人工知能学会研究会資料
JSAI Technical Report
SIG-Challenge-046-02 (11/09)

量子化 Deep Neural Network のための 有界重みモデルに基づく音響モデル学習

Acoustic Model Training based on Weight Boundary Model for Discrete Deep Neural Networks

武田 龍 [*1], **中臺一博** [*2], **駒谷和範** [*1]

Ryu TAKEDA[*1], **Kazuhiro NAKADAI**[*2], **Kazunori KOMATANI**[*1]

大阪大学 産業科学研究所 [*1], (株) ホンダ・リサーチ・インスティチュート・ジャパン [*2]

The Institute of Scientific and Industrial Research, Osaka University[*1]

Honda Research Institute Japan Co., Ltd.[*2]

rtakeda@sanken.osaka-u.ac.jp, nakadai@jp.honda-ri.com, komatani@sanken.osaka-u.ac.jp

Abstract

本研究では，Deep Neural Network (DNN) に基づく音響モデルの省メモリ化と高速化のため，パラメータを数ビットに量子化した DNN の構築を目指す．それには，1) 量子化に伴う認識誤りの低減と，2) 高速演算が可能な実装方法の開発，が必要である．1) に対しては，DNN の重みパラメータの正規化を，層単位ではなくノード単位に行う有界重みモデルに基づく学習アルゴリズムを提案する．2) に対しては，量子化した重みパラメータを索引に用いるルックアップテーブルを用いた実装方法を提案する．これらにより，少ないビット数でのパラメータ表現が可能となり，また，複数変数の高速な積和演算が実現できる．評価実験により，単語正解精度の低下を抑えて重みパラメータを 2-bit まで量子化でき，DNN のフォワード計算を 40% 高速化できることを確認した．

1 はじめに

1.1 背景

Deep neural network (DNN) は，高い識別精度により Gaussian mixture model (GMM) に代わって音声認識の音響モデルに広く使われている [1, 2, 3, 4]．一方，その膨大なパラメータ数により，使用メモリ量と計算コストは高く，DNN を適用可能な計算機はまだ限られている．GPU や分散処理を用いた実装 [5, 6] は，DNN の処理速度向上に効果的だが，リソースが制限されたシステムには適用が難しい．例えば，通常の CPU を備えた小型計算機や組み込みシステムなどがある．それゆえ，計算コストとメモリ量の観点で**軽量な DNN** が必要である．

量子化 DNN は，パラメータを量子化することで，GPU や分散処理がなくとも合理的な計算コストとメモリ量で DNN 計算を可能とする．Fixed-point DNN は，重みやバイアスパラメータ，中間層の入力変数を n ビット固定小数点で表現しており，Very Large Scale Integration(VLSI) 実装や CPU 上の Supplemental Streaming SIMD Extensions 3 (SSSE3) 命令を用いた実装により，DNN の高速処理を実現した [7, 8]．Fixed-point DNN のパラメータは，n ビットへの量子化と誤差逆伝播法を繰り返すことで学習される [7, 9]．しかし，最適な量子化ビット数 n の選択は経験的であり，多くの実験を必要とする．また，上記の実装は CPU の特殊命令セットや特殊なデバイスを活用している．CPU のみを搭載した小型計算機上や CPU を伴った FPGA への実装では，特殊な命令セットを前提としない DNN の方が適用しやすい[1]．

ルックアップテーブル（LUT）は DNN の高速計算に有効な方法の 1 つであり，重みパラメータからなる値を LUT の索引として用いる．例えば，一般的な CPU 上でもキャッシュメモリを活用することで，LUT を用いた DNN は SSSE3 命令と同等の高速処理が可能である．もちろん，LUT 技術はハードウェア実装にも適用できる．我々は，これまで重み有界モデルと境界収縮に基づいた学習アルゴリズムを提案していた [10]．このモデルでは，各層の重みの値はある範囲内に制限されており（層単位の有界重みモデル），層単位で重みを正規化する機能を持つ．学習された重みは一様分布またはベルヌーイ分布に従っているため，学習の後に一度だけ量子化するだけでよい．また，大語彙音声認識実験により，単語正解精度を落とすことなく，パラメータを 4 ビットまで量子化可能なことを確認した．一方，4 ビット量子化における致命的な問題は，高速計算のために必要な巨大な LUT サイズにある．その LUT サイズは 32M バイトを超えており，CPU のキャッシュメモリを有効活用できず，高速計算のネックとなっていた．そのため，より少ないビット数，たとえば 2 ビットで量子化

[1] http://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

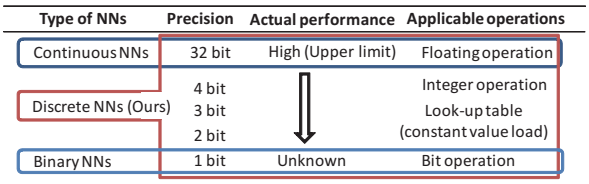

図 1: Properties of neural networks

した DNN が，高速計算に実現に必要である．

本稿では，2 ビット量子化 DNN を達成するため，新たにノード単位の有界重みモデルに基づく学習アルゴリズムと計算機上での実装手法を提案する．2 ビット量子化のキーは，従来研究では層単位で正規化をして量子化を行っていた点にある．実際，各ノードで重みの分布とダイナミックレンジが異なるので，層単位の正規化は量子化誤差を増加する．それゆえ，各ノードに適した重みの範囲で正規化することで，量子化誤差の低減が期待される．ノード単位の有界重みモデルに基づた学習アルゴリズムにより，上記の正規化に適した重みパラメータを学習できる．また，本稿では，LUT に適した重みのエンコード・デコード方法を 2 種類提案するが，そのうち 1 つは LUT サイズを大きく削減できる．提案した学習アルゴリズムと実装方法を，大語彙音声認識実験および DNN フォワード計算の real-time factor (RTF) で検証する．

量子化 DNN の利点としては，量子化ビット数を調整することで，必要な性能で多くのタスクに適応可能な点が挙げられる（図 1）．量子化 DNN は，バイナリ DNN と通常の（連続）DNN の中間的な位置付けである [11, 12]．バイナリ DNN では，すべてのパラメータがバイナリ $\{0, 1\}$ で表現されており，最小の量子化 DNN と言える．一方，著者の知る限り，音声認識でのパフォーマンスは評価されていない．Wang らは，DNN の重みパラメータをベクトル量子化し，軽量な音響モデルを構築している [13] ．音声認識精度を維持し，使用メモリサイズを 90% 削減しているが，実際の処理速度の改善は CPU キャッシュのために課題とされている．

