nabla} l_1)_{\boldsymbol{w}_i} = \sum_{j=1}^{m} \left[b_{ij} \frac{\partial l_1}{\partial \boldsymbol{w}_j} - b_{ij} \sum_{k=1}^{m} \left(\boldsymbol{v}_k^T \frac{\partial l_1}{\partial \boldsymbol{w}_j} \right) \boldsymbol{u}_k + \sum_{k,l=1}^{m} b_{ij}^{kl} \left(\boldsymbol{v}_l^T \frac{\partial l_1}{\partial \boldsymbol{w}_j} \right) \boldsymbol{u}_k \right]. \quad (5.3)$$

It requires only the vector inner product, the vector addition, and no matrix multiplication to compute the above expression, so the time complexity for computing the natural gradient $\tilde{\nabla} l_1$ is $O(n)$.

In summary, for the committee machine, $O(n^2)$ flops due to the vector outer product and the matrix addition are required to compute the inverse of the Fisher information matrix, and $O(n)$ flops due to vector inner product and vector addition are needed to compute the natural gradient by formula 5.3. In the next section, we show that this conclusion is also true for the multilayer perceptron.

5.2 Multilayer Perceptron. Recalling that

$$\boldsymbol{A}(\boldsymbol{\theta}) = [\boldsymbol{A}_{ij}]_{[(m+2)\times(m+2)]}$$

is a partition of $\boldsymbol{A}(\boldsymbol{\theta})$ corresponding to the partition of

$$\boldsymbol{\theta} = (\boldsymbol{w}_1^T, \cdots, \boldsymbol{w}_m^T, \boldsymbol{a}^T, \boldsymbol{b}^T)^T,$$

we consider a new partition of $\boldsymbol{A}(\boldsymbol{\theta})$ as:

$$\boldsymbol{A}(\boldsymbol{\theta}) = \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{12}^T & \boldsymbol{B}_{22} \end{bmatrix},$$

where

$$\begin{aligned}
\boldsymbol{B}_{11} &= [\boldsymbol{A}_{ij}]_{i=1,\cdots,m, j=1,\cdots,m},\\
\boldsymbol{B}_{12} &= [\boldsymbol{A}_{ij}]_{i=1,\cdots,m, j=m+1,m+2},\\
\boldsymbol{B}_{22} &= [\boldsymbol{A}_{ij}]_{i=m+1,m+2, j=m+1,m+2}.
\end{aligned}$$

Assuming that both $\boldsymbol{B}_{11}$ and $\boldsymbol{B}_{22} - \boldsymbol{B}_{21}\boldsymbol{B}_{11}^{-1}\boldsymbol{B}_{12}$ are nonsingular and applying the four-block inverse formula, we have

$$\boldsymbol{A}^{-1}(\boldsymbol{\theta}) = \begin{bmatrix} \boldsymbol{B}^{11} & \boldsymbol{B}^{12} \\ \boldsymbol{B}^{21} & \boldsymbol{B}^{22} \end{bmatrix},$$

where $\boldsymbol{B}^{ij}$ are defined in the same way as lemma 1.

To characterize the structure of these blocks $\boldsymbol{B}^{ij}$ in $\boldsymbol{A}^{-1}(\boldsymbol{\theta})$ we need some notations and lemmas.

Define

$$\begin{aligned}
\mathcal{M}_{\boldsymbol{u}} &= \{\boldsymbol{A} \in \Re^{n\times m} : \text{ each column of } \boldsymbol{A} \text{ belongs to } R(\boldsymbol{U}_1)\ \},\\
\mathcal{M}_{\boldsymbol{v}} &= \{\boldsymbol{A} \in \Re^{n\times m} : \text{ each column of } \boldsymbol{A} \text{ belongs to } R(\boldsymbol{V}_1)\ \}.
\end{aligned}$$

Here, $R(\boldsymbol{U}_1)$ is the range of $\boldsymbol{U}_1$ generated by all linear combinations of the columns in $\boldsymbol{U}_1$. $R(\boldsymbol{V}_1)$ is defined in the same way from $\boldsymbol{V}_1$.

