Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Binary neural networks for speech recognition[*#]

Yan-min QIAN[‡1,2], Xu XIANG[1,2]

[1]*Key Laboratory of Shanghai Education Commission for Intelligent Interaction and Cognitive Engineering, Shanghai Jiao Tong University, Shanghai 200240, China*

[2]*SpeechLab, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China*

E-mail: yanminqian@sjtu.edu.cn; chinoiserie@sjtu.edu.cn

Received Aug. 6, 2018; Revision accepted Dec. 23, 2018; Crosschecked May 13, 2019

Abstract: Recently, deep neural networks (DNNs) significantly outperform Gaussian mixture models in acoustic modeling for speech recognition. However, the substantial increase in computational load during the inference stage makes deep models difficult to directly deploy on low-power embedded devices. To alleviate this issue, structure sparseness and low precision fixed-point quantization have been applied widely. In this work, binary neural networks for speech recognition are developed to reduce the computational cost during the inference stage. A fast implementation of binary matrix multiplication is introduced. On modern central processing unit (CPU) and graphics processing unit (GPU) architectures, a 5–7 times speedup compared with full precision floating-point matrix multiplication can be achieved in real applications. Several kinds of binary neural networks and related model optimization algorithms are developed for large vocabulary continuous speech recognition acoustic modeling. In addition, to improve the accuracy of binary models, knowledge distillation from the normal full precision floating-point model to the compressed binary model is explored. Experiments on the standard Switchboard speech recognition task show that the proposed binary neural networks can deliver 3–4 times speedup over the normal full precision deep models. With the knowledge distillation from the normal floating-point models, the binary DNNs or binary convolutional neural networks (CNNs) can restrict the word error rate (WER) degradation to within 15.0%, compared to the normal full precision floating-point DNNs or CNNs, respectively. Particularly for the binary CNN with binarization only on the convolutional layers, the WER degradation is very small and is almost negligible with the proposed approach.

Key words: Speech recognition; Binary neural networks; Binary matrix multiplication; Knowledge distillation; Population count
https://doi.org/10.1631/FITEE.1800469 **CLC number:** TP391.4

1 Introduction

Over the last few years, various kinds of deep neural networks, including deep neural networks (DNNs), convolutional neural networks (CNNs), long-short term memory recurrent neural networks (LSTM-RNNs), and residual networks (ResNets) have achieved tremendous success in acoustic modeling for speech recognition (Dahl et al., 2012; Hinton et al., 2012; Jaitly et al., 2012; Mohamed et al., 2012; Sainath et al., 2013; Sak et al., 2014; Bi et al., 2015; Qian et al., 2016; Yu et al., 2016; Chen et al., 2017, 2018a, 2018b, 2018c; Saon et al., 2017; Xiong et al., 2017). On several large-scale speech recognition benchmarks, these methods can outperform the best Gaussian mixture model-hidden Markov model (GMM-HMM) system by a wide margin. However,

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (Nos. 61603252 and U1736202) and experiments have been carried out on the Pi supercomputer at Shanghai Jiao Tong University

# A preliminary version was presented at the 18th Annual Conference of the International Speech Communication Association, August 20–24, 2017, Sweden

ORCID: Yan-min QIAN, http://orcid.org/0000-0002-0314-3790 these models usually incur excessively high computational cost during the inference stage (mainly due to large matrix multiplication), which makes them infeasible to deploy on low-power embedded devices, such as Internet of Things (IoT) devices without WiFi connection. Hence, there is great interest in speeding up the inference of DNNs while keeping the performance degradation in an acceptable range.

Denil et al. (2013) have demonstrated that there is a significant redundancy in the parameterization of conventional deep learning models, which leads to a waste of computation. Based on this phenomenon, deep models can be aggressively simplified, which can reduce the computational cost and accelerate the model inference. In general, the current work on acoustic modeling can be roughly grouped into two categories: reconstructing the model by exploring the sparseness property and quantizing the model parameters with a low precision. The speedup with advanced hardwares, such as field-programmable gate array (FPGA) and application specific integrated circuit (ASIC), is not considered in this study.

The first approach is to decrease the number of parameters by exploiting the sparseness property in deep models. Xue et al. (2013) applied singular value decomposition (SVD) on the weight matrices of DNNs to restructure the model based on the inherent sparseness of the original matrices. After reconstruction, the new model was re-finetuned to improve the model accuracy. Yu et al. (2012) incorporated a soft regularization aiming at minimizing the number of nonzero elements of weight matrices during DNN training. To save storage and speed up calculation, they proposed a new data structure that can efficiently process the weight matrices with many zero elements. He et al. (2014) and Qian et al. (2015) defined three importance functions on network nodes and pruned the original DNN models according to these measures. Han et al. (2015) explored an iterative process to remove unimportant connections in DNNs based on the weights. Novikov et al. (2015) represented the weight matrices in a multi-linear format, and with this representation, a large weight matrix could be described by much fewer parameters. The other approach is to quantize the model parameters with low precision fixed-point values. Han et al. (2017) proposed an efficient speech recognition engine using an FPGA and explored different data quantization strategies and widths. Experiments showed that, on the TIMIT phone recognition task, a 12-bit width was enough for the weight and activation quantization in an LSTM network with little loss in accuracy. Wang et al. (2015) achieved a promising result for small-footprint speech recognition using the split vector quantization based network quantization. Gupta et al. (2015) studied the effect of low-bit width data representation and computation on neural network training. Using stochastic rounding, deep networks could be trained using only 16-bit fixed-point representation, with little degradation on MNIST and CIFAR10 tasks. Courbariaux et al. (2016), Hubara et al. (2016), and Rastegari et al. (2016) quantized both weights and activations of neural network models into low-bit width values. Zhou et al. (2016) further quantized the gradients and tested different quantization configurations for weights, activations, and gradients.

More recently, the binarization of deep models has attracted interest in image processing, with both binary parameters and activations. Researchers from the image community have shown that binary deep models can obtain significant acceleration during model inference while still having a competitive accuracy (Hubara et al., 2016; Rastegari et al., 2016). Inspired by this attempt on image processing, in this study, binary neural networks are comprehensively explored for acoustic modeling in speech recognition for the first time. With binarization, the new models show superiority in inference speed. The optimization algorithms for binary neural networks are first designed for acoustic modeling, including several types of deep models. Then a knowledge distillation framework using normal full precision floating-point deep models is developed to boost the performance of binary neural networks.

The main contributions of this work are as following: (1) Binary matrix multiplication is implemented and optimized so that it can run 5–7 times faster than highly optimized floating-point baselines on desktop central processing unit (CPU), mobile CPU, and desktop graphics processing unit (GPU); (2) Binary neural networks are comprehensively explored for speech recognition for the first time, while binarization on various types of deep models is developed and compared for acoustic modeling; (3) A knowledge distillation framework is designed for binary deep models to keep the accuracy degradation within a small range.

2 Neural network based acoustic modeling

In speech recognition, the speech signal (observation sequence) is assumed to be generated by a hidden Markov process which is modeled by an HMM. In the HMM with hidden states $S = \{s_1, s_2, \ldots, s_K\}$, the parameters include the initial state probability $p(q_0 = s_i)$, the state transition probability $p(q_t = s_j|q_{t-1} = s_i)$, and the observation probability $p(\boldsymbol{x}_t|s_t)$, where $1 \leq i, j \leq K$, t indicates a time step, and $\boldsymbol{x}_t$ represents the observation value at time step t. Traditionally, the observation probability is modeled by a GMM (Young et al., 2006). To achieve a better performance with deep models, a GMM can be replaced by a DNN (Hinton et al., 2012). In practice, a neural network is used to model $p(s_t|\boldsymbol{x}_t)$ instead of $p(\boldsymbol{x}_t|s_t)$, since by the Bayes rule, we have

$$p(\boldsymbol{x}_t|s_t) = \frac{p(s_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t)}{p(s_t)}, \tag{1}$$

where $p(s_t)$ is the prior probability, which can be easily collected from the training transcriptions.

We take a DNN based acoustic model as an example of neural network based acoustic modeling. A DNN is a feed-forward multi-layer perceptron with several hidden layers. The output of the previous layer is first linearly transformed with a weight matrix and a bias vector, and then nonlinearly transformed element-wise by an activation function. Since the DNN is used to model $p(s_t|\boldsymbol{x}_t)$, the output layer of the DNN is typically a softmax layer. Activation functions are usually rectified linear unit (ReLU), Sigmoid, and Tanh.