1.2 関連研究

音声認識や画像処理分野では，軽量な DNN を達成可能な量子化以外の方法が提案されており，1) 重み行列の低ランク近似，2) ノード削減，3) 特殊なネットワーク構造，および，4) ノード削除と量子化の組み合わせ，に基づく手法がある．これらの手法は，使用メモリ量の高い削減率と高速な DNN 計算を実現している．主として，標準的な学習手順で構築した DNN に対してパラメータ圧縮処理を施し，圧縮後の DNN パラメータを再学習する流れとなっている．これらの DNN を比較する場合，同じタスクの条件下でメモリ量や速度の相対および絶対的な改善量を比較する必要がある．特に，圧縮前の DNN サイズには注意が必要で，元の DNN パラメータ（例えば，ノード数）が冗長だと，相対的な圧縮率は高くなるからである．

最初の手法は，特異値分解（Singular value decomposition; SVD）[14] や行列分解 [15] を用いて重みパラメータ数を削減する方法で，音声認識実験で評価されている．Xue らは，SVD を中間層の重み行列に適用することで，認識精度の低下なく，使用メモリ量を 80% 削減している [14]．削減率は，重み行列を近似する際に用いる特異値の数に依存し，256 や 196 などが使われている．次の方法は，ノード削減 [16, 17] や模倣学習 [18] であり，音声認識タスクで評価されている．He らは，重みの値に基づいてノードの重要度を定義し，不要なノードを削除することで，中間層のノード数を約 62% 削減している [16]．Li らは，元の DNN を模倣する小さな DNN を学習する方法を提案しているが，精度向上にはより多くの学習データが必要となる [18]．3 番目の方法は，特殊なネットワーク構造に基づくコンパクトな DNN[19] である．彼らは Toeplitz や Vandermonde 行列といった，特殊な行列を重み行列の表現として用いている．これらの行列構造は，行列・ベクトル積といった線形演算の高速化を可能にしている．最後の方法は，ノード削減と量子化の組み合わせにより，Convolutional NN (CNN) と DNN の高い圧縮率を達成しており，画像認識タスクで評価している [20]．彼らは，4 または 5 ビットで量子化しており，また，SVD による圧縮は量子化とノード削減と比べて効果的でないことを示している．

効率的な量子化方法の追求は，他の手法と組み合わせる場合でも，軽量な DNN を達成する上で重要であり，Han らによる実験でも示唆されている [20]．もちろん，ノード削除や DNN の構造も重要であるが，ビットとノードに関する冗長性の性質はやや異なる．例えば，ノード数を最適化した後でも，パラメータの量子化ビット数はある程度の最適化は可能である．本稿では，提案する DNN 量子化の効果を明らかにするため，同じ音声認識タスクにおいて SVD およびノード削減手法との性能も比較する．

2 層単位の有界重みモデルに基づく量子化 DNN

本章では最初に，層単位の正規化をほどこした量子化 DNN のフォワードモデルについて説明する．次に，有界重みモデルと収縮写像に基づくパラメータ学習を説明する．最後に，より小さいビット数へ量子化する際の問題点を説明する．図 2 に，量子化 DNN の学習手順の概要を示す．

2.1 フォワードモデル

DNN のフォワード計算は層 l に関して再帰的に定義される．入力ベクトル $\mathbf{x}_l = [x_{l,1}, ..., x_{l,N_l}]^T \in \mathbb{R}^{N_l}$ はアフィン変換され，活性化関数 $\mathbf{h}_l : \mathbb{R}^{M_l} \to \mathbb{R}^{N_{l+1}}$ が適用される．

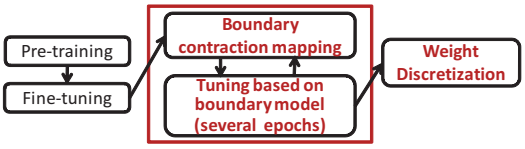

図 2: Training process of discrete NNs

ここで，$\cdot^T$ は転置を，N_l と M_l は l 番目の層への入力ベクトル $\mathbf{x}_l$ と中間ベクトル $\mathbf{z}_l = [z_{l,1}, ..., z_{l,M_l}]^T$ の次元を表す．最終層 L の出力は，初期入力ベクトル $\mathbf{x}_0$ が与えられた元で，層 $l = 0, ..., L-1$ に関する再帰的な計算で得られる．

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

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

ここで，行列 $\mathbf{W}_l = (w_{l,ij}) \in \mathbb{R}^{M_l \times N_l}$ とベクトル $\mathbf{b}_l = [b_{l,1}, ..., b_{l,M_l}]^T \in \mathbb{R}^{M_l}$ は，第 l 層目の重みとバイアスパラメータである．シグモイド関数とソフトマックス関数が活性化関数 $\mathbf{h}_l$ として用いられる．音響モデルにおいては，入力ベクトル $\mathbf{x}_0$ が音声特徴量，出力ベクトル $\mathbf{x}_L$ が Hidden Markov Model (HMM) の状態に関する音響尤度に対応する [3].

