Ph.D. DISSERTATION

Quantization of Deep Neural Networks for Improving the Generalization Capability

일반화 능력의 향상을 위한 깊은 신경망 양자화

BY

Sungho Shin

February 2020

DEPARTMENT OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Abstract

Deep neural networks (DNNs) achieve state-of-the-art performance for various applications such as image recognition and speech synthesis across different fields. However, their implementation in embedded systems is difficult owing to the large number of associated parameters and high computational costs. In general, DNNs operate well using low-precision parameters because they mimic the operation of human neurons; therefore, quantization of DNNs could further improve their operational performance. In many applications, word-length larger than 8 bits leads to DNN performance comparable to that of a full-precision model; however, shorter word-length such as those of 1 or 2 bits can result in significant performance degradation. To alleviate this problem, complex quantization methods implemented via asymmetric or adaptive quantizers have been employed in previous works.

In contrast, in this study, we propose a different approach for quantization of DNNs. In particular, we focus on improving the generalization capability of quantized DNNs (QDNNs) instead of employing complex quantizers. To this end, first, we analyze the performance characteristics of quantized DNNs using a retraining algorithm; we employ layer-wise sensitivity analysis to investigate the quantization characteristics of each layer. In addition, we analyze the differences in QDNN performance for different quantized network sizes. Based on our analyses, two simple quantization training techniques, namely *adaptive step size retraining* and *gradual quantization* are proposed. Furthermore, a new training scheme for QDNNs is proposed, which is referred to as high-low-high-low-precision (HLHLp) training scheme, that allows the network to achieve flat minima on its loss surface with the aid of quantization noise. As the name suggests, the proposed training method employs high-low-high-low precision for network training in an alternating manner. Accordingly, the learning rate is also abruptly changed at each stage. Our obtained analysis results include that the

proposed training technique leads to good performance improvement for QDNNs compared with previously reported fine tuning-based quantization schemes.

Moreover, the knowledge distillation (KD) technique that utilizes a pre-trained teacher model for training a student network is exploited for the optimization of the QDNNs. We explore the effect of teacher network selection and investigate that of different hyperparameters on the quantization of DNNs using KD. In particular, we use several large floating-point and quantized models as teacher networks. Our experiments indicate that, for effective KD training, softmax distribution produced by a teacher network is more important than its performance. Furthermore, because softmax distribution of a teacher network can be controlled using KD hyperparameters, we analyze the interrelationship of each KD component for QDNN training. We show that even a small teacher model can achieve the same distillation performance as a larger teacher model. We also propose the gradual soft loss reducing (GSLR) technique for robust KD-based QDNN optimization, wherein the mixing ratio of hard and soft losses during training is controlled.

In addition, we present a new QDNN optimization approach, namely *stochastic quantized weight averaging* (SQWA), to design low-precision DNNs with good generalization capability using model averaging. The proposed approach includes (1) floating-point model training, (2) direct quantization of weights, (3) capture of multiple low-precision models during retraining with cyclical learning rate, (4) averaging of the captured models, and (5) re-quantization of the averaged model and its fine-tuning with low learning rate. Additionally, we present a loss-visualization technique for the quantized weight domain to elucidate the behavior of the proposed method. Our visualization results indicate that a QDNN optimized using our proposed approach is located near the center of the flat minimum on the loss surface.

keywords: Quantized Deep Neural Networks, Fixed-point Optimization, Generalization Capability, High-low-high-low-precision Training, Stochastic Quantized Weight Averaging, Knowledge Distillation

student number: 2013-23122

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

1.1 Quantization of Deep Neural Networks

Deep neural networks (DNNs) employ artificial neurons that contain many synaptic weights and have a considerably good generalization capability for many applications across different fields [5, 6, 7]. However, significant memory requirements and computational costs hinder the implementation of DNN-based models for applications in limited resource environments such as on mobile phones or Internet of Things (IoT) devices. For example, state-of-the-art architectures, such as ResNet [5], DenseNet [8], and PyramidNet [9], contain 6.8, 25.6, and 116.4 million parameters, respectively. Quantization of DNNs is among the most popular and effective approaches to alleviate the abovementioned problem.

Fixed-point implementation of signal processing algorithms has long been of interest for VLSI-based design of multimedia and communication systems. Some early works on this used statistical modeling of quantization noise that had been originally developed for linear digital filters. Furthermore, a previously proposed simulation-based word-length optimization method utilized simulation tools to evaluate fixed-point performance of the system [10]. Ternary (+1, 0, -1) coefficients-based digital filters were used to eliminate multiplications at the cost of higher quantization noise.

The implementation of adaptive filters with ternary weights were developed, but it required oversampling to reduce quantization noise [11].

Fixed-point shallow neural network design has also been studied to reduce hardware implementation costs [12]. In [13], backpropagation simulation with 16-bit integer arithmetic was conducted for several applications such as NetTalk, Parity, and Protein. The authors conducted experiments with different number of hidden units, but the number of hidden units was relatively small. The integer simulations showed good results for NetTalk and Parity benchmarks, but not for Protein benchmark. Furthermore, with direct quantization of trained weights, this work also showed satisfactory operational neural network performance with 8-bit precision. An implementation with ternary weights was reported for neural network design with optical fiber networks [14]. In this ternary network design, the authors employed retraining after direct quantization of weights to improve performance of a shallow network.

Recently, fixed-point design of DNNs was revisited in [1] and [15], by which considerably good performance similar to floating-point models was achieved using the quantization training algorithm. Based on these works, further research has been conducted employing this quantization training scheme by combining various quantizers including uniform [16, 17, 18, 19, 20], asymmetric uniform [21], non-uniform [22], and differential [23, 24, 25, 26, 27] quantizers. Furthermore, elaborate techniques for optimization of DNN have been combined with quantization training, including knowledge distillation (KD) [2], weight normalization [28], and stochastic weight averaging [3]. Quantization training of DNNs with KD loss was studied in [29] and [30]. Apprentice [30] showed that employing a pre-trained full-precision teacher and student models are advantageous to achieve high accuracy of the quantized student network. Moreover, weight normalization can improve the performance of quantized DNNs (QDNNs) because it can help in eliminating long-tail distribution of network weights [31]. Employing a stochastic weight averaging technique to train QDNNs in low-precision (*i.e.*, 8-bit) environments, including their weights, activations, and gra-

dients, can improve their training efficacy [32].

1.2 Generalization Capability of DNNs

Unlike traditional machine learning algorithms, DNNs trained with stochastic gradient descent (SGD) do not easily overfit even when the the network size increases significantly, *i.e.*, these neural networks have a high generalization capability. Many previous studies have closely explored the reason for this high generalization capability [33, 34, 35]. An early work claimed that a flat minimum of the loss or error function in such networks is the cause of their typically high generalization capability [33]. In addition, a recent work revealed that the ratio of learning rate to batch size is key to determine the flatness of the loss surface of a neural network [34]. A flatness of the trained model is evaluated with the sharpness of the loss function via a heuristic metric (*i.e.*, $\epsilon-$sharpness) in [35].

