ACOUSTIC MODEL TRAINING BASED ON NODE-WISE WEIGHT BOUNDARY MODEL INCREASING SPEED OF DISCRETE NEURAL NETWORKS

Ryu Takeda and Kazunori Komatani

Osaka University
The Institute of Scientific and Industrial Research
8-1, Mihogaoka, Ibaraki, Osaka 567-0047, Japan

Kazuhiro Nakadai

Honda Research Institute Japan Co., Ltd.
Wako, Saitama 351-0114, Japan

ABSTRACT

Our purpose is to realize discrete neural networks (NNs), whose some parameters are discretized, as a low-resource and fast NNs for acoustic models. Two essential problems should be tackled for its realization; 1) the reduction of discretization errors and 2) the implementation method for fast processing. We propose a new parameter training algorithm for 1) and an implementation using look-up table (LUT) on general-purpose CPUs for 2), respectively. The former can set proper boundaries of discretization at each node of NNs, resulting in the reduction of discretization error. The latter can reduce the memory usage of NNs within the cache size of CPU by encoding parameters of NNs. Experiments with 2-bit discrete NNs showed that our algorithm maintained almost the same word accuracy as 8-bit discrete NNs and achieved a 40% increase in speed of the NN's forward calculation.

Index Terms— Deep Neural Network, Acoustic Model, Quantization, Discretization

1. INTRODUCTION

Deep neural networks (DNNs) are widely used as acoustic models in automatic speech recognition (ASR) instead of Gaussian mixture models (GMMs) because of their high word accuracy (WA) [1, 2, 3, 4]. However, the applicable computer architecture of DNNs is still restricted because of their high computational cost. Although implementation using a graphics processing unit (GPU) [5] and distributed computing [6] is effective for increasing the speed of DNNs, such an expensive approach cannot be applied to resource-restricted systems, such as small/micro computers or embedded systems with ordinary CPUs. Thus, light DNNs in terms of memory usage and computational cost are required.

Parameter discretization can drastically reduce memory usage and computational cost, and thus enables DNNs to finish processing within a reasonable time without GPUs and distributed computing. A fixed-point DNN, whose weights, bias parameters and middle layer inputs are linearly quantized to n bits, enables fast processing on a Very Large Scale Integration [7] or a CPU with Supplemental Streaming SIMD Extensions 3 (SSSE3) instruction set [8]. The parameters of fixed-point DNNs are trained by iterating the quantization of weights to n bits and usual back propagation [7, 9]. However, experimental selection of n for the best performance requires massive quantity of experiments. Moreover, the above implementation still requires a special instruction set of CPUs or special devices. Neural networks implemented on general-purpose CPUs without a special instruction set will be helpful for small/micro computers and embedded systems.

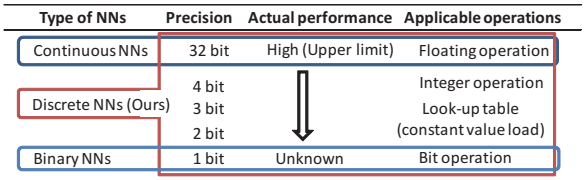

Fig. 1. Property of neural networks

The use of look-up tables (LUTs) is a promising approach, and we call such NNs 'discrete NNs' because their weights are treated as an encoded address for an LUT. For example, the speed of NNs can increase by SSSE3-like processing with an LUT on a general-purpose CPU by exploiting its cache memory. Of course, such techniques can be applied to hardware, too. Our previously proposed training algorithm is based on weight boundary model and control of weight boundary [10]. In this model, the boundary of weights at each layer is restricted to a certain range (*layer-wise* weight boundary model), and it plays a role of layer-wise weight normalization. Weights are discretized only once after training because the distribution of weights has already become uniform or Bernoulli through training. We confirmed that 4-bit discrete NNs actually worked without degradation in a large vocabulary ASR task. A critical issue of 4-bit discrete NNs is that use of an LUT for achieving the fast processing is not realistic because the table size becomes more than 32 Mbytes where a CPU cache no longer works well. Therefore, smaller-bit, such as 2 bits, discrete NNs are required.

We propose a parameter training algorithm and implementation scheme based on *node-wise* weight boundary model to achieve 2-bit discrete NNs. The key for 2-bit discretization is the fact that our previous model normalized weights at each layer in quantization. The errors of quantization with the layer-wise normalization increase because the dynamic range and distribution of weights differ at each node. Therefore, node-wise normalization is expected to adjust weight's range not for a set of all nodes in the layer but for each individual node. Such normalization is incorporated in the node-wise weight boundary model. We also discuss two weight encoding methods using LUTs, one of which can dramatically reduce the LUT size. The training algorithm and implementation are validated through experiments involving a large vocabulary ASR task and the real-time factor (RTF) of forward calculation of NNs.

One of the advantages of the discrete NNs is that they can be applied to various tasks by adjusting their precision of weights, i.e., the number of bits, according to their required performance (Fig. 1). In other words, the discrete NNs lie between continuous NNs and

binary NNs. Binary NNs [11, 12], whose parameters are all binary, i.e. $\{0, 1\}$, are considered an ultimate form of NNs, but their actual performance for an large vocabulary ASR has not been shown as far as we know.