量子化 DNN のフォワード計算は，DNN の重みと中間入力ベクトルが数ビットに量子化された状態で定義される．**この式は，学習時ではなく，認識時のフォーワード計算に用いられる．**第 l 層の中間ベクトル $\mathbf{z}_l$ の i 番目の要素は，層単位の正規化に基づき，次のように計算される．

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

ここで，$\mathcal{Q}_y$ と $\mathcal{Q}_x$ は，正規化された重み $\mathbf{Y}_l = (y_{l,ij}) \in \mathbb{R}^{M_l \times N_l}$ (i.e. $y_{l,ij} = w_{l,ij} / \max_{i,j} |w_{l,ij}|$) と第 l 層の入力ベクトル $x_{l,j}$ をバイナリや整数にエンコードする関数である．$\mathcal{F}(a, b)$ は，2 つのバイナリ値 a と b のデコードと積演算を行う関数である．α_l は第 l 層の重みの正規化パラメータであり，$\mathbf{Y}_l$, $\mathcal{Q}_y$, $\mathcal{Q}_x$ 及び $\mathcal{F}$ の定義に依存する．

式 (3) における複数の $\mathcal{F}$ と加算演算は，目的の計算機の構造に応じていくつか実現方法がある．例えば，$\mathcal{Q}_y, \mathcal{Q}_x$ が n ビット整数に線形量子化する演算，$\mathcal{F}$ が単純な整数積演算であるとする．もし，SSSE3 のような特殊命令を用いると，8 変数の積和は 3 命令で実現できる．他にも LUT を用いれば，重みと入力ベクトルを結合した索引によって，予め計算された複数変数の積和結果を読み出すだけでよい．

2.2 パラメータ学習

量子化 DNN の学習は，連続領域で次の 2 ステップを何回か繰り返して行われる．1) 逆誤差伝播法 [21]，2) 重み境界の収縮．これ以降，登場するパラメータの初期値は，すでにファインチューニングされた DNN のものだと仮定する．

量子化に適したパラメータを学習するために，式 (3) に類似した重みの制約を導入する．潜在的なパラメータ行列 $\mathbf{V}_l = (v_{l,ij}) \in \mathbb{R}^{M_l \times N_l}$ を用いて，フォワード計算を次のように表現する．

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

ここで，$\mathbf{g}_l(\mathbf{x}) = (g_l(x_{ij}))$ は，$\tanh(x)$ のような値域が $[-1, 1]$ であるような，要素毎の有界な関数行列である．また，$\tanh(x)$ の導関数は $1 - \tanh^2(x)$ であり，$y_{l,ij} = g_l(v_{l,ij})$ を満たす．

パラメータは通常の NN の学習と同様に，コスト関数 E と教師信号 $\mathbf{r} = [r_1, ..., r_{N_L}]^T \in \mathbb{R}^{N_L}$ に基づく逆誤差伝播法に基づいて最適化される．本稿では，E としてクロスエントロピーを使う．初期誤差ベクトル $\boldsymbol{\epsilon}_L = \left(\frac{\partial E}{\partial \mathbf{x}}(\mathbf{r}, \mathbf{x}_L)\right)$ を計算した後，次のように $l = L-1, ..., 0$ に対する各パラメータを更新する:

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

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

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

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

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

ここで，$\circ$ は行列の要素毎の積を表す演算子であり，η は学習の速度と精度を制御する学習係数である．$\boldsymbol{\epsilon}_l$ は逆誤差伝播のための第 l 層における誤差ベクトルである．なお，我々の有界重みモデル [10] は，Kim らが議論したバイナリ NN の学習モデルの一部と類似している [12].

正規化パラメータ α_l を小さくするため，適当なエポック毎に次の境界収縮写像を行う．これは過去の実験から得られた次の事実によって導入している．1) 小さな α_l は式 (3) の量子化誤差を減少させる，2) 学習後の重み $w_{l,ij}$ の分布が非ガウス分布，特に線形量子化 $\mathcal{Q}_y$ に有効に働くベルヌーイ分布に近くなる．

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

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

ここで，$\mathbf{W}_l = \alpha_l \mathbf{g}_l(\mathbf{V}_l)$ である．g に tanh を用いるので，この操作で α_l は単調に減少する．

2.3 問題点

層単位の有界重みモデルに基づく学習は，音声認識精度を下げることなく 4 ビットの量子化 DNN まで実現できた．しかし，LUT を用いたフォワード計算の高速化は非現実的であった．なぜなら，4 ビット量子化では LUT に

必要なメモリ量が 32M バイトとなり，一般的な CPU のキャッシュメモリサイズを大きく超えているからである．キャッシュメモリを活用するには，より小さなビット数での量子化が不可欠である．

実際の重みのダイナミックレンジと分布はノード毎に異なっている．そのため，層単位で重みを正規化すると情報損失が大きくなり，小さなビット数では量子化誤差が増大する．その改善には，学習と認識において，ノード毎に異なる重みのダイナミックレンジを考慮する必要がある．

3 ノード単位の有界重みモデルに基づく量子化 DNN

本研究では，重みのダイナミックレンジと分布を均一化可能な，ノード単位の有界重みモデルに基づく学習アルゴリズムと実装方法を提案する．はじめに，フォワード計算におけるノード単位での重みの正規化と，学習のためのノード単位での有界重みモデルを説明する．次に，任意の計算機構造で適用可能な LUT を用いた実装方法に関しても議論する．最後に，量子化対象の層に関して議論を行う．

3.1 フォワードモデルとパラメータ学習

本研究では，層単位ではなく，ノード単位で重みのダイナミックレンジをより適切に設定する．そのため，α_l の代わりに，ノード毎に異なる正規化パラメータ $\boldsymbol{\lambda}_l = [\lambda_{l,1}, ..., \lambda_{l,M_l}]^T$ を導入する．ノード単位の正規化は次式で表現される．

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

正規化パラメータの対角行列表記 $\boldsymbol{\Lambda}_l = \mathrm{diag}(\lambda_{l,1}, ..., \lambda_{l,M_l})$ を用いることで，ノード単位の有界モデルは次のように表現できる．

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

各パラメータは確率勾配に基づき逆誤差伝播法で学習される．新たに導入したパラメータに関する更新則は次のように導出される．

$$\boldsymbol{\delta}_l = \left(\frac{\partial \mathbf{h}_l^T}{\partial \mathbf{z}}(\mathbf{z}_l)\right) \boldsymbol{\epsilon}_{l+1}, \quad (14)$$

$$\boldsymbol{\epsilon}_l = \boldsymbol{\Lambda}\mathbf{g}(\mathbf{V}_l)^T \boldsymbol{\delta}_l, \quad (15)$$

$$\boldsymbol{\lambda}_l \leftarrow \boldsymbol{\lambda}_l - \eta \boldsymbol{\delta}_l \circ (\mathbf{g}_l(\mathbf{V}_l)\mathbf{x}_l), \quad (16)$$

$$\mathbf{V}_l \leftarrow \mathbf{V}_l - \eta \frac{\partial \mathbf{g}_l}{\partial \mathbf{V}}(\mathbf{V}_l) \circ \left(\boldsymbol{\Lambda}_l \boldsymbol{\delta}_l \mathbf{x}_l^T\right). \quad (17)$$

