
Hessian-Free Optimization For Learning Deep Multidimensional Recurrent Neural Networks

Minhyung Cho Chandra Shekhar Dhir Jaehyung Lee

Applied Research Korea, Gracenote Inc.

{mhyung.cho, shekhardhir}@gmail.com jaehyung.lee@kaist.ac.kr

Abstract

Multidimensional recurrent neural network (MDRNN) has shown a remarkable performance in speech and handwriting recognition. The performance of MDRNN is improved by further increasing its depth, and the difficulty of learning the deeper network is overcome by Hessian-free (HF) optimization. Considering that connectionist temporal classification (CTC) is utilized as an objective of learning MDRNN for sequence labelling, the non-convexity of CTC poses a problem to apply HF to the network. As a solution to this, a convex approximation of CTC is formulated and its relationship with the EM algorithm and the Fisher information matrix is discussed. MDRNN up to the depth of 15 layers is successfully trained using HF, resulting in improved performance for sequence labelling.

1 Introduction

Multidimensional recurrent neural network (MDRNN) is an efficient architecture to build multidimensional context into recurrent neural networks [1]. End-to-end training of MDRNN in conjunction with connectionist temporal classification (CTC) has shown the state-of-the-art performance in on/off-line handwriting recognition [2, 3] and speech recognition [4].

In previous approaches, the performance of MDRNN has been demonstrated with the networks having up to depth of 5 layers, which are relatively limited compared to the recent progress on feed-forward networks [5]. The effectiveness of deeper MDRNNs beyond 5 layers has been unknown.

Training a deeper architecture has always been a challenging topic in machine learning. Notable breakthrough was made where deep feedforward neural networks were initialized using layer-wise pre-training [6]. Recently, there has been approaches to add supervision to intermediate layers to train deep networks [5, 7]. To our knowledge, no such pre-training or bootstrapping method has been developed for MDRNN which potentially utilizes LSTM cells [8] as its hidden unit.

Alternatively, HF optimization is an appealing approach to train deep neural networks due to its ability to overcome pathological curvature of the objective function [9]. Furthermore it can be applied to any connectionist model as long as its objective function is differentiable. The recent success of HF to deep feedforward and recurrent neural networks [9, 10] encourages the use of HF to MDRNN.

In this paper, we claim that MDRNN can benefit from deeper architecture, and applying second order optimization like HF enables its successful learning. First we offer details to develop HF optimization for MDRNN. Then, to apply HF optimization for sequence labelling tasks, we address the problem of non-convexity of CTC, and formulate a convex approximation. Also, its relationship with the EM algorithm and the Fisher information matrix is discussed. Experimental results of offline handwriting recognition and phoneme recognition show that MDRNN with HF performs better as the depth of the network increases up to fifteen.

2 Multidimensional recurrent neural networks

MDRNN is a generalization of RNN to process multidimensional data by replacing the single recurrent connection with as many connections as dimensions of the data [1]. The network can access the contextual information from 2^N directions, allowing to make a collective decision based on rich context information. To enhance its ability of exploiting context information, long short-term memory (LSTM) [8] cells are usually utilized as hidden units. In addition, stacking MDRNNs to construct deeper networks further improves the performance as the depth increases, reporting the state-of-the-art performance in phoneme recognition [4]. For sequence labelling, CTC is applied as a loss function of MDRNN. The important advantage of using CTC is that any pre-segmented sequences are not required, and the entire transcription of the input sample is sufficient.

2.1 Learning MDRNN

A d -dimensional MDRNN with M inputs and K outputs is regarded as a mapping from an input sequence $\mathbf{x} \in \mathbb{R}^{M \times T_1 \times \dots \times T_d}$ to an output sequence $\mathbf{a} \in (\mathbb{R}^K)^T$ of length T , where input data for M input neurons is given by vectorization of a d -dimensional data and T_1, \dots, T_d is the length of the sequence in each dimension. All learnable weights and biases are concatenated to obtain a parameter vector $\theta \in \mathbb{R}^N$. In the learning phase with fixed training data, MDRNN is formalized as a mapping $\mathcal{N} : \mathbb{R}^N \rightarrow (\mathbb{R}^K)^T$ from the parameters θ to the output sequence \mathbf{a} , i.e. $\mathbf{a} = \mathcal{N}(\theta)$. The scalar loss function is defined over the output sequence as $\mathcal{L} : (\mathbb{R}^K)^T \rightarrow \mathbb{R}$. Learning MDRNN is viewed as an optimization of the objective function $\mathcal{L}(\mathcal{N}(\theta)) = \mathcal{L} \circ \mathcal{N}(\theta)$ with respect to θ .

2.2 Notation

The Jacobian $J_{\mathcal{F}}$ of a function $\mathcal{F} : \mathbb{R}^m \rightarrow \mathbb{R}^n$ is the $n \times m$ matrix where each element is a partial derivative of an element of output with respect to an element of input. The Hessian $H_{\mathcal{F}}$ of a scalar function $\mathcal{F} : \mathbb{R}^m \rightarrow \mathbb{R}$ is the $m \times m$ matrix of second-order partial derivatives of the output with respect to its inputs. Throughout the paper, a symbol $^\top$ is used for denoting the transpose of a vector or matrix. For variables, a sequence of vector is denoted by boldface \mathbf{a} , a vector at time t in \mathbf{a} is denoted by a^t , and the k -th element of a^t is denoted by a_k^t .

3 Hessian-free optimization for MDRNN

In this section, we discuss two main points to develop HF optimization for MDRNN. One is obtaining a local quadratic approximation for MDRNN, and the other is an efficient calculation of the matrix-vector product used at each iteration of the conjugate gradient (CG) method.

HF minimizes an objective by constructing a local quadratic approximation to the objective function and minimizing the approximate function instead of the original one. The loss function $\mathcal{L}(\theta)$ needs to be approximated at each point θ_n of the n -th iteration as follows:

$$Q_n(\theta) = \mathcal{L}(\theta_n) + \nabla_{\theta} \mathcal{L}|_{\theta_n}^\top \delta_n + \frac{1}{2} \delta_n^\top G \delta_n, \quad (1)$$

where $\delta_n = \theta - \theta_n$ is the search direction, i.e. parameters of the optimization, and G is a local approximation to the curvature of $\mathcal{L}(\theta)$ at θ_n , which is typically obtained by the generalized Gauss-Newton (GGN) matrix as an approximation of the Hessian.