Thus, visualization of the loss surface is useful for understanding the generalization capability of DNNs. A three-dimensional visualization method for the loss surface of a DNN was proposed by [36]; they employed a filter normalization method to represent loss function curvatures. In addition, they showed that the residual connection in ResNet architectures [5] leads to flattening of their loss surfaces. Visualizing three models on a single loss surface is also exploited to understand the relationship between the networks [37, 3]. They found that the minima of these models on the loss surface trained using SGD were closely-connected.

1.3 Improved Generalization Capability of QDNNs

As discussed in the previous subsections, the quantization method for, and generalization capability of DNNs have been actively studied in recent years. In the same vein, in this study, we attempt to improve the performance of QDNNs by increasing their generalization capability. To this end, in Chapter 2, we investigate the performance

resiliency of QDNNs for retraining-based quantization [1] with varying widths and depths. We observed that a small quantized model shows more performance degradation than a large one. Therefore, we proposed two simple quantization methods that perform well for small networks. The first method involves adjusting the quantization step size, while the second involves gradually decreasing the word-length from 8 to 2 bits during quantization training.

Most previous studies related to the generalization capability of DNNs trained with SGD have typically reported that batch size and learning rate are important hyperparameters that determine the flatness of minima on the loss surface of those DNNs. We showed that learning rate and quantization bit precision is related to the generalization capability of the quantized models and proposed a new QDNN training method, referred to as high-low-high-low-precision (HLHLp), which controls both learning rate and word-length during quantization training; this has been discussed in Chapter 3.

Chapter 4 presents detailed information regarding quantization training with KD. In particular, we showed that the distribution produced by a teacher network is more important than the performance of the teacher network itself to obtain high accuracy of student model. In addition, we determine that if the produced distribution is appropriately adjusted using the hyperparameters for KD, then the student network can be trained well even with a poor teacher network.

In Chapter 5, we discuss our proposed stochastic quantized weight averaging (SQWA) method that applies the recently reported stochastic weight averaging (SWA) technique to quantization training. The SWA technique employs cyclical learning rate scheduling and captures models when the learning rate is lowest in the cycle. The captured models are then averaged to obtain the final model. This averaging technique leads to the resulting model located at the center of the basin on the loss surface. We observe that this significantly improves the performance of QDNNs. In general, quantization can be interpreted as adding quantization noise to the weights of a DNN. The lower the bit-precision is, the stronger the quantization noise is. Thus, quantized weights may

locate the model at the minima's edge on the basin. Thus, forcing to locate the network to the center of the basin can greatly improve the generalization capability of the quantized model. Furthermore, we presented a loss surface visualization method for three QDNNs on a single loss surface in order to analyze their inter-relationship on the quantization domain.

1.4 Outline of the Dissertation

This dissertation is organized as follows. Chapter 2 analyzes the performance resiliency of QDNNs using the retraining-based quantization. In addition, we discuss two proposed simple techniques, namely *adaptive step size retraining* and *gradual quantization*, to improve the performance of small QDNNs. In Chapter 3, we present information showing that the learning rate and quantization precision are important hyperparameters that affect the loss surface of QDNNs. Based on our analyses, we introduce the HLHLp training scheme that helps to improve the generalization capability of QDNNs. In our study, we employ KD to improve the performance of QDNNs: this is discussed in Chapter 4. Furthermore, our proposed SQWA training scheme is described in Chapter 5. Moreover, we propose a visualization method for QDNNs. Finally, Chapter 6 concludes our dissertation.

It should be noted that significant portions of the information presented in Chapters 2 and 3 was previously published in [38, 18, 19, 39]; in addition, Chapters 4 and 5 have been submitted to ICASSP 2020 and CVPR 2020, respectively.

Chapter 2

Analysis of Fixed-point Quantization of Deep Neural Networks

2.1 Introduction

Real-time implementation of DNNs usually demands many arithmetics and weight fetch operations. Thus, word-length optimization is needed in embedded applications to reduce the strength of arithmetic and the size of the weight storage. However, direct quantization of deep neural networks usually does not show satisfactory performance with very low precision weights. Thus, retraining on quantized domain should be employed. With the algorithm, even ternary valued weights (+1, 0, and -1) for a DNN have yielded satisfactory performance [18, 1]. Recently, several improved fixed-point optimization methods are developed by employing retraining based fine tuning [40, 41]. Also, VLSI and FPGA based deep neural networks have been implemented using fixed-point weights [42, 43, 44, 45, 46].

In this chapter, we try to investigate the retraining algorithm that can recover the performance of feed-forward DNNs (FFDNNs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs) with low-precision weights. For this study, the network complexity is changed to analyze their effects on the performance

gap between floating-point and retrained low-precision fixed-point deep neural networks. We also conduct layer-wise sensitivity analysis to investigate which layer is more robust to quantization.

We conduct our experiments with an FFDNN for phoneme recognition, a CNN for image classification, and RNNs for phoneme recognition and language modeling. To control the network size, not only the number of units in each layer but also the number of hidden layers are varied in the FFDNN. For CNN, the number of feature maps for each layer and the number of layers are both changed. The RNN employs the long short-term memory (LSTM) model to analyze the sensitivity of each layer. This analysis intends to find an insight into the knowledge representation capability of highly quantized networks, and also provides a guideline to network size and word-length determination for efficient hardware implementation of DNNs.

Based on the analysis, improved retraining algorithms are developed for fixed-point optimization of deep neural networks. The previous works determine the optimum quantization step size based on the distribution of floating-point weights and freeze the step-size during the retraining period [1, 40]. The proposed algorithm adaptively determines the step-size at the re-quantization step during retraining. Since the weight values change much at the beginning of retraining, this approach is especially effective when applied at initial retraining epochs. In order to change the weight values less abruptly, we also propose and evaluate the gradual quantization method. In this scheme, floating-point weights are converted to, for example, 6-bit weights, which are then converted to 4-bit weights, and so on. We evaluate the proposed schemes in three different networks: FFDNNs, CNNs, and RNNs. The proposed methods yielded better results compared to the previous retrain-based quantization schemes.

2.2 Fixed-point Performance Analysis of Deep Neural Networks

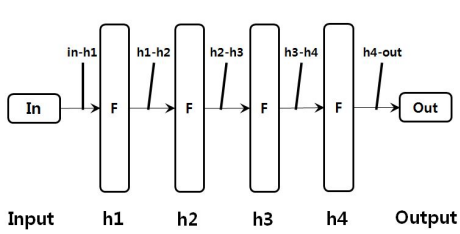
This section explains the design of FFDNN, CNN, and RNN. We also review the fixed-point optimization procedure and analyze their resilient properties with the retraining algorithm [1].

2.2.1 Model Design of Deep Neural Networks

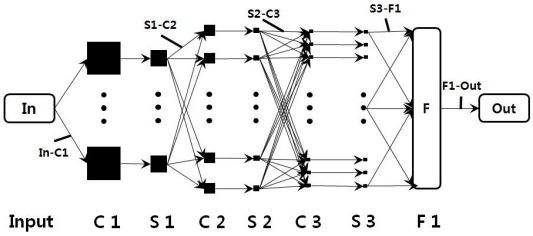
To analyze the properties of DNN under quantization, we employ three tasks including phoneme recognition, image classification, and language modeling. We use FFDNN and RNN for phoneme recognition, CNN for image classification, and language modeling for RNN. The FFDNN, CNN, and RNN architectures are depicted in Figure 2.1.