以上の式より，式 (7) では 1 つの α_l に集約していた伝播誤差ベクトル δ_l が，それぞれの正規化パラメータに影響していることがわかる．$\mathbf{W}_l = \boldsymbol{\Lambda}_l \mathbf{g}_l(\mathbf{V}_l)$ における境界収縮は同様に修正され，

$$\lambda_{l,i} \leftarrow \max_j |w_{l,ij}|, \quad (18)$$

$$v_{l,ij} \leftarrow w_{l,ij} / \max_j |w_{l,ij}|. \quad (19)$$

となる．

3.2 ルックアップテーブルを用いた実装方法

LUT の実装方法は，エンコード関数 $\mathcal{Q}_x$ がバイナリ値 $\{0, 1\}$ を出力するか否かで 2 種類に分けられる．初めに，一般的なモデルで LUT のエンコード・デコード方法を説明する．そのあと，特別なバイナリモデルの場合について議論する．

3.2.1 一般モデル

説明のため，まず，重みの行ベクトルと入力ベクトルをいくつかのグループに分け，各グループで D 個の要素を持っていると仮定する．それらを用いたバイナリコードをそれぞれ次のように記述する．

$$\bar{\mathbf{y}}_{(l,i,k)} = \{\mathcal{Q}_y[y_{l,ij}]\}_{j=Dk}^{D(k+1)-1}, \quad (20)$$

$$\bar{\mathbf{x}}_{(l,k)} = \{\mathcal{Q}_x[x_{l,i}]\}_{i=Dk}^{D(k+1)-1}. \quad (21)$$

LUT，T，は，以上のバイナリコードの組み合わせで参照される．例えば，$\bar{\mathbf{y}}_{(l,i,k)} = [110, 101]$ と $\bar{\mathbf{x}}_{(l,k)} = [010, 101]$ の場合，LUT のインデックスはバイナリ表現で [110101010101] となる．式 (12) から，量子化後のフォワード計算は次のように表現できる．

$$z_{l,i} = \lambda_{l,i} \sum_{k=0}^{N_l/D} T[\bar{\mathbf{y}}_{(l,i,k)}\bar{\mathbf{x}}_{(l,k)}] + b_{l,i}. \quad (22)$$

加算回数は明らかに $1/D$ に削減されている．

LUT は事前にすべてのバイナリコードのパターンを予め計算することで構築する．$\bar{\mathbf{a}}, \bar{\mathbf{b}}$ のすべてのパターン, 例えば, 0 から $2^n - 1$ を具体的に列挙して構築する．

$$T[\bar{\mathbf{a}}\bar{\mathbf{b}}] = \sum_{j=0}^{D-1} \mathcal{F}(a_j, b_j) = \sum_{j=0}^{D-1} \mathcal{Q}_y^{-1}[a_j]\mathcal{Q}_x^{-1}[b_j] \quad (23)$$

ここで，a_j と b_j は $\bar{\mathbf{a}}$ と $\bar{\mathbf{b}}$ に対応する第 j 要素を表現する．$\mathcal{Q}_y^{-1}$ と $\mathcal{Q}_x^{-1}$ は，$\mathcal{Q}_y$ と $\mathcal{Q}_x$ に対応するデコード関数であり，バイナリパターンの連続値表現を返す.

$g_l(x) = \tanh(x)$ で，$h_l(x) = 1/(1 + \exp(-x))$ である場合には, n ビットの各デコード・エンコード演算は次のように定義される．

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

$$\mathcal{Q}_y[y] = \mathrm{floor}[(2^n - 1)(y + 1)/2 + 0.5], \quad (25)$$

$$\mathcal{Q}_x^{-1}[x] = x/(2^n - 1), \quad (26)$$

$$\mathcal{Q}_y^{-1}[y] = 2y/(2^n - 1) - 1. \quad (27)$$

なお，本稿では LUT 自体の最適化は扱わない．

表 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}

3.2.2 バイナリモデル

もし，$\mathbf{x}_l$ にバイナリ量子化を採用するなら，テーブルサイズはビットマスク演算により半分にできる．例えば，$\bar{\mathbf{y}}_{(l,i,k)} = [110, 101]$ と $\bar{\mathbf{x}}_{(l,k)} = [0, 1]$ の場合，T のインデックスは通常は [11010101] となる．デコード関数が $Q_y^{-1}[000] = 0$, $Q_x^{-1}[0] = 0$ かつ $Q_x^{-1}[1] = 1$ を満たす限り，$\bar{\mathbf{y}}_{(l,i,k)}$ のビットを $\bar{\mathbf{x}}_{(l,k)}$ でマスクすることで，バイナリモデルの LUT, T_b, のインデックスは [000101] とできる．これは次式により理解できる．

$$\begin{aligned} T[11010101] &= Q_y^{-1}[110]Q_x^{-1}[0] + Q_y^{-1}[101]Q_x^{-1}[1] && (28)\\ &= 0 + Q_y^{-1}[101] && (29)\\ &= Q_y^{-1}[000] + Q_y^{-1}[101] && (30)\\ &= T_b[000101]. && (31) \end{aligned}$$

なお，このバイナリモデルにおける $\mathcal{Q}_y$ と $\mathcal{Q}_y^{-1}$ の定義は，式 (25) と (27) から適切に変更する．使用メモリ量は削減できるが，量子化誤差の増加と音声認識率の低下を招く可能性があるため，その効果は実験的に確認する必要がある．

3.2.3 使用メモリ量の比較

表 1 に，各実装方法における DNN 重みと LUT のメモリ使用量を示す．*32-bit float* と *8-bit SSSE* は，通常の浮動小数点の積和演算と SSSE3 命令セットによる方法で，LUT 自体を必要としない．*3-bit-bin (binary LUT)* はバイナリモデルの LUT を意味し，その他の項目は一般モデルの LUT の結果を示している．*32-bit float* では $4N_lM_l$ のメモリが重みに必要であるが，他の方法ではそれの 1/8 以下のメモリ量で済む．なお，LUT のサイズはパラメータ D にも依存する．全体のメモリ使用量は，実際の CPU キャッシュサイズを考えると，1, 2 M バイト以下が理想的である．