HF uses the CG method in a subroutine to minimize the quadratic objective above for utilizing the complete curvature information and achieving computational efficiency. CG requires the computation of Gv for an arbitrary vector v , but not the explicit evaluation of G . For neural networks, an efficient way to compute Gv was proposed by [11], extending the work of [12]. In section 3.2, we provide the details for the efficient computation of Gv for MDRNN.

3.1 Quadratic approximation of loss function

The Hessian matrix, $H_{\mathcal{L} \circ \mathcal{N}}$, of the objective $\mathcal{L}(\mathcal{N}(\theta))$ is written as

$$H_{\mathcal{L} \circ \mathcal{N}} = J_{\mathcal{N}}^\top H_{\mathcal{L}} J_{\mathcal{N}} + \sum_{i=1}^{KT} [J_{\mathcal{L}}]_i H_{[\mathcal{N}]_i}, \quad (2)$$

where $J_{\mathcal{N}} \in \mathbb{R}^{KT \times N}$, $H_{\mathcal{L}} \in \mathbb{R}^{KT \times KT}$, and $[q]_i$ denotes the i -th component of the vector q . An indefinite Hessian matrix is problematic for 2nd-order optimization because it defines an unbounded local quadratic approximation [13]. For nonlinear systems, the Hessian is not necessarily positive semidefinite, thus the GGN matrix is used as an approximation of the Hessian [11, 9]. The GGN matrix is obtained by ignoring the second term in Eq. (2), as given by

$$G_{\mathcal{L} \circ \mathcal{N}} = J_{\mathcal{N}}^{\top} H_{\mathcal{L}} J_{\mathcal{N}}. \quad (3)$$

The sufficient condition for the GGN approximation to be exact is that the network makes a perfect prediction for every given sample, that is, $J_{\mathcal{L}} = 0$, or $[\mathcal{N}]_i$ stays in the linear region for all i , that is, $H_{[\mathcal{N}]_i} = 0$.

$G_{\mathcal{L} \circ \mathcal{N}}$ has less rank than KT and is positive semidefinite as long as $H_{\mathcal{L}}$ is. Thus, \mathcal{L} is chosen to be a convex function so that $H_{\mathcal{L}}$ is positive semidefinite. In principle, it is best to define \mathcal{L} and \mathcal{N} in a way that \mathcal{L} performs as much of the computation as possible, with the positive semidefiniteness of $H_{\mathcal{L}}$ as a minimum requirement [13]. In practice, a nonlinear output layer along with its matching loss function [11], such as the softmax function with cross-entropy loss, is widely used.

Considering that MDRNN is normally applied to model sequential data such as speech or handwriting, complex loss functions need to be adopted, like the one provided by CTC. A detailed discussion of approximating the Hessian for CTC is provided in section 4.

3.2 Computation of matrix-vector product for MDRNN

The product of an arbitrary vector v by the GGN matrix, $Gv = J_{\mathcal{N}}^{\top} H_{\mathcal{L}} J_{\mathcal{N}} v$, amounts to the sequential multiplication of v by three matrices. First, the product $J_{\mathcal{N}} v$ is a Jacobian times vector and is therefore equal to the directional derivative of $\mathcal{N}(\theta)$ along the direction of v . Thus, $J_{\mathcal{N}} v$ can be written using a differential operator $J_{\mathcal{N}} v = \mathcal{R}_v(\mathcal{N}(\theta))$ [12], and the properties of the operator can be utilized for efficient computation. Because MDRNN is a composition of differentiable components, the computation of $\mathcal{R}_v(\mathcal{N}(\theta))$ throughout the whole network can be accomplished by repeatedly applying the sum, product, and chain rules starting from the input layer. The detailed derivation of \mathcal{R} operator to LSTM, normally used as a hidden unit in MDRNN, is provided in appendix A.

Next, the multiplication of $J_{\mathcal{N}} v$ by $H_{\mathcal{L}}$ can be done by direct computation. At first sight, the dimension of $H_{\mathcal{L}}$ could be seen problematic since the dimension of the output vector used by the loss function \mathcal{L} can be as high as KT , especially if CTC is adopted as an objective for MDRNN. If the loss function can be expressed as the sum of individual loss functions with restricted domain in time, the computation can be reduced significantly. For example, with the commonly used cross-entropy loss function, $KT \times KT$ matrix $H_{\mathcal{L}}$ can be turned into a block diagonal matrix with T blocks of $K \times K$ Hessian matrix. Let $H_{\mathcal{L},t}$ be the t -th block in $H_{\mathcal{L}}$. Then, the GGN matrix can be written as

$$G_{\mathcal{L} \circ \mathcal{N}} = \sum_t J_{\mathcal{N}_t}^{\top} H_{\mathcal{L},t} J_{\mathcal{N}_t}, \quad (4)$$

where $J_{\mathcal{N}_t}$ is the Jacobian of the network at time t .

Finally, the multiplication of a vector $u = H_{\mathcal{L}} J_{\mathcal{N}} v$ by the matrix $J_{\mathcal{N}}^{\top}$ is calculated using the back-propagation through time algorithm by propagating u instead of the error at the output layer.

4 Convex approximation of CTC for application to HF optimization

Connectionist temporal classification (CTC) [14] provides an objective function of learning MDRNN for sequence labelling. In this section, we derive a convex approximation of CTC inspired by the GGN approximation according to the following steps. First, the non-convex part from the original objective is separated out by reformulating the softmax part. Next, the remaining convex part is approximated without altering its Hessian, making it well matched to the non-convex part. Finally, the convex approximation is obtained by reuniting the convex and non-convex part.

4.1 Connectionist temporal classification

CTC is formulated as the mapping from an output sequence of the recurrent network, $\mathbf{a} \in (\mathbb{R}^K)^T$, to a scalar loss. The output activations at time t are normalized using the softmax function

$$y_k^t = \frac{\exp(a_k^t)}{\sum_{k'} \exp(a_{k'}^t)}, \quad (5)$$

where y_k^t is the probability of label k given \mathbf{a} at time t .