Lemma 5. *If $\boldsymbol{A} \in \overline{\mathcal{M}}$ and $\boldsymbol{B} \in \mathcal{M}_{\boldsymbol{v}}$, then $\boldsymbol{AB} \in \mathcal{M}_{\boldsymbol{u}}$ and $\boldsymbol{AB}$ has the following form,*

$$\left[\sum_{i=1}^{m} a_1^i \boldsymbol{u}_i, \cdots, \sum_{i=1}^{m} a_m^i \boldsymbol{u}_i\right],$$

where the coefficients a_j^i are computed in $O(n)$ flops.

The proof of lemma 5 is given in appendix A.

Lemma 6. *Let $\boldsymbol{A} \in \mathcal{M}_{\boldsymbol{u}}$ and $\boldsymbol{B} \in \mathcal{M}_{\boldsymbol{v}}$; then the complexity for computing the $m \times m$ matrix $\boldsymbol{B}^T\boldsymbol{A}$ is $O(n)$.*

The proof of lemma 6 is given in appendix B.

Lemma 7. *If $\boldsymbol{A} \in \mathcal{M}_{\boldsymbol{u}}$ and $\boldsymbol{B} \in \Re^{m\times m}$, then $\boldsymbol{AB} \in \mathcal{M}_{\boldsymbol{u}}$. $\boldsymbol{AB}$ has the following form,*

$$\left[\sum_{i=1}^{m} b_1^i \boldsymbol{u}_i, \cdots, \sum_{i=1}^{m} b_m^i \boldsymbol{u}_i\right],$$

where the coefficients b_j^i are computed in m^3 flops.

The proof of lemma 7 is given in appendix C.

Let us recall the definition of $\boldsymbol{B}^{11}$:

$$\begin{aligned}\boldsymbol{B}^{11} &= \boldsymbol{B}_{11}^{-1} + \boldsymbol{B}_{11}^{-1}\boldsymbol{B}_{12}\boldsymbol{B}_{22,1}^{-1}\boldsymbol{B}_{21}\boldsymbol{B}_{11}^{-1}, \\ \boldsymbol{B}_{22,1} &= \boldsymbol{B}_{22} - \boldsymbol{B}_{21}\boldsymbol{B}_{11}^{-1}\boldsymbol{B}_{12}.\end{aligned} \tag{5.4}$$

The matrix $\boldsymbol{B}_{11}$ has the same structure as the Fisher information matrix for the committee machine, so $\boldsymbol{B}_{11}^{-1}$ is an $m \times m$ matrix with each of its blocks belonging to $\overline{\mathcal{M}}$. Therefore, the time complexity for computing $\boldsymbol{B}_{11}^{-1}$ is the same as that for the committee machine. However, we shall show below that the blocks in $\boldsymbol{B}_{11}^{-1}\boldsymbol{B}_{12}\boldsymbol{B}_{22,1}^{-1}\boldsymbol{B}_{21}\boldsymbol{B}_{11}^{-1}$ do not belong to $\overline{\mathcal{M}}$, and the time complexity for computing these blocks needs to be discussed separately.

To discuss the time complexity of computing the blocks $\boldsymbol{B}^{ij}$, we need the following lemma:

Lemma 8. *The $m \times 2$ block matrix $\boldsymbol{B}_{11}^{-1}\boldsymbol{B}_{12}$ has the form*

$$\boldsymbol{B}_{11}^{-1}\boldsymbol{B}_{12} = [\boldsymbol{X}_{ij}]_{[m\times 2]},$$

where

$$\boldsymbol{X}_{ij} = \left[\sum_{k=1}^{m} x_{ij}^{k1}\boldsymbol{u}_k, \cdots, \sum_{k=1}^{m} x_{ij}^{km}\boldsymbol{u}_k\right], \tag{5.5}$$

and $O(n)$ flops are needed to compute the coefficients x_{ij}^{kl}.

