

A Neural Network model with Bidirectional Whitening

Yuki Fujimoto* and Toru Ohira**

Graduate School of Mathematics, Nagoya University, Nagoya, Japan

*E-mail: m15042x@math.nagoya-u.ac.jp **E-mail: ohira@math.nagoya-u.ac.jp

Abstract

We present here a new model and algorithm which performs an efficient Natural gradient descent for Multilayer Perceptrons. Natural gradient descent was originally proposed from a point of view of information geometry, and it performs the steepest descent updates on manifolds in a Riemannian space. In particular, we extend an approach taken by the “Whitened neural networks” model. We make the whitening process not only in feed-forward direction as in the original model, but also in the back-propagation phase. Its efficacy is shown by an application of this “Bidirectional whitened neural networks” model to a handwritten character recognition data (MNIST data).

1 Introduction

Interests for developing and efficient learning algorithm for multilayer neural networks have grown rapidly due to recent upheaval of the deep learning and other machine learnings. Natural gradient descent(NGD) is considered as one of the strong methods. It was proposed from a point of view of information geometry[1], where neural networks are considered as manifolds in a Riemannian space with a measure given by the Fisher information matrix (FIM). Then, the learning process can be interpreted as an optimization problem of a function in a Riemannian space. The idea of applying the NGD to multilayer neural networks was initiated by Amari. Recently, it has regained interests from machine learning researchers[6, 9].

However, difficulty exists for using the NGD: the computational costs of estimating the FIM and obtaining its inverse is high. Much attention and research efforts have gone into solving this difficulty[4, 7, 8, 5, 10].

In this paper, we will focus on one of such approaches, and extend the work of [4]. In their approach “Whitened neural networks” model was proposed. There, a neural network architecture, whose FIM is closer to the identity matrix with less computational demands, is explored. Extra neurons and connections are

added to achieve this whitening approximation. In particular, they have used this scheme for the forward direction of inputs to neurons and achieved lower computational costs.

Our main proposal in this paper is to further push the approximation of the FIM being closer to the identity by implementing the whitening process also in the back-propagation phase. This model, which we term as the “bidirectional whitened neural networks” model, will be described in the following. Its efficacy is also shown through its application to a handwritten character recognition data (MNIST data).

2 Multilayer Perceptron and Natural Gradient Descent

We present here a brief review of the Multilayer Perceptron and the Natural Gradient Descent, which we focus in this paper. The first level of approximation for the FIM is also discussed.

2.1 Multilayer Perceptron

Multilayer Perceptron is a model of neural networks which has feed-forward structure with no recurrent loops. They have multiple layers called input, hidden, and output, and neurons have all to all connections between successive layers. Let us consider a N layer Perceptron, and set the values of the input as $\mathbf{z}^{(0)} = \mathbf{x}$, the hidden layer values as $\mathbf{z}^{(i)} = \mathbf{h}^{(i)}$, ($1 \leq i \leq N - 1$), and the output of the entire network as $\mathbf{z}^{(N)} = f(\mathbf{x}; \mathbf{w})$.

This $f(\mathbf{x}; \mathbf{w})$ can be viewed as a function of \mathbf{x} by fixing the parameters \mathbf{w} , and thus called as a “multilayer Perceptron function”. The rules of computing the value of the i layer from the $i - 1$ in the network is given as follows ($1 \leq i \leq N$).

$$\mathbf{a}^{(i)} = \mathbf{W}^{(i)} \mathbf{z}^{(i-1)} + \mathbf{b}^{(i)} \quad (1)$$

$$= \bar{\mathbf{W}}^{(i)} \bar{\mathbf{z}}^{(i-1)} \quad (2)$$

$$\mathbf{z}^{(i)} = \phi^{(i)}(\mathbf{a}^{(i)}) \quad (3)$$

Here, $\phi^{(i)}(\cdot)$ is an activation function applied to each element of \mathbf{a} . Typically, the sigmoid function or *ReLU* function are used for this activation function. Also, (2) is a shortened notation by setting $\bar{\mathbf{W}}^{(i)} \equiv (\mathbf{b}^{(i)}, \mathbf{W}^{(i)})$, $\bar{\mathbf{z}}^{(i)} \equiv (1, \mathbf{z}^{(i)T})^T$.