The conditional probability of the path π is calculated by the multiplication of the label probabilities at each timestep, as given by

$$p(\pi|\mathbf{a}) = \prod_{t=1}^T y_{\pi_t}^t, \quad (6)$$

where π_t is the label observed at time t along the path π . The path π of length T is mapped to a label sequence of length $M \leq T$ by an operator \mathcal{B} which removes the repeated labels and then the blanks. Several mutually exclusive paths can map to the same label sequence. Let S be a set containing every possible sequence mapped by \mathcal{B} , that is, $S = \{s | s \in \mathcal{B}(\pi) \text{ for some } \pi\}$, and let $|S|$ denote the cardinality of the set.

The conditional probability of a label sequence \mathbf{l} is given by

$$p(\mathbf{l}|\mathbf{a}) = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{l})} p(\pi|\mathbf{a}), \quad (7)$$

which is the sum of probabilities of all the paths mapped to a label sequence \mathbf{l} by \mathcal{B} .

The cross-entropy loss assigns a negative log probability to the correct answer. Given a target sequence \mathbf{z} , the loss function of CTC for the sample is written as

$$\mathcal{L}(\mathbf{a}) = -\log p(\mathbf{z}|\mathbf{a}). \quad (8)$$

From the description above, CTC is composed of the sum of the product of softmax components. The function $-\log(y_k^t)$, corresponding to the softmax with cross-entropy loss, is convex [11]. Therefore, y_k^t is log-concave. Whereas log-concavity is closed under multiplication, the sum of log-concave functions is not log-concave in general [15]. As a result, the CTC objective is not convex in general because it contains the sum of softmax components in Eq. (7).

4.2 Reformulation of CTC objective function

We reformulate the CTC objective Eq. (8) to separate terms which are responsible for the non-convexity of the function. By reformulation, the softmax function is defined over the categorical label sequences.

By substituting Eq. (5) into Eq. (6), it follows that

$$p(\pi|\mathbf{a}) = \frac{\exp(b_\pi)}{\sum_{\pi' \in \text{all}} \exp(b_{\pi'})}, \quad (9)$$

where $b_\pi = \sum_t a_{\pi_t}^t$. By substituting Eq. (9) into Eq. (7) and setting $\mathbf{l} = \mathbf{z}$, $p(\mathbf{z}|\mathbf{a})$ can be re-written as

$$p(\mathbf{z}|\mathbf{a}) = \frac{\sum_{\pi \in \mathcal{B}^{-1}(\mathbf{z})} \exp(b_\pi)}{\sum_{\pi \in \text{all}} \exp(b_\pi)} = \frac{\exp(f_{\mathbf{z}})}{\sum_{\mathbf{z}' \in S} \exp(f_{\mathbf{z}'}), \quad (10)$$

where S is the set of every possible label sequence and $f_{\mathbf{z}} = \log \left(\sum_{\pi \in \mathcal{B}^{-1}(\mathbf{z})} \exp(b_\pi) \right)$ is the *log-sum-exp* function¹, which is proportional to the probability of observing the label sequence \mathbf{z} among all the other label sequences.

With the reformulation above, the CTC objective can be regarded as the cross-entropy loss with the softmax output which is defined over all the possible label sequences. Because the cross-entropy

¹ $f(x_1, \dots, x_n) = \log(e^{x_1} + \dots + e^{x_n})$ is the *log-sum-exp* function defined on \mathbb{R}^n

loss function matches the softmax output layer [11], the CTC objective is convex except the part which computes $f_{\mathbf{z}}$ for each of the label sequences. At this point, an obvious candidate for the convex approximation of CTC is the GGN matrix separating the convex part and non-convex part.

Let the non-convex part be \mathcal{N}_c and the convex part be \mathcal{L}_c . The mapping $\mathcal{N}_c : (\mathbb{R}^K)^T \rightarrow \mathbb{R}^{|S|}$ is defined by

$$\mathcal{N}_c(\mathbf{a}) = F = [f_{\mathbf{z}_1}, \dots, f_{\mathbf{z}_{|S|}}]^\top, \quad (11)$$

where $f_{\mathbf{z}}$ is given above, and $|S|$ is the number of all the possible label sequences. For given F as above, the mapping $\mathcal{L}_c : \mathbb{R}^{|S|} \rightarrow \mathbb{R}$ is defined by

$$\mathcal{L}_c(F) = -\log \frac{\exp(f_{\mathbf{z}})}{\sum_{\mathbf{z}' \in S} \exp(f_{\mathbf{z}'})} = -f_{\mathbf{z}} + \log \left(\sum_{\mathbf{z}' \in S} \exp(f_{\mathbf{z}'}) \right), \quad (12)$$

where \mathbf{z} is the label sequence corresponding to \mathbf{a} .

The final reformulation for the loss function of CTC is given by

$$\mathcal{L}(\mathbf{a}) = \mathcal{L}_c \circ \mathcal{N}_c(\mathbf{a}). \quad (13)$$

4.3 Convex approximation of CTC loss function

The GGN approximation of Eq. (13) immediately gives a convex approximation of the Hessian for CTC as $G_{\mathcal{L}_c \circ \mathcal{N}_c} = J_{\mathcal{N}_c}^\top H_{\mathcal{L}_c} J_{\mathcal{N}_c}$. Although $H_{\mathcal{L}_c}$ has the form of a diagonal matrix plus a rank-1 matrix, i.e. $\text{diag}(Y) - YY^\top$, the dimension of $H_{\mathcal{L}_c}$ is $|S| \times |S|$ where $|S|$ becomes exponentially large as the length of the sequence increases. It makes the practical calculation of $H_{\mathcal{L}_c}$ difficult.