Our contributions are as follows:

1. Achieve fast processing of 2-bit discrete NNs for large vocabulary ASR task on a general-purpose CPU
2. Discuss the minimum number of bits for the discrete NNs through experiments

2. DISCRETE NEURAL NETWORKS BASED ON LAYER-WISE WEIGHT BOUNDARY MODEL

We first explain the forward model of discrete NNs with layer-wise normalization by contrasting it to that of continuous NNs. Next, we introduce its parameter training algorithm based on weight boundary model and *contraction mapping*. Finally, we explain the problem with previous work regarding smaller-bit discrete NNs. Figure 2 gives an overview of the training process of discrete NNs .

2.1. Forward Model in Semi-Continuous Domain

The structure of continuous NNs is defined recursively on the layer index l. First, the input vector $\mathbf{x}_l = [x_{l,1}, ..., x_{l,N_l}]^T \in \mathbb{R}^{N_l}$ is affine transformed, then activation functions $\mathbf{h}_l : \mathbb{R}^{M_l} \rightarrow \mathbb{R}^{N_{l+1}}$ are applied. Here, $\cdot^T$ denotes the transportation operator, N_l and M_l are dimensions of $\mathbf{x}_l$ and temporal vector $\mathbf{z}_l = [z_{l,1}, ..., z_{l,M_l}]^T$ at l-th layer, respectively. Therefore, the output of the L-th layer can be recursively described for $l = 0, ..., L-1$ given the initial input vector $\mathbf{x}_0$.

$$\mathbf{z}_l = \mathbf{W}_l \mathbf{x}_l + \mathbf{b}_l \quad (1)$$

$$\mathbf{x}_{l+1} = \mathbf{h}_l(\mathbf{z}_l) \quad (2)$$

where the matrix $\mathbf{W}_l = (w_{l,ij}) \in \mathbb{R}^{M_l \times N_l}$ and vector $\mathbf{b}_l = [b_{l,1}, ..., b_{l,M_l}]^T \in \mathbb{R}^{M_l}$ are the weight and bias parameters at the l-th layer, respectively. The sigmoid function and soft-max function are often used as an activation function $\mathbf{h}_l$. In ASR, the input vector $\mathbf{x}_0$ corresponds to temporal speech features, and the final output vector $\mathbf{x}_L$ is used for the acoustic likelihood of Hidden Markov Model (HMM) states [1].

The forward calculation of discrete NNs is defined to approximate the continuous NNs in a semi-continuous domain where weights and middle input vectors are encoded into several bits. **Note that this formula is used only for forward calculation in the recognition phase not for training.** The i-th element of $\mathbf{z}_l$ is calculated based on layer-wise normalization as follows:

$$z_{l,i} = \alpha_l \sum_{j=0}^{N_l - 1} \mathcal{F}(\mathcal{Q}_y[y_{l,ij}], \mathcal{Q}_x[x_{l,j}]) + b_{l,i} \quad (3)$$

where $\mathcal{Q}_y$ and $\mathcal{Q}_x$ are the encoding function to binary or integer code for normalized weights $\mathbf{Y}_l = (y_{l,ij}) \in \mathbb{R}^{M_l \times N_l}$ (i.e. $y_{l,ij} = w_{l,ij} / \max_{i,j} |w_{l,ij}|$) and middle input variable $x_{l,j}$, respectively. The $\mathcal{F}(a, b)$ includes the decoding and multiplication functions of two binary-coded values, a and b. The α_l is a normalization parameter of the l-th weights and depends on the definition of $\mathbf{Y}_l$, $\mathcal{Q}_y$, $\mathcal{Q}_x$ and $\mathcal{F}$.

The well designed operators for the target computer architecture increase the computation speed based on the instance calculation of several $\mathcal{F}$s and the summation in Eq.(3). For example, we use the case when $\mathcal{Q}_y$, $\mathcal{Q}_x$ are linear quantization into a n-bit integer and $\mathcal{F}$ is just an integer multiplication. If we use a special instruction set, such as the SSSE3, only three instructions result in multiply-and-add of eight variables. If we use an LUT, the pre-calculated result of multiply-and-add of several variables is loaded at the index of connected binary-codes of weights and middle inputs.

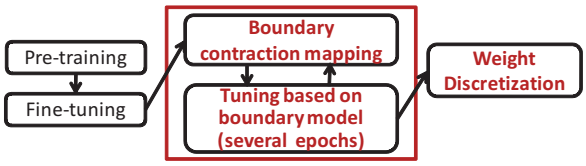

Fig. 2. Training process of discrete NNs

2.2. Parameter Training in Continuous Domain

The parameters for discrete NNs are trained in the continuous domain in two steps, 1) back propagation [13] based on the layer-wise boundary model for several epochs, and 2) contraction of the weight boundary. Steps 1) and 2) are conducted iteratively several times. We assume to already have fine-tuned parameters with continuous NNs.