For an L-layer DNN, assume the input is $\boldsymbol{x}$, the i^{th} layer linear output is $\boldsymbol{o}_i$, the layer nonlinear output is $\boldsymbol{a}_i$, and the weight matrices and biases are $\boldsymbol{W}_{i,i-1}$ and $\boldsymbol{b}_i$, respectively. Omitting the time step t, the forward calculation of a DNN can be described as

$$\boldsymbol{a}_1 = \boldsymbol{x}, \tag{2}$$

for $2 \leq l \leq L-1$,

$$\begin{cases} \boldsymbol{o}_l = \boldsymbol{W}_{l,l-1}\boldsymbol{a}_{l-1} + \boldsymbol{b}_l, \\ \boldsymbol{a}_l = \text{sigmoid}(\boldsymbol{o}_l), \end{cases} \tag{3}$$

for $l = L$,

$$\begin{cases} \boldsymbol{o}_L = \boldsymbol{W}_{L,L-1}\boldsymbol{a}_{L-1} + \boldsymbol{b}_{L-1}, \\ \boldsymbol{a}_L = \text{softmax}(\boldsymbol{o}_L), \end{cases} \tag{4}$$

where $\text{sigmoid}(\boldsymbol{o}) = [\frac{1}{1+\mathrm{e}^{-o_1}}, \frac{1}{1+\mathrm{e}^{-o_2}}, \ldots, \frac{1}{1+\mathrm{e}^{-o_n}}]$, and $\text{softmax}(\boldsymbol{o}) = [\frac{\mathrm{e}^{o_1}}{\sum_{j=1}^{n}\mathrm{e}^{o_j}}, \frac{\mathrm{e}^{o_2}}{\sum_{j=1}^{n}\mathrm{e}^{o_j}}, \ldots, \frac{\mathrm{e}^{o_n}}{\sum_{j=1}^{n}\mathrm{e}^{o_j}}]$.

In the above equations, $\boldsymbol{x}$ is the input feature $\boldsymbol{x}_t$, and $\boldsymbol{a}_L$ gives the conditional probability distribution $p(s_t|\boldsymbol{x}_t)$. Cross entropy (CE) is usually used for model optimization, which can be described as

$$L_{\text{CE}} = -\sum_{t=1}^{T}\sum_{s=1}^{S}\eta_t(s)\ln P(s_t = s|\boldsymbol{x}_t), \tag{5}$$

where T is the number of speech frames in the training set, S is the number of DNN output units, and $\eta_t(s) = 1$ if s is the index of the ground-truth label at time step t, and otherwise $\eta_t(s) = 0$.

Using the back propagation algorithm, the gradients of L_{CE} with respect to all weight matrices and bias vectors can be obtained. Then the DNN parameters are updated by stochastic gradient descent (SGD) or more advanced techniques like Adam (Kingma and Ba, 2014) and AdaGrad (Duchi et al., 2011).

The DNN-HMM approach has shown great success in speech recognition with a large improvement compared to the traditional GMM-HMM. However, the additional computational load stemming from weight matrix multiplication leads to a slowdown during inference.

3 Binary matrix multiplication

3.1 Population count based binary matrix multiplication

In practical applications, the multiplication of two matrices $\boldsymbol{A} \in \mathbb{R}^{m\times k}$ and $\boldsymbol{B} \in \mathbb{R}^{k\times n}$ requires $2 \times m \times n \times k$ floating-point arithmetic operations (multiplications and additions). The algorithm Strassen runs in $O(n^{\ln 7}) = O(n^{2.81})$ time. However, it has a large hidden constant factor and is extremely cache unfriendly, which makes it difficult to obtain a good performance on contemporary processors. Because of the hardware limitation of these operations, it is hard to improve the speed of floating-point matrix multiplication.

However, in the binary case, it is possible to replace most multiplications and additions with bit operations, as each element of the resulting matrix can be represented by an inner product: $C_{ij} = \sum_k A_{ik}B_{kj}$. To explain this in detail, two bit operations "xor" and "popcnt" are defined: (1) xor(x, y) is the element-wise exclusive or of integers x and y; (2) popcnt(x) is the number of bits set to 1 in integer x. Assume that there are two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ with length n such that their elements are constrained to be either $+1$ or -1 (binary values). This representation of $\boldsymbol{a}$ or $\boldsymbol{b}$ is not suitable for fast computation, so an extra processing step is needed. All -1's are replaced with zero, and then the n bits are packed into $\lceil n/k \rceil$ ($k = 32$ or 64) k-bit integers. This process converts $\boldsymbol{a}$ and $\boldsymbol{b}$ to $\bar{\boldsymbol{a}}$ and $\bar{\boldsymbol{b}}$, respectively. The inner product of $\boldsymbol{a}$ and $\boldsymbol{b}$ can be calculated as

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle = n - 2 \times \text{popcnt}(\text{xor}(\bar{\boldsymbol{a}}, \bar{\boldsymbol{b}})). \tag{6}$$

For a better understanding, an example is given here. Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be two vectors with length eight: $\boldsymbol{a} = (1, -1, 1, 1, 1, 1, 1, 1)$ and $\boldsymbol{b} = (-1, 1, 1, -1, -1, 1, -1, 1)$. It is easy to find that their inner product is -2, which is equal to

$$\begin{aligned} & 8 - 2 \times \text{popcnt}(\text{xor}(\bar{\boldsymbol{a}}, \bar{\boldsymbol{b}})) \\ =& 8 - 2 \times \text{popcnt}(\text{xor}(10111111_2, 01100101_2)) \\ =& 8 - 2 \times \text{popcnt}(11011010_2) \\ =& 8 - 2 \times 5 \\ =& -2. \end{aligned} \tag{7}$$

3.2 Binary matrix multiplication on CPU

Recent CPUs have built-in support for population count instructions for both 32- and 64-bit operands. On the Intel Haswell microarchitecture, two eight-wide fused multiply-add (FMA) instructions can be performed every cycle, yielding 32 32-bit floating-point operations per cycle (Hammarlund et al., 2014). In contrast, the 64-bit population count instruction can be issued every cycle, yielding 128 binary operations per cycle (other instructions like xor can be issued simultaneously due to superscalar execution). Based on this observation, the theoretical peak performance (TPP) of an Intel Haswell CPU can be obtained. For example, with an Intel i3-4150 CPU's clock speed being 3.5 GHz, the floating-point TPP and binary TPP are 3.5×32=112.0 giga floating-point/binary arithmetic operations per second (GOPS) per core and 3.5×128=448.0 GOPS per core, respectively. As a result, the population count instruction based binary matrix multiplication has 128/32 = 4.0 times throughput compared with optimized floating-point matrix multiplication. On an ARM Cortex A72 microarchitecture, one four-wide FMA instruction, or up to eight floating-point operations can be performed per cycle. Although the precise timing of population count instruction on ARM cannot be estimated, empirical evaluation shows that it can deliver a considerable speedup.

To achieve compute-bound performance, the implementation of binary matrix multiplication uses packing to ensure consecutive memory locations access and cache- and register-aware blocks to maximize data reuse. Algorithm 1 shows the implementation of binary matrix multiplication. To reduce cache miss penalty, the submatrix sizes $k_\text{c} \times n_\text{c}$ and $m_\text{c} \times k_\text{c}$ are designed to best fit the CPU cache (Goto and van de Geijn, 2008; Low et al., 2016). Note that the constant n shown in Eq. (6) is omitted since n can be treated as a bias.

An Intel i3-4150 CPU (Intel Haswell microarchitecture) running at 3.50 GHz and a HiSilicon Kirin 950 CPU (ARM Cortex A72 microarchitecture) run-