Hence, the multilayer Perceptron function (MPF) is defined by setting $\{(\mathbf{W}^{(i)}, \mathbf{b}^{(i)})\}$. It is often convenient to denote these parameters by \mathbf{w} , defined by

$$\mathbf{w} \equiv (\text{vec}(\bar{\mathbf{W}}^{(1)})^T, \dots, \text{vec}(\bar{\mathbf{W}}^{(N)})^T)^T \quad (4)$$

where $\text{vec}(A)$ means a compound vector of column vectors of a matrix A

The learning process of multilayer Perceptrons is an optimization problem set by the following statistical inference. The training data of input and output

pairs is given as $D \equiv \{(\mathbf{x}_k, \mathbf{y}_k)\}_{k=1}^K$. We assume this data set is generated by the same joint distribution $Q(X, Y)$ independently. In order to estimate this input output probabilistic relations, a statistical model $\{p(\mathbf{x}, \mathbf{y}; \mathbf{w})\}_{\mathbf{w} \in \Theta}$ is considered using the MPF. Here $p(\mathbf{x}, \mathbf{y}; \mathbf{w})$ is a joint probability density function and $\Theta \subset \mathbb{R}^M$ is a set of parameters. The problem is to find the parameter \mathbf{w} which makes $p(\mathbf{x}, \mathbf{y}; \mathbf{w})$ as a best estimate of $Q(X, Y)$. The maximum likelihood method is employed to obtain such \mathbf{w}^* .

$$\mathbf{w}^* \equiv \arg \max_{\mathbf{w} \in \Theta} \prod_{k=1}^K p(\mathbf{x}_k, \mathbf{y}_k; \mathbf{w}) \quad (5)$$

It is known that this estimation is the same as the following minimization problem.

$$\mathbf{w}^* \equiv \arg \min_{\mathbf{w} \in \Theta} \sum_{k=1}^K -\log p(\mathbf{x}_k, \mathbf{y}_k; \mathbf{w}) \quad (6)$$

$$= \arg \min_{\mathbf{w} \in \Theta} M(\mathbf{w}) \quad (7)$$

Here, we have set the target function to minimize as $M(\mathbf{w})$. Research on efficient algorithms for this optimization problem is the central issue in the following.

2.2 Natural Gradient Method

Natural Gradient Method is a steepest descent method in a Riemannian space. It is proposed from the information geometry where statistical models are manifolds in a Riemannian space with a metric of the Fisher Information Matrices[2]. Thus, we can view the learning by the multilayer Perceptrons as an optimization problem in a Riemannian space as presented in 2.1.

Let us start by defining the Fisher information matrix and the Natural Gradient Descent.

Definition: Fisher Information Matrix

We set $l(\mathbf{x}; \mathbf{w}) \equiv \log p(\mathbf{x}; \mathbf{w})$. For $\mathbf{w} \in \Theta$, a square matrix $G(\mathbf{w}) = (g_{ij}(\mathbf{w}))$ is defined as follows.

$$G(\mathbf{w}) \equiv E [\nabla l(X; \mathbf{w}) \nabla l(X; \mathbf{w})^T] \quad (8)$$

(8) can be expressed by each elements as,

$$g_{ij}(\mathbf{w}) = E \left[\frac{\partial l}{\partial w_i}(X; \mathbf{w}) \frac{\partial l}{\partial w_j}(X; \mathbf{w}) \right] = \int \frac{\partial l}{\partial w_i}(\mathbf{x}; \mathbf{w}) \frac{\partial l}{\partial w_j}(\mathbf{x}; \mathbf{w}) p(\mathbf{x}; \mathbf{w}) d\mathbf{x} \quad (9)$$

We call this matrix G the Fisher information matrix (FIM).

Definition: Natural Gradient Descent

We call the following gradient method as the Natural Gradient Descent (NGD).

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta(t) G^{-1}(\mathbf{w}(t)) \nabla M(\mathbf{w}(t)) \quad (10)$$

Here $\eta(t)$ is a rate of the learning.

Then, $-G^{-1}(\mathbf{w}(t)) \nabla M(\mathbf{w}(t))$ is the direction of the maximal decrease of the target function M given a fixed step size. We note that this NGD reduces to the ordinary gradient descent, when G is the identity matrix.

2.3 Approximation of the Fisher Information Matrix

As discussed in the previous section, the FIM and its inverse play important roles in the calculation in the NGD. We, thus, present a preliminary approximation of the FIM in order to lessen the computational burdens[7].

Let us first compute the FIM for the multilayer Perceptrons. The probability density function associated with the multilayer Perceptron function (MPF) is given as follows.

$$p(\mathbf{x}, \mathbf{y}; \mathbf{w}) = p(\mathbf{y} | f(\mathbf{x}; \mathbf{w})) p(\mathbf{x}) \quad (11)$$

Also, the gradient vector are written concisely as in (4),

$$\frac{\partial l}{\partial \mathbf{w}} \equiv \left(\text{vec} \left(\frac{\partial l}{\partial \bar{W}^{(1)}} \right)^T, \dots, \text{vec} \left(\frac{\partial l}{\partial \bar{W}^{(N)}} \right)^T \right)^T \quad (12)$$

Then, the FIM for the MLP is given as follows.

$$G(\mathbf{w}) = \begin{pmatrix} G_{1,1} & G_{1,2} & \cdots & G_{1,N} \\ G_{2,1} & G_{2,2} & \cdots & G_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ G_{N,1} & G_{N,2} & \cdots & G_{N,N} \end{pmatrix} \quad (13)$$

$$G_{i,j} \equiv E \left[\text{vec} \left(\frac{\partial l}{\partial \bar{W}^{(i)}} \right) \text{vec} \left(\frac{\partial l}{\partial \bar{W}^{(j)}} \right)^T \right] \quad (14)$$

Hence, the FIM for the MLP is composed of the block matrices $G_{i,j}$.