The reference FFDNN has four hidden layers. Each of the hidden layers has N_h units; the value of N_h is changed to control the complexity of the network. We conduct experiments with the N_h size of 32, 64, 128, 256, 512, and 1024. The number of hidden layers is also reduced. The input layer of the network has 1,353 units to accept 11 frames of a Fourier-transform-based filter-bank with 40 coefficients (+energy) distributed on a mel-scale, together with their first and second temporal derivatives. The output layer consists of 61 softmax units which correspond to 61 target phoneme labels. Phoneme recognition experiments were performed on the TIMIT corpus. The standard 462 speaker set with all SA records removed was used for training, and a separate development set of 50 speakers was used for early stopping. Results are reported for the 24-speaker core test set.

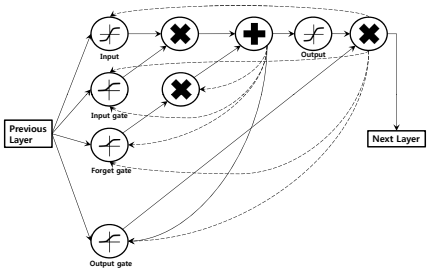
The CNN consists of three convolution and pooling layers and a fully connected hidden layer with 64 units, and the output has 10 softmax units. We control the number of feature maps in each convolution layer. The reference size has 32-32-64 feature maps as used in [47]. We did not perform any preprocessing and data augmentation such as zero-phase component analysis (ZCA) whitening and global contrast normal-



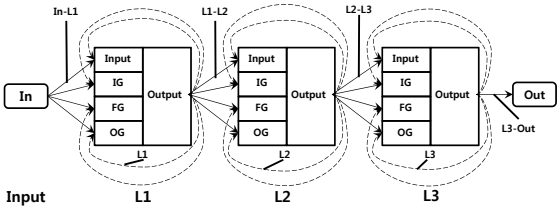
(a) A FFDNN with 4 hidden layers.



(b) A CNN with 3 convolution layers and 1 fully-connected layers.



(c) A structure of LSTM layer



(d) An RNN with 3 LSTM layers.

Figure 2.1: The architectures of the DNNs that we employ in this chapter. The text in the each figure represents sensitivity analysis group. In (c), each circle represents one layer which is a part of the LSTM. A dotted line means a backward path and a solid line is a forward path. The plus and multiplication signs show aggregation functions for summing and multiplication, respectively. The graph in the circles means an activation function for logistic sigmoid or tanh.

ization [48]. To know the effects of network size variation, the number of feature maps is reduced or increased. The configurations of the feature maps used for the experiments are 8-8-16, 16-16-32, 32-32-64, 64-64-128, 96-96-192, and 128-128-256. The number of feature map layers is also changed, resulting in 32-32-64, 32-64, and 64 map configurations. Note that the fully connected layer on the CNN is not changed.

We employ two RNNs for phoneme recognition and language modeling. For both applications, we construct three LSTM layers. The RNNs for acoustic and language models have 512 and 1024 memory cells, respectively. For the data preprocessing in acoustic modeling, we employ exactly the same setup with the FFDNN case. English Wikipedia dataset is used for language modeling. Since we use the character-level language model, the input and output layers contain 256 linear units to accept ASCII code.

2.2.2 Retrain-based Weight Quantization

The retrain based quantization method includes the fixed-point conversion process inside of the training procedure so that the network learns the quantization effects [1]. This method shows much better performance when the number of bits is small.

A symmetric uniform quantization function, $Q(\cdot)$, is defined as follows:

$$Q(w) = sgn(w) \cdot \Delta \cdot min\left(\left\lfloor \frac{|w|}{\Delta} + 0.5 \right\rfloor, \frac{M-1}{2} \right) = \Delta \cdot z, \tag{2.1}$$

where $sgn(\cdot)$ is the sign function, Δ is a quantization step size, w is the set of the floating-point weights, and M represents the number of quantization levels. Note that M is normally an odd number since the weight values can be positive or negative. When M is 5, the weights are represented by -2Δ, $-\Delta$, 0, Δ, and 2Δ, which can be stored in 3 bits.

For selecting a proper step size Δ, the L2 error minimization criteria is applied as

adopted in [1]. The quantization error E is represented as follows:

$$E = \frac{1}{2}\sum_{i=1}^{N}\big(Q(w_i) - w_i\big)^2 = \frac{1}{2}\sum_{i=1}^{N}\big(\Delta \cdot z_i - w_i\big)^2, \tag{2.2}$$

where N is the number of weights in each layer, w_i is the i-th weight value in the floating-point precision, and z_i is the integer membership of w_i. The quantization error E is minimized by the following two step iterative computation.

$$\begin{aligned} \boldsymbol{z}^{(t)} &= \underset{\boldsymbol{z}}{\operatorname{argmin}}\, E(\boldsymbol{w}, \boldsymbol{z}, \Delta^{(t-1)}) \\ &= sgn(w_i) \cdot min\left(\left\lfloor \frac{|w_i|}{\Delta^{(t-1)}} + 0.5 \right\rfloor, \frac{M-1}{2}\right) \end{aligned} \tag{2.3}$$

$$\Delta^{(t)} = \underset{\Delta}{\operatorname{argmin}}\, E(\boldsymbol{w}, \boldsymbol{z}^{(t)}, \Delta) = \frac{\sum_{i=1}^{N} w_i \cdot z_i^{(t)}}{\sum_{i=1}^{N}\big(z_i^{(t)}\big)^2}, \tag{2.4}$$

where the superscript (t) indicates the iteration step. Equation (2.3) can be computed using Equation (2.1) and Equation (2.4) can be solved by using the derivative of the error with respect to $\Delta^{(t)}$ to be zero. The iteration stops when $\Delta^{(t)}$ is converged.

Activation can be quantized with a symmetric uniform quantizer. In the standard DNNs, the popular activation functions are logistic sigmoid, rectified linear unit (Relu), or tanh.

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}} \tag{2.5}$$

$$\tanh(x) = \frac{e^{x}+e^{-x}}{e^{x}-e^{-x}} \tag{2.6}$$

$$\text{Relu}(x) = \max(0, x) \tag{2.7}$$

The output ranges of the sigmoid and the tanh are limited by 0 to 1 and -1 to 1, respectively. The quantization step size Δ is determined by the quantization level M. For example, if the signal word-length is two bits (M is four), the quantization points are 0/3, 1/3, 2/3, and 3/3 for the sigmoid and -1/1, 0/1, and 1/1 for the tanh. However signals of Relu units are not bounded and their quantization range should be determined

empirically. Thus, Relu can follows the same scheme with the weight quantization strategy.