3.3 量子化対象の層の選択

本研究では，量子化したフォワード計算はすべての層に適用しない．というのは，少なくとも入力層への入力は $[0, 1]$ の範囲に収まっていないからである [8, 10]．また，重みを量子化する層を選択することで，音声認識性能を改善できる可能性はある．以前の報告では，入力層を除いたすべての層の重みを量子化していた．最終層の出力は識別に用いられるので，本稿では最終層の重みも量子化を行わず浮動小数点で表現する．

表 2: Configuration

Item	Value
Audio data	16 bits, 16 kHz sampling
STFT analysis	hamming window: 25 ms, shift: 10 ms
Features for GMM	MFCC 39 dim. [13+ Δ 13 + ΔΔ 13]
Features for DNN	FBANK 825 dim. [(25+ Δ 25 + ΔΔ 25) × 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 × 825 middle layer: 1024 × 1024 output layer: 4000 × 1024
Training set	clean speech 223 hours (799 males and 168 females)
Test set	clean speech 3.5 hours (15 males and 5 females)

4 評価実験

4.1 実験設定

日本語話し言葉コーパス（CSJ）[22] を用いた連続音声認識実験により，ノード単位の有界重みモデルに基づく音響モデルを評価した．まず，n ビット量子化におけるノード単位・層単位モデルの単語正解精度（WA）を比較する．この時，学習フェーズ (Eq.(4) or Eq.(13)) と，認識フェーズ (Eq.(3) or Eq.(12)) のそれぞれにおいて，ノード単位・層単位のモデルの各組み合わせにおける正解精度を調査する．正規化した重みの平均量子化誤差，$|y - \mathcal{Q}_y^{-1}[\mathcal{Q}_y[y]]|$ と，ノード単位の重み統計量として正規化尖度 $\mathbb{E}[(x - \mathbb{E}[x])^4]/\mathbb{E}[(x - \mathbb{E}[x])^2]^2 - 3$ に関しても議論する．$\mathbb{E}$ は期待値演算を表す．LUT の一般・バイナリモデルの各実装におけるフォワード計算のリアルタイムファクタ (RTF) も比較した．RTF は次式で定義される．

$$\text{RTF} = (\text{processing time})/(\text{data duration}). \qquad (32)$$

ノード削減 DNN [16] と SVD-DNN[14] の WA，RTF およびメモリ使用量も調査した．

DNN 音響モデルの学習用データとして，約 223 時間分の学術講演音声を用いた．評価用データは，CSJ テストセット 1 および 2 の約 3.5 時間分，男性 15 名・女性 5 名分の音声である．言語モデルの学習データは，評価用データを除いた CSJ に含まれるすべての書き起こしテキストを用いた．語彙サイズは 65000 であり，トライグラム言語モデルを用いた．音声認識器は Julius (ver. 4.3.1) [23] を用い，言語モデル重みと挿入ペナルティはデフォルト値である 8 と −2 に設定した．また，ビーム幅は 4000 にした．

表 3: Word accuracy (WA) vs. quantization bits for clean speech task

		Applied model		Discrete layer l	Word accuracy (%)							
		Training	Recog.		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	
	P4	re-train. (2-bit quan.)		1-5	–	–	**80.15**	–	–	–	–	

DNN 音響モデルは，GMM-HMM を用いたラベル付与とフレーム単位での識別に基づき学習した．初めに，混合数 32・共有状態数 4000 のトライフォン GMM-HMM を HTK (Hidden Markov Model Toolkit) [2]を用いて学習した．音声特徴量は，13 次元の Mel-frequency cepstral coefficients (MFCCs)，その 1 次差分および 2 次差分の計 39 次元である．音声データのサンプリング周波数は 16kHz であり，窓長 25 ミリ秒・シフト長 10 ミリ秒で短時間フーリエ解析を行った．また，発話単位毎に平均・分散正規化を行っている．DNN-HMM は，GMM-HMM と同じ HMM パラメータを利用し，DNN の中間層のノード数は 1024 で，$L = 7$ とした．出力ノードの次元は，HMM の状態を識別するため 4000 に設定した．DNN への入力特徴量は，基本特徴量における中心フレームと前後 5 フレームを連結した計 11 フレームの 825 次元となっている．その基本特徴量は，25 次元の対数メルフィルタバンク係数と 1 次差分，2 次差分で構成される．平均・分散正規化を同様に適用した．これらの設定を表 2 にまとめる．最後に，GMM-HMM に基づくビタビアライメントにより得られた状態ラベルを用いて，DNN パラメータを学習した．プレトレーニングは段階的に識別学習を行う方法を用い [3]，学習係数の半自動調整を行うため AdaGrad [24] を用いた．ミニバッチサイズは 64 に設定した．ファインチューニングの後，有界モデルを用いた学習を行った．重みの量子化後，入力層などの量子化を行わなかった層のパラメータの再学習も検討した．ドロップアウト [25] も DNN の絶対的な性能を改善する可能性があるが，Kim らが議論しているように [7]，その有無で手法間の相対的な差には致命的な影響を及ぼさないと考えている．

ノード削減 DNN と SVD-DNN においても，初期値に用いる DNN のパラメータ値，構造，特徴量などは提案法の場合と同様にした．ノード削減 DNN は，出力重みノルム基準 [16] によって行い，中間層のノード削減率は 10, 20, 40, 60, 80% に設定した．SVD-DNN [14] も，SVD は中間層の重み $\mathbf{W}_l$ $(l = 1, ..., 5)$ に対して適用し，用いる特異値数は 32, 64, 128, 256, 512 および 768 に設定した．

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

従来研究と同様，性能改善のため，ノード削除と SVD を適用した後に，DNN パラメータを再学習した．

4.2 実験結果

4.2.1 単語正解精度

表 3 に，異なる量子化ビット数に対する WA を示す．*Normal Training* の行は，有界重みモデルではなく，通常の DNN モデルで学習したパラメータでの認識結果を表す．*Discrete layer* の列は，量子化対象の層を表している．*2-bit-bin* と *3-bit-bin* は，重みは 2・3 ビットに量子化し，中間層への入力ベクトルを 1 ビットに量子化した場合の結果を示す．*2-bit* などの他の表記では，重みと中間層への入力ベクトルの量子化ビット数は同じである．*32-bit float* は量子化していない浮動小数点で表現された重みを意味する．なお，GMM-HMM の単語正解精度は 75.8 % であった．