On the other hand, removing the linear term $-f_{\mathbf{z}}$ from $\mathcal{L}_c(F)$ in Eq. (12) does not alter its Hessian. The resulting formula is $\mathcal{L}_p(F) = \log \left(\sum_{\mathbf{z}' \in S} \exp(f_{\mathbf{z}'}) \right)$. The GGN matrices of $\mathcal{L} = \mathcal{L}_c \circ \mathcal{N}_c$ and $\mathcal{M} = \mathcal{L}_p \circ \mathcal{N}_c$ are exactly the same, i.e. $G_{\mathcal{L}_c \circ \mathcal{N}_c} = G_{\mathcal{L}_p \circ \mathcal{N}_c}$. Therefore the Hessian matrices of them are approximations of each other. The condition that the two Hessian matrices, $H_{\mathcal{L}}$ and $H_{\mathcal{M}}$, converges to the same matrix is discussed later.

Interestingly, \mathcal{M} is given as a compact formula $\mathcal{M}(\mathbf{a}) = \mathcal{L}_p \circ \mathcal{N}_c(\mathbf{a}) = \sum_t \log \sum_k \exp(a_k^t)$, where a_k^t is the output unit k at time t . Its Hessian $H_{\mathcal{M}}$ can be directly computed, resulting in a block diagonal matrix. Each block is restricted in time, and the t -th block is given by

$$H_{\mathcal{M},t} = \text{diag}(Y^t) - Y^t Y^{t\top}, \quad (14)$$

where $Y^t = [y_1^t, \dots, y_K^t]^\top$ and y_k^t is given in Eq. (5). Because the Hessian of each block is positive semidefinite, $H_{\mathcal{M}}$ is positive semidefinite. A convex approximation of the Hessian of MDRNN using the CTC objective can be obtained by substituting $H_{\mathcal{M}}$ for $H_{\mathcal{L}}$ in Eq. (3). Note that the resulting matrix is block diagonal, and Eq. (4) can be utilized for efficient computation.

Summary of our derivation is as follows:

1. $H_{\mathcal{L}} = H_{\mathcal{L}_c \circ \mathcal{N}_c}$ is not positive semidefinite.
2. $G_{\mathcal{L}_c \circ \mathcal{N}_c} = G_{\mathcal{L}_p \circ \mathcal{N}_c}$ is positive semidefinite but is not computationally tractable.
3. $H_{\mathcal{L}_p \circ \mathcal{N}_c}$ is positive semidefinite and computationally tractable.

4.4 Sufficient condition for the proposed approximation to be exact

From Eq. (2), the condition $H_{\mathcal{L}_c \circ \mathcal{N}_c} = H_{\mathcal{L}_p \circ \mathcal{N}_c}$ holds if and only if $\sum_{i=1}^{KT} [J_{\mathcal{L}_c}]_i H_{[\mathcal{N}_c]_i} = \sum_{i=1}^{KT} [J_{\mathcal{L}_p}]_i H_{[\mathcal{N}_c]_i}$. Since $J_{\mathcal{L}_c} \neq J_{\mathcal{L}_p}$ in general, we consider only the case of $H_{[\mathcal{N}_c]_i} = 0$ for all i , which corresponds to the case that \mathcal{N}_c is a linear mapping.

$[\mathcal{N}_c]_i$ contains a *log-sum-exp* function mapping from paths to a label sequence. Let \mathbf{z} be the label sequence corresponding to $[\mathcal{N}_c]_i$, then $[\mathcal{N}_c]_i = f_{\mathbf{z}}(\dots, b_\pi, \dots)$ for $\pi \in \mathcal{B}^{-1}(\mathbf{z})$. If the probability of one path π' is large enough to ignore all the other paths, that is, $\exp(b_{\pi'}) \gg \exp(b_\pi)$ for $\pi \in \{\mathcal{B}^{-1}(\mathbf{z}) \setminus \pi'\}$, it follows that $f_{\mathbf{z}}(\dots, b_{\pi'}, \dots) = b_{\pi'}$. This is a linear mapping, which results in $H_{[\mathcal{N}_c]_i} = 0$.

In conclusion, the condition $H_{\mathcal{L}_c \circ \mathcal{N}_c} = H_{\mathcal{L}_p \circ \mathcal{N}_c}$ holds if one dominant path $\pi \in \mathcal{B}^{-1}(\mathbf{z})$ exists such that $f_{\mathbf{z}}(\dots, b_\pi, \dots) = b_\pi$ for every label sequence \mathbf{z} .

4.5 Derivation of the proposed approximation from the Fisher information matrix

The identity of the GGN and the Fisher information matrix [16] has been shown for the network using the softmax with cross-entropy loss [17, 18]. Thus, it follows that the GGN matrix of Eq. (13) is identical to the Fisher information matrix. Now we show that the Fisher information matrix is equivalent to the proposed matrix in Eq. (14) under the condition in section 4.4. The Fisher information matrix of MDRNN using CTC is written as

$$F = \mathbb{E}_{\mathbf{x}} \left[J_{\mathcal{N}}^{\top} \mathbb{E}_{\mathbf{l} \sim p(\mathbf{l}|\mathbf{a})} \left[\left(\frac{\partial \log p(\mathbf{l}|\mathbf{a})}{\partial \mathbf{a}} \right)^{\top} \left(\frac{\partial \log p(\mathbf{l}|\mathbf{a})}{\partial \mathbf{a}} \right) \right] J_{\mathcal{N}} \right], \quad (15)$$

where $\mathbf{a} = \mathbf{a}(\mathbf{x}, \theta)$ is the KT -dimensional output of the network \mathcal{N} . CTC assumes output probabilities at each timestep to be independent of those at other timesteps [1], therefore its Fisher information matrix is given as the sum of every timestep. It follows that

$$F = \mathbb{E}_{\mathbf{x}} \left[\sum_t J_{\mathcal{N}_t}^{\top} \mathbb{E}_{\mathbf{l} \sim p(\mathbf{l}|\mathbf{a})} \left[\left(\frac{\partial \log p(\mathbf{l}|\mathbf{a})}{\partial a^t} \right)^{\top} \left(\frac{\partial \log p(\mathbf{l}|\mathbf{a})}{\partial a^t} \right) \right] J_{\mathcal{N}_t} \right]. \quad (16)$$

Under the condition in section 4.4, the Fisher information matrix is given by

$$F = \mathbb{E}_{\mathbf{x}} \left[\sum_t J_{\mathcal{N}_t}^{\top} (\text{diag}(Y^t) - Y^t Y^{t\top}) J_{\mathcal{N}_t} \right], \quad (17)$$

which is the same form as Eq. (4) and (14) combined. See appendix B for the detailed derivation.