We set the boundary constraint of weights similar to Eq.(3) to obtain parameters appropriate for bit encoding. By using the latent matrix $\mathbf{V}_l = (v_{l,ij}) \in \mathbb{R}^{M_l \times N_l}$, the forward model can be written as

$$\mathbf{z}_l = \alpha_l \mathbf{g}_l(\mathbf{V}_l)\mathbf{x}_l + \mathbf{b}_l, \quad (4)$$

where $\mathbf{g}_l(\mathbf{x}) = (g_l(x_{ij}))$ is a element-wise bounded function matrix with the range of $[-1, 1]$, such as $\tanh(x)$ whose derivative is $1 - \tanh^2(x)$. Here, $y_{l,ij} = g_l(v_{l,ij})$.

The parameters are optimized by supervised back propagation with the cost function E and supervisory signal vector $\mathbf{r} = [r_1, ..., r_{N_L}]^T \in \mathbb{R}^{N_L}$, the same as continuous NNs. The cross entropy function is used as E. After calculating the initial error vector $\boldsymbol{\epsilon}_L = \left(\frac{\partial E}{\partial \mathbf{x}}(\mathbf{r}, \mathbf{x}_L)\right)$, we update each parameter for $l = L-1, ..., 0$ as follows:

$$\boldsymbol{\delta}_l = \left(\frac{\partial \mathbf{h}_l^T}{\partial \mathbf{z}}(\mathbf{z}_l)\right) \boldsymbol{\epsilon}_{l+1}, \quad (5)$$

$$\boldsymbol{\epsilon}_l = \alpha_l \mathbf{g}(\mathbf{V}_l)^T \boldsymbol{\delta}_l, \quad (6)$$

$$\alpha_l \leftarrow \alpha_l - \eta \boldsymbol{\delta}_l^T \mathbf{g}_l(\mathbf{V}_l)\mathbf{x}_l, \quad (7)$$

$$\mathbf{V}_l \leftarrow \mathbf{V}_l - \eta \frac{\partial \mathbf{g}_l}{\partial \mathbf{V}}(\mathbf{V}_l) \circ \left(\alpha_l \boldsymbol{\delta}_l \mathbf{x}_l^T\right), \quad (8)$$

$$\mathbf{b}_l \leftarrow \mathbf{b}_l - \eta \boldsymbol{\delta}_l, \quad (9)$$

where $\circ$ is an operator of the element-wise multiplication of matrices, and η is a learning parameter that controls the speed and precision of training. Note that the part of the training algorithm for binary NNs recently discussed in [12] is similar to our weight boundary model, but we have proposed it before [10].

At an appropriate epoch, we apply boundary contraction mapping to make the normalization parameter α_l smaller. This is because of the following two facts from the previous experiments; 1) a smaller α_l reduces the quantization errors (QEs) in Eq.(3), and 2) the distribution of $w_{l,ij}$ becomes non-Gaussian, nearly Bernoulli, which

is better for a linear quantizer $\mathcal{Q}_y$.

$$\alpha_l \leftarrow \max_{i,j} |w_{l,ij}|, \quad (10)$$

$$v_{l,ij} \leftarrow w_{l,ij} / \max_{i,j} |w_{l,ij}|, \quad (11)$$

where $\mathbf{W}_l = \alpha_l \mathbf{g}_l(\mathbf{V}_l)$. Since we use $\tanh$ for g, α_l decreases monotonically with this operation.

2.3. Problem with Layer-wise Boundary Model

The training based on the layer-wise boundary model enables us to construct a discrete NN and achieve 4-bit quantization of parameters without degradation of ASR performance. However, the increase in forward calculation speed using an LUT has not been possible because the memory size of an LUT becomes 32 Mbytes even for 4-bit quantization. Such a large table size is usually more than the cache size of CPUs. Therefore, we need smaller-bit NNs without ASR performance degradation. The limitation of the quantization is mainly due to the normalization of weights at each layer while the actual dynamic range and distribution of weights differ at each node of NNs. We should consider the different dynamic ranges of weights in the training and implementation for further improvement.

3. DISCRETE NEURAL NETWORKS BASED ON NODE-WISE WEIGHT BOUNDARY MODEL

We propose a training algorithm and implementation based on the node-wise weight boundary model for discrete NNs to adjust the dynamic range of weights among nodes, which results in equalizing weight distributions at each node. We first explain the node-wise weight normalization in forward calculation, node-wise boundary model for training and its parameter update. Next, we discuss the implementation using LUT, which can be applied to any computer architecture. Finally, the selection of layers for discretization is discussed.