Algorithm 1 High performance implementation of binary matrix multiplication

```
Require: binary matrix $\boldsymbol{A}$ with size $m \times k$, and
    binary matrix $\boldsymbol{B}$ with size $k \times n$
Ensure: matrix $\boldsymbol{C} = \boldsymbol{A} \times \boldsymbol{B}$ with size $m \times n$
 1: for $j_\text{c} = 0$ to $n - 1$ in steps of $n_\text{c}$ do
 2:   for $p_\text{c} = 0$ to $k - 1$ in steps of $k_\text{c}$ do
 3:     Pack size $k_\text{c} \times n_\text{c}$ submatrix of $\boldsymbol{B}$ to $\boldsymbol{B}_\text{c}$
 4:     for $i_\text{c} = 0$ to $m - 1$ in steps of $m_\text{c}$ do
 5:       Pack size $m_\text{c} \times k_\text{c}$ submatrix of $\boldsymbol{A}$ to $\boldsymbol{A}_\text{c}$
 6:       for $j_\text{r} = 0$ to $m_\text{c} - 1$ in step of 1 do
 7:         for $i_\text{r} = 0$ to $n_\text{c} - 1$ in step of 1 do
 8:           $x = 0$
 9:           for $p_\text{r} = 0$ to $k_\text{c} - 1$ in step of 1 do
10:             $x$+=popcnt(xor($\boldsymbol{A}_\text{c}(i_\text{r}, p_\text{r}), \boldsymbol{B}_\text{c}(p_\text{r}, j_\text{r})$))
11:           end for
12:           $\boldsymbol{C}_\text{c}(i_\text{r}, j_\text{r})$ −= $2x$ // $\boldsymbol{C}_\text{c}$ is a subma-
              // trix of $\boldsymbol{C}$ that relates to $\boldsymbol{A}_\text{c}\boldsymbol{B}_\text{c}$
13:         end for
14:       end for
15:     end for
16:   end for
17: end for
```

ning at 2.30 GHz were used to compare the speed of binary matrix multiplication and floating-point matrix multiplication. The baseline floating-point implementation used Intel Math Kernel Library 11.3 Update 3 on the Intel platform and OpenBLAS 0.2.19 on the ARM platform to achieve the maximum speed. To determine the maximum real performance, a matrix multiplication of size $(m, n, k) = (2048, 2048, 2048)$ is used. It is worth noting that the batch size m is much smaller than 2048 during inference in a real implementation because of latency (m utterances are packed in one parallel batch for the forwarding computation). Hence, the performance of matrix multiplication with size $(16, 2048, 2048)$, corresponding to $m = 16$, was also evaluated.

Table 1 reports the single thread GOPS. For each matrix size, the calculation was repeated 100 times to obtain the average GOPS on an Intel i3-4150 CPU. This shows that binary matrix multiplication can achieve a 7.2× speedup when the batch size is 16. Table 2 reports single thread GOPS on a HiSilicon Kirin 950 CPU. Similar to the results in Table 1, binary matrix multiplication is 6.7× faster when the batch size is 16.

3.3 Binary matrix multiplication on GPU

Population count instructions are natively supported by recent NVIDIA GPUs via intrinsics, including `popc__()` (an intrinsic function that maps to a single instruction) for 32-bit operands and `popcll__()` (an intrinsic function that maps to multiple instructions) for 64-bit operands.

On the NVIDIA Pascal microarchitecture Tesla P100 GPU, up to 64 32-bit FMA instructions (128 floating-point operations) can be issued every cycle per streaming multiprocessor. In contrast, 16 32-bit population count instructions, 64 32-bit integer add instructions, and 64 32-bit bit-wise xor instructions can be issued every cycle per streaming multiprocessor, yielding $2 \times 64 \times 32/(1 + 4 + 1) = 683$ binary operations per cycle. Hence, in theory, binary matrix multiplication can achieve a $683/128 = 5.3$ times speedup.

The implementation of binary matrix multiplication uses a modified version of an example program in CUDA toolkit document that replaces multiply-add operations with bit operations. The floating-point baseline used NVIDIA cuBLAS library to achieve the best performance. Table 3 describes the GOPS of both implementations. When the batch size is 16, the proposed binary matrix multiplication is able to achieve a 5.4× speedup.

Table 1 Speed comparison on an Intel i3-4150 CPU (single thread)

Size	FMM	BMM	Speedup
16, 2048, 2048	34.5	249.0	7.2×
2048, 2048, 2048	91.2	263.5	2.9×
TPP	112.0	448.0	4.0×

FMM: floating-point matrix multiplication; BMM: binary matrix multiplication; TPP: theoretical peak performance

Table 2 Speed comparison on a HiSilicon Kirin 950 CPU (single thread)

Size	FMM	BMM	Speedup
16, 2048, 2048	3.8	25.4	6.7×
2048, 2048, 2048	12.0	29.1	2.4×
TPP	18.4	$\approx$ 39.5	2.1×

FMM: floating-point matrix multiplication; BMM: binary matrix multiplication; TPP: theoretical peak performance

Table 3 Speed comparison on NVIDIA Tesla P100 GPU

Size	FMM	BMM	Speedup
16, 2048, 2048	1.3×10^3	7.0×10^3	5.4×
2048, 2048, 2048	7.6×10^3	30.6×10^3	4.0×
TPP	9.3×10^3	49.6×10^3	5.3×

FMM: floating-point matrix multiplication; BMM: binary matrix multiplication; TPP: theoretical peak performance

4 Binary neural networks for speech recognition

Binarization on a feed-forward DNN is first introduced for speech recognition. In a DNN with L layers, let the activation in layer l be $\boldsymbol{a}_l$, the weight matrix between layers l and $l + 1$ be $\boldsymbol{W}_{l+1,l}$, and the bias in layer $l + 1$ be $\boldsymbol{b}_{l+1}$, where $1 \leq l \leq L - 1$. The binarized weight matrices and activations are denoted by $\hat{\boldsymbol{W}}_{l+1,l}$ and $\hat{\boldsymbol{a}}_l$, respectively. Note that weight matrix $\boldsymbol{W}_{2,1}$ (the network input layer) is not binarized in most cases, since binarization on this input layer will cause an obvious degradation, which will be shown in the experimental section.

Algorithms 2 and 3 describe the forward pass and backward pass during the training of a binary DNN, respectively, where BatchNormGradient($\cdot$) calculates the gradient with respect to the input of BatchNorm($\cdot$). Note that weights are stored as floating-point values. In the forward pass and back-

Algorithm 2 Forward pass to train a binary DNN

```
Require: input $\boldsymbol{a}_1$, weight matrices $\boldsymbol{W}_{2,1}, \boldsymbol{W}_{3,2}, \ldots, \boldsymbol{W}_{L,L-1}$, and biases $\boldsymbol{b}_2, \boldsymbol{b}_3, \ldots, \boldsymbol{b}_L$
Ensure: activations $\boldsymbol{a}_2, \boldsymbol{a}_3, \ldots, \boldsymbol{a}_L$
  1. Input layer: $\boldsymbol{W}_{2,1}$ is not binarized
    $\boldsymbol{c}_2 = \boldsymbol{W}_{2,1}\boldsymbol{a}_1 + \boldsymbol{b}_2$
    $\boldsymbol{d}_2 = \mathrm{BatchNorm}(\boldsymbol{c}_2)$
    $\boldsymbol{a}_2 = \mathrm{HardTanh}(\boldsymbol{d}_2)$
    $\hat{\boldsymbol{a}}_2 = \mathrm{Binarize}(\boldsymbol{a}_2)$
  2. Hidden layers: $\boldsymbol{W}_{l,l-1}$ and $\boldsymbol{a}_l$ $(3 \le l \le L-1)$ are binarized
    for $l = 3$ to $L-1$ do
      $\widehat{\boldsymbol{W}}_{l,l-1} = \mathrm{Binarize}(\boldsymbol{W}_{l,l-1})$
      $\boldsymbol{c}_l = \widehat{\boldsymbol{W}}_{l,l-1}\hat{\boldsymbol{a}}_{l-1} + \boldsymbol{b}_l$
      $\boldsymbol{d}_l = \mathrm{BatchNorm}(\boldsymbol{c}_l)$
      $\boldsymbol{a}_l = \mathrm{HardTanh}(\boldsymbol{d}_l)$
      $\hat{\boldsymbol{a}}_l = \mathrm{Binarize}(\boldsymbol{a}_l)$
    end for
  3. Output layer: $\boldsymbol{a}_L$ is not binarized
    $\widehat{\boldsymbol{W}}_{L,L-1} = \mathrm{Binarize}(\boldsymbol{W}_{L,L-1})$
    $\boldsymbol{c}_L = \widehat{\boldsymbol{W}}_{L,L-1}\hat{\boldsymbol{a}}_{L-1} + \boldsymbol{b}_L$
    $\boldsymbol{d}_L = \mathrm{BatchNorm}(\boldsymbol{c}_L)$
    $\boldsymbol{a}_L = \mathrm{Softmax}(\boldsymbol{d}_L)$
```