If we further set $\delta_j^{(i)} = \frac{\partial l}{\partial a_j^{(i)}}$, the following is obtained.

$$\frac{\partial l}{\partial \bar{W}^{(i)}} = \boldsymbol{\delta}^{(i)} \mathbf{z}^{(i-1)T} \quad (15)$$

By putting (15) into (14), the $G_{i,j}$ can now be expressed as

$$G_{i,j} = E \left[\bar{\mathbf{z}}^{(i-1)} \bar{\mathbf{z}}^{(j-1)T} \otimes \boldsymbol{\delta}^{(i)} \boldsymbol{\delta}^{(j)T} \right] \quad (16)$$

(Here, \otimes is the Kronecker product.)

For the efficient computation, it is essential to approximate this FIM. The preliminary approximation consists of two steps.

The first step approximation of $G_{i,j}$ is given as $\tilde{G}_{i,j}$ which is defined as follows.

$$\begin{aligned} G_{i,j} &\approx E \left[\bar{\mathbf{z}}^{(i-1)} \bar{\mathbf{z}}^{(j-1)T} \right] \otimes E \left[\boldsymbol{\delta}^{(i)} \boldsymbol{\delta}^{(j)T} \right] \\ &\equiv \bar{Z}_{i-1,j-1} \otimes D_{i,j} \\ &\equiv \tilde{G}_{i,j} \end{aligned} \quad (17)$$

This approximation means that we are inter-changing the expectation of the Kronecker products with the Kronecker products of the expectations. The matrix \tilde{G} , whose elements are given by replacing $G_{i,j}$ of (13) with $\tilde{G}_{i,j}$, is the first step approximation of the FIM. We note that the FIM is decomposed into two parts by this approximation: $\bar{\mathbf{z}}^{(i-1)}$ (the feed-forward phase part) and $\boldsymbol{\delta}^{(i)}$ (the back-propagating phase part).

We perform the second step approximation on \tilde{G} to obtain \check{G} .

$$\check{G} \equiv \text{diag}(\tilde{G}_{1,1}, \tilde{G}_{2,2}, \dots, \tilde{G}_{N,N}) \quad (18)$$

Here, $\text{diag}(\dots)$ denotes a block diagonal matrix, whose non-zero diagonals are given by the elements. In other words, \check{G} is obtained from \tilde{G} by setting non-diagonal elements as the zero matrix,

$$\tilde{G}_{i,j} = O \quad (i \neq j) \quad (19)$$

This approximation allows us to compute the FIM layer by layer independently.

3 Whitened Neural Networks

In this section, we present algorithms which aim to perform Natural Gradient Descent efficiently with the approximated FIM, \check{G} .

3.1 Natural Gradient Descent by Whitening

Let us first describe Whitened Neural Networks[4]. The main idea of this method is to perform the NGD by reconfiguring the network and parameters, so that the FIM becomes closer to the identity matrix. When the FIM is the identity matrix, the NGD is the same as the ordinary gradient descent, thus can be implemented simply with less computational costs.

3.1.1 Whitened Neural Network

The architecture of the Whitened Neural Networks (WNN) is obtained by changing (1) through (3) into the following form.

$$\mathbf{z}^{\dagger(i-1)} = U^{(i-1)}(\mathbf{z}^{(i-1)} - \mathbf{c}^{(i-1)}) \quad (20)$$

$$\mathbf{a}^{(i)} = W^{\dagger(i)} \mathbf{z}^{\dagger(i-1)} + \mathbf{b}^{\dagger(i)} \quad (21)$$

$$\mathbf{z}^{(i)} = \phi^{(i)}(\mathbf{a}^{(i)}) \quad (22)$$

Here $\{(U^{(i-1)}, \mathbf{c}^{(i-1)})\}$ are the new parameters introduced as “Whitening” parameters. $\{(W^{\dagger(i)}, \mathbf{b}^{\dagger(i)})\}$ are the new model parameters associated with this new architecture. These are the ones which we want to estimate and update using gradient descent methods as in the normal multilayer Perceptrons.

We present in the Figure 1 the new architecture defined by (20), (21), (22). It shows the $i-1$ th layer to the i th layer. We note the gray layer in the Figure 1 is the new inserted layer for the purpose of “whitening”. This change of network configuration is the essence of WNN.