3.1. Forward Model and Parameter Optimization

We control the boundary of weights more flexibly and adequately for each node, not for each layer. Therefore, different normalization parameters $\boldsymbol{\lambda}_l = [\lambda_{l,1}, ..., \lambda_{l,M_l}]^T$ are introduced to each node, instead of α_l. The node-wise normalization in a semi-continuous domain is represented as

$$z_{l,i} = \lambda_{l,i} \sum_{j=0}^{N_l-1} \mathcal{F}(\mathcal{Q}_y[y_{l,ij}], \mathcal{Q}_x[x_{l,j}]) + b_{l,i}. \quad (12)$$

By using the notation of the diagonal matrix of normalization parameters $\boldsymbol{\Lambda}_l = \mathrm{diag}(\lambda_{l,1}, ..., \lambda_{l,M_l})$, the corresponding node-wise weight boundary model in the continuous domain is expressed as

$$\mathbf{z}_l = \boldsymbol{\Lambda}_l \mathbf{g}_l(\mathbf{V}_l)\mathbf{x}_l + \mathbf{b}_l. \quad (13)$$

The parameters are also trained using back propagation based on the stochastic gradient. The update rules concerning new parameters are derived straightforwardly as

$$\boldsymbol{\epsilon}_l = \boldsymbol{\Lambda}\mathbf{g}(\mathbf{V}_l)^T \boldsymbol{\delta}_l, \quad (14)$$

$$\boldsymbol{\lambda}_l \leftarrow \boldsymbol{\lambda}_l - \eta \boldsymbol{\delta}_l \circ (\mathbf{g}_l(\mathbf{V}_l)\mathbf{x}_l), \quad (15)$$

$$\mathbf{V}_l \leftarrow \mathbf{V}_l - \eta \frac{\partial \mathbf{g}_l}{\partial \mathbf{V}}(\mathbf{V}_l) \circ \left(\boldsymbol{\Lambda}_l \boldsymbol{\delta}_l \mathbf{x}_l^T\right). \quad (16)$$

We can see that the propagated errors δ_l affects each normalization parameter, whereas they are concentrated on one α_l in Eq.(7). The boundary contraction mapping under $\mathbf{W}_l = \boldsymbol{\Lambda}_l \mathbf{g}_l(\mathbf{V}_l)$ is also modified as

$$\lambda_{l,i} \leftarrow \max_j |w_{l,ij}|, \quad (17)$$

$$v_{l,ij} \leftarrow w_{l,ij} / \max_j |w_{l,ij}|. \quad (18)$$

3.2. Implementation using Look-up Table

There are two types of LUT implementation; whether $\mathcal{Q}_x$ outputs a binary $\{0, 1\}$ or not. We first explain the encoding and decoding processes for an LUT in a general case. After that, we discuss a specific binary case.

3.2.1. General Model

For the following explanation, we divide the row-vector of weights and middle input vector into several groups, and assume each group has D elements. Their binary code sets can be described as

$$\bar{\mathbf{y}}_{(l,i,k)} = \{\mathcal{Q}_y[y_{l,ij}]\}_{j=Dk}^{D(k+1)-1}, \text{and} \quad (19)$$

$$\bar{\mathbf{x}}_{(l,k)} = \{\mathcal{Q}_x[x_{l,i}]\}_{i=Dk}^{D(k+1)-1}, \quad (20)$$

respectively. The LUT, T, is referred by the combination of the binary code sets. For example, when $\bar{\mathbf{y}}_{(l,i,k)} = [110, 101]$ and $\bar{\mathbf{x}}_{(l,k)} = [010, 101]$, the index for T becomes $[110101010101]$ in binary code. The forward calculation from Eq.(12) is re-described as

$$z_{l,i} = \lambda_{l,i} \sum_{k=0}^{N_l/D} T[\bar{\mathbf{y}}_{(l,i,k)}\bar{\mathbf{x}}_{(l,k)}] + b_{l,i}. \quad (21)$$

Obviously, the number of summations decreases to $1/D$.

The T has the calculation results of every pattern of binary code sets in advance. The table is built up by enumerating every possible pattern of $\bar{\mathbf{a}}, \bar{\mathbf{b}}$, i.e. from 0 to $2^n - 1$ in binary, as follows

$$T[\bar{\mathbf{a}}\bar{\mathbf{b}}] = \sum_{j=0}^{D-1} \mathcal{F}(a_j, b_j) = \sum_{j=0}^{D-1} \mathcal{Q}_y^{-1}[a_j]\mathcal{Q}_x^{-1}[b_j] \quad (22)$$

where a_j and b_j denote the corresponding elements of $\bar{\mathbf{a}}$ and $\bar{\mathbf{b}}$. Since $\mathcal{Q}_y^{-1}$ and $\mathcal{Q}_x^{-1}$ are the decoding function corresponding to $\mathcal{Q}_y$ and $\mathcal{Q}_x$, they return the continuous representation of the binary pattern.

Given $g_l(x) = \tanh(x)$ and $\mathbf{h}_l(x) = (1/(1 + \exp(-x))$, each n-bit decoding and encoding operator is defined as

$$\mathcal{Q}_x[x] = \mathrm{floor}[(2^n - 1)x + 0.5], \quad (23)$$

$$\mathcal{Q}_y[y] = \mathrm{floor}[(2^n - 1)(y + 1)/2 + 0.5], \quad (24)$$

$$\mathcal{Q}_x^{-1}[x] = x/(2^n - 1), \quad (25)$$

$$\mathcal{Q}_y^{-1}[y] = 2y/(2^n - 1) - 1. \quad (26)$$

We did not consider optimizing these LUTs in this paper.