After the direct quantization, the retraining procedure follows. We maintain both floating-point and quantized weights, since applying the backpropagation algorithm directly to quantized weights usually does not work. The reason is that the amount of weights to be changed on each training step is much smaller than the quantization step size Δ. The entire retraining based quantization algorithm can be described as follows:

$$net_i = \sum_{j \in A_i} w_{ij}^{(q)} y_j^{(q)}$$

$$y_i^{(q)} = R_i(\phi_i(net_i)) \tag{2.8}$$

$$\delta_j = \phi_j^{'}(net_j) \sum_{i \in P_j} \delta_i w_{ij}^{(q)} \tag{2.9}$$

$$\frac{\partial E}{\partial w_{ij}} = -\delta_i y_j^{(q)} \tag{2.10}$$

$$w_{ij,new} = w_{ij} - \alpha \left\langle \frac{\partial E}{\partial w_{ij}} \right\rangle$$

$$w_{ij,new}^{(q)} = Q_{ij}(w_{ij,new}) \tag{2.11}$$

where net_i is the summed input value of the unit i, δ_i is the error signal of the unit i, w_{ij} is the weight from the unit j to the unit i, y_j is the output signal of the unit j, α is the learning rate, A_i is the set of units anterior to the unit i, P_j is the set of units posterior to the unit j, $R(\cdot)$ is the signal quantizer, $Q(\cdot)$ is the weight quantizer, $\phi(\cdot)$ is the activation function, the superscript (q) indicates quantization, and $\langle \cdot \rangle$ is an average operation via the mini-batch. Equation (2.8), (2.9), (2.10), and (2.11) represent the forward, backward, gradient calculation, and weights update phases each.

2.2.3 Quantization Sensitivity Analysis

DNNs usually contain millions of weights and thousands of signals. Therefore, it is necessary to group them according to their range and the quantization sensitivity [10]. Fortunately, a neural network can easily be grouped layerwisely. Throughout this sen-

sitivity check, we can identify which layer in the neural network needs more bits for quantization. For example, the RNN for the phoneme recognition contains three hidden LSTM layers. Thus the weights can be grouped into 7 groups, which are In-L1, L1, L1-L2, L2, L2-L3, L3, and L3-Out groups, where In-L1 connects input and the first LSTM layer and L1 is the recurrent path in the first LSTM layer. Figure 2.1 (a), (b), and (d) illustrate the weight and signal grouping for FFDNN, CNN, and RNN, respectively. In the sensitivity analysis, we only quantize the selected group while those in other groups are remaining in full-precision.

2.2.4 Empirical Analysis

Results of Sensitivity Analysis

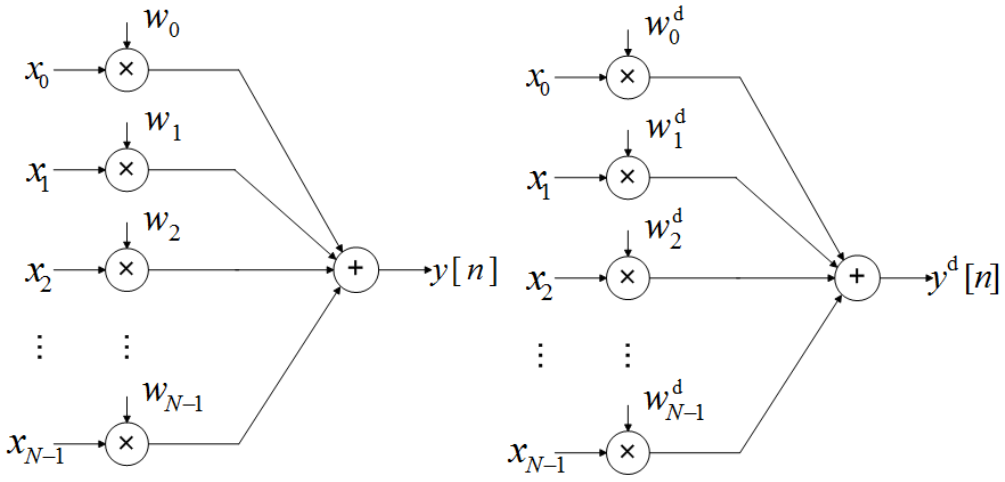
The quantized weight can be represented as follows,

$$w_i^q = w_i + w_i^d \tag{2.12}$$

where w_i^d is the distortion of each weight due to quantization. In the direct quantization, we can assume that the distortion w_i^d is not dependent each other.

Consider a computation procedure for a unit in a hidden layer, the signals from the previous layer are summed up after multiplication with the weights as illustrated in Figure 2.2 (a). We can also assemble a model for distortion, which is shown in Figure 2.2 (b). In the distortion model, since w_i^d is independent of each other, we can assume that the effects of the summed distortion are reduced according to the random process theory. This analysis means that the quantization effects are reduced when the number of units in the anterior layer increases, but slowly.

Figure2.3 (a) illustrates the performance of the FFDNN with floating-point arithmetic, 2-bit direct quantization of all the weights, and 2-bit direct quantization only on the weight group 'In-h1', 'h1-h2', and 'h4-out'. Consider the quantization performance of the 'In-h1' layer, the phone-error rate is higher than the floating-point result with an almost constant amount, about 10%. Note that the number of input to the



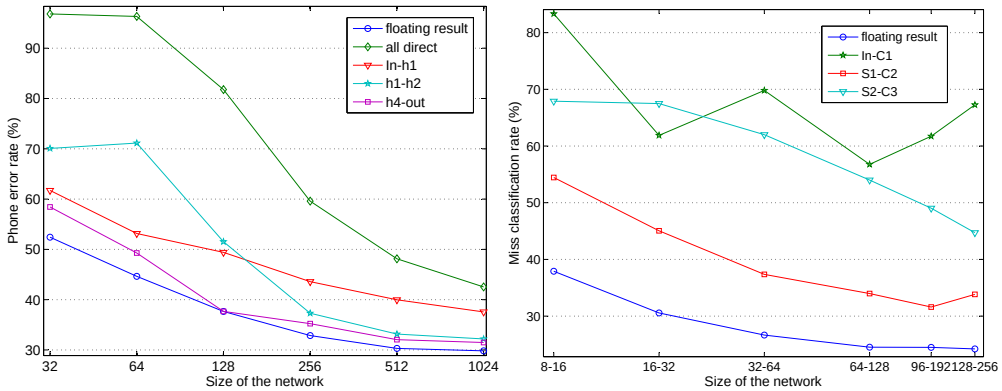
(a) Floating-point (b) Distortion

Figure 2.2: Computation model for a unit in the hidden layer j.

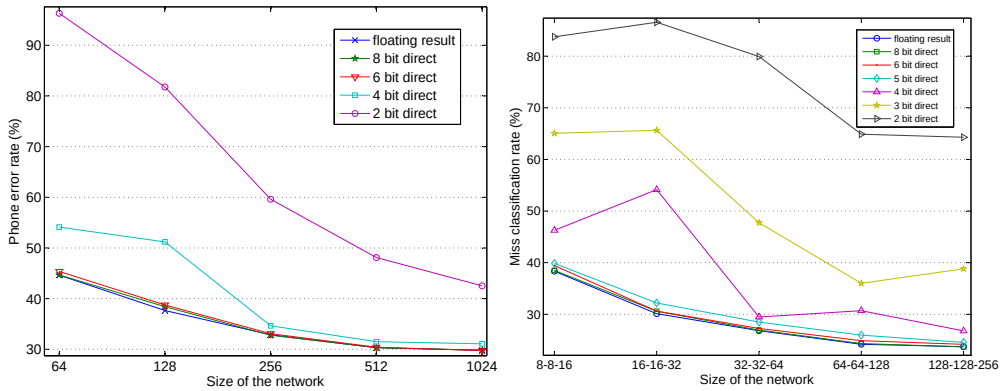
'In-h1' layer is fixed, 1353, regardless of the hidden unit size. Thus, the amount of distortion delivered to each unit of the hidden layer 1 can be considered unchanged.