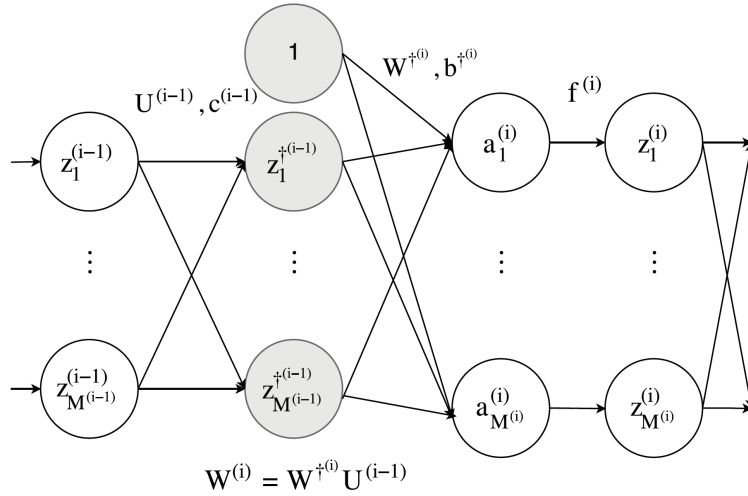


Figure 1: Architecture of Whitened Neural Networks

From (17), the approximated FIM $\tilde{G}_{i,i}$ in the WNN, then, is expressed as the following.

$$\tilde{G}_{i,i} = E \left[\tilde{\mathbf{z}}^{\dagger(i-1)} \tilde{\mathbf{z}}^{\dagger(i-1)T} \right] \otimes E \left[\boldsymbol{\delta}^{(i)} \boldsymbol{\delta}^{(i)T} \right] \quad (23)$$

The essential idea of the whitening is to make \tilde{G} closer to the identity by defining

the whitening parameters $\{(U^{(i-1)}, \mathbf{c}^{(i-1)})\}$ as

$$E \left[\bar{\mathbf{z}}^{\dagger(i-1)} \bar{\mathbf{z}}^{\dagger(i-1)T} \right] = I \quad (24)$$

for each i and performs the gradient descent. (Our idea, which will be described later in 3.2, is to further extend the whitening to the latter factor $E \left[\boldsymbol{\delta}^{(i)} \boldsymbol{\delta}^{(i)T} \right]$ in (23))

3.1.2 Updating of the Whitening Parameters

We calculate here explicitly $\{(U^{(i-1)}, \mathbf{c}^{(i-1)})\}$, which satisfies the condition (24).

As $\bar{\mathbf{z}}^{\dagger(i-1)} = (1, \mathbf{z}^{\dagger(i-1)T})^T$, (24) can be decomposed into

$$\begin{pmatrix} 1 & E \left[\mathbf{z}^{\dagger(i-1)T} \right] \\ E \left[\mathbf{z}^{\dagger(i-1)} \right] & E \left[\mathbf{z}^{\dagger(i-1)} \mathbf{z}^{\dagger(i-1)T} \right] \end{pmatrix} = I \quad (25)$$

Thus,

$$E \left[\mathbf{z}^{\dagger(i-1)} \right] = \mathbf{0} \quad (26)$$

$$E \left[\mathbf{z}^{\dagger(i-1)} \mathbf{z}^{\dagger(i-1)T} \right] = I \quad (27)$$

are required to satisfy this condition.

Let us look at these conditions. (26) can be satisfied by

$$\mathbf{c}^{(i-1)} \leftarrow E \left[\mathbf{z}^{(i-1)} \right] \quad (28)$$

Also, for (27), we first set the matrix $\check{Z}_{i-1, i-1}$ by the following

$$\check{Z}_{i-1, i-1} \equiv E \left[(\mathbf{z}^{(i-1)} - \mathbf{c}^{(i-1)})(\mathbf{z}^{(i-1)} - \mathbf{c}^{(i-1)})^T \right] \quad (29)$$

Then, (27) becomes

$$E \left[\mathbf{z}^{\dagger(i-1)} \mathbf{z}^{\dagger(i-1)T} \right] = U^{(i-1)} \check{Z}_{i-1, i-1} U^{(i-1)T} = I \quad (30)$$

Because $\check{Z}_{i-1, i-1}$ is a symmetric matrix, there exists a orthogonal matrix P , which makes it diagonal.

$$\check{Z}_{i-1, i-1} = P \Lambda P^T \quad (31)$$

Here Λ is the diagonalized matrix. Then, if we set

$$U^{(i-1)} \leftarrow (\Lambda + \varepsilon I)^{-\frac{1}{2}} \cdot P^T \quad (32)$$

the condition (30) is approximately satisfied. (Here, ε is a small positive constant to avoid division by zero.)

By this process, called the whitening process, according to (28) and (32), we update the whitening parameters satisfying (24). We note that, in this updating, the calculation of $\mathbf{z}^{(i-1)}$ in feed-forward phase is essential.