3.2.2. Binary Model

If we adopt the binary quantization for $\mathbf{x}_l$, the table size halves through bit mask operation. For example, when $\bar{\mathbf{y}}_{(l,i,k)} = [110, 101]$ and $\bar{\mathbf{x}}_{(l,k)} = [0, 1]$, the index for T is usually $[11010101]$. By masking bits of $\bar{\mathbf{y}}_{(l,i,k)}$ with $\bar{\mathbf{x}}_{(l,k)}$, the index for binary model table

Table 1. Memory requirement per layer in bytes

	for weights	for LUT
32-bit float	$4N_lM_l$	–
8-bit SSSE	N_lM_l	–
4-bit general LUT	$N_lM_l/2$	2^{8D}
3-bit general LUT	$N_lM_l3/8$	2^{6D}
3-bit-bin binary LUT	$N_lM_l3/8$	2^{3D}
2-bit general LUT	$N_lM_l/4$	2^{4D}
1-bit general LUT	$N_lM_l/8$	2^{2D}

Table 2. Configuration

Audio data STFT analysis	16 bits, 16 kHz sampling hamming window: 25 ms, shift: 10 ms
Features for GMM	MFCC 39 dim. [13+ Δ 13 + $\Delta\Delta$ 13]
Features for DNN	FBANK 825 dim. [(25+ Δ 25 + $\Delta\Delta$ 25) $\times$ 11 frames]
Language model	3-gram statistical 65000 words
GMM-HMM	3-state tri-phone 4000 tied-states 32 mixtures
# of DNN layer (L)	7
DNN network size	input layer: 1024 $\times$ 825 middle layer: 1024 $\times$ 1024 output layer: 4000 $\times$ 1024
Training set Test set	223 hours (799 males and 168 females) 3.5 hours (15 males and 5 females)

T_b becomes $[000101]$ as long as the decoding functions satisfy $Q_y^{-1}[000] = 0$, $Q_x^{-1}[0] = 0$, and $Q_x^{-1}[1] = 1$. This can be understood by following equations

$$T[11010101] = Q_y^{-1}[110]Q_x^{-1}[0] + Q_y^{-1}[101]Q_x^{-1}[1], \quad (27)$$
$$= 0 + Q_y^{-1}[101] = Q_y^{-1}[000] + Q_y^{-1}[101], \quad (28)$$
$$= T_b[000101]. \quad (29)$$

Therefore, the definition of $\mathcal{Q}_y$ and $\mathcal{Q}_y^{-1}$ should be modified heuristically from Eqs. (24) and (26) to realize this binary model. Note that this will increase the QE and lead to performance degradation of ASR.

The memory usages are compared in Table 1. The terms *32-bit float* and *8-bit SSSE* do not require memory for an LUT. The term *3-bit-bin (binary LUT)* represents a binary case, and others are general cases. The *32-bit float* requires $4N_lM_l$ but others require less than $1/8$ that. The size of an LUT depends on parameter D. The total memory usage should be less than 1 or 2 Mbytes considering the size of the actual CPU cache.

3.3. Selection of Discrete Layer

The discrete forward calculation is not applied to every layer because at least the inputs of the first layer are not bounded $[0, 1]$ [8, 10]. However, there is a possibility to improve ASR performance by selecting layers for weight discretization. In our previous work, weights of all layers except for the first layer were discretized. For this study, we left the last layer continuous because the final outputs are used for classification.

4. EXPERIMENTS

4.1. Experimental Setup

We evaluated our node-wise boundary model by conducting large-vocabulary continuous-speech recognition experiments using the Corpus of Spontaneous Japanese (CSJ), which is a collection of Japanese lecture recordings [14]. We first compare WA of node-wise model and layer-wise model at n-bit discretization. We investigate WAs of all the combinations of applying the layer-wise or node-wise model to the training (Eq.(4) or Eq.(13)) and recognition phases (Eq.(3) or Eq.(12)). Next, we discuss the average QE of normalized weights, $|y - \mathcal{Q}_y^{-1}[\mathcal{Q}_y[y]]|$, and the node-wise weight statistics by using normalized kurtosis, $\mathbb{E}[(x - \mathbb{E}[x])^4]/\mathbb{E}[(x - \mathbb{E}[x])^2]^2 - 3$. Here, $\mathbb{E}$ represents the expectation operator. Finally, the RTF of forward calculations of each implementation, LUT in general or binary model, are compared. The RTF is defined as

$$\text{RTF} = (\text{processing time})/(\text{data duration}). \quad (30)$$

The training data for the acoustic model of the DNNs contained 223 hours of *Academic-Presentation-Speech* recordings [1]. The evaluation data were test sets 1 and 2 of the CSJ, i.e., 3.5 hours of lectures featuring 20 speakers (15 males and 5 females). The training data for the language model contained all transcriptions in CSJ except the evaluation data. We used a tri-gram language model with 65,000 words. Julius (ver. 4.3.1) was used for decoding [15], and the language model weight and insertion penalty were set to default, 8 and -2, respectively.