Algorithm 3 Backward pass to train a binary DNN

```
Require: gradient $\nabla_{\boldsymbol{d}_L}$, input $\boldsymbol{a}_1$, activations $\hat{\boldsymbol{a}}_2, \hat{\boldsymbol{a}}_3, \ldots, \hat{\boldsymbol{a}}_L$, intermediate activations $\boldsymbol{c}_2, \boldsymbol{c}_3, \ldots, \boldsymbol{c}_L$, $\boldsymbol{d}_2, \boldsymbol{d}_3, \ldots, \boldsymbol{d}_L$, weight matrices $\widehat{\boldsymbol{W}}_{2,1}, \widehat{\boldsymbol{W}}_{3,2}, \ldots, \widehat{W}_{L,L-1}$, and biases $\boldsymbol{b}_2, \boldsymbol{b}_3, \ldots, \boldsymbol{b}_L$
Ensure: gradients $\nabla_{\boldsymbol{W}_{2,1}}, \nabla_{\boldsymbol{W}_{3,2}}, \ldots, \nabla_{\boldsymbol{W}_{L,L-1}}$ and $\nabla_{\boldsymbol{b}_2}, \nabla_{\boldsymbol{b}_3}, \ldots, \nabla_{\boldsymbol{b}_L}$
  1. Output layer:
    $\nabla_{\boldsymbol{c}_L} = \mathrm{BatchNormGradient}(\nabla_{\boldsymbol{d}_L})$
    $\nabla_{\boldsymbol{b}_L} = \nabla_{\boldsymbol{c}_L}$
    $\nabla_{\boldsymbol{W}_{L,L-1}} = \nabla_{\boldsymbol{c}_L}\hat{\boldsymbol{a}}_{L-1}^{\mathrm{T}}$
  2. Hidden layers:
    for $l = L-1$ to 3 do
      $\nabla_{\hat{\boldsymbol{a}}_l} = \widehat{\boldsymbol{W}}_{l,l-1}\nabla_{\boldsymbol{c}_{l+1}}$
      $\nabla_{\boldsymbol{d}_l} = \nabla_{\hat{\boldsymbol{a}}_l} \odot \mathbb{1}\{|\boldsymbol{d}_l| \le 1\}$
      $\nabla_{\boldsymbol{c}_l} = \mathrm{BatchNormGradient}(\nabla_{\boldsymbol{d}_l})$
      $\nabla_{\boldsymbol{b}_l} = \nabla_{\boldsymbol{c}_l}$
      $\nabla_{\boldsymbol{W}_{l,l-1}} = \nabla_{\boldsymbol{c}_l}\hat{\boldsymbol{a}}_{l-1}^{\mathrm{T}}$
    end for
  3. Input layer:
    $\nabla_{\hat{\boldsymbol{a}}_2} = \boldsymbol{W}_{2,1}\nabla_{\boldsymbol{c}_3}$
    $\nabla_{\boldsymbol{d}_2} = \nabla_{\hat{\boldsymbol{a}}_2} \odot \mathbb{1}\{|\boldsymbol{d}_2| \le 1\}$
    $\nabla_{\boldsymbol{c}_2} = \mathrm{BatchNormGradient}(\nabla_{\boldsymbol{d}_2})$
    $\nabla_{\boldsymbol{b}_2} = \nabla_{\boldsymbol{c}_2}$
    $\nabla_{\boldsymbol{W}_{2,1}} = \nabla_{\boldsymbol{c}_2}\boldsymbol{a}_1^{\mathrm{T}}$
```

ward pass, before the calculation, weights are binarized on the fly. The original floating-point weights are updated with the gradients.

During the training of a binary DNN, Binarize$(\cdot)$ is used to transform each element of $\boldsymbol{W}_{l+1,l}$ or $\boldsymbol{a}_l$ to $+1$ or -1. The deterministic binarization function is defined by

$$\mathrm{Binarize}(x) = \begin{cases} +1, & \text{if } x > 0, \\ -1, & \text{otherwise}, \end{cases} \tag{8}$$

where x is a floating-point value. In this work we use a stochastic version which is defined by

$$\mathrm{Binarize}(x) = \begin{cases} +1, & \text{if } x - p > 0, \\ -1, & \text{otherwise}, \end{cases} \tag{9}$$

where p is a random value that is drawn from a normal distribution with zero mean and unit variance. This is different from the approach in Courbariaux et al. (2016). Stochastic binarization incurs more computational cost than the deterministic version, but it reduces overfitting, and hence it is used to binarize the activations during training. Note that although $\mathrm{HardTanh}(x) = \max(-1, \min(x, 1))$ has no effect on the forward pass since $\mathrm{Binarize}(\mathrm{HardTanh}(\cdot)) = \mathrm{Binarize}(\cdot)$, it sets the gradient to zero in the backward pass when the corresponding input is too large.

4.1 Straight through estimator

While the forward calculation of the binary DNN is straightforward, there is an issue in backward calculation. Mathematically, the gradient of Binarize$(\cdot)$ is always zero with respect to its input (except at 0), which makes gradient based training impossible. However, this can be resolved using the straight through estimator (STE) (Bengio et al., 2013).

STE is a gradient estimator that allows the use of a threshold function in neural networks trained by back propagation. Although STE is a biased estimator of the gradient, it performs well in practice. In the forward pass, Binarize$(\cdot)$ acts normally to produce the binary output. In the backward pass, Binarize$(\cdot)$ is replaced with the identity function. In other words, it just allows the gradient with respect to the binary output to pass through.

In this work, a variant of STE (Courbariaux et al. 2016) is used to cancel the gradient when the magnitude of the input is too large:

$$\begin{cases} q = \mathrm{Binarize}(p), \\ \nabla_p = \nabla_q \odot \mathbb{1}\{|p| \le 1\}, \end{cases} \tag{10}$$

where p and q are the input and output of Binarize$(\cdot)$ respectively, and "$\odot$" denotes the Hadamard product. When the absolute value of p exceeds 1, the gradient with respect to q is set to zero; otherwise, the gradient with respect to q is copied to the gradient with respect to p. The indicator function $\mathbb{1}\{|p| \le 1\}$

is exactly the gradient of HardTanh(·). Hence, the composition of HardTanh(·) and Binarize(·) directly implements the variant of STE mentioned above.

4.2 Optimization for fast inference

4.2.1 Module reforming

Batch normalization (Ioffe and Szegedy, 2015) used in Algorithm 2 can be described as

$$\mathrm{BatchNorm}(x)=\frac{x-\mu}{\sqrt{\sigma^2+\epsilon}}\gamma+\beta, \tag{11}$$

where x is the layer input, γ and β are learnable parameters, and ϵ is a small value to avoid underflow. During inference, the mean μ and variance σ^2 are replaced with fixed values that are estimated over the training data, which leads to an efficient and compact representation:

$$\mathrm{BatchNorm}(x)=\xi x+\delta, \tag{12}$$

where $\xi=\frac{\gamma}{\sqrt{\sigma^2+\epsilon}}$ and $\delta=\beta-\frac{\mu\gamma}{\sqrt{\sigma^2+\epsilon}}$, both of which can be pre-computed before deployment.

4.2.2 Module compacting

During training, floating-point weights are essential to do the update, but during inference, only their binarized versions are used. For this reason, by replacing $\boldsymbol{W}_{l,l-1}$ with $\widehat{\boldsymbol{W}}_{l,l-1}$ in Algorithm 2, module $\widehat{\boldsymbol{W}}_{l,l-1}=\mathrm{Binarize}(\boldsymbol{W}_{l,l-1})$ can be removed. Moreover, considering Binarize(HardTanh(·)) = Binarize(·), module $\boldsymbol{a}_l=\mathrm{HardTanh}(\boldsymbol{a}_l)$ can be removed. Furthermore, module $\boldsymbol{a}_l=\mathrm{BatchNorm}(\boldsymbol{a}_l)$ and $\hat{\boldsymbol{a}}_l=\mathrm{Binarize}(\boldsymbol{a}_l)$ can be seamlessly integrated into one module so that unnecessary computation can be avoided.

4.3 Binarization on other advanced neural networks

For other more advanced neural networks, such as CNNs and LSTMs, it is also possible to build a binary counterpart. Considering the great success of deep CNNs in speech recognition recently (Bi et al., 2015; Qian et al., 2016; Sercu et al., 2016; Xiong et al., 2016; Yu et al., 2016), the binarization of CNNs is further explored here.

Typically, there are two kinds of layers in CNNs, fully connected layers and convolutional layers. For the fully connected layers, the same binalization procedure as described above for DNNs is implemented, and the binarization method described in Algorithm 1 can be applied directly. For convolutional layers, the nonlinear function ReLU(·) is replaced by a composition of HardTanh(·) with Binarize(·). Similar to the binarization of fully connected layers, batch normalization is used to limit the linear outputs to a relatively small range, which makes the training process more stable. The kernel weights are stored and updated in the floating-point format. During forward and backward propagation, the kernel weights are first binarized before calculation. The first layer of a binary CNN is not binarized, just as with a binary DNN. Using this binarization on CNNs, the computational load can be significantly reduced in model inference. Moreover, the speedup is especially large for very deep CNNs with many convolutional layers, such as the visual geometry group network (VGG) (Bi et al., 2015; Qian et al., 2016; Sercu et al., 2016) or ResNet (Yu et al., 2016) based acoustic models.