3.1.3 Updating of the model parameters

We now turn our attention to the updating of the model parameters $\{(W^{\dagger(i)}, \mathbf{b}^{\dagger(i)})\}$. We need to pay attention so that the inclusion of the whitening process and the associated layer does not change the value of the multilayer Perceptron function (MPF) itself. In concrete, we need to do the following. Let us assume the whitening parameters $\{(U^{(i-1)}, \mathbf{c}^{(i-1)})\}$ are updated to $\{(U_{new}^{(i-1)}, \mathbf{c}_{new}^{(i-1)})\}$. We want to keep the value of (21) unchanged by this updating. This places a constraint in the way we update the model parameters $\{(W_{new}^{\dagger(i)}, \mathbf{b}_{new}^{\dagger(i)})\}$. Namely, for any value of $\mathbf{z}^{(i-1)}$, the following must be satisfied.

$$W^{\dagger(i)} U^{(i-1)} (\mathbf{z}^{(i-1)} - \mathbf{c}^{(i-1)}) + \mathbf{b}^{\dagger(i)} = W_{new}^{\dagger(i)} U_{new}^{(i-1)} (\mathbf{z}^{(i-1)} - \mathbf{c}_{new}^{(i-1)}) + \mathbf{b}_{new}^{\dagger(i)} \quad (33)$$

We can obtain the following by solving these equations.

$$W_{new}^{\dagger(i)} = W^{\dagger(i)} U^{(i-1)} U_{new}^{(i-1)-1} \quad (34)$$

$$\mathbf{b}_{new}^{\dagger(i)} = \mathbf{b}^{\dagger(i)} - W^{\dagger(i)} U^{(i-1)} \mathbf{c}^{(i-1)} + W_{new}^{\dagger(i)} U_{new}^{(i-1)} \mathbf{c}_{new}^{(i-1)} \quad (35)$$

By putting together (20) and (21), we can set $\{(W^{(i)}, \mathbf{b}^{(i)})\}$ as

$$W^{(i)} = W^{\dagger(i)} U^{(i-1)} \quad (36)$$

$$\mathbf{b}^{(i)} = \mathbf{b}^{\dagger(i)} - W^{\dagger(i)} U^{(i-1)} \mathbf{c}^{(i-1)} \quad (37)$$

Using these $\{(W^{(i)}, \mathbf{b}^{(i)})\}$, we can re-write (34) and (35) as

$$W^{\dagger(i)} \leftarrow W^{(i)} U_{new}^{(i-1)-1} \quad (38)$$

$$\mathbf{b}^{\dagger(i)} \leftarrow \mathbf{b}^{(i)} + W^{(i)} \mathbf{c}_{new}^{(i-1)} \quad (39)$$

Thus, we can keep MPF the same by updating whitening parameters first as in (28) and (32) and then update model parameters with (38) and (39).

As we change model parameters, the values of $E[\mathbf{z}^{(i-1)}]$, $\check{Z}_{i-1, i-1}$ changes, which in turn requires the update of the whitening parameters to keep the FIM close to the identity matrix. However, it is computationally expensive to update both set of parameters at every iterations. In particular, the update of the whitening parameters for a layer of M neurons takes computation of the order of $O(M^3)$. Thus, in actual implementations, the update of the whitening parameters are performed at certain fixed time intervals[4], though this makes a gradual digression from the NGD for that time interval between the successive updating of the whitening parameters.

The method and algorithm described above is called ‘‘Projected Natural Gradient Descent’’(PRONG)[4], which is outlined in Algorithm 1.

Algorithm 1 Projected Natural Gradient Descent (PRONG)[4].

Input: training set D , initial parameter $\mathbf{w}(0)$
Hyper parameters: updating period of whitening parameters τ
• $U^{(i)} \leftarrow I; \mathbf{c}^{(i)} \leftarrow \mathbf{0}; t \leftarrow 0$
while ending condition not satisfied **do**
 if $\text{mod}(t, \tau) = 0$ **then**
 for all layers i **do**
 • Computation of standard parameters $\{(W^{(i)}, \mathbf{b}^{(i)})\}$.
 • Estimations of $E[\mathbf{z}^{(i-1)}], \tilde{Z}_{i-1, i-1}$.
 • Updating of the Whitening parameters $\{(U^{(i-1)}, \mathbf{c}^{(i-1)})\}$.
 • Updating of the model parameters $\{(W^{\dagger(i)}, \mathbf{b}^{\dagger(i)})\}$.
 end for
 end if
 • Updating of $\{(W^{\dagger(i)}, \mathbf{b}^{\dagger(i)})\}$ by the ordinary gradient descent.
 • $t \leftarrow t + 1$
end while

3.2 Extension of Whitening

Here, we describe our proposal of the new extended whitening algorithms based on 3.1.

In the whitening method described above, in order to keep the approximated FIM, \tilde{G} , closer to the identity matrix, updating of the whitening parameters $\{(U^{(i)}, \mathbf{c}^{(i)})\}$ are performed. This makes the first factor $E[\bar{\mathbf{z}}^{\dagger(i-1)} \bar{\mathbf{z}}^{\dagger(i-1)T}]$ in

$$\tilde{G}_{i,i} = E[\bar{\mathbf{z}}^{\dagger(i-1)} \bar{\mathbf{z}}^{\dagger(i-1)T}] \otimes E[\boldsymbol{\delta}^{(i)} \boldsymbol{\delta}^{(i)T}] \quad (40)$$

closer to the identity matrix.