The proof of lemma 8 is given in appendix D.

Lemma 9. *It takes $O(n)$ flops to compute the $2m \times 2m$ matrix $\boldsymbol{B}_{22,1}$. $\boldsymbol{B}_{11}^{-1}\boldsymbol{B}_{12}\boldsymbol{B}_{22,1}^{-1}\boldsymbol{B}_{21}\boldsymbol{B}_{11}^{-1} = [\boldsymbol{Z}_{ij}]_{[m\times m]}$ is an $m \times m$ block matrix and each block $\boldsymbol{Z}_{ij}$*

has the following form:

$$\boldsymbol{Z}_{ij} = \sum_{k,l=1}^{m} z_{ij}^{kl} \boldsymbol{u}_k \boldsymbol{u}_l^T.$$

It takes $O(n)$ flops to compute these coefficients z_{ij}^{kl} in the above expression.

The proof of lemma 9 is given in appendix E.

Theorem 1. *The complexity for computing $\boldsymbol{B}^{22} = \boldsymbol{B}_{22,1}^{-1}$ is $O(n)$. The $m \times 2$ block matrix $\boldsymbol{B}^{12}$ has the following form:*

$$\boldsymbol{B}^{12} = -\boldsymbol{B}_{11}^{-1} \boldsymbol{B}_{12} \boldsymbol{B}_{22,1}^{-1} = -[\boldsymbol{Y}_{ij}]_{[m\times 2]},$$

where

$$\boldsymbol{Y}_{ij} = \left[\sum_{k=1}^{m} y_{ij}^{k1} \boldsymbol{u}_k, \cdots, \sum_{k=1}^{m} y_{ij}^{km} \boldsymbol{u}_k \right], \tag{5.6}$$

and $O(n)$ flops are needed to compute the coefficients y_{ij}^{kl}.

The $m \times m$ block matrix $\boldsymbol{B}^{11}$ has the following form:

$$\boldsymbol{B}^{11} = [\widetilde{\boldsymbol{B}}_{ij}]_{[m\times m]},$$

where

$$\widetilde{\boldsymbol{B}}_{ij} = a_{ij}\Omega_0 + \sum_{k,l=1}^{m} b_{kl}^{ij} \boldsymbol{u}_k \boldsymbol{v}_l^T + \sum_{k,l=1}^{m} z_{kl}^{ij} \boldsymbol{u}_k \boldsymbol{u}_l^T, \tag{5.7}$$

and $O(n)$ flops are needed to compute these coefficients a_{ij}, b_{kl}^{ij} and z_{kl}^{ij}.

Proof. By *(1)* in lemma 9, it takes $O(n)$ flops to compute $\boldsymbol{B}_{22,1}$, which is of $2m \times 2m$. Therefore, $O(n)$ flops are needed to compute $\boldsymbol{B}_{22,1}^{-1}$. This is shown in the second part of the proof for lemma 9. By equation 5.4 and the second statement in lemma 9, $\boldsymbol{B}^{11}$ is an $m \times m$ block matrix. The (i, j)th block in $\boldsymbol{B}^{11}$ is expressed as equation 5.7, and $O(n)$ flops are needed to compute these coefficients a_{ij}, b_{kl}^{ij} and z_{kl}^{ij}.

The computation of the natural gradient is based on theorem 1. Let

$$\widetilde{\nabla} l_1 = \begin{bmatrix} (\widetilde{\nabla} l_1)_{\boldsymbol{w}_1} \\ \vdots \\ (\widetilde{\nabla} l_1)_{\boldsymbol{w}_m} \\ (\widetilde{\nabla} l_1)_{\boldsymbol{a}} \\ (\widetilde{\nabla} l_1)_{\boldsymbol{b}} \end{bmatrix}$$

be the partition of $\widetilde{\nabla} l_1$. By theorem 1,

$$(\widetilde{\nabla} l_1)_{\boldsymbol{w}_1} = \sum_{i=1}^{m} \widetilde{\boldsymbol{B}}_{1i} \frac{\partial l_1}{\partial \boldsymbol{w}_i} - \boldsymbol{Y}_{11} \frac{\partial l_1}{\partial \boldsymbol{a}} - \boldsymbol{Y}_{12} \frac{\partial l_1}{\partial \boldsymbol{b}}.$$