In addition, to achieve a trade-off between accuracy and speed, we can binarize a portion of CNNs. In this study, experimental results show that a partly binarized CNN or a very deep CNN can achieve a good accuracy compared to the fully binarized CNN.

5 Knowledge distillation from full precision deep models to binary neural networks

By convention, a neural network based acoustic model with parameter θ typically maps a D-dimensional acoustic feature vector $\boldsymbol{x}\in\mathbb{R}^D$ into a distribution y over K classes, denoted as $p(y|\boldsymbol{x},\theta)$. The goal of neural network training is to find a set of parameters θ so that the outputs closely match the labels on the training data. The parameters θ are usually updated by minimizing the cross entropy loss between the empirical posterior distribution and the predicted posterior distribution for the current batch (X,Y) consisting of N training samples:

$$J(\theta;X,Y)=\sum_{n=1}^{N}\mathrm{CE}\big(p(y_n^{\mathrm{empirical}}),p(y_n|\boldsymbol{x}_n,\theta)\big). \tag{13}$$

Normally, the empirical posterior distribution is derived from the ground-truth label and represented by one-hot encoding (only one of all K classes has the probability of 1). Under this condition, the above cross entropy loss can be rewritten as

$$\begin{aligned}J_{\mathrm{hard}}(\theta;X,Y)&=\sum_{n=1}^{N}\mathrm{CE}\big(p(y_n^{\mathrm{label}}),p(y_n|\boldsymbol{x}_n,\theta)\big)\\&=\sum_{n=1}^{N}\sum_{k=1}^{K}-\mathbb{1}\{y_{n,k}^{\mathrm{label}}=1\}\ln p(y_{n,k}|\boldsymbol{x}_n,\theta),\end{aligned}\tag{14}$$

where the subscript "hard" means that the loss function uses hard targets, and "hard" is the one-hot encoding of labels.

Hinton et al. (2015) introduced the idea that the correlations among the classes in the softmax outputs of a classifier can be very informative, since it is more reasonable to misclassify a dog as a cat rather than an apple. In contrast, to train the model against hard targets, one can train a new simple model using the posteriors (soft targets) from a well-trained model. This mode is also named teacher-student learning (Li et al., 2017; Lu and Renals, 2017; Lu et al., 2017), in which the new simple model is the student model and the well-trained model is the teacher model. Usually the teacher is a more powerful model using a complex structure having a significantly better accuracy, while the student is relatively simple with a small scale. To boost the performance of the student model to be deployed in a resource-limited embedded device, the training criterion can be changed to

$$\begin{aligned}J_{\mathrm{soft}}(\theta;X,Y)&=\sum_{n=1}^{N}\mathrm{CE}\big(p(y_n|\boldsymbol{x}_n,\theta^{\mathrm{teacher}}),p(y_n|\boldsymbol{x}_n,\theta)\big)\\&=\sum_{n=1}^{N}\sum_{k=1}^{K}-p(y_{n,k}|\boldsymbol{x}_n,\theta^{\mathrm{teacher}})\ln p(y_{n,k}|\boldsymbol{x}_n,\theta).\end{aligned}\tag{15}$$

In this study, the full precision floating-point deep model and the binary deep model are used as the teacher and student, respectively. This framework can be used for knowledge distillation from the full precision floating-point deep model to the binary model. Experiments show that with this knowledge distillation approach, the performance of a binary neural network can be significantly improved. Specifically, a normal full precision floating-point deep model is first trained with hard targets, and then used to generate soft labels for the training data. By combining these two loss functions corresponding to the hard and soft labels separately, the new system can be optimized according to

$$\begin{aligned}J(\theta;X,Y)\triangleq&\lambda J_{\mathrm{hard}}(\theta;X,Y)\\&+(1-\lambda)J_{\mathrm{soft}}(\theta;X,Y),\end{aligned}\tag{16}$$

where λ is an interpolation parameter. When $\lambda=0$, the criterion is equal to $J_{\mathrm{soft}}(\theta;X,Y)$, which just uses the soft labels; otherwise, when $\lambda=1$, the criterion is equal to $J_{\mathrm{hard}}(\theta;X,Y)$, which uses just the ground-truth hard labels (the same as the normal CE training).

6 Experimental results

To validate the effectiveness of binary neural networks for speech recognition, the proposed binary DNNs and binary CNNs were evaluated on the standard Switchboard telephone speech recognition task. Kaldi (Povey et al., 2011) was used to train the GMM-HMM model and decode the speech, while Torch7 (Collobert et al., 2011) was used to train all types of neural network models. An LDA-MLLT-SAT GMM-HMM model with 8876 tied states was first built with the standard Kaldi recipe for Switchboard data. The GMM-HMM model was then used to generate the state level alignments for neural network training. The Switchboard/CallHome (SWB/CH) portions of the NIST Hub5 2000 evaluation set (Hub5'00) and the Fisher/Switchboard (FSH/SWB) portions of the Rich Transcription 2003 evaluation set (RT03S) were used as the test sets. The 309-h speech data were divided into training data (90%) and cross validation data (10%) for neural network training. One-pass decoding with a trigram language model trained on the Switchboard transcripts was applied in testing. Note that the main focus is the binary deep model based acoustic modeling algorithm, so we ran only the first decoding pass with the trigram language model to do the fair comparison.

6.1 Model description

The detailed architecture configurations of the baseline full precision DNN, CNN, and very deep CNN (VDCNN) models are shown in Table 4.

6.1.1 DNN

Thirty-six-dimensional log-mel frequency filter bank (FBANK) along with their first- and

Table 4 Architecture configurations of different deep models

Model	VDCNN	CNN	DNN
Number of convolutions	10	2	–
Number of max poolings	5	1	–
Padding	Frequency & time	None	–
Input size	17 × 64 × 1	11 × 36 × 3	1188
	Conv, 3 × 3, 64		
	Conv, 3 × 3, 64		
	Pool, 1 × 2		
	Conv, 3 × 3, 128		
	Conv, 3 × 3, 128		
	Pool, 1 × 2		
Convolutional &	Conv, 3 × 3, 128	Conv, 8 × 7, 256	–
max pooling layer	Conv, 3 × 3, 128	Pool, 1 × 3	
	Pool, 2 × 2	Conv, 4 × 3, 256	
	Conv, 3 × 3, 256		
	Conv, 3 × 3, 256		
	Pool, 2 × 2		
	Conv, 3 × 3, 256		
	Conv, 3 × 3, 256		
	Pool, 2 × 2		
Output size	2 × 2 × 256	1 × 8 × 256	–
			1188 × 2048
	1024 × 2048	2048 × 2048	2048 × 2048
	2048 × 2048	2048 × 2048	2048 × 2048
Fully connected layer	2048 × 2048	2048 × 2048	2048 × 2048
	2048 × 2048	2048 × 2048	2048 × 2048
	2048 × 8876	2048 × 8876	2048 × 2048
			2048 × 8876
Softmax	8876	8876	8876

The model configuration, such as 3 × 3 and 64, indicates that the layer uses a 3 × 3 filter and the output contains 64 feature maps. CNN: convolutional neural network; DNN: deep neural network; VDCNN: very deep CNN; Conv: convolutional layer; Pool: max pooling layer

second-order derivatives were used to train the full precision DNN model. The acoustic features were mean normalized per speaker. A temporal context of 11 consecutive frames was used as the input for a DNN. The full precision DNN has six fully connected layers with 2048 sigmoid hidden units in each layer. The last layer is a softmax layer with the same output targets as the senones.

6.1.2 CNN

The full precision CNN model used the same expanded feature as the input. The feature was organized into three channels with an 11 × 36 image size in each channel. The full precision CNN model was built using the classic structure (Sainath et al., 2013), which has two convolutional layers followed by four fully connected layers. Max pooling was used after the first convolutional operation and all hidden layers used ReLU as the activation function. There were 256 feature maps in each convolutional layer.

6.1.3 VDCNN

Following our previous work on VDCNNs (Qian et al., 2016), a 64-dimensional static FBANK with 17 context window was formed as the single input channel. The full precision VDCNN has 10 layers of convolutional layers and four fully connected layers. All hidden units used ReLU as the activation function. Compared to the shallow CNN, a VDCNN used a smaller kernel size and a larger input channel size which can increase the model depth. The feature map number was increased gradually to keep a reasonable model scale. The superiority of a VDCNN has been demonstrated on several speech tasks, e.g., telephone speech recognition (Sercu et al., 2016) and noise-robust speech recognition (Qian and Woodland, 2016; Qian et al., 2016). Details on VDCNN can be found in Qian et al. (2016).