The main idea of our method is to make the second factor $E[\boldsymbol{\delta}^{(i)} \boldsymbol{\delta}^{(i)T}]$ toward the identity as well, so that $\tilde{G}_{i,i}$ is even better approximated by the identity matrix. This turns out that we implement whitening process not only in the feed-forward phase but also in the back-propagating phase.

3.2.1 Bidirectional Whitened Neural Networks

In order to perform the back-whitening, we modify the forward-whitening process described by (20), (21) and (22) into the following.

$$\mathbf{z}^{\dagger(i-1)} = U^{(i-1)}(\mathbf{z}^{(i-1)} - \mathbf{c}^{(i-1)}) \quad (41)$$

$$\mathbf{a}^{\dagger(i)} = W^{\dagger(i)} \mathbf{z}^{\dagger(i-1)} + \mathbf{b}^{\dagger(i)} \quad (42)$$

$$\mathbf{a}^{(i)} = R^{(i)T} \mathbf{a}^{\dagger(i)} \quad (43)$$

$$\mathbf{z}^{(i)} = \phi^{(i)}(\mathbf{a}^{(i)}) \quad (44)$$

Here, $\{R^{(i)T}\}$ is a newly introduced parameter, called the back-whitening parameter.

We show, as in Figure 1, the architecture of this extended method defined by (41), (42), (43), (44) in the Figure 2. The dark gray part in the Figure 2 is the newly introduced layer to accommodate the back-whitening parameter $\{R^{(i)T}\}$.

As mentioned above, this proposed method performs whitening process both in feed-forward and back-propagating phase. Thus, we call this new architecture as the Bidirectional Whitened Neural Networks (BWNN).

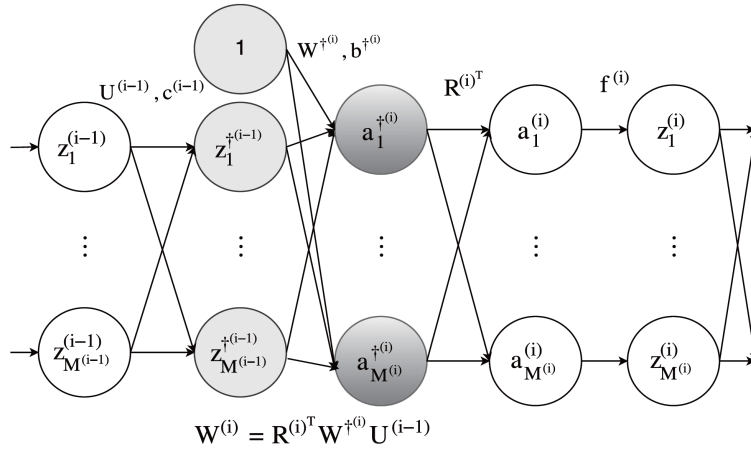


Figure 2: Architecture of the Bidirectional Whitened Neural Networks

We introduce a new parameter $\delta^{\dagger(i)}$ in place of $\delta^{(i)}$ as in the following.

$$\delta^{\dagger(i)} \equiv \frac{\partial l}{\partial a^{\dagger(i)}} \quad (45)$$

Then, the approximation of $\tilde{G}_{i,i}$ is then expressed as

$$\tilde{G}_{i,i} = E \left[\tilde{z}^{\dagger(i-1)} \tilde{z}^{\dagger(i-1)T} \right] \otimes E \left[\delta^{\dagger(i)} \delta^{\dagger(i)T} \right] \quad (46)$$

In analogy with Section 3.1, we will fix the back-whitening parameter $\{R^{(i)T}\}$ so that

$$E \left[\delta^{\dagger(i)} \delta^{\dagger(i)T} \right] = I \quad (47)$$

3.2.2 Updating of the back-whitening parameter

Let us explicitly find $\{R^{(i)T}\}$ to satisfy (47). From (45), we have

$$\delta_j^{\dagger(i)} = \sum_k \frac{\partial l}{\partial a_k^{(i)}} \frac{\partial a_k^{(i)}}{\partial a_j^{\dagger(i)}} = \sum_k \delta_k^{(i)} r_{kj}^{(i)T} \quad (48)$$

Thus, $\delta^{\dagger(i)}$ is a linear transformation of $\delta^{(i)}$, which can be written as

$$\delta^{\dagger(i)} = R^{(i)} \delta^{(i)} \quad (49)$$

By inserting (49) into (47), we obtain

$$E \left[\delta^{\dagger(i)} \delta^{\dagger(i)T} \right] = R^{(i)} D_{i,i} R^{(i)T} = I \quad (50)$$

Hence, in analogy with (32), $R^{(i)}$ which satisfies (47) is given by the following

$$R^{(i)} \leftarrow (\Lambda + \varepsilon I)^{-\frac{1}{2}} \cdot P^T \quad (51)$$

Here, Λ, P are the diagonalized and the orthogonal matrices associated with $D_{i,i}$, and ε is the small positive parameter to avoid a division by zero.