We first trained a GMM-HMM with a tri-phone, 4000 tied-states, and 32 Gaussian mixtures by using HTK[2]. The 13 Mel-frequency cepstral coefficients (MFCCs) and delta and delta-delta coefficients with mean and variance normalization per utterance were used as speech features. The features were extracted at every 10-ms interval from the speech signal, whose sampling rate was 16 kHz. The window size for short-time Fourier Transformation (STFT) was 25 ms. The DNN-HMM used the same HMM with a GMM-based model and $L = 7$ layers with 1024 hidden nodes. The output dimensions were 4000 to classify the tied-states of the HMM. There were a total of 825 dimensions of features for DNNs input, including 11 frames (previous 5 frames and following 5 frames) of basic features. The basic features consisted of 25 log filter bank coefficients, and the delta and delta-delta coefficients. The mean and variance normalization were applied to features. These configurations are summarized in Tab. 2.

Next, the DNNs were trained with Viterbi force alignment labels by using the GMM-HMM. We used a discriminative pre-training method [1] for the pre-training and used the AdaGrad method [16] for scheduling learning rate parameters. The mini-batch size was 64. After the fine tuning of the parameters, the training with the boundary model was applied. Although the drop out [17] will improve the absolute performance of DNNs, we believe that it does not critically affect the relative results among the methods, as discussed in [7].

4.2. Results

4.2.1. Word Accuracy

Table 3 specifies the WAs with different number of bits for discretization. The *normal training* row represents the weights that

[1] Since the amount of data and the ASR decoder are different from those in our previous paper[10], WA was a little worse than that of the previous paper.

[2] http://htk.eng.cam.ac.uk/

Table 3. Word accuracy vs. quantization bits.

		Applied model		Discrete layer l	Word accuracy (%)							
		Training	Recognition		1-bit	2-bit -bin	2-bit	3-bit -bin	3-bit	4-bit	8-bit	32-bit float
Normal training		–	layer-wise	1-6	–	–	-2.24	–	-1.16	-3.35	80.63	81.86
		–	layer-wise	1-5	1.10	–	2.17	–	**2.80**	**49.54**	81.74	
		–	node-wise	1-6	–	–	2.13	–	57.47	79.07	81.82	
		–	node-wise	1-5	1.07	0.69	2.17	1.33	**62.35**	**79.83**	81.82	
Baseline	P0	layer-wise	layer-wise	1-5	2.98	–	77.78	–	80.07	81.07	81.32	81.33
Proposed	P1	layer-wise	node-wise	1-5	2.73	41.17	77.55	59.45	80.82	81.47	81.36	
	P2	node-wise	layer-wise	1-5	2.06	–	44.35	–	66.13	80.14	81.52	81.53
	P3	node-wise	node-wise	1-5	1.45	58.45	**79.37**	**61.40**	**81.05**	81.10	81.52	

Table 4. Quantization error (QE) vs. quantization bits. Discrete layers are 1-5 in all cases.

		Applied model		Quantization error				
		Training	Recognition	1-bit	2-bit	3-bit	4-bit	8-bit
Normal training		–	layer-wise	9.13E-01	2.50E-01	8.42E-02	3.43E-02	1.96E-03
		–	node-wise	8.31E-01	**1.98E-01**	**7.25E-02**	3.33E-02	1.96E-03
Baseline	P0	layer-wise	layer-wise	3.10E-01	1.89E-01	8.16E-02	3.29E-02	1.97E-03
Proposed	P1	layer-wise	node-wise	2.26E-01	**1.27E-01**	**7.25E-02**	3.58E-02	1.96E-03
	P2	node-wise	layer-wise	7.24E-01	1.38E-01	6.83E-02	3.35E-02	1.96E-03
	P3	node-wise	node-wise	2.92E-01	1.52E-01	7.85E-02	3.44E-02	1.96E-03

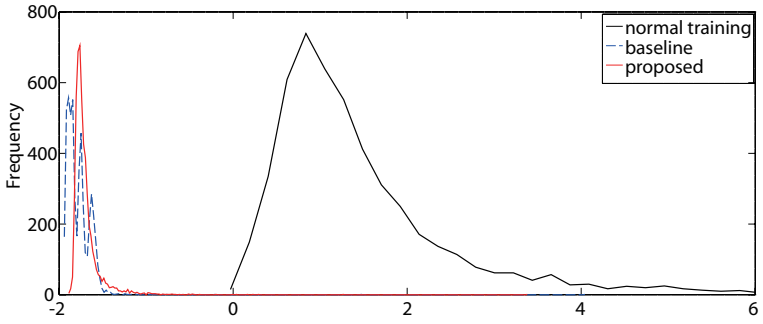

Fig. 3. Kurtosis distributions

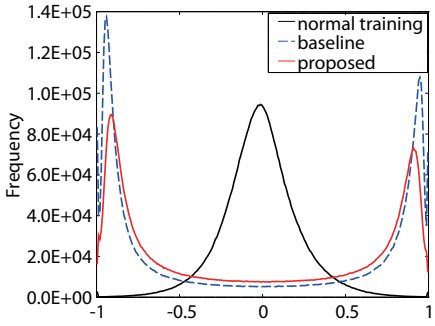

Fig. 4. Weight distributions

were not trained for discrete NNs. The *Discrete Layer* column represents the layers were discretized. The *3-bit-bin* means the bits used for weight was 3 and that for middle inputs was 1. The *2-bit-bin* means the bits used for weight was 2 and that for middle inputs was 1. For other notations, such as *2-bit*, the bits for weight and middle inputs were the same. The *32-bit float* means un-quantized weights. Note that the WA of GMM-HMM was 75.8 %.