Applying equations 5.6 and 5.7, we have

$$(\widetilde{\nabla} l_1)_{\boldsymbol{w}_1} = \sum_{k=1}^{m} \left(a_{1k} \frac{\partial l_1}{\partial \boldsymbol{w}_k} + \widetilde{a}_{1k} \boldsymbol{u}_k \right), \tag{5.8}$$

where a_{1k} are the coefficients of $\widetilde{\boldsymbol{B}}_{1k}$ in equation 5.7 and

$$\widetilde{a}_{1k} = -\sum_{i=1}^{m} a_{1i} \boldsymbol{v}_k^T \frac{\partial l_1}{\partial \boldsymbol{w}_i} + \sum_{i,l=1}^{m} \left(b_{kl}^{1i} \boldsymbol{v}_l^T \frac{\partial l_1}{\partial \boldsymbol{w}_i} + z_{kl}^{1i} \boldsymbol{u}_l^T \frac{\partial l_1}{\partial \boldsymbol{w}_i} \right)$$
$$- \sum_{l=1}^{m} \left(y_{11}^{kl} \frac{\partial l_1}{\partial \boldsymbol{a}_l} + y_{12}^{kl} \frac{\partial l_1}{\partial \boldsymbol{b}_l} \right).$$

Therefore, $(\widetilde{\nabla} l_1)_{\boldsymbol{w}_1}$ is a linear combination of $\frac{\partial l_1}{\partial \boldsymbol{w}_i}$ and $\boldsymbol{u}_k$, and it requires $O(n)$ flops to compute a_{1k} and $\widetilde{a}_{1k}$ due to the vector inner product and the vector addition. This is also true for computing other $(\widetilde{\nabla} l_1)_{\boldsymbol{w}_i}$.

Let $\boldsymbol{B}^{22} = [\boldsymbol{C}_{ij}]_{[2\times 2]}$. Again applying equation 5.6, we have

$$(\widetilde{\nabla} l_1)_{\boldsymbol{a}} = \sum_{i=1}^{m} \boldsymbol{Y}_{i1}^T \frac{\partial l_1}{\partial \boldsymbol{w}_i} + \boldsymbol{C}_{11} \frac{\partial l_1}{\partial \boldsymbol{a}} + \boldsymbol{C}_{12} \frac{\partial l_1}{\partial \boldsymbol{b}}, \tag{5.9}$$

where

$$\boldsymbol{Y}_{i1}^T \frac{\partial l_1}{\partial \boldsymbol{w}_i} = \left[\sum_{k=1}^{m} y_{i1}^{k1} \boldsymbol{u}_k^T \frac{\partial l_1}{\partial \boldsymbol{w}_i}, \cdots, \sum_{k=1}^{m} y_{i1}^{km} \boldsymbol{u}_k^T \frac{\partial l_1}{\partial \boldsymbol{w}_i} \right]^T$$

involving the vector inner product $\boldsymbol{u}_k^T \frac{\partial l_1}{\partial \boldsymbol{w}_i}$. Note that each $\boldsymbol{C}_{ij}$ is of $m \times m$. So, it only needs $O(n)$ flops to compute $(\widetilde{\nabla} l_1)_{\boldsymbol{a}}$. This is also true for $(\widetilde{\nabla} l_1)_{\boldsymbol{b}}$.