Figure2.3 (a) also shows the quantization performance on 'h1-h2' and 'h4-out' layers, which informs the trend of the reduced gap to the floating-point performance as the network size increases. This can be explained by the sum of the increased number of independent distortions when the network size grows. The performance of all 2-bit quantization also shows a similar trend of the reduced gap to the floating-point performance. But, apparently, the performance of 2-bit directly quantized networks is not satisfactory.

In Figure2.3 (b), a similar analysis is conducted to the CNN with direct quantization when the number of feature maps increases or decreases. In the CNN, the number of input to each output is determined by the number of input feature maps and the kernel size. For example, at the first layer C1, the number of input signals for computing one output is only 75 (=3×25) regardless of the network size, where the input map size is always 3 and the kernel size is 25. However, at the second layer C2, the number of input feature maps increases as the network size grows. When the feature map of

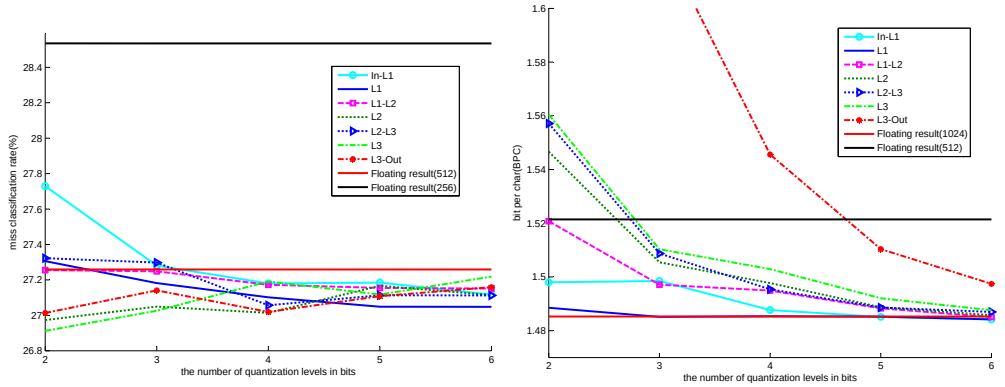


(a) FFDNN (b) CNN



(c) FFDNN (d) CNN

Figure 2.3: Results of the sensitivity analysis for the FFDNN (a) and the CNN (b). (c) and (d) show the performance of the direct quantization with multiple precision for the FFDNN and the CNN, respectively.



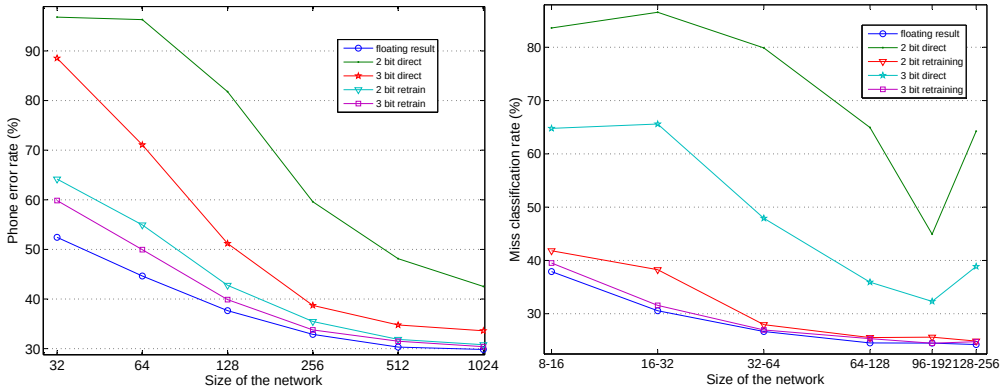
(a) (b)

Figure 2.4: Layerwise sensitivity analysis results of the weights in the phoneme recognition and language model for RNN examples. In (a), the red and black horizontal lines indicate the floating-point results for 512 and 256 LSTM size each. Similarly in (b), they indicate 1024 and 512 LSTM size each.

32-32-64 is considered, the number of inputs for the C2 layer grows to 800 (=32×25). Thus, we can expect a reduced distortion as the number of feature maps increases.

Figure 2.3 (c) shows the performance of direct quantization with 2, 4, 6, and 8-bit precision when the network complexity varies. In the FFDNN, 6-bit direct quantization seems enough when the network size is larger than 128. But, small FFDNNs demand 8 bits for obtaining a near floating-point performance. The CNN in Figure 2.3 (d) also shows the similar trend. The direct quantization requires about 6 bits when the feature map configuration is 16-16-32 or larger.

Figure 2.4 (a) shows the result of the layerwise sensitivity analysis of RNN phoneme recognition. The original phoneme error rate was 27.26% and 28.63% for the LSTM RNN with 512 and 256 memory cells, respectively. The results indicate that all layers except the input-LSTM1 group shows almost the same quantization sensitivity and requires only two bits for weight representation. Input-LSTM1 weights group demands at least three quantization bits. In the signal sensitivity analysis, all layers need only three to five quantization bits.



(a) DNN (b) CNN

Figure 2.5: Comparison of retrain-based and direct quantization for DNN. All the weights are quantized with ternary and 7-level weights.

Figure 2.4 (b) shows the layerwise sensitivity analysis results of the RNN language model. The bit per character (BPC) of the floating-point language model with the layer size of 1024 was 1.485. The analysis shows that the most sensitive weights group of the network is the L3-Out layer. Note that the last RNN layer is connected to the softmax layer. The first weights group shows low sensitivity when compared to the phoneme recognition example. This is because the one-hot encoding of the ASCII code is used for the input in this language model. The LSTM RNN has two types of paths, the forward and the recurrent connections. For both paths, the layer that is close to the output layer shows higher quantization sensitivity. The sensitivity analysis of signals shows that a minimum of four or five bits is needed for the activation quantization.

Results on Fully Quantized DNNs

The fixed-point performance of the FFDNN is shown in Figure 2.5 (a) when the number of hidden units in each layer varies. The performance of direct 2 bits (ternary levels), direct 3 bits (7-levels), retrain-based 2 bits, and retrain-based 3 bits are compared with the floating-point simulation. We can find that the performance gap between the floating-point and the retrain-based fixed-point networks converges very fast as the

network size grows. Although the performance gap between the direct and the floating-point networks also converges, the rate of convergence is significantly different. In this figure, the performance of the floating-point network almost saturates when the network size is about 1024. Note that the TIMIT corpus that is used for training has only 3 hours of data. Thus, the network with 1024 hidden units can be considered in the 'training-data limited region'. Here, the gap between the floating-point and fixed-point networks almost vanishes when the network is in the 'training-data limited region'. However, when the network size is limited, such as 32, 64, 128, or 256, there is some performance gap between the floating-point and highly quantized networks even if retraining algorithm is performed.