The WAs only slightly differed among all methods in the 8-bit case, but the difference became clearer at smaller bits. We focus on the rows of *normal training* in this table. The normal trained weights did not work well with less than 4-bit discretization. By just applying the node-wise model in *recognition*, WAs improved dramatically, indicating the importance of node-wise normalization. The discrete layer selection seemed to improve WA at a sensitive discretization, such as at 3 bits. Therefore, we now compare the results whose discrete layers were from 1 to 5.

The WAs of our node-wise model outperformed the baseline by a maximum of about 1.8 points in the smaller-bit case, except the mismatched combination case, P2. Moreover, P3 maintained high accuracy even in the 2-bit case and degraded by only 2 points from 8-bit discretization which is almost the same as continuous NNs. The WA of the *3-bit-bin* and *3-bit-bin* was more than 50%, although the middle inputs were quantized to $\{0, 1\}$. These results show the advantages of our node-wise boundary model. However, nothing was recognized with the *1-bit* quantized weights, and there is a large gap between 2-bit and 1-bit quantization. Therefore, more precise training or quantization is required for the 1-bit NNs.

4.2.2. Quantization Error and Weight Distribution

The average QEs at each bit are listed in Table 4 to understand the relationship between QE and WA. The QEs of the normal training and the baseline (P0) decreased by applying node-wise model in the recognition phase. However, the *low average QE* did not necessarily mean a high WA. For example, the QE of P1 improved from that of P0, but their WAs were the same.

We then focused on the distribution of the *node-wise* kurtosis of weights to analyze the difference among normal training, node-wise and layer-wise boundary model training. The kurtosis of Gaussian, uniform and Bernoulli are 0, -1, -2 respectively. Figure 3 shows

Table 5. Computer specifications

OS	Ubuntu 14.04.2 LTS
CPU	Intel Core i5-4690 3.50GHz
Memory	32GB
Cache	6M

Table 6. RTF of forward calculation and memory usage on single CPU

	Methods	RTF	D	Memory usage (bytes) LUT	Memory usage (bytes) total weights ($l = 1, ..., 5$)
Standard	32-bit float	**0.981**	1	–	20 M
Baseline	**4-bit (general LUT)**	**1.222**	3	**32 M**	2.5 M
	4-bit (general LUT)	**0.709**	2	**128 K**	2.5 M
Proposed	3-bit (general LUT)	0.644	3	512 K	1.85 M
	3-bit-bin (binary LUT)	0.500	7	4 M	1.85 M
	2-bit (general LUT)	**0.561**	4	**128 K**	**1.25 M**
	1-bit (general LUT)	**0.484**	8	**128 K**	**640 K**
Special	8-bit SSSE	**0.496**	8	–	5 M

the kurtosis distribution of normal training, the baseline P0 and ours P3 after training. Since the kurtosis of normal training was over 0, its weight distribution looks like Gaussian or more sparse one. On the other hand, the weight distributions of baseline and ours were likely Bernoulli because most of the kurtosis were less than -1. We can see the distribution of the baseline's kurtosis had several peaks while that of ours had one. This suggests that the "*outlier*", such as one of several peaks at higher kurtosis, increases the non-uniform and unexpected prediction errors of NNs after quantization, and the range of errors are also different among nodes. Therefore, the *outlier* errors are unfortunately accumulated, not canceled by each other, and the errors exceed the acceptable range that can be smoothed with an HMM. The node-wise boundary training equalizes the weight distribution at each node and fill the gap between continuous and discretized weight, which avoids such case and improves WA.

Finally, we just confirm the actual weight distributions of each methods in Fig. 4. While the distribution of normally trained weights is like Gaussian, the distribution of weights trained with boundary model become like Bernoulli. The two peaks near -1 and 1 tend to go outside as the normalization parameters $\alpha_l, \lambda_{l,i}$ become smaller. Therefore, they are considered to control the form of distribution.

4.2.3. Real Time Factor of Forward Calculation with LUT

We reveal the actual increase in speed of the forward calculation of discrete NNs with two implementation schemes we used, *general model* and *binary model*. The memory usage of LUT and weights are also shown. The computer specifications are listed in Table 5, and the frequency and cache size of CPU were sufficient for using an LUT.