Corollary 1. *The time complexity for inverting the Fisher information matrix is $O(n^2)$. The time complexity for computing the natural gradient $\boldsymbol{A}(\boldsymbol{\theta})^{-1} \frac{\partial l_1}{\partial \boldsymbol{\theta}}$ is $O(n)$.*

In summary, it is easy to generalize formulas 5.8 and 5.9 to compute $(\widetilde{\nabla} l_1)_{\boldsymbol{w}_i}$ and $(\widetilde{\nabla} l_1)_{\boldsymbol{b}}$. For the multilayer perceptron, the natural gradient can be computed in $O(n)$ flops by these formulas.

6 Conclusion

The complexity for computing the Fisher information matrix and the natural gradient is discussed. Based on the efficient scheme to represent the Fisher information matrix, the process of inverting the Fisher information matrix is examined in detail. The structure of the Fisher information matrix and the natural gradient for the multilayer perceptron is elaborated by this analysis. We show that the time complexity for inverting the Fisher information matrix is $O(n^2)$ and the time complexity for computing the natural gradient is $O(n)$.

Appendix A:

Proof of Lemma 5

Extend $\{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m\}$ to $\{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m, \boldsymbol{u}_{m+1}, \cdots, \boldsymbol{u}_n\}$, a basis in $\Re^{n\times n}$, such that

$$\text{for } j > m,\ \boldsymbol{u}_j \perp \mathcal{L}(\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m) \tag{A.1}$$

$$\boldsymbol{u}_j^T \boldsymbol{u}_k = \delta_{j,k} \text{ (delta notation)}, \quad \text{for } j, k = m+1, \cdots, n, \tag{A.2}$$

where $\mathcal{L}(\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m)$ is the vector space spanned by $\{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m\}$.

Let $\boldsymbol{U} = [\boldsymbol{U}_1, \boldsymbol{u}_{m+1}, \cdots, \boldsymbol{u}_n]$ and $\boldsymbol{V} = [\boldsymbol{V}_1, \boldsymbol{u}_{m+1}, \cdots, \boldsymbol{u}_n]$. Then noticing that each column of $\boldsymbol{V}_1$ is in $\mathcal{L}(\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m)$, we have $\boldsymbol{V}^T\boldsymbol{U} = \boldsymbol{I}$. Therefore, $\boldsymbol{V}^T = \boldsymbol{U}^{-1}$ and $\boldsymbol{U}\boldsymbol{V}^T = I$, which implies that

$$\sum_{i=1}^{m} \boldsymbol{u}_i \boldsymbol{v}_i + \sum_{i=m+1}^{n} \boldsymbol{u}_i \boldsymbol{u}_i^T = \boldsymbol{I},$$

that is,

$$\Omega_0 = \boldsymbol{I} - \sum_{i=1}^{m} \boldsymbol{u}_i \boldsymbol{v}_i = \sum_{i=m+1}^{n} \boldsymbol{u}_i \boldsymbol{u}_i^T. \tag{A.3}$$

Let $\boldsymbol{A} = a_0\Omega_0 + \sum_{ij} a_{ij} \boldsymbol{u}_i \boldsymbol{v}_j^T$ and $\boldsymbol{B} \in \mathcal{M}_{\boldsymbol{v}}$. Each column of $\boldsymbol{B}$ is a linear combination of $\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m\}$. Let $\boldsymbol{b} = \sum_{i=1}^{m} c_i \boldsymbol{v}_i$. Then

$$\boldsymbol{A}\boldsymbol{b} = \left(a_0 \sum_{i=m+1}^{n} \boldsymbol{u}_i \boldsymbol{u}_i^T + \sum_{i,j=1}^{m} a_{ij} \boldsymbol{u}_i \boldsymbol{v}_j^T \right) \sum_{i=1}^{m} c_i \boldsymbol{v}_i$$

$$= \sum_{i,j,k=1}^{m} a_{ij} c_k \boldsymbol{v}_j^T \boldsymbol{v}_k \boldsymbol{u}_i$$
$$= \sum_{i=1}^{m} a^i \boldsymbol{u}_i,$$

where $a^i = \sum_{j,k=1}^{m} a_{ij} c_k \boldsymbol{v}_j^T \boldsymbol{v}_k$ and $O(n)$ flops are needed to compute these coefficients a^i. Expressing $\boldsymbol{AB}$ as the following form,

$$\boldsymbol{AB} = \left[\sum_{i=1}^{m} a_1^i \boldsymbol{u}_i, \cdots, \sum_{i=1}^{m} a_m^i \boldsymbol{u}_i \right],$$

we need $O(n)$ flops to compute the coefficients a_j^i.