The similar experiments are conducted for the CNN while varying the number of the feature maps, and the results are shown in Figure 2.5 (b). The configurations of the feature maps used for the experiments are 8-8-16, 16-16-32, 32-32-64, 64-64-128, 96-96-192, and 128-128-256. The size of the fully connected layer is not changed. In this figure, the floating-point and the fixed-point performances with retraining also converge very fast as the number of feature maps increases. The floating-point performance saturates when the feature map size is 128-128-256, and the gap is less than 1% when comparing the floating-point and the retrain-based 2-bit networks. However, also, there is some performance gap when the number of feature maps is reduced. This suggests that a fairly high-performance feature extraction can be designed even using very low-precision weights if the number of feature maps can be increased.

Using the sensitivity analysis results, we construct a fully quantized LSTM RNN and the results are shown in Table 2.1 and Table 2.2 for phoneme recognition and language modeling, respectively. The results of the phoneme recognition show that the phoneme error rate of 27.74% is achieved with only about 10% of the weight capacity of the floating-point model. When compared to the phoneme recognition example, the language modeling needs more quantization bits over the results as observed in the sensitivity analysis. A reasonable BPC was achieved with two more bits for both

weights and signals than the sensitivity analysis result. The memory space needed for weights is only 16.75% when compared to the floating-point model.

Fixed-point performances when varying the depth

It is well known that increasing the depth usually results in positive effects on the performance of a DNN [49]. The network complexity of a DNN is changed by increasing or reducing the number of hidden layers or feature map levels. The result of fixed-point and floating-point performances when varying the number of hidden layers for the FFDNN is summarized in Table 2.3. The number of units in each hidden layer is 512. This table shows that both the floating-point and the fixed-point performances of the FFDNN increase when adding hidden layers from 1 to 4. The performance gap between the floating-point and the fixed-point networks shrinks as the number of levels increases.

The network complexity of the CNN is also varied by reducing the number of levels as reported in Table 2.4. As expected, the performance of both the floating-point and retrain-based low-precision networks degrades as the number of levels is reduced. The performance gap between them is very small with 7-level quantization for all feature map levels.

These results for the FFDNN and the CNN with a varied number of levels also indicate that the effects of quantization can be much reduced by retraining when the network contains some redundant complexity.

Discussion

In this section, we control the network size by changing the number of units in each hidden layer, the number of feature maps, or the number of levels. At any case, reduced network complexity lowers the resiliency to quantization. This work seems to be directly related to several network optimization methods, such as pruning, fault tolerance, and decomposition [50, 41, 51, 52]. In the pruning, retraining of weights is

Table 2.1: Frame-level phoneme error rates (%) on the test set with the TIMIT phoneme recognition with the RNN. Numbers in the parenthesis indicate the ratio of the weights capacity compared to the floating-point version

Layerwise quantization bits		FER(%)	
Weights bits (In-L1, L1, L1-L2, L2, L2-L3, L3, L3-Out)	Signal bits (Input, L1, L2, L3)	Direct	Retrain
3-2-2-2-2-2-2 (6.39%)	4-4-3-5	48.00	28.74
4-3-3-3-3-3-3 (9.52%)	4-4-3-5	34.37	28.87
3-2-2-2-2-2-2 (6.39%)	5-5-4-6	31.65	27.79
4-3-3-3-3-3-3 (9.52%)	5-5-4-6	31.54	27.74

Table 2.2: Bit per character on the test set with the English Wikipedia language model with the RNN. Numbers in the parenthesis indicate the ratio of the weights capacity compared to the floating-point version

Layerwise quantization bits		BPC	
Weights bits (In-L1, L1, L1-L2, L2, L2-L3, L3, L3-Out)	Signal bits (L1, L2, L3)	Direct	Retrain
2-2-3-4-4-4-6 (10.52%)	6-6-7	3.623	1.546
3-3-4-5-5-5-7 (13.64%)	6-6-7	1.641	1.510
4-4-5-6-6-6-8 (16.75%)	6-6-7	1.517	1.499
2-2-3-4-4-4-6 (10.52%)	7-7-8	3.613	1.545
3-3-4-5-5-5-7 (13.64%)	7-7-8	1.639	1.508
4-4-5-6-6-6-8 (16.75%)	7-7-8	1.517	1.499

Table 2.3: Depth change in DNN

Number of layers (Floating-point result)	**Quant Level**	**Direct**	**Retraining**	**Difference**
4	3-level	48.13%	31.86%	1.55%
(30.31%)	7-level	34.77%	31.49%	1.18%
3	3-level	49.27%	33.05%	2.24%
(30.81%)	7-level	36.58%	31.72%	0.91%
2	3-level	47.74%	33.89%	2.38%
(31.51%)	7-level	36.99%	33.04%	1.53%
1	3-level	69.88%	38.58%	3.91%
(34.67%)	7-level	56.81%	36.57%	1.90%

conducted after zeroing small valued weights. The effects of pruning, fault tolerance, and network decomposition efficiency would be dependent on the redundant representation capability of DNNs.

This study can be applied to hardware efficient DNN design. For design with limited hardware resources, when the size of the reference DNN is relatively small, it is advised to employ a very low-precision arithmetic and, instead, increase the network complexity as much as the hardware capacity allows. But, when the DNNs are in the performance saturation region, increasing the arithmetic precision is not recommended because growing the 'already-big' network size brings almost no performance advantages.

Even though the retraining based quantization can alleviate the loss due to quantization, small networks are not resilient. Thus, it is very important to find the quantization method that also works well with small networks.

Table 2.4: Depth change in CNN

Layer (Floating-point result)	Quant Level	Direct	Retraining	Difference
64	3-level	72.95%	35.37%	1.18%
(34.19%)	7-level	46.60%	34.15%	-0.04%
32-64	3-level	55.30%	29.51%	0.22%
(29.29%)	7-level	39.80%	29.32%	0.03%
32-32-64	3-level	79.88%	27.94%	1.07%
(26.87%)	7-level	47.91%	26.95%	0.08%

2.3 Step Size Adaptation and Gradual Quantization for Retraining of Deep Neural Networks

2.3.1 Step-size adaptation during retraining

As described in Section 2.2.2, the conventional method freezes the step size during the retraining. However, in many cases, the weight values change much by retraining. Note that the amount of change decreases as the retraining iteration progresses. Thus, it is advantageous for improving the performance to adjust the quantization step size during the retraining. Especially, the need for step size adaptation is greater at the beginning of retraining. The proposed scheme adds the determination of Δ_{new} at the weight update stage of Figure 2.6.