For binary neural networks, the basic model configuration is the same as that in the corresponding full precision deep models shown in Table 4. The only difference is the activation functions, which are replaced by three new functions in the proposed binary deep neural networks, HardTanh(·), Binarize(·), and BatchNorm(·).

6.2 Evaluation on binary DNN

The comparisons on accuracy and speed between the full precision DNN (FPDNN) and proposed binary DNN (BDNN) are shown in Table 5. Word error rate (WER) was used as the performance metric for speech recognition. To test the speed of inference, FPDNN and BDNN models were evaluated on an Intel i3-4150 CPU. The batch size was set to 16. Frames per second (FPS) was used to measure the speed. As shown in Table 5, compared with FPDNN, BDNN can achieve about 4.0 times speedup with about 25.0% WER degradation.

There are some considerations on the architecture design for the binary DNN, which have significant impacts on accuracy.

6.2.1 Using full precision weight matrix in the input layer

Since the input features for DNN training are floating-point values, to exploit binary matrix multiplication, these should be quantized to k-bit values. However, this is not very helpful. If k is large, to adopt k-bit values in binary matrix multiplication, the calculation amount is k times larger than that from 1-bit binary matrix multiplication, in which case there will be little speedup compared with floating-point matrix multiplication. In contrast, if k is small, the information loss caused by quantization on the original feature is very large, which is very harmful for model optimization.

The binary DNNs with full precision weight matrix (BDNN) or with binary weight matrix in the input layer (BDNN-inBin) are also shown in Table 5. It is observed that there is an obvious WER degradation when doing the binarization on the first input layer. In the following experiments, the weight matrix in the first input layer was not binarized for all the proposed binary deep models.

Table 5 WER and FPS comparison of DNN models on Switchboard telephone speech recognition task

Model	WER (%)				FPS
	Hub5'00		RT03S		
	SWB	CH	FSH	SWB	
FPDNN	15.6	27.9	20.7	30.2	459
BDNN	20.8	34.6	26.6	36.6	1679
BDNN-inBin	21.8	35.6	27.4	37.4	–
BDNN-noBN	73.1	81.9	75.2	78.9	–
FPDNN-3072	15.9	27.9	20.7	30.1	229
BDNN-3072	19.5	32.7	25.0	34.8	1080

WER: word error rate; FPS: frames per second; FPDNN: full precision DNN; BDNN: binary DNN; BDNN-inBin: BDNN with binary weight matrix in the input layer; BDNN-noBN: BDNN without using batch normalization; FPDNN-3072: FPDNN with 3072 hidden nodes; BDNN-3072: BDNN with 3072 hidden units per hidden layer

6.2.2 Using batch normalization

In the architectures of binary DNN and binary CNN, before nonlinear transform, the linear output of each layer ($\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}$) was first batch normalized. This is critical to train a good binary neural network model. Since $\boldsymbol{W}$ and $\boldsymbol{x}$ are binary, the linear outputs from $\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}$ can have very large or very small values that heavily deviate from $+1$ or -1. Without batch normalization, the dynamic range of the linear output is too large and the training process is not stable. The related binary DNN without using batch normalization (BDNN-noBN) is listed in Table 5. We can see that the binary model without batch normalization does not converge well, leading to high WERs. Note that using batch normalization will introduce little overhead during model inference. For instance, given the batch size k, the number of hidden units in the last layer m, and the number of hidden units in the current layer n, the computational complexity of binary matrix multiplication and batch normalization are $O(kmn)$ and $O(kn)$, respectively. Usually m is very large ($m = 2048$) so the calculation time of batch normalization can be omitted when compared to that of matrix multiplication.

6.2.3 Using more hidden units in hidden layers

Since the weights are represented with only 1-bit binary values, the expressive power of binary DNNs is much smaller than that of full precision DNNs. This is a major reason for the degraded WER of the binary DNN. To achieve an improved performance, a binary DNN with more hidden units in the hidden layer was built. Here a binary DNN model with 3072 hidden units per hidden layer (BDNN-3072) was constructed. For a fair comparison, the full precision DNN with 3072 hidden nodes (FPDNN-3072) was also built. The results are shown in Table 5. It shows that enlarging the hidden layer does not achieve any further improvement on the normal full precision DNN, which leads to the same conclusion as in Dahl et al. (2012) and Mohamed et al. (2012). In contrast, using more hidden units in the binary DNN obtains more than 6.0% WER improvement over all test sets. Moreover, BDNN-3072 achieves more than 4.0 times speedup compared to FPDNN-3072, and is still 2.5 times faster than FPDNN.

6.3 Evaluation on binary CNN and VDCNN

In this subsection the binarization on CNN is performed. The comparison results between the proposed binary CNN (BCNN) and full precision CNN (FPCNN) are shown in Table 6. Similar to BDNN, a full precision matrix was always used in the first input layer of all binary CNN models. It can be observed that the normal full precision CNN has better results than the full precision DNN shown in Table 5.

Table 6 WER comparison of CNN and VDCNN models on Switchboard telephone speech recognition task

Model	WER (%)			
	Hub5'00		RT03S	
	SWB	CH	FSH	SWB
FPCNN	15.1	26.9	20.3	29.6
BCNN	21.1	34.7	26.8	37.1
BCNN-FPFC	16.3	28.8	21.4	31.2
FPVDCNN	13.8	24.6	18.7	27.9
BVDCNN	21.6	33.5	26.1	35.7

WER: word error rate; FPCNN: full precision CNN; BCNN: binary CNN; BCNN-FPFC: BCNN with full precision values on fully connected layers; FPVDCNN: full precision very deep CNN; BVDCNN: binary very deep CNN

6.3.1 With/without binarization on fully connected layers

Two kinds of binary CNN were implemented, and the results are shown in Table 6, indicated by BCNN and BCNN-FPFC. BCNN denotes the new binary model with binarization on all layers (except the input layer as stated above), including both convolutional and fully connected layers. It is observed that the degradation is about 25.0% from FPCNN to BCNN, which is consistent with the observation on DNN shown in Table 5.

We know that the computational load on the convolutional layers is higher than that on the fully connected layers for CNNs. Therefore, a BCNN-FPFC was built, which performs only the binarization on the convolutional layers and still has full precision values on fully connected layers. BCNN-FPFC makes a good trade-off between accuracy and speed. The results show that the performance is substantially improved with this partly binarized CNN. Compared to FPCNN, the WER gap is reduced to 5.0% over all test sets, which is very promising.

6.3.2 Binary VDCNN

The binarization was further extended to the more powerful VDCNN, with results shown in Table 6. We can observe that very deep CNNs with more convolutional layers significantly achieve better performance than shallow ones using either the full precision VDCNN (FPVDCNN) or the binary VDCNN (BVDCNN).

6.4 Evaluation on knowledge distillation for binary neural networks

The knowledge distillation from the normal full precision deep models using the teacher-student learning framework was evaluated and the performance comparisons of binary DNNs are shown in Table 7. We used FPDNN as the teacher model. Different interpolation weight λ settings in Eq. (16) were studied to find the best one. Note that using $\lambda = 1.0$ actually corresponds to the normal cross entropy (CE) training using the purely hard labels. The results show that the knowledge in full precision DNN (FPDNN) can be effectively transferred with the proposed training scheme, resulting in a significant and consistent improvement for all test sets. Using purely soft labels ($\lambda = 0.0$) seems sufficient when doing knowledge distillation between full

Table 7 WER comparison of BDNN with knowledge distillation from the FPDNN on Switchboard telephone speech recognition task

Model	Teacher	λ	WER (%)			
			Hub5'00		RT03S	
			SWB	CH	FSH	SWB
FPDNN	–	–	15.6	27.9	20.7	30.2
BDNN+TS	FPDNN	1.0	20.8	34.6	26.6	36.6
BDNN+TS	FPDNN	0.5	19.6	33.5	25.3	35.3
BDNN+TS	FPDNN	0.0	19.7	33.0	25.1	35.1

WER: word error rate; FPDNN: full precision DNN; λ: interpolation weight; BDNN+TS: binary DNN using the teacher-student learning method precision and binary DNNs.

Increasing the number of hidden units in BDNN was also investigated to examine whether the student model can learn the knowledge from the teacher more effectively with an enlarged model scale. The results of two BDNNs with 2048 or 3072 hidden units are shown in Table 8. It is observed that although the same teacher model is used, the performance is obviously further boosted with more hidden units in the student model, which demonstrates more effective knowledge distillation. Compared with FPDNN, there is a 10.0%–15.0% performance deterioration with our proposed binary DNNs over all the test sets, but a large acceleration in model inference.