4.6 EM interpretation of the proposed approximation

The goal of the Expectation-Maximization (EM) algorithm is to find the maximum likelihood solution for models having latent variables [19]. Given an input sequence \mathbf{x} , and its corresponding target label sequence \mathbf{z} , the log likelihood of \mathbf{z} is given by $\log p(\mathbf{z}|\mathbf{x}, \theta) = \log \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{z})} p(\pi|\mathbf{x}, \theta)$, where θ represents the model parameters. For each observation \mathbf{x} , we have a corresponding latent variable q which is a 1-of- k binary vector where k is the number of all the paths mapped to \mathbf{z} . The log likelihood can be written in terms of q as $\log p(\mathbf{z}, q|\mathbf{x}, \theta) = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{z})} q_{\pi|\mathbf{x}, \mathbf{z}} \log p(\pi|\mathbf{x}, \theta)$.

EM algorithm starts with an initial parameter $\hat{\theta}$, and repeats the following process until convergence.

Expectation step calculates: $\gamma_{\pi|\mathbf{x}, \mathbf{z}} = \frac{p(\pi|\mathbf{x}, \hat{\theta})}{\sum_{\pi \in \mathcal{B}^{-1}(\mathbf{z})} p(\pi|\mathbf{x}, \hat{\theta})}$.

Maximization step updates: $\hat{\theta} = \text{argmax}_{\theta} \mathcal{Q}(\theta)$, where $\mathcal{Q}(\theta) = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{z})} \gamma_{\pi|\mathbf{x}, \mathbf{z}} \log p(\pi|\mathbf{x}, \theta)$.

In the context of CTC and RNN, $p(\pi|\mathbf{x}, \theta)$ is given as $p(\pi|\mathbf{a}(\mathbf{x}, \theta))$ as in Eq. (6), where $\mathbf{a}(\mathbf{x}, \theta)$ is the KT -dimensional output of the neural network. Taking the second-order derivative of $\log p(\pi|\mathbf{a})$ with respect to a^t gives $\text{diag}(Y^t) - Y^t Y^{t\top}$, with Y^t as in Eq. (14). Because this term is independent of π and $\sum_{\pi \in \mathcal{B}^{-1}(\mathbf{z})} \gamma_{\pi|\mathbf{x}, \mathbf{z}} = 1$, the Hessian of \mathcal{Q} with respect to a^t is given by

$$H_{\mathcal{Q}, t} = \text{diag}(Y^t) - Y^t Y^{t\top}, \quad (18)$$

which is the same as the convex approximation in Eq. (14).

5 Experiments

In this section, we present the experimental results on two different tasks of sequence labelling, offline handwriting recognition and phoneme recognition. The performance of Hessian-free optimization for MDRNN with the proposed matrix is compared with the one of stochastic gradient descent (SGD) optimization on the same settings.

5.1 Database and preprocessing

IFN/ENIT Database [20] is a database of handwritten Arabic words, which consists of 32,492 images written by 411 writers. The entire dataset has 5 subsets (a, b, c, d, e). The 25,955 images corresponding to the subsets ($b - e$) are used for training. The validation set consists of 3,269 images

corresponding to the first half of the sorted list in alphabetical order (ae07_001.tif – ai54_028.tif) in set a . Rest of the images in set a , which amounts to 3,268, are used for test. The intensity of pixels is centered and scaled using the mean and standard deviation calculated from the training set.

TIMIT corpus [21] is a benchmark database for evaluating speech recognition performance. The standard training, validation, and core dataset are used for performance evaluation. Each set contains 3,696 sentences, 400 sentences, and 192 sentences respectively. Mel spectrum with 26 coefficients is used as a feature vector with a pre-emphasis filter, 25 ms window size, and 10 ms shift size. Each input feature of the training set is normalized to have zero mean and unit variance. Similarly, the features of core and validation sets are centered and scaled using the mean and standard deviation of the training set.

5.2 Experimental setup

For handwriting recognition, the basic architecture was adopted from the one proposed in [3]. Deeper networks were constructed by replacing the top layer by more layers. The number of LSTM cells in the augmented layer was chosen to make the total number of weights between different networks similar to each other. Detailed architectures are described in Table 1 with results.

For phoneme recognition, deep bidirectional LSTM and CTC in [4] was adopted as the basic architecture. Additionally, the memory cell block [8], in which the cells share the gates, was applied for efficient information sharing. Each LSTM block was constrained to have 10 memory cells.

We have found that using a large value of bias for input/output gates is beneficial for training deep MDRNN. A possible explanation is that the activation of neurons is exponentially decayed by input/output gates during the propagation. Thus, setting large bias values for those gates may help sending information through many layers at the beginning of the learning. For this reason, biases of input and output gates were initialized to 2, whereas the ones for forget gates and memory cells were initialized to 0. All the other weight parameters of MDRNN were initialized randomly from a uniform distribution in the range of $[-0.1, 0.1]$.

Label error rate was used as the metric for performance evaluation along with the average loss of CTC in Eq. (8). It is defined by the edit distance which sums the total number of insertions, deletions, and substitutions required to match two given sequences. The final performance in Table 1 and 2 was evaluated using the weight parameters which gave the best label error rate on the validation set. To map output probabilities to a label sequence, best path decoding [1] was used for Arabic handwriting, and beam search decoding [4, 22] with the beam width of 100 was used for phoneme recognition. For phoneme recognition, 61 phoneme labels were used during training and decoding, and then mapped to 39 classes for calculating the phoneme error rate (PER) in Table 2 [4, 23].

For phoneme recognition, the regularization method suggested in [24] was used. We applied Gaussian weight noise of standard deviation $\sigma = \{0.03, 0.04, 0.05\}$ along with L2 regularization of strength 0.001. Table 2 presents the best result from different values of σ . The network was first trained without noise, then it was initialized to the weights that gave the lowest CTC loss on the validation set. After that, the network was retrained with Gaussian weight noise [4].