We update the quantization step size during retraining by using the L2 error minimization between the floating-point and fixed-point weights. We consider two different quantization step size update timing. The first one is 'epoch-level update', and the other is '1 epoch update & fix'. The 'epoch-level update' changes the step size at every epoch. The '1 epoch update & fix' updates the step size only during one or two epochs and freezes it for the remaining epochs. In our empirical evaluation, the first scheme is

- Quantization step size determining:

$$\Delta = QStep(\boldsymbol{w}) = \underset{\Delta}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{N} \big(Q(w_i, \Delta) - w_i\big)^2$$

- Quantized weights:

$$\boldsymbol{w}^{(q)} = Q(\boldsymbol{w}, \Delta) = sgn(\boldsymbol{w}) \cdot \Delta \cdot min\left(\left\lfloor \frac{|\boldsymbol{w}|}{\Delta} + 0.5 \right\rfloor, \frac{M-1}{2} \right)$$

- Forward:

$$net_i = \sum_{j \in A_i} w_{ij}^{(q)} y_j$$

$$y_i = \phi_i(net_i)$$

- Backward:

$$\delta_j = \phi_j^{'}(net_j) \sum_{i \in P_j} \delta_i w_{ij}^{(q)}$$

- Gradient calculation:

$$\frac{\partial E}{\partial w_{ij}} = -\delta_i y_j$$

- Weights update:

$$w_{ij,new} = w_{ij} - \alpha \frac{\partial E}{\partial w_{ij}}$$

$\Delta_{new} = QStep(w_{ij,new})$ **(Proposed scheme)**

$$w_{ij,new}^{(q)} = Q_{ij}(w_{ij,new}, \Delta_{new})$$

Figure 2.6: Overall fixed-point retraining algorithm with step size adaptation scheme, where Δ is the quantization step size, $\boldsymbol{w}$ is the weight groups, net_i is the summed input value of unit i, δ_i is the error signal of unit i, M is quantization points (2-bit quantization = 3 points, 3-bit quantization = 7 points), α is the learning rate, N is the number of the weights in each layer, A_i and P_j represent the activation of next and previous layer, $\phi(\cdot)$ is the activation function, E is the output error, and superscript (q) means the value is quantized.

good for FFDNNs, but the second one shows better results for CNNs and RNNs.

2.3.2 Gradual quantization scheme

We also propose another step size adaptation approach which is similar to the curriculum learning. The curriculum learning is a training strategy to move the goal from an easy level to a more complex one gradually [53]. One of the important points in curriculum learning is how to organize the tasks from easy to complex ones. We consider that the fixed-point optimization with a small number of bits is a more difficult problem than that with a large one.

In the proposed scheme, we begin fixed-point optimization with fairly high precision, such as 6 bits, and then keep lowering the word-length by one bit with retraining for each precision. At each retraining process with a given precision, we also combine the proposed quantization step size adaptation scheme. The experiments are conducted for FFDNNs.

2.3.3 Experimental Results

The proposed step size adaptation is evaluated for three applications. We employ FFDNNs for phoneme recognition, CNNs for house number recognition, and RNNs for language modeling. To analyze the effect of step size adaptation, we change the size of networks and their word lengths.

Phoneme recognition using feed-forward deep neural networks

The FFDNN is trained with the TIMIT corpus [54], and the detailed experimental condition for the data preprocessing is the same with [55]. We construct 11 consecutive frames as the network input. The output layer supports 61 labels, and the labels are merged into 39 classes for the final evaluation. For performance evaluation, the number of units in each layer increases from 64 to 1024. We train the floating-point networks using the stochastic gradient descent (SGD) with Nesterov momentum [56].

The learning rate decreases from 2e-3 to 3.90625e-6 with a factor of 2 when the development set does not show improvements for 4 consecutive evaluations. For fixed-point networks training, all other conditions are the same with the floating-point case but the initial learning rate is 5e-4.

The results of fixed-point optimization for FFDNNs with and without the step size adaptation are reported in Table 2.5. The experiments also show the results with batch normalization (BN) [57]. The step size is updated using the 'epoch-level update' until the end of the retraining. Table 2.5 shows that the floating-point network performance saturates at 512 units size when BN is applied, and at 256 units when BN is not used. When the unit size in each layer is 512 or smaller, the proposed algorithm yields better performance in both cases. For example, if the 512 units size network is quantized in 2-bit without BN, the differences between the floating-point and the fixed-point networks are 1.82% and 1% for 'conventional' and 'adaptive' schemes, respectively. In addition, the phoneme error rate of the 3-bit network optimized with the 'adaptive' scheme (29.83%) is lower than that of the 4-bit quantized network with the 'conventional' scheme (29.95%).

BN improves the performance of both floating-point and fixed-point networks. Applying the 'adaptive' method improves the performance. For example, if the layer unit size is 128 and 2-bit quantization is used, BN brings the performance gain of 3.42% when 'adaptive' scheme is used. Therefore, the proposed 'adaptive' method can be efficiently used with BN.

When the unit size is large enough, the quantization scheme does not affect the performance much because a larger size network has strong resiliency to quantization [20]. Even the performance of 4-bit quantized 512 units size network without BN is almost comparable to that of the floating-point 1024 units size network. When the network is trained with BN, it shows a similar trend.

Figure 2.7 shows the quantization step size, Δ, of the proposed adaptive scheme as the retraining progresses. Note that the step size is renewed at each epoch during

Table 2.5: Frame-level phoneme error rate (%) on the test set with the TIMIT phoneme recognition examples. Note that 'conventional' is the baseline [1] and 'adaptive' is the proposed scheme.

		Without BN					With BN				
	Model size	64	128	256	512	1024	64	128	256	512	1024
	Full-precision	**34.38**	**31.63**	**30.17**	**29.61**	**29.53**	**33.82**	**30.81**	**29.79**	**29.77**	**29.59**
2-bit	Direct	80.25	84.12	81.92	83.30	75.05	89.82	88.79	87.57	85.73	86.10
2-bit	Conventional	43.73	37.80	33.70	31.43	29.99	41.81	35.88	33.12	31.21	30.22
2-bit	Adaptive	**42.06**	**36.88**	**32.61**	**30.61**	**29.49**	**37.87**	**33.46**	**31.48**	**30.73**	**30.09**
3-bit	Direct	68.13	63.65	60.33	51.46	48.61	80.41	69.55	69.42	81.60	64.55
3-bit	Conventional	40.63	34.73	31.41	30.49	**29.33**	36.88	32.58	30.53	30.14	29.76
3-bit	Adaptive	**37.89**	**33.80**	**30.74**	**29.83**	29.40	**35.29**	**31.94**	**30.32**	**30.10**	**29.65**
4-bit	Direct	58.90	50.58	42.15	38.05	36.53	65.63	50.43	46.46	43.80	39.77
4-bit	Conventional	36.51	32.65	30.79	29.95	29.44	34.17	31.34	29.86	**29.81**	29.70
4-bit	Adaptive	**35.50**	**32.09**	**30.50**	**29.54**	**29.29**	**33.91**	**30.86**	**29.47**	29.87	**29.52**

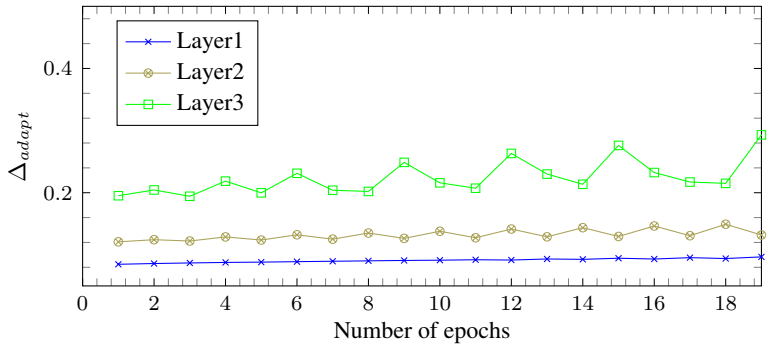


Figure 2.7: Training curves in terms of Δ_{adapt} for the FFDNN with the size of 256.

retraining. As shown in this figure, the step size of the last layer varies much, while that of the first layer is almost constant. The step size adaptation is much needed for the last layer.