Altogether, as in the case of the forward-whitening parameters, (47) is satisfied by updating of the back-whitening parameters according to (51), which, in turn, depends on the calculation of $\delta^{(i)}$ in the back-propagating phase.

3.2.3 Updating of the model parameters

As in the feed-forward phase, we update the model parameters $\{(W^{\dagger(i)}, \mathbf{b}^{\dagger(i)})\}$ so that the values of the multilayer Perceptron function are kept the same when the back-whitening parameters are updated.

In order to achieve this, the model parameters $\{(W^{\dagger(i)}, \mathbf{b}^{\dagger(i)})\}$ need to be updated as follows, given the back-whitening parameters are updated from $R^{(i)}$ to $R_{new}^{(i)}$.

$$W^{\dagger(i)} \leftarrow (R_{new}^{(i)})^{-1} R^{(i)T} W^{\dagger(i)} \quad (52)$$

$$\mathbf{b}^{\dagger(i)} \leftarrow (R_{new}^{(i)})^{-1} R^{(i)T} \mathbf{b}^{\dagger(i)} \quad (53)$$

We will call the above algorithm as ‘‘Bidirectional Projected Natural Gradient Descent’’(BPRONG) because it performs whitening both in feed-forward and back-propagating phase. Its outline is shown in Algorithm 2. Also, as in the forward-whitening, we can perform the back-whitening update in a fixed intervals. They can both be done at the same time, or independently. In the following section, we will employ the latter method for a numerical application.

Algorithm 2 Bidirectional Projected Natural Gradient Descent(BPRONG).

Input: training set D , initial parameter $\mathbf{w}(0)$
Hyper parameters: parameters for forward-whitening τ_1, c_1 , parameters for back-whitening τ_2, c_2
• $U^{(i)} \leftarrow I; \mathbf{c}^{(i)} \leftarrow \mathbf{0}; R^{(i)T} \leftarrow I; t \leftarrow 0$
while ending condition not satisfied **do**
 if $\text{mod}(t, \tau_1) = c_1$ **then**
 • forward-whitening (cf. Algorithm 1).
 end if
 if $\text{mod}(t, \tau_2) = c_2$ **then**
 for all layers i **do**
 • Estimation of $D_{i,i}$.
 • Computation of the back-whitening parameters $\{R_{new}^{(i)T}\}$.
 • Updating the model parameters $\{(W^{\dagger(i)}, \mathbf{b}^{\dagger(i)})\}$.
 • Updating the back-whitening parameters $\{R^{(i)T}\}$.
 end for
 end if
 • Updating of $\{(W^{\dagger(i)}, \mathbf{b}^{\dagger(i)})\}$ by the ordinary gradient descent.
 • $t \leftarrow t + 1$
end while

4 Numerical Experiment

In order to see the efficacy of our proposed method BPRONG in 3.2, we have applied it to a problem of hand-written character (digits) recognition using the MNIST data set (<http://yann.lecun.com/exdb/mnist/>) and compared against three other methods: ordinary Stochastic Gradient Descent(SGD), Batch Normalization(BN)[5], and PRONG. The network architecture is common to all the compared methods with 5 layers of 784-100-100-100-10 neurons from input to output. Also, common learning rate of 0.01 is taken and the mini-batch size is 100. The training data contains 60000 sets and the test data has 10000. We call updates of 600 as 1 epoch, and plot, at each epoch, the training loss with the training set, and the validation loss with the test data sets.

We observe the advantage of BPRONG with respect to the iteration numbers both in the training and the validation losses as shown in Figures 3 and 4. With respect to the actual computation times, BPRONG is faster than PRONG, and about the same speed as the BN (Figures 5 and 6). This is due to the fact that eigenvalue decomposition associated with the whitening is computationally costly to offset the advantage over BN with respect to iteration numbers.

Altogether, our proposed method, BPRONG, has shown its potential. If we can find methods to speed up the whitening process, BPRONG can show its effectiveness further.