5.2.1 Parameters

For HF optimization, we followed the basic setup described in [9], but different parameters were utilized. Tikhonov damping were used along with Levenberg-Marquardt heuristics. The value of the damping parameter λ was initialized to 0.1, and adjusted according to the reduction ratio ρ (multiplied by 0.9 if $\rho > 0.75$, divided by 0.9 if $\rho < 0.25$, and unchanged otherwise). The initial search direction for each run of CG was set to the CG direction found by the previous HF iteration decayed by 0.7. To ensure that CG follows the descent direction, we continued to perform minimum 5 and maximum 30 more CG iterations after it found the first descent direction. We terminated CG at iteration i before reaching the maximum iteration if the following condition is satisfied: $(\phi(x_i) - \phi(x_{i-5}))/\phi(x_i) < 0.005$ where ϕ is the quadratic objective of CG without offset. The training data was divided into 100 and 50 mini-batches for handwriting and phoneme recognition experiments respectively, and used for both of the gradient and matrix-vector product calculation. The learning was stopped if any of two criteria did not improve for 20 epochs in handwriting recognition and for 10 epochs in phoneme recognition, respectively.

For SGD optimization, the learning rate ϵ was chosen from $\{10^{-4}, 10^{-5}, 10^{-6}\}$, and the momentum μ from $\{0.9, 0.95, 0.99\}$. For handwriting recognition, the best performance from all the possible combinations of parameters is presented in Table 1. For phoneme recognition, the best parameters out of 9 candidates for each network were selected after initialization (training without weight noise) based on the CTC loss. Then the networks were trained with weight noise. Additionally, the back-propagated error in LSTM layer was clipped to stay in the range $[-1, 1]$ for stable learning [25]. The learning was stopped after 1000 epochs had been processed. Note that in order to guarantee the convergence, we selected a conservative criteria compared to the reference where the network converged after 85 epochs in handwriting recognition [3] and after 55-150 epochs in phoneme recognition [4].

5.3 Results

Table 1 presents the label error rate on the test set for handwriting recognition. In all cases, the networks trained using HF optimization outperformed the ones using SGD. The advantage of using HF is more pronounced as the depth increases. The improvements from deeper architecture can be seen with the error rate dropping from 6.1% to 4.5% as the depth increases from 3 to 13.

Table 2 shows the phoneme error rate (PER) on the core set for phoneme recognition. The improved performance according to the depth can be observed for both optimization methods. The best PER for HF is 18.54% at 15 layers, and the one for SGD is 18.46% at 10 layers, which are comparable to the one in [4] where the reported results are PER 18.6% from the network with 3 layers having 3.8 million weights and PER 18.4% from the network with 5 layers having 6.8 million weights. The benefit from deeper network is obvious in terms of the number of weight parameters, although this is not meant to be the definitive performance comparison due to different preprocessing. The advantage of HF is not prominent for the experiments using TIMIT database. One explanation is that the networks tend to overfit to relatively small number of the training data samples, which removes the advantage of using advanced optimization techniques.

Table 1: Experimental results on Arabic offline handwriting recognition. The label error rate is presented with the different depth of networks for each optimization method. A^B means a stack of B layers having A hidden LSTM cells in each layer. ‘Epochs’ is the number of epochs required by the network using HF for the stopping criteria fulfilled. ϵ is the learning rate and μ is the momentum.

NETWORKS	DEPTH	WEIGHTS	HF (%)	EPOCHS	SGD (%)	$\{\epsilon, \mu\}$
2-10-50	3	159,369	6.10	77	9.57	$\{10^{-4}, 0.9\}$
2-10-21 ³	5	157,681	5.85	90	9.19	$\{10^{-5}, 0.99\}$
2-10-14 ⁶	8	154,209	4.98	140	9.67	$\{10^{-4}, 0.95\}$
2-10-12 ⁸	10	154,153	4.95	109	9.25	$\{10^{-4}, 0.95\}$
2-10-10 ¹¹	13	150,169	4.50	84	10.63	$\{10^{-4}, 0.9\}$
2-10-9 ¹³	15	145,417	5.69	84	12.29	$\{10^{-5}, 0.99\}$

Table 2: Experimental results on phoneme recognition using TIMIT corpus. PER is presented with the different MDRNN architectures (Depth \times Block \times Cell/Block). σ is the standard deviation of Gaussian weight noise. The other parameters are the same as in Table 1.

NETWORKS	WEIGHTS	HF (%)	EPOCHS	$\{\sigma\}$	SGD (%)	$\{\epsilon, \mu, \sigma\}$
$3 \times 20 \times 10$	771,542	20.14	22	$\{0.03\}$	20.96	$\{10^{-5}, 0.99, 0.05\}$
$5 \times 15 \times 10$	795,752	19.18	30	$\{0.05\}$	20.82	$\{10^{-4}, 0.9, 0.04\}$
$8 \times 11 \times 10$	720,826	19.09	29	$\{0.05\}$	19.68	$\{10^{-4}, 0.9, 0.04\}$
$10 \times 10 \times 10$	755,822	18.79	60	$\{0.04\}$	18.46	$\{10^{-5}, 0.95, 0.04\}$
$13 \times 9 \times 10$	806,588	18.59	93	$\{0.05\}$	18.49	$\{10^{-5}, 0.95, 0.04\}$
$15 \times 8 \times 10$	741,230	18.54	50	$\{0.04\}$	19.09	$\{10^{-5}, 0.95, 0.03\}$

6 Conclusion

Hessian-free optimization as an approach for successful learning of deep MDRNN, in conjunction with CTC, has been presented. To apply HF to CTC, a convex approximation of its objective function has been explored. Improvements in performance are seen as the depth of the network increases for both HF and SGD. HF shows significantly better performance for handwriting recognition compared to SGD, and comparable performance for speech recognition.