We also evaluate the performance of the gradual quantization scheme. The results are reported in Table 2.6. The floating-point results show a 29.61% error rate on the test set. The 6-bit word length shows slightly better accuracy than the floating-point. Thus, we define the easiest task as the 6-bit quantization. In Table 2.6, the 'gradual' scheme yields better performance than the 'conventional' strategy, but shows worse or similar results compared to the 'adaptive' quantization. The combined strategy of the 'adaptive' and 'gradual' shows slightly better accuracy than the 'adaptive' strategy in 4- and 3-bit quantization, but it is worse than the 'adaptive' scheme in 2-bit quantization. Since there is no performance difference between the 'adaptive' and 'adaptive & gradual' scheme, we only employ the 'adaptive' scheme for CNN and RNN experiments.

Image classification using convolutional neural networks

Image classification experiments are performed on the SVHN dataset [58]. The dataset includes 600,000 labeled 32x32 RGB images from real-world house numbers. For the data preprocessing, we employ the same method with [59]. The output label has ten

Table 2.6: The error rate of the proposed quantization strategies on TIMIT phoneme recognition task. The network is FFDNN with two 512 size hidden layers, and the floating-point result is 29.61%. 'Conventional' is general retraining based quantization, 'adaptive' conducts proposed step size adaptation, 'gradual' is curriculum learning style quantization scheme, and 'adaptive & gradual' represents mixed approach using both techniques.

	Conventional	Adaptive	Gradual	Adaptive & Gradual
6-bit	**29.32**	**29.32**	**29.32**	**29.32**
4-bit	29.95	29.54	29.53	**29.49**
3-bit	30.49	29.83	29.90	**29.61**
2-bit	31.43	**30.61**	30.69	30.62

units which represent the numbers from 0 to 9. For the evaluation of the proposed scheme, we employ three different structures. We name the networks as 'L', 'C', and 'V' which have the trainable parameters of 60k, 84k, and 435k, respectively. The 'L' network is Lenet5 [60], 'C' network is from [61], and 'V' network is constructed as VGG style [15]. We train the floating-point networks using SGD with Nesterov momentum. The learning rate is decreased from 2-e2 to 3.125e-4 with a factor of 2 when the development set does not show improvement for 4 consecutive evaluations. For the fixed-point network training, the initial learning rate was 5e-4. The effects of step size adaptation in the CNNs are examined in Table 2.7. The step size is updated using the '1 epoch update & fix' strategy. Our algorithm works well for 'L' and 'V' networks regardless of the weight precision, 2, 3, or 4 bits. However, the 'C' networks with the conventional retraining show a better result when the weight precision is 4bits. Overall, the proposed method yields improved performances.

Table 2.7: Miss classification rate on the test set with the SVHN house number recognition example. The alphabets 'L', 'C', and 'V' represent specific structure of the CNN. The 'L' is the most smallest network and the 'V' is the biggest network. Please refer Section 2.3.3 for details.

	Type of network	L	C	V
	Floating result	***6.45***	***5.65***	***4.50***
2-bit (3 point)	Direct	45.68	23.17	73.55
	Conventional	8.37	7.10	5.24
	Adaptive	**8.01**	**6.65**	**5.02**
3-bit (7 point)	Direct	10.14	7.88	6.73
	Conventional	7.04	5.97	4.57
	Adaptive	**6.92**	**5.91**	**4.53**
4-bit (15 point)	Direct	7.85	6.03	4.79
	Conventional	6.60	**5.76**	4.74
	Adaptive	**6.46**	5.86	**4.60**

Language modeling using recurrent neural networks

Character-level language modeling predicts the next character and is used for speech recognition and text generation. Since the input and output layers consider only alphabets, the input and output complexities are much lower than the word level language model. We adopt the English Wikipedia dataset for training the character-level language modeling. The dataset contains 100 MB English Wikipedia text. The input and output layers are composed of 256 units for the one-hot encoded ASCII code. The RNN consists of three Long Short-Term Memory (LSTM) layers with a different number of memory cells ranging from 64 to 256 [62]. We train the RNNs using AdaDelta based SGD with 64 parallel input streams. The networks are unrolled 256 times and weights update is performed for128 forward steps. The learning rate starts from 5e-4

Table 2.8: Bit per character (BPC) on the test set with the English Wikipedia language model.

	Size of each layer	64	128	256
	Floating result	***2.07***	***1.81***	***1.65***
2-bit (3 point)	Direct	8.46	9.53	7.26
	Conventional	2.48	2.49	1.89
	Adaptive	**2.42**	**2.16**	**1.86**
3-bit (7 point)	Direct	7.176	6.84	4.35
	Conventional	2.52	2.10	1.91
	Adaptive	**2.35**	**2.06**	**1.82**
4-bit (15 point)	Direct	4.49	5.50	2.59
	Conventional	2.43	2.04	**1.83**
	Adaptive	**2.32**	**1.95**	1.86
6-bit (63 point)	Direct	2.56	3.73	1.73
	Conventional	**2.11**	**1.87**	**1.67**
	Adaptive	**2.11**	1.89	1.68

and decreases until 5e-8. For the step size adaptation, '1 epoch update & fix' strategy is employed. The fixed-point optimization results are reported in Table 2.8. As with our previous FFDNN and CNN results, it shows much improved performances on low-precision weights or small size networks.

2.4 Concluding remarks

This chapter investigates the fixed-point characteristics of deep neural networks. The retraining-based fixed-point optimization greatly reduces the word-length of weights and signals. The performance gap between the floating-point and the fixed-point neural networks with severe quantization almost vanishes when the DNNs are in the perfor-

mance saturation region for the given training data. However, when the complexity of DNNs is reduced, by lowering either the number of units, feature maps, or hidden layers, the performance gap between them increases. To solve this problem, we developed improved fixed-point optimization methods. The first one adaptively determines the quantization step size by measuring the weight distribution during the retraining procedure. The second one is a curriculum learning style fixed-point optimization technique, which conducts fixed-point optimization from high- to low-precision gradually. The proposed work yields better quantization results in FFDNNs, CNNs, and RNNs. Especially the effectiveness of the proposed techniques increases when the number of quantization levels is small and the network size is not large enough.

Chapter 3

HLHLp:Quantized Neural Networks Training for Reaching Flat Minima in Loss Surface

3.1 Introduction

Many previous QDNN optimization algorithms consist of three steps: training a floating-point network, quantizing the model, and improving the performance of the quantized network by fine-tuning. As for the fine-tuning, usually low learning rates are used to limit the deviation from the floating-point model as small as possible [63, 1, 64, 65, 21]. However, when only very low-precision weights are employed, the loss surface may differ from that with high precision. Therefore, fine-tuning the QDNN with quantization error feedback is not sufficient to design well-generalized QDNN.

The generalization capability of a DNN has been actively discussed [33, 34, 35]. The generalization capability of a DNN was explained in relation to the flat minimum of the error or loss function [33]. The study by [34] reveals that the ratio of learning rate to batch size is a key determinant of flatness of loss surface and generalization. Recent studies schedule the learning rate to improve the generalization capability [66, 67].

In this chapter, we propose a QDNN training algorithm that is intended to avoid sharp minima and reach flat