まず，量子化ビット数と各手法の WA の関係に関して述べる．8 ビット量子化ではすべての手法はほぼ同程度の性能であり，より小さいビット数では性能差が明確になっている．表中の *Normal Training* に着目すると，通常モデルで学習し，認識時に層単位の正規化を行った場合（*Recog.*），4 ビット量子化以下では認識に失敗していることがわかる．認識時にノード単位の正規化を適用するだけで，WA は大きく改善しており，ノード単位の正規化の重要性が示唆される．量子化対象の層を制限すると，3 ビットのようなシビアなビット数においても WA を改善している．これ以降は，層 $l = 1, ..., 5$ における重みに量子化を行った結果を比較する．

次に，ノード単位での有界重みモデルと各 LUT の実装を用いた場合の WA に着目する．ノード単位モデルの WA は，P2 のミスマッチ状況を除き，2・3 ビットの量子化において，ベースラインの WA を最大 1.8 ポイント上回っている．特に，P3 では 2 ビット量子化においても十分に高い精度を保っており，通常の DNN と比較して 2 ポイントしか WA は低下していない．P3 から量子化を行っていない層のパラメータ再学習することで，少し性能は改善している (P4)．*2-bit-bin* および *3-bit-bin* の WA は，中

表 4: Quantization error (QE) vs. quantization bits. Discrete layers were 1-5 in all cases for clean speech task.

		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, P4	node-wise	node-wise	2.92E-01	1.52E-01	7.85E-02	3.44E-02	1.96E-03

表 5: Computer specifications for forward calculation

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

間層への入力ベクトルを $\{0, 1\}$ に量子化したにも関わらず 50% 以上となった．*2-bit-bin* では，層単位モデルから約 16 ポイント WA が改善しており，ノード単位モデルに基づく学習の優位性を示している．しかし，1 ビットに重みを量子化する（*1-bit*）と全く認識されず，2 ビットと 1 ビット量子化の間に大きな隔たりがある．1 ビット量子化 DNN を実現するには，より精度を保てるパラメータ学習と量子化方法が必要となる．

4.2.2 量子化誤差および重み分布

量子化誤差 (QE) と WA の関係を理解するため，各ビット数における平均量子化誤差を表 4 に示す．normal training と baseline (P0) の QE は，認識時でノード単位の正規化を行うことで減少している．しかし，低い量子化誤差が必ずしも高い WA に繋がっているとは限らない．例えば，P1 における QE は P0 と比較して改善しているが，それらの WA は同じ程度である．

次に，通常学習，層単位およびノード単位の有界重みモデルに基づく学習の違いを分析するため，重みに対するノード単位の尖度の分布に注目 s する．ガウス分布，一様分布，ベルヌーイ分布の尖度はそれぞれ 0, -1 および -2 である．図 3 に，学習後における normal training，P0 (baseline, layer-wise) および P3 (node-wise) の尖度の分布を示す．normal training の尖度は 0 以上であるため，重みはガウス分布やよりスパースな分布をしていることがわかる．一方，baseline と提案モデルの重み分布はベルヌーイ分布に近い．というのは，ほとんどの尖度が -1 よりも小さいからである．また，baseline の尖度分布には提案モデルの方にはない複数のピークが見られる．この内，比較的尖度が大きいピークは，量子化 DNN の中間出力における誤差を増大させ，その誤差の範囲や分布も各ノードで異なると考えられる．このような誤差が途中で打ち消し合うこともなく蓄積していき，HMM や言語モデルで平滑化可能な範囲を超え，層単位での正規化方法の認識精度低下を招いたと推察される．ノード単位の有界重みモデルに基づく学習は，各ノードの重みの分布を均一にし，連続値と量子化した値の隔たりを埋め，そのような状況を避けられたと考えられる．

最後に，層単位およびノード単位モデルの実際の重み分布を図 4 に示す．通常学習 (normal training) の重み分布はガウス分布に似ているが，有界モデルを用いた学習された重みの分布はベルヌーイ分布に近くなっている．-1 と 1 付近にある 2 つのピークは，正規化パラメータ $\alpha_l, \lambda_{l,i}$ が小さくなるにつれて，外側へ移動する傾向にある．それゆえ，それらは分布の形状を制御していると考えられる．

4.2.3 LUT を用いたフォワード計算の RTF

LUT における一般的モデル（*general model*）とバイナリモデル（*binary model*）の 2 つの実装方法を用いて，量子化 DNN のフォワード計算の速度向上率を明らかにする．また，重みと LUT の使用メモリ量も同時に示す．計算機のスペックは表 5 に示す通りである．CPU のキャッシュサイズと周波数は LUT を用いた計算に十分な値である．

RTF と LUT・重みに必要なメモリ量の関係を表 6 に示す．表中では 2 つの RTF が示されているおり，1 つは中間層 ($l = 1, ..., 5$) に対する RTF(partial)，もう一つは DNN 処理全体 ($l = 0, ..., 6$) に対する RTF(all) である．これは，量子化が中間層の重みに対して適用されており，実質的な速度改善はこれらの層（$l = 1, ..., 5$）に限られるからである．*32-bit float* は，LUT や特殊命令セットを用いない通常の積和実装を表し，*8-bit SSSE* は [8] で提案された SSSE 命令セットを用いた実装である．*baseline* は 4 ビットの量子化 DNN の結果であり，層単位モデルの限界値である．表中の D は，式 (22) において同時に計算される変数の数を意味する．

$D = 3, 4$ において，一般・バイナリの両実装モデルに対する RTF(all) は，baseline と比較して約 30〜40% の処理速度向上を達成し，SSSE 命令を用いた実装に近い処理速度となっている．さらに，3-bit-bin で $D = 7$ の RTF は，WA は十分ではないものの，SSSE と同等の処理速度となっている．しかし，3 ビットの量子化では 4M バイトの LUT を必要とする．もし，2 ビット量子化かつバイナリモデルを適用し，$D = 8$ に設定すると，SSSE と同等