References

- [1] Alex Graves. *Supervised sequence labelling with recurrent neural networks*, volume 385. Springer, 2012.
- [2] Alex Graves, Marcus Liwicki, Horst Bunke, Jürgen Schmidhuber, and Santiago Fernández. Unconstrained on-line handwriting recognition with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 577–584, 2008.
- [3] Alex Graves and Jürgen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 545–552, 2009.
- [4] Alex Graves, Abdel-ranhman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Proceedings of ICASSP*, pages 6645–6649. IEEE, 2013.
- [5] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *CoRR*, abs/1412.6550, 2014. URL <http://arxiv.org/abs/1412.6550>.
- [6] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.
- [7] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL <http://arxiv.org/abs/1409.4842>.
- [8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [9] James Martens. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning*, pages 735–742, 2010.
- [10] James Martens and Ilya Sutskever. Learning recurrent neural networks with Hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning*, pages 1033–1040, 2011.
- [11] Nicol N Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.
- [12] Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural Computation*, 6(1):147–160, 1994.
- [13] James Martens and Ilya Sutskever. Training deep and recurrent networks with Hessian-free optimization. In *Neural Networks: Tricks of the Trade*, pages 479–535. Springer, 2012.
- [14] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 369–376, 2006.
- [15] Stephen Boyd and Lieven Vandenbergh, editors. *Convex Optimization*. Cambridge University Press, 2004.
- [16] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.
- [17] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. In *International Conference on Learning Representations*, 2014.
- [18] Hyeyoung Park, S-I Amari, and Kenji Fukumizu. Adaptive natural gradient learning algorithms for various stochastic models. *Neural Networks*, 13(7):755–764, 2000.
- [19] Christopher M. Bishop, editor. *Pattern Recognition and Machine Learning*. Springer, 2007.
- [20] Mario Pechwitz, S Snoussi Maddouri, Volker Märgner, Noureddine Ellouze, and Hamid Amiri. IFN/ENIT-database of handwritten arabic words. In *Proceedings of CIFED*, pages 129–136, 2002.
- [21] DARPA-ISTO. The DARPA TIMIT acoustic-phonetic continuous speech corpus (TIMIT). In *speech disc cd1-1.1 edition*, 1990.
- [22] Alex Graves. Sequence transduction with recurrent neural networks. In *ICML Representation Learning Workshop*, 2012.
- [23] Kai-Fu Lee and Hsiao-Wuen Hon. Speaker-independent phone recognition using hidden markov models. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(11):1641–1648, 1989.
- [24] Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.
- [25] Alex Graves. Rnnlib: A recurrent neural network library for sequence learning problems, 2008.

A \mathcal{R} operator to LSTM

We follow the version of LSTM in [4]. The forward pass of LSTM is to calculate the following functions:

$$\begin{aligned} i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i), \\ f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f), \\ c_t &= f_t \cdot c_{t-1} + i_t \cdot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c), \\ o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o), \\ h_t &= o_t \cdot \tanh(c_t), \end{aligned}$$

where \cdot denotes the element-wise vector product, σ is the logistic sigmoid function, x , h , and c are the input, hidden, and cell activation vector respectively, and i , o , and f are the input, output, and forget gates respectively. All the gates and cells have the same size as the hidden vector h .

Applying \mathcal{R} operator to the above equations gives

$$\begin{aligned} \mathcal{R}_v(i_t) &= \sigma'(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\ &\quad \cdot (V_{xi}x_t + V_{hi}h_{t-1} + V_{ci}c_{t-1} + V_i + W_{hi}\mathcal{R}_v(h_{t-1}) + W_{ci}\mathcal{R}_v(c_{t-1})), \\ \mathcal{R}_v(f_t) &= \sigma'(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\ &\quad \cdot (V_{xf}x_t + V_{hf}h_{t-1} + V_{cf}c_{t-1} + V_f + W_{hf}\mathcal{R}_v(h_{t-1}) + W_{cf}\mathcal{R}_v(c_{t-1})), \\ \mathcal{R}_v(c_t) &= \mathcal{R}_v(f_t) \cdot c_{t-1} + f_t \cdot \mathcal{R}_v(c_{t-1}) + \mathcal{R}_v(i_t) \cdot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\ &\quad + i_t \cdot \tanh'(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \cdot (V_{xc}x_t + V_{hc}h_{t-1} + V_c + W_{hc}\mathcal{R}_v(h_{t-1})), \\ \mathcal{R}_v(o_t) &= \sigma'(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \\ &\quad \cdot (V_{xo}x_t + V_{ho}h_{t-1} + V_{co}c_t + V_o + W_{ho}\mathcal{R}_v(h_{t-1}) + W_{co}\mathcal{R}_v(c_t)), \\ \mathcal{R}_v(h_t) &= \mathcal{R}_v(o_t) \cdot \tanh(c_t) + o_t \cdot \tanh'(c_t) \cdot \mathcal{R}_v(c_t), \end{aligned}$$

where V_{ij} and V_i are taken from v at the same point of W_{ij} and b_i in θ , respectively. Note that θ and v have the same dimension.

B Detailed derivation of the proposed approximation from the Fisher information matrix

The derivative of the negative log probability of Eq. (7) is given by

$$-\frac{\partial \log p(\mathbf{l}|\mathbf{a})}{\partial a_k^t} = y_k^t - \frac{1}{p(\mathbf{l}|\mathbf{a})} \sum_{s \in \text{lab}(\mathbf{l}, k)} \alpha_t(s) \beta_t(s). \quad (19)$$

where $\alpha_t(s)$ and $\beta_t(s)$ denote forward and backward variables respectively, and $\text{lab}(\mathbf{l}, k) = \{u | \mathbf{l}_u = k\}$ is the set of positions where label k occurs in \mathbf{l} [1, 3]. For compact notation, let Y^t denote a column matrix containing y_k^t as its k -th element, and let V^t denote a column matrix containing $v_k^t = \frac{1}{p(\mathbf{l}|\mathbf{a})} \sum_{s \in \text{lab}(\mathbf{l}, k)} \alpha_t(s) \beta_t(s)$ as its k -th element.