We focused on the relationship between the RTF and its required memory for LUT size and weights, as shown in Table 6. The term *32-bit float* denotes the usual implementation without special instruction sets or LUT, and *8-bit SSSE* is the implementation using [8]. The *baselines* are the results of 4-bit discrete NNs that was the limitation of layer-wise model. The notation D means the number of variables calculated at the same time with Eq. (21). The RTF of our implementations with $D = 3, 4$ achieved a 30% to 40% increase in speed from that with the baseline and became closer to that of the SSSE, which uses the CPU's special instruction set. Moreover, the RTF of the 3-bit-bin, whose WA was still not sufficient, was almost the same as that of the SSSE, although it required a 4 M LUT. If we can achieve 2-bit discrete NNs, the LUT size with $D = 8$ becomes 512 K, and will run at the same speed as the SSSE. Since *1-bit* discrete NNs outperformed 8-bit SSSE, the improvement of WA of it is expected.

5. DISCUSSION

We obtained two facts about WA, QE, and weight statistics from the experiments, 1) the actual relationship between QE and WA, and 2) effects of weight normalization. The former affects the strategy of parameter training of NNs. Since QE does not necessary improve WA directly, it is no longer effective just to add the minimizing constraint of QE to the cost function of NNs. The latter indicates the possibility of another constraint of weights for discrete NNs. If we use this fact directly, minimization of the mean of the node-wise kurtosis may be effective in improving WA. Of course, the non-linear quantization of weights using an optimized LUT will also improve WA of low-bit discrete NNs, such as *1-bit* discrete NNs.

On the other hand, there is a possibility of an implementation based on a *binary case* LUT because our node-wise model does not take into account the quantization effect of middle inputs. Ideally, the distribution of its output signal should also be Bernoulli. Therefore, it is important to find or develop a useful constraint to control their distribution. This may be possible with the Bayesian approach or just controlling the scaling parameter of the sigmoid function used as the activation function. After addressing these issues, we will conduct an implementation based on a 3-bit and 2-bit binary or 1-bit LUT.

6. CONCLUSION

Our goal is the development of discrete neural networks (NNs) with reduced memory and computational cost for acoustic models on CPUs without special processors. Two-bit discrete NNs are required for realistic implementation in terms of memory usage because even 4-bit discretization requires a large look-up table. The key to using two bits is normalizing variation of weights to reduce the influence of discretization error. We developed a parameter training algorithm based on the node-wise weight boundary, not layer-wise as in previous studies. The algorithm resulted in the node-wise equalization of weight distribution. We also implemented discrete NNs with two different schemes in terms of memory usage. Experiments with 2-bit discrete NNs showed that our algorithm maintained high word accuracy and achieved a 40% increase the speed of the NNs' forward calculation.

The main future work is the 1-bit discretization of middle inputs to achieve dramatic improvement in memory usage and processing speed. Key approaches making this possible are non-linear weight quantization and giving appropriate constraint to the distribution of middle inputs during training.

Acknowledgment This work was partially supported by JSPS KAKENHI Grant Number 15K16051.

7. REFERENCES

[1] F. Seide, G. Li, , and X. Chen D. Yu, "Feature engineering in context-dependent deep neural networks for conversational speech transaction," in *Proc. of ASRU*, 2011, pp. 24–29.

[2] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using contex-dependent deep neural network," in *Proc. of Interspeech*, 2011, pp. 437–440.

[3] G. Hinton, L. Deng, D. Yu, G. E. Geroge, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and othres, "Deep neural networks for acuostic modelling in speech recognition," *Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[4] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocaburary speech recognition," *IEEE Trans. on ASPS*, vol. 20, no. 6, pp. 82–97, 2012.

[5] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proc. of ICML*, 2009, pp. 873–880.

[6] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Ng, "Large scale distributed deep neural networks," in *Advances in NIPS*, 2012, pp. 1223–1231.

[7] J. Kim, K. Hwang, and W. Sung, "X1000 real-time phoneme recognition VLSI using feed-forward deep neural networks," in *Proc. of ICASSP*, 2014, pp. 7510–7514.

[8] V. Vanhouche, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," in *Proc. of Deep Learning and Unsupervised Feature Learning NIPS Workshop*, 2011.

[9] C.Z. Tang and H.K. Kwang, "Multilayer feedforward neural networks with single powers-of-two weights," *IEEE Trans. on Signal Processing*, vol. 41, no. 8, pp. 2724–2727, 1993.

[10] R. Takeda, N. Kanda, and N. Nukaga, "Boundary contraction training for acoustic model based on discrete deep neural networks," in *Proc. of Interspeech*, 2014, pp. 1063–1067.

[11] D. Soudry, I. Hubara, and R. Meir, "Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights," in *Advances in NIPS*, 2014, pp. 963–971.

[12] M. Kim and P. Smaragdis, "Bitwise neural networks," in *Proc. of ICML Workshop on Resource-Efficient Machine Learning*, 2015.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating erros," *Nature*, pp. 533–536, 1986.

[14] K. Maekawa, "Corpus of spontaneous Japanese: Its design and evaluation," in *ISCA & IEEE Workshop on Spontaneous Speech Processing and Recognition*, 2003.

[15] A. Lee and T. Kawahara, "Recent development of open-source speech recognition engine Julius," in *Proc. of APSIPA ASC*, 2009, pp. 131–137.

[16] J. Duchi, Elad. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.