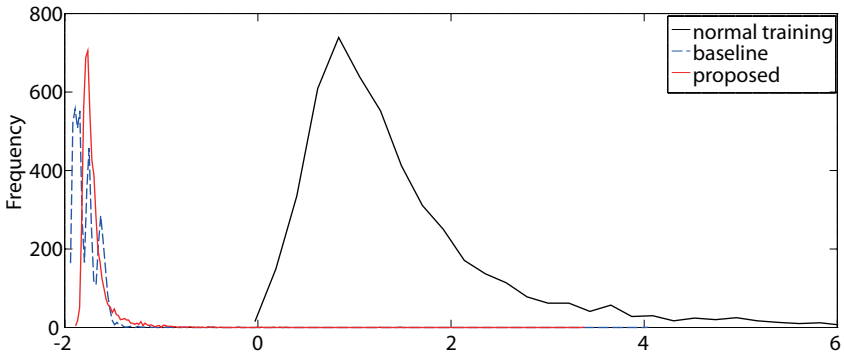

図 3: Kurtosis distributions

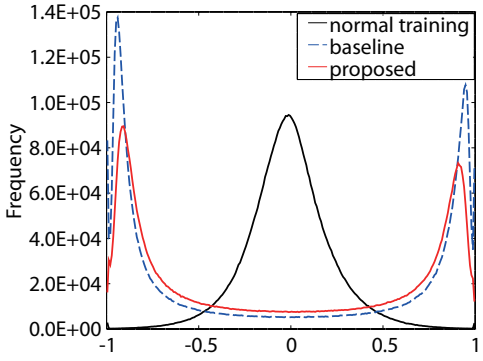

図 4: Weight distributions

表 6: Real-time factor of forward calculation and memory usage on single CPU

	Methods	RTF (partial) ($l = 1, ..., 5$)	RTF (all) ($l = 0, ..., 6$)	D	Memory usage (bytes) LUT	Memory usage (bytes) total weights ($l = 1, ..., 5$)
Standard	32-bit float	**0.435**	**0.841**	1	–	20 M
Baseline	**4-bit (general LUT)**	0.653	**1.060**	3	**32 M**	2.5 M
	4-bit (general LUT)	0.220	**0.641**	2	**128 K**	2.5 M
Proposed	3-bit (general LUT)	0.255	0.661	3	512 K	1.85 M
	3-bit-bin (binary LUT)	0.096	0.501	7	4 M	1.85 M
	2-bit (general LUT)	**0.111**	**0.517**	4	**128 K**	**1.25 M**
	1-bit (general LUT)	0.084	**0.489**	8	**128 K**	**640 K**
Special	8-bit SSSE	0.092	**0.498**	8	–	5 M

の処理速度を達成しつつ，LUT サイズは 512K バイトに抑えられる．1 ビット量子化 DNN は 8-bit SSSE の処理速度を上回っているので，その WA の改善が期待される．中間層だけの RTF(partial) に注目すると，DNN の量子化ビット数に比例して，RTF が改善していることがわかる．例えば，2 ビット実装の RTF(partial) は，32 ビット実装から約 75% 処理速度の向上がみられる．ハードウェア上での論理回路を用いた実装では，さらなる RTF の改善が期待される

4.2.4 ノード削減 DNN と SVD-DNN との比較

表 7 と表 8 に，ノード削減 DNN と SVD-DNN の WA，メモリ使用量および RTF を示す．これらの表でも 2 種類の RTF が示されており，1 つは中間層 ($l = 1, ..., 5$) に対する RTF(partial)，もう一つは DNN 処理全体 ($l = 0, ..., 6$) での RTF(all) である．再学習を行わない場合の WA は，すべての設定において元の DNN よりも低くなっている．SVD-DNN での性能低下はノード削減 DNN よりも深刻でない．これは，SVD は元の重み行列の値を保つような次元圧縮方法だからである．多くの場合，再学習の後で WA は 81% まで改善している．

まず，各手法の使用メモリ量と WA の関係について比較する．量子化 DNN では約 1.4M バイトの使用メモリ量で WA 80% を達成していたが，ノード削減 DNN および SVD-DNN では，3M のような少ない使用メモリ量の場合，WA は 80% 未満となっている．量子化 DNN と同等の使用メモリ量および RTF(partial) を達成するには，さらにノード削減率を高くする必要があり，使用する特異値数も極端に減らす必要がある．例えば，これまで報告されているような 80% の圧縮率を達成するためには，ノード削減率では 60% 程度必要であり，特異値数では 64 に設定する必要がある．しかし，それらの設定での WA は 80% を下回っている．この結果は，本研究での用いた中間層におけるノード数が他の報告で用いられている数，例えば 2048，よりも相対的に小さい点が一因している．以上より，使用メモリ量の削減において，量子化の効果は大きいと言える．

次に，各手法の RTF に注目する．ノード削減法は，通常多くのノードを含む出力層の重み行列の次元を大きく削減可能なので，RTF(all) は量子化 DNN と比較すると大きく改善している．量子化ビット数は，DNN のノード数を最適化した後でも，最適化するとは可能であるため，ノード削除と量子化の組み合わせは，DNN の構造を改善する効率的な戦略であるといえる [20]．そのような場合でも，量子化の方法は重みの圧縮率と認識性能に影響を与えるため，量子化アルゴリズムの改善は不可欠である．

5 議論

実験結果から，WA，量子化誤差および重みの統計量に関して，1) 量子化誤差と WA の関係と 2) 重みの正規化の効果に関する事実を得た．前者は，DNN のパラメータ学習の戦略に影響を与える．QE は必ずしも直接的に WA を改善するわけではないので，パラメータ学習のコストに

表 7: The Performance of Node-pruning: word accuracy, memory usage, and RTF

Pruning-ratio	Word Accuracy (%) w/o retrain	Word Accuracy (%) w/ retrain	Memory usage (bytes) ($l = 1, ..., 5$)	RTF (partial) ($l = 1, ..., 5$)	RTF (all) ($l = 0, ..., 6$)
80%	0	74.48	0.8 M	0.018	0.101
60%	0	78.29	3.2 M	0.070	0.233
40%	0.56	**80.28**	**7.2 M**	**0.155**	**0.397**
20%	1.35	81.29	12.8 M	0.274	0.592
10%	58.06	81.64	16.2 M	0.346	0.702