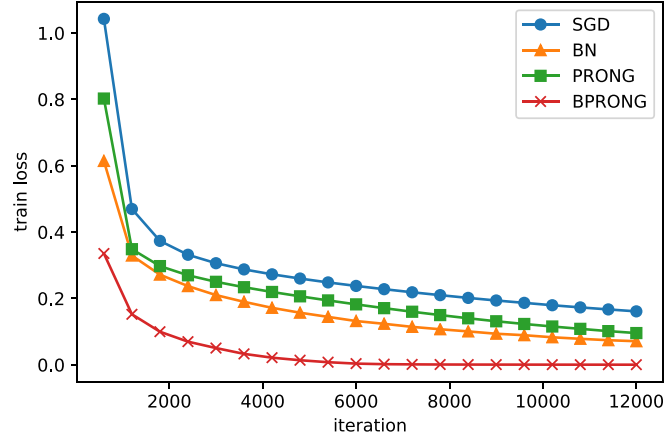


Figure 3: Training loss as a function of the iteration numbers

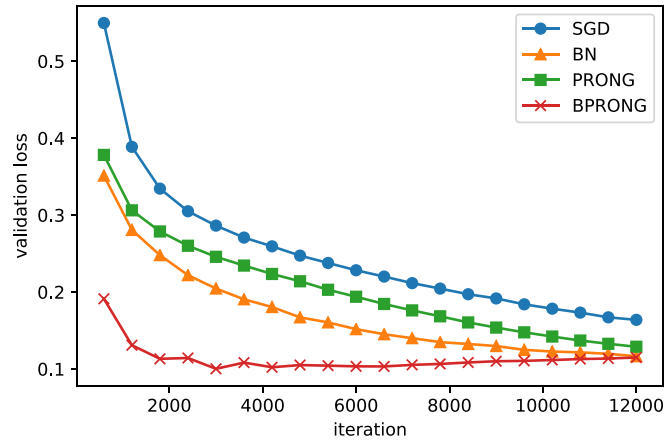


Figure 4: Validation loss as a function of the iteration numbers

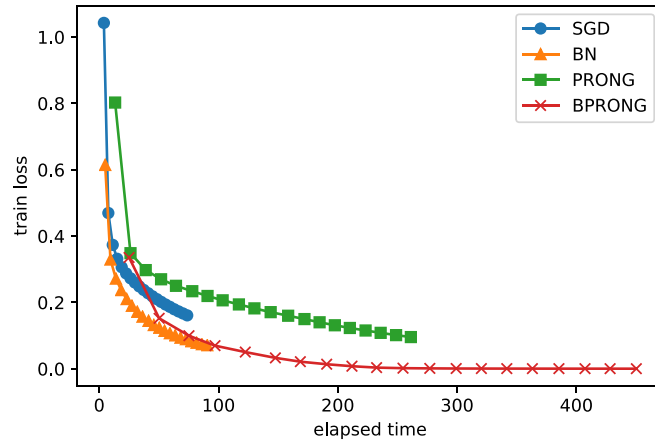


Figure 5: Training loss as a function of the computational time

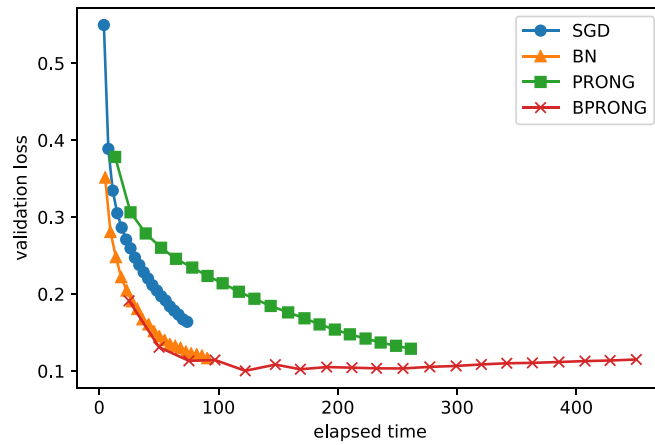


Figure 6: Validation loss as a function of the computational time

5 Discussion

We presented here an extended model of the previously proposed Whitened Neural Networks[4] as a method to realize the Natural Gradient Descent. Our extension, which we call Bidirectional Whitened Neural Networks, aims to make the Fisher Information Matrix closer to the identity matrix. It has shown its potential as an efficient method thorough a numerical application to a hand-written digits recognition problem.

We note two points as topics to be investigated further. First, the proposed model should be tested for larger and deeper network architectures for a check of its efficacy and stability. It may require further modifications for improvements on these aspects, particularly by exploring matrix decomposition methods. Secondly, we want to find more dynamical way for whitening process. In other words, we would like to keep the Fisher Information Matrix constantly closer to the identity by continuous whitenings. Though it is computationally more expensive, we may build on some previous studies, such as adaptive calculations of transforming matrices[3].

References

- [1] Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.
- [2] Shun-ichi Amari and Hiroshi Nagaoka. *Methods of Information Geometry (Translations of Mathematical Monographs)*. American Mathematical Society, 2007.
- [3] J-F Cardoso and Beate H Laheld. Equivariant adaptive source separation. *IEEE Transactions on signal processing*, 44(12):3017–3030, 1996.
- [4] Guillaume Desjardins, Karen Simonyan, Razvan Pascanu, and Koray Kavukcuoglu. Natural neural networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2071–2079. Curran Associates, Inc., 2015.
- [5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 448–456, 2015.
- [6] James Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.
- [7] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. *arXiv preprint arXiv:1503.05671*, 2015.

- [8] Hyeyoung Park, Shun-ichi Amari, and Kenji Fukumizu. Adaptive natural gradient learning algorithms for various stochastic models. *Neural Networks*, 13(7):755–764, 2000.
- [9] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.
- [10] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 901–909. Curran Associates, Inc., 2016.