Appendix B:

Proof of Lemma 6

Let $\boldsymbol{A} = [\sum_{i=1}^{m} a_1^i \boldsymbol{u}_i, \cdots, \sum_{i=1}^{m} a_m^i \boldsymbol{u}_i]$ and $\boldsymbol{B} = [\sum_{i=1}^{m} b_1^i \boldsymbol{v}_i, \cdots, \sum_{i=1}^{m} b_m^i \boldsymbol{v}_i]$. Then the (k,l)th element of $\boldsymbol{B}^T \boldsymbol{A}$ is $\sum_{i,j=1}^{m} b_k^i a_l^j \boldsymbol{v}_i^T \boldsymbol{u}_j$, which is computed in $O(n)$ flops.

Appendix C:

Proof of Lemma 7

For $\boldsymbol{A} \in \mathcal{M}_{\boldsymbol{u}}$ and $\boldsymbol{B} \in \Re^{m\times m}$, each column of $\boldsymbol{AB}$ has the form $\boldsymbol{Ax}$ for $\boldsymbol{x} \in \Re^m$. Let $\boldsymbol{A} = [\sum_{i=1}^{m} c_1^i \boldsymbol{u}_i, \cdots, \sum_{i=1}^{m} c_m^i \boldsymbol{u}_i]$ and $\boldsymbol{x} = [x_1, \cdots, x_m]^T$. Then

$$\boldsymbol{Ax} = \sum_{i=1}^{m} \left(\sum_{k=1}^{m} c_k^i x_k \right) \boldsymbol{u}_i = \sum_{i=1}^{m} b^i \boldsymbol{u}_i,$$

and m^2 flops are needed to compute these coefficients b^i. Hence, when $\boldsymbol{AB}$ is expressed as the following form,

$$\boldsymbol{AB} = \left[\sum_{i=1}^{m} b_1^i \boldsymbol{u}_i, \cdots, \sum_{i=1}^{m} b_m^i \boldsymbol{u}_i \right],$$

m^3 flops are needed to compute the coefficients b_j^i.

Appendix D:

Proof of Lemma 8

$\boldsymbol{B}_{11}$ has the same structure as that of the Fisher information matrix for the committee machine. $\boldsymbol{B}_{11}^{-1}$ is an $m \times m$ block matrix, and each of its m^2 blocks has the following form,

$$\widetilde{a}_0 \Omega_0 + \sum_{l,k=1}^{m} \widetilde{c}_{ij}^{lk} \boldsymbol{u}_l \boldsymbol{v}_k^T,$$

and the complexity for computing these coefficients $\widetilde{a}_0$ and $\widetilde{c}_{ij}^{lk}$ does not depend on n.

$\boldsymbol{B}_{12}$ is an $m \times 2$ block matrix with blocks belonging to $\mathcal{M}_{\boldsymbol{v}}$. By lemma 5, $\boldsymbol{B}_{11}^{-1}\boldsymbol{B}_{12}$ is an $m \times 2$ block matrix, and each of its $2m$ blocks belongs to $\mathcal{M}_{\boldsymbol{u}}$ with the form of equation 5.5 and the complexity for computing x_{ij}^{kl} is $O(n)$.