The same knowledge distillation framework was also performed and validated on CNNs. In this case, the full precision VDCNN was used as the teacher model to improve the shallow binary CNN. The shallow CNN is more feasible than the VDCNN for the real applications on the low-power embedded devices, so only the shallow CNN was explored as the student model here. The comparison results are shown in Table 9. The same observation and conclusion as those for binary DNNs are obtained:

Table 8 WER comparison of different BDNN models with knowledge distillation from the same FPDNN on Switchboard telephone speech recognition task

Student	Teacher	WER (%)			
		Hub5'00		RT03S	
		SWB	CH	FSH	SWB
FPDNN	–	15.6	27.9	20.7	30.2
BDNN-2048	FPDNN	19.7	33.0	25.1	35.1
BDNN-3072	FPDNN	18.6	31.5	23.6	33.6

WER: word error rate; FPDNN: full precision DNN; BDNN-2048/-3072: binary DNN with 2048/3072 hidden units

Table 9 WER comparison of different BCNN models with knowledge distillation from the FPVDCNN on Switchboard telephone speech recognition task

Model	Teacher	WER (%)			
		Hub5'00		RT03S	
		SWB	CH	FSH	SWB
FPCNN	–	15.1	26.9	20.3	29.6
BCNN	–	21.1	34.7	26.8	37.1
BCNN	FPVDCNN	18.8	31.5	23.9	33.8
BCNN-FPFC	–	16.3	28.8	21.4	31.2
BCNN-FPFC	FPVDCNN	15.2	27.4	20.3	29.8

WER: word error rate; FPCNN: full precision CNN; BCNN: binary CNN; BCNN-FPFC: BCNN with full precision values on fully connected layers; FPVDCNN: full precision very deep CNN

doing knowledge distillation from a powerful full precision deep model to the binary CNN will obviously improve the WER, and the performance gap between BCNNs and normal FPCNNs is reduced significantly. Specifically for the BCNN without fully connected layer binarization (BCNN-FPFC), the WER degradation is very small and negligible (almost the same over three test sets), and this is very promising and further demonstrates the effectiveness of the proposed method.

6.5 Other investigations on binary neural networks

The training curves for the different models and different optimization approaches are compared in Figs. 1 and 2, for DNN and CNN, respectively. There is a large degradation in the basic binary DNN or CNN, while the basic BDNN/BCNN converges much slower than the normal full precision deep models. In contrast, training binary neural networks with soft labels from a teacher model can converge much faster and achieve a better accuracy.

The weight parameter distribution is then compared for FPDNN and BDNN. The histogram distributions of FPDNN and BDNN are illustrated in Figs. 3 and 4, respectively. For the full precision DNN, most of the parameters are close to 0 (more than 80.0% parameters are within [−0.1, 0.1]). The same property on DNN sparseness was observed in Yu et al. (2012). For the proposed binary DNN, the distribution is changed. Note that the temporary floating-point parameters are stored in BDNN training as described in Section 4, so the statistics

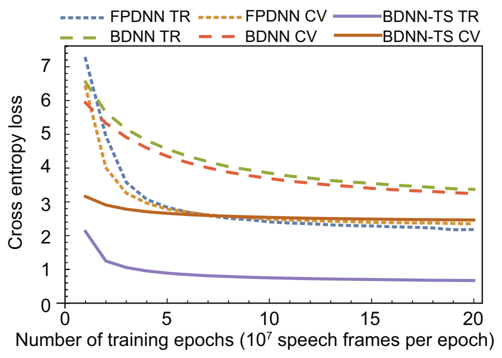

Fig. 1 Training curve comparison of different DNNs
References to color refer to the online version of this figure. FPDNN: full precision DNN; BDNN: binary DNN; BDNN-TS: binary DNN with the knowledge distillation from the FPDNN using teacher-student learning; TR: results in the training set; CV: results in the validation set

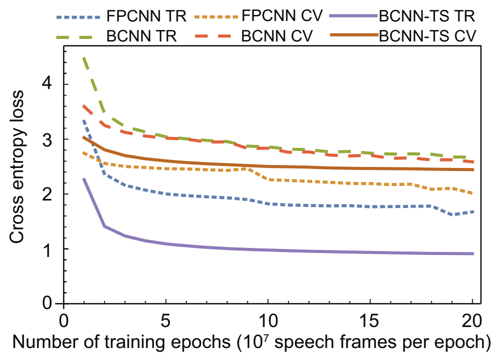

Fig. 2 Training curve comparison of different CNNs
References to color refer to the online version of this figure. FPCNN: full precision CNN; BCNN: binary CNN; BCNN-TS: binary CNN with knowledge distillation from the full precision very deep CNN using teacher-student learning; TR: results in the training set; CV: results in the validation set

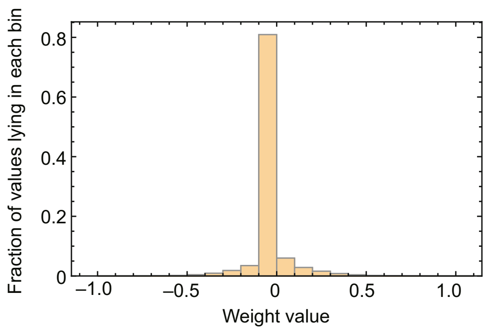

Fig. 3 Histogram of the weight values of a full precision DNN (FPDNN)

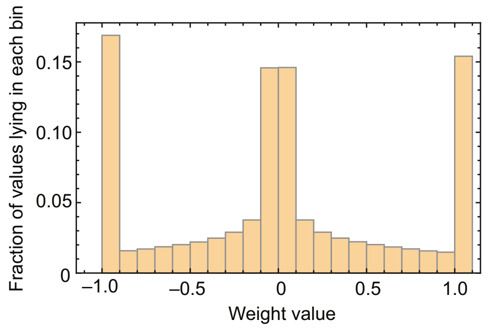

Fig. 4 Histogram of the weight values of a binary DNN (BDNN)

are performed on the finally optimized model before implementing binarization operation. There are still many parameters around 0, but the percentage is significantly decreased (fewer than 30.0% of parameters are within [−0.1, 0.1]). In contrast, more than 30.0% of parameters are close to −1.0 or 1.0. This observation also matches the optimization criterion in binary neural networks.

7 Conclusions and future work

To better deploy deep models on low-power embedded devices with limited computational resources, we have introduced binary neural networks for acoustic modeling in automatic speech recognition. With binary weights and activations during the inference, binary matrix multiplication provides a 5–7 times speedup over highly optimized floating-point matrix multiplication on modern CPUs and GPUs. This benefit results in a 3–4 times speedup for model inference in real scenarios. Different types of binary neural networks, including DNNs, CNNs, and VDCNNs, have been developed and compared for speech recognition. In addition, knowledge distillation from full precision deep models to binary models has been explored to further boost the performance of binary neural networks. Experiments on the standard Switchboard telephone speech recognition task showed that the proposed binary neural networks approach obtains considerable acceleration on model inference, while the binary DNN or binary CNN constrains the accuracy degradation to within 15.0%, compared to the normal full precision DNN or CNN. Particularly, for the binary CNN with binarization only on the convolutional layers, the WER degradation is very small and almost negligible with the proposed approach, which is very promising.

Currently, the proposed binary neural networks have been evaluated only on existing hardware, which were originally designed and highly optimized for normal numerical calculation. It is reasonable to speculate that the acceleration will be much larger if there are new hardware platforms specifically designed for binarization operation. Therefore, further implementation of binary neural networks on such specifically designed hardware would be valuable for future research. Moreover, integrating the proposed binarization with model sparseness reconstruction and implementing binarization on the recurrent neural networks (e.g., LSTM-RNN) are other interesting topics which will be conducted in our future work.

References

Bengio Y, Léonard N, Courville A, 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. https://arxiv.org/abs/1308.3432

Bi MX, Qian YM, Yu K, 2015. Very deep convolutional neural networks for LVCSR. 16th Annual Conf of Int Speech Communication Association, p.3259-3263.

Chen ZH, Zhuang YM, Qian YM, et al., 2017. Phone synchronous speech recognition with CTC lattices. *IEEE/ACM Trans Audio Speech Lang Process*, 25(1): 90-101. https://doi.org/10.1109/TASLP.2016.2625459

Chen ZH, Luitjens J, Xu HN, et al., 2018a. A GPU-based WFST decoder with exact lattice generation. https://arxiv.org/abs/1804.03243

Chen ZH, Liu Q, Li H, et al., 2018b. On modular training of neural acoustics-to-word model for LVCSR. IEEE Int Conf on Acoustics, Speech, and Signal Processing, p.4754-4758. https://doi.org/10.1109/ICASSP.2018.8461361

Chen ZH, Droppo J, Li JY, et al., 2018c. Progressive joint modeling in unsupervised single-channel overlapped speech recognition. *IEEE/ACM Trans Audio Speech Lang Process*, 26(1):184-196. https://doi.org/10.1109/TASLP.2017.2765834

Collobert R, Kavukcuoglu K, Farabet C, 2011. Torch7: a Matlab-like environment for machine learning. BigLearn NIPS Workshop.