The Fisher information matrix [16] is defined by

$$F = \mathbb{E}_{\mathbf{x}} \left[\mathbb{E}_{\mathbf{l} \sim p(\mathbf{l}|\mathbf{x})} \left[\left(\frac{\partial \log p(\mathbf{l}|\mathbf{x}, \theta)}{\partial \theta} \right)^\top \left(\frac{\partial \log p(\mathbf{l}|\mathbf{x}, \theta)}{\partial \theta} \right) \right] \right]. \quad (20)$$

The Fisher information matrix of MDRNN using CTC is written as

$$F = \mathbb{E}_{\mathbf{x}} \left[\mathbb{E}_{\mathbf{l} \sim p(\mathbf{l}|\mathbf{x})} \left[\left(\frac{\partial \log p(\mathbf{l}|\mathbf{a})}{\partial \mathbf{a}} J_{\mathcal{N}} \right)^\top \left(\frac{\partial \log p(\mathbf{l}|\mathbf{a})}{\partial \mathbf{a}} J_{\mathcal{N}} \right) \right] \right] \quad (21)$$

$$= \mathbb{E}_{\mathbf{x}} \left[J_{\mathcal{N}}^\top \mathbb{E}_{\mathbf{l} \sim p(\mathbf{l}|\mathbf{a})} \left[\left(\frac{\partial \log p(\mathbf{l}|\mathbf{a})}{\partial \mathbf{a}} \right)^\top \left(\frac{\partial \log p(\mathbf{l}|\mathbf{a})}{\partial \mathbf{a}} \right) \right] J_{\mathcal{N}} \right], \quad (22)$$

where $\mathbf{a} = \mathbf{a}(\mathbf{x}, \theta)$ is the KT -dimensional output of the network \mathcal{N} . The last step follows from that $J_{\mathcal{N}}$ is independent of \mathbf{l} .

CTC assumes output probabilities at each timestep to be independent of those at other timesteps [1], therefore its Fisher information matrix is given as the sum of every timestep. It follows that

$$F = \mathbb{E}_{\mathbf{x}} \left[\sum_t J_{\mathcal{N}_t}^\top \mathbb{E}_{\mathbf{l} \sim p(\mathbf{l}|\mathbf{a})} \left[\left(\frac{\partial \log p(\mathbf{l}|\mathbf{a})}{\partial a^t} \right)^\top \left(\frac{\partial \log p(\mathbf{l}|\mathbf{a})}{\partial a^t} \right) \right] J_{\mathcal{N}_t} \right] \quad (23)$$

$$= \mathbb{E}_{\mathbf{x}} \left[\sum_t J_{\mathcal{N}_t}^\top \mathbb{E}_{\mathbf{l} \sim p(\mathbf{l}|\mathbf{a})} \left[(Y^t - V^t) (Y^t - V^t)^\top \right] J_{\mathcal{N}_t} \right] \quad (24)$$

$$= \mathbb{E}_{\mathbf{x}} \left[\sum_t J_{\mathcal{N}_t}^\top \left(Y^t Y^{t\top} - Y^t \mathbb{E}_{\mathbf{l}} [V^t]^\top - \mathbb{E}_{\mathbf{l}} [V^t] Y^{t\top} + \mathbb{E}_{\mathbf{l}} [V^t V^{t\top}] \right) J_{\mathcal{N}_t} \right], \quad (25)$$

where Y^t and V^t are defined above.

$\mathbb{E}_{\mathbf{l}}[v_k^t]$ is given by

$$\mathbb{E}_{\mathbf{l}}[v_k^t] = \mathbb{E}_{\mathbf{l} \sim p(\mathbf{l}|\mathbf{a})} \left[\frac{1}{p(\mathbf{l}|\mathbf{a})} \sum_{s \in \text{lab}(\mathbf{l}, k)} \alpha_t(s) \beta_t(s) \right] \quad (26)$$

$$= \sum_{\mathbf{l}} \sum_{s \in \text{lab}(\mathbf{l}, k)} \alpha_t(s) \beta_t(s) \quad (27)$$

$$= y_k^t. \quad (28)$$

$\mathbb{E}_{\mathbf{l}}[v_i^t v_j^t]$ is given by

$$\mathbb{E}_{\mathbf{l}}[v_i^t v_j^t] = \mathbb{E}_{\mathbf{l} \sim p(\mathbf{l}|\mathbf{a})} \left[\frac{1}{p(\mathbf{l}|\mathbf{a})^2} \sum_{s \in \text{lab}(\mathbf{l}, i)} \alpha_t(s) \beta_t(s) \sum_{s \in \text{lab}(\mathbf{l}, j)} \alpha_t(s) \beta_t(s) \right]. \quad (29)$$

Unfortunately Eq. (29) cannot be analytically calculated in general. We apply the sufficient condition for the proposed approximation to be exact in section 4.4. By the assumption of one dominant path in a label sequence, $\mathbb{E}_{\mathbf{l}}[v_i^t v_j^t] = 0$ for $i \neq j$. If the dominant path visits i at time t , $\sum_{s \in \text{lab}(\mathbf{l}, i)} \alpha_t(s) \beta_t(s) = p(\mathbf{l}|\mathbf{a})$. Otherwise $\sum_{s \in \text{lab}(\mathbf{l}, i)} \alpha_t(s) \beta_t(s) = 0$. Under this condition, Eq. (29) can be written as

$$\mathbb{E}_{\mathbf{l}}[v_i^t v_j^t] = \delta_{ij} \sum_{\mathbf{l}} \sum_{s \in \text{lab}(\mathbf{l}, i)} \alpha_t(s) \beta_t(s) \quad (30)$$

$$= \delta_{ij} y_i^t, \quad (31)$$

where δ_{ij} is Kronecker delta. Substituting $\mathbb{E}_{\mathbf{l}}[V^t] = Y^t$ and $\mathbb{E}_{\mathbf{l}}[V^t V^{t\top}] = \text{diag}(Y^t)$ into Eq. (25) gives

$$F = \mathbb{E}_{\mathbf{x}} \left[\sum_t J_{\mathcal{N}_t}^\top (\text{diag}(Y^t) - Y^t Y^{t\top}) J_{\mathcal{N}_t} \right], \quad (32)$$

which is the same form as Eq. (4) and (14) combined.