Appendix E:

Proof of Lemma 9

Applying lemmas 6 and 8, it takes $O(n)$ flops to compute the $2m \times 2m$ matrix $\boldsymbol{B}_{12}^T(\boldsymbol{B}_{11}^{-1}\boldsymbol{B}_{12})$, so $O(n)$ flops are needed to compute $\boldsymbol{B}_{22,1} = \boldsymbol{B}_{22} - \boldsymbol{B}_{12}^T(\boldsymbol{B}_{11}^{-1}\boldsymbol{B}_{12})$.

By lemmas 7 and 8,

$$\boldsymbol{B}_{11}^{-1}\boldsymbol{B}_{12}\boldsymbol{B}_{22,1}^{-1} = \begin{bmatrix} \boldsymbol{Y}_{11} & \boldsymbol{Y}_{12} \\ \vdots & \vdots \\ \boldsymbol{Y}_{m1} & \boldsymbol{Y}_{m2} \end{bmatrix},$$

where $\boldsymbol{Y}_{ij} = [\sum_{k=1}^m y_{ij}^{k1}\boldsymbol{u}_k, \cdots, \sum_{k=1}^m y_{ij}^{km}\boldsymbol{u}_k]$.

Also by lemma 8,

$$\boldsymbol{B}_{21}\boldsymbol{B}_{11}^{-1} = \boldsymbol{B}_{12}^T\boldsymbol{B}_{11}^{-1} = (\boldsymbol{B}_{11}^{-1}\boldsymbol{B}_{12})^T = \begin{bmatrix} \boldsymbol{X}_{11}^T & \cdots & \boldsymbol{X}_{m1}^T \\ \boldsymbol{X}_{12}^T & \cdots & \boldsymbol{X}_{m2}^T \end{bmatrix}.$$

Hence,

$$\boldsymbol{B}_{11}^{-1}\boldsymbol{B}_{12}\boldsymbol{B}_{22,1}^{-1}\boldsymbol{B}_{21}\boldsymbol{B}_{11}^{-1} = [\boldsymbol{Z}_{ij}]_{[m\times m]},$$

where $\boldsymbol{Z}_{ij} = \boldsymbol{Y}_{i1}\boldsymbol{X}_{j1}^T + \boldsymbol{Y}_{i2}\boldsymbol{X}_{j2}^T = \sum_{k,l=1}^m z_{ij}^{kl}\boldsymbol{u}_k\boldsymbol{u}_l^T$ and $O(n)$ flops are needed to compute the coefficients z_{ij}^{kl}.

References

Amari, S. (1997). Neural learning in structured parameter spaces—Natural Riemannian gradient. In M. C. Mozer, M. I. Jordan, & T. Petsche (Eds.), *Advances in neural information processing systems, 9* (pp. 127–133) Cambridge, MA: MIT Press.

Amari, S. (1998). Natural gradient works efficiently in learning. *Neural Computation, 10,* 251–276.

Orr, G. B., & Leen, T. K. (1997). Using curvature information for fast stochastic search. In M. C. Mozer, M. I. Jordan & T. Petsche (Eds.), *Advances in neural information processing systems, 9.* Cambridge, MA: MIT Press.

Saad, D., & Solla, S. A. (1995). On-line learning in soft committee machines. *Physical Review E, 52,* 4225–4243.

Stewart, G. W. (1973). *Introduction to matrix computations.* New York: Academic Press.

Yang, H. H., & Amari, S. (1997a). Natural gradient descent for training multi-layer perceptrons. Unpublished manuscript.

Yang, H. H., & Amari, S. (1997b). Training multi-layer perceptrons by natural gradient descent. In *ICONIP'97 Proceedings,* New Zealand.

Yang, H. H., & Amari, S. (1998). The efficiency and the robustness of natural gradient descent learning rule. In M. I. Jordan, M. J. Kearns, & S. A. Solla (Eds.), *Advances in neural information processing systems, 10.* Cambridge, MA: MIT Press.

Yang, H. H., Murata, N., & Amari, S. (1998). Statistical inference: Learning in artificial neural networks. *Trends in Cognitive Sciences, 2*(1), 4–10.

Received August 11, 1997; accepted April 16, 1998.