Courbariaux M, Hubara I, Soudry D, et al., 2016. Binarized neural networks: training deep neural networks with weights and activations constrained to $+1$ or -1. https://arxiv.org/abs/1602.02830

Dahl GE, Yu D, Deng L, et al., 2012. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Trans Audio Speech Lang Process*, 20(1):30-42. https://doi.org/10.1109/tasl.2011.2134090

Denil M, Shakibi B, Dinh L, et al., 2013. Predicting parameters in deep learning. 26th Int Conf on Neural Information Processing Systems, p.2148-2156.

Duchi J, Hazan E, Singer Y, 2011. Adaptive subgradient methods for online learning and stochastic optimization. *J Mach Learn Res*, 12:2121-2159.

Goto K, van de Geijn RA, 2008. Anatomy of high-performance matrix multiplication. *ACM Trans Math Softw*, 34(3), Article 12. https://doi.org/10.1145/1356052.1356053

Gupta S, Agrawal A, Gopalakrishnan K, et al., 2015. Deep learning with limited numerical precision. Proc 32nd Int Conf on Machine Learning, p.1737-1746.

Hammarlund P, Martinez AJ, Bajwa AA, et al., 2014. Haswell: the fourth-generation Intel core processor. *IEEE Micro*, 34(2):6-20. https://doi.org/10.1109/MM.2014.10

Han S, Pool J, Tran J, et al., 2015. Learning both weights and connections for efficient neural network. Proc 28th Int Conf on Neural Information Processing Systems, p.1135-1143.

Han S, Kang JL, Mao HZ, et al., 2017. ESE: efficient speech recognition engine with sparse LSTM on FPGA. Proc ACM/SIGDA Int Symp on Field-Programmable Gate Arrays, p.75-84. https://doi.org/10.1145/3020078.3021745

He TX, Fan YC, Qian YM, et al., 2014. Reshaping deep neural network for fast decoding by node-pruning. Proc IEEE Int Conf on Acoustics, Speech, and Signal Processing, p.245-249. https://doi.org/10.1109/ICASSP.2014.6853595

Hinton G, Deng L, Yu D, et al., 2012. Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. *IEEE Signal Process Mag*, 29(6):82-97. https://doi.org/10.1109/msp.2012.2205597

Hinton G, Vinyals O, Dean J, 2015. Distilling the knowledge in a neural network. https://arxiv.org/abs/1503.02531

Hubara I, Courbariaux M, Soudry D, et al., 2016. Quantized neural networks: training neural networks with low precision weights and activations. https://arxiv.org/abs/1609.07061

Ioffe S, Szegedy C, 2015. Batch normalization: accelerating deep network training by reducing internal covariate shift. 32nd Int Conf on Machine Learning, p.448-456.

Jaitly N, Nguyen P, Senior A, et al., 2012. Application of pretrained deep neural networks to large vocabulary speech recognition. Proc 13th Annual Conf of the Int Speech Communication Association.

Kingma D, Ba J, 2014. Adam: a method for stochastic optimization. https://arxiv.org/abs/1412.6980

Li JY, Seltzer ML, Wang X, et al., 2017. Large-scale domain adaptation via teacher-student learning. Proc 18th Annual Conf of Int Speech Communication Association, p.2386-2390. https://doi.org/10.21437/Interspeech.2017-519

Low TM, Igual FD, Smith TM, et al., 2016. Analytical modeling is enough for high-performance BLIS. *ACM Trans Math Softw*, 43(2), Article 12. https://doi.org/10.1145/2925987

Lu L, Renals S, 2017. Small-footprint highway deep neural networks for speech recognition. *IEEE/ACM Trans Audio Speech Lang Process*, 25(7):1502-1511. https://doi.org/10.1109/TASLP.2017.2698723

Lu L, Guo M, Renals S, 2017. Knowledge distillation for small-footprint highway networks. Proc IEEE Int Conf on Acoustics, Speech and Signal Processing, p.4820-4824. https://doi.org/10.1109/ICASSP.2017.7953072

Mohamed AR, Dahl GE, Hinton GE, 2012. Acoustic modeling using deep belief networks. *IEEE Trans Audio Speech Lang Process*, 20(1):14-22. https://doi.org/10.1109/TASL.2011.2109382

Novikov A, Podoprikhin D, Osokin A, et al., 2015. Tensorizing neural networks. Advances in Neural Information Processing Systems, p.442-450.

Povey D, Ghoshal A, Boulianne G, et al., 2011. The Kaldi speech recognition toolkit. Proc IEEE Workshop on Automatic Speech Recognition and Understanding.

Qian YM, Woodland PC, 2016. Very deep convolutional neural networks for robust speech recognition. Proc IEEE Spoken Language Technology Workshop, p.481-488. https://doi.org/10.1109/SLT.2016.7846307

Qian YM, He TX, Deng W, et al., 2015. Automatic model redundancy reduction for fast back-propagation for deep neural networks in speech recognition. Proc Int Joint Conf on Neural Networks, p.1-6. https://doi.org/10.1109/IJCNN.2015.7280335

Qian YM, Bi MX, Tan T, et al., 2016. Very deep convolutional neural networks for noise robust speech recognition. *IEEE/ACM Trans Audio Speech Lang Process*, 24(12):2263-2276. https://doi.org/10.1109/TASLP.2016.2602884

Rastegari M, Ordonez V, Redmon J, et al., 2016. XNOR-Net: ImageNet classification using binary convolutional neural networks. Proc 14th European Conf on Computer Vision, p.525-542. https://doi.org/10.1007/978-3-319-46493-0_32

Sainath TN, Mohamed AR, Kingsbury B, et al., 2013. Deep convolutional neural networks for LVCSR. Proc IEEE Int Conf on Acoustics, Speech and Signal Processing, p.8614-8618. https://doi.org/10.1109/ICASSP.2013.6639347

Sak H, Senior A, Beaufays F, 2014. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. Proc 15th Annual Conf of Int Speech Communication Association, p.338-342.

Saon G, Kurata G, Sercu T, et al., 2017. English conversational telephone speech recognition by humans and machines. https://arxiv.org/abs/1703.02136

Sercu T, Puhrsch C, Kingsbury B, et al., 2016. Very deep multilingual convolutional neural networks for LVCSR. Proc IEEE Int Conf on Acoustics, Speech, and Signal Processing, p.4955-4959. https://doi.org/10.1109/icassp.2016.7472620

Wang YQ, Li JY, Gong YF, 2015. Small-footprint high-performance deep neural network-based speech recognition using split-VQ. Proc IEEE Int Conf on Acoustics, Speech and Signal Processing, p.4984-4988. https://doi.org/10.1109/ICASSP.2015.7178919

Xiong W, Droppo J, Huang X, et al., 2016. Achieving human parity in conversational speech recognition. https://arxiv.org/abs/1610.05256

Xiong W, Droppo J, Huang X, et al., 2017. The Microsoft 2016 conversational speech recognition system. Proc IEEE Int Conf on Acoustics, Speech, and Signal Processing, p.5255-5259. https://doi.org/10.1109/icassp.2017.7953159

Xue J, Li JY, Gong YF, 2013. Restructuring of deep neural network acoustic models with singular value decomposition. Proc 14th Annual Conf of Int Speech Communication Association, p.2365-2369.

Young S, Evermann G, Gales M, et al., 2006. The HTK Book. Cambridge University Engineering Department, Cambridge, UK.

Yu D, Seide F, Li G, et al., 2012. Exploiting sparseness in deep neural networks for large vocabulary speech recognition. Proc IEEE Int Conf on Acoustics, Speech, and Signal Processing, p.4409-4412. https://doi.org/10.1109/ICASSP.2012.6288897

Yu D, Xiong W, Droppo J, et al., 2016. Deep convolutional neural networks with layer-wise context expansion and attention. Proc 17th Annual Conf of Int Speech Communication Association, p.17-21. https://doi.org/10.21437/Interspeech.2016-251

Zhou SC, Wu YX, Ni ZK, et al., 2016. DoReFa-Net: training low bitwidth convolutional neural networks with low bitwidth gradients. https://arxiv.org/abs/1606.06160