表 8: The Performance of SVD-DNN: word accuracy, memory usage, and RTF

SVD dim.	Word Accuracy (%) w/o retrain	Word Accuracy (%) w/ retrain	Memory usage (bytes) ($l = 1, ..., 5$)	RTF (partial) ($l = 1, ..., 5$)	RTF (all) ($l = 0, ..., 6$)
32	4.35	48.81	1.25 M	0.034	0.431
64	3.35	79.74	2.5 M	0.056	0.456
128	2.96	**80.52**	**5.0 M**	**0.111**	**0.508**
256	11.41	81.78	10.0 M	0.215	0.611
512	79.36	81.81	20.0 M	0.421	0.817
768	81.62	82.01	30.0 M	0.628	1.024

QE を最小化するような制約を加えるだけでは，効果的でないと考えられる．後者は，量子化 DNN に対する他の重み制約の可能性を示唆している．もし，WA を改善するためにこの事実を直接的に用いるのであれば，ノード単位の重みの尖度の平均を最小化するアプローチもある．LUT の最適化や，重みの非線形量子化なども，1 ビット量子化を含む低ビット量子化 DNN の WA 改善に効果的である．また， sequence training も WA の向上に効果があると推察される．なぜなら，HMM と言語モデルによるフィルタ効果により，量子化誤差の影響は小さくなるからである．量子化後のフレーム単位での状態識別率はとても低い，という事実もこの仮説を支持している．

バイナリモデル LUT に基づく実装も改善の余地があり，その実現可能性は残されている．提案したノード単位のモデルは，中間層への入力ベクトルの量子化を考慮していないため，その影響を学習に組み込むことで性能改善が可能だと考えられる．理想的には，そのベクトルの要素もベルヌーイ分布に従っていることである．そのような分布の形成に適した制約を，発見または開発することが必要である．これは，単に活性化関数で用いられるシグモイド関数などのスケールパラメータを調整するだけで十分な可能性もある．以上の課題を解決すれば，2 ビット重み量子化とバイナリモデル LUT に基づいた実装が実現できる．

最後に，DNN の本質的な「記憶」力も，より進んだ研究を行うために考慮すべきである．例えば，もし，巨大なネットワークを中間層に用いると，従来研究 [9] でも述べられているように，重みの量子化はより簡単になると予想される．彼らの研究では，10 種の数字画像の識別精度を，より多くの中間層のノード数を用いることで改善できたと報告している．そのため，ノード数と重み表現に必要なビット数，必要なメモリ量および演算効率などの関係をより詳細に調べる必要がある．また，雑音や残響音声を含むような，入力データのパターン数が増加する場合，それらを記憶または識別するためのより多くの「記憶容量」が必要である．そのため，ヘテロなデータを扱う場合に，クリーン音声用に設計された DNN 構造を流用すると，少ないビット数での量子化はより難しくなると考えられる．使用メモリ量と計算効率の観点でより一般的に最適化を行うには，入力データと演算の実装方法に合わせた DNN 構造の適応（ノード削減など）と量子化を同時に扱う枠組みが不可欠である．

6 結論

本研究の目的は，省メモリ・低計算コストな音響モデルのための量子化 DNN の開発である．4 ビットの量子化では LUT サイズが巨大となるので，使用メモリ量の観点で 2 ビットでの量子化が必要となった．2 ビット量子化のカギは，重みパラメータのバラつきの正規化方法にあった．本研究では，層単位ではなく，ノード単位の有界重みモデルに基づいてパラメータを学習するアルゴリズムを開発した．このアルゴリズムは，ノード単位で重み分布のバラつきを均一化し，量子化に有効な重みの学習を可能にした．また，量子化 DNN の実装として，メモリ使用量が異なる，一般的な LUT およびバイナリ LUT に基づくの 2 種類の実装方法を提案した．2 ビット量子化 DNN の評価実験では，高い単語正解精度を保ちつつ，DNN のフォワード計算の計算速度を約 40% 向上した．

残された主な課題は，メモリ量と処理速度を改善するために，ノード削減法との組み合わせや中間層への入力ベクトルの 1 ビット量子化である．そのカギとなるアプローチは sequence training，重みの非線形量子化などがあり，これらの技術を用いてより省メモリかつ低計算コストな DNN 構築を目指す予定である．

謝辞

本研究は JSPS 科研費 15K16051 の助成を受けたものです.

参考文献

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

[2] 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.

[3] F. Seide, G. Li, , and X. Chen D. Yu, "Feature engineering in context-dependent deep neural networks for conversational speech transaction," in *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*, 2011, pp. 24–29.

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

[5] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th Annual International Conference on Machine Learning*, 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 Neural Information Processing Systems*, 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 *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2014, pp. 7510–7514.

[8] V. Vanhouche, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," in *Proceedings of the Deep Learning and Unsupervised Feature Learning NIPS Workshop*, 2011, vol. 1.

[9] C. Z. Tang and H. K. Kwang, "Multilayer feedforward neural networks with single powers-of-two weights," *IEEE Transactions 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 *Proceedings of the 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 Neural Information Processing Systems*, 2014.

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

[13] Y. Wang, J. Li, and Y. Gong, "Small-footprint high-performance deep neural network-based speech recognition using split-vq," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2015, pp. 4984–4988.

[14] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition.," in *Proceedings of Interspeech*, 2013, pp. 2365–2369.

[15] T.N Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6655–6659.

[16] T. He, Y. Fan, Y. Qian, T. Tan, and K. Yu, "Reshaping deep neural network for fast decoding by node-pruning," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2014, pp. 245–249.

[17] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, 2015.

[18] J. Li, R. Zhao, J.-T. Huang, and Y. Gong, "Learning small-size DNN with output-distribution-based criteria.," in *Proceedings of Interspeech*, 2014, pp. 1910–1914.

[19] V. Sindhwani, T. Sainath, and S. Kumar, "Structured transforms for small-footprint deep learning," in *Advances in Neural Information Processing Systems*, 2015.

[20] S. Han, H. Mao, and W.J Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *ICLR*, 2015.

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

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

[23] A. Lee and T. Kawahara, "Recent development of open-source speech recognition engine Julius," in *Proceedings of the Asia-Pacific Signal and Information Processing Association, Annual Summit and Conference*, 2009, pp. 131–137.

[24] 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.

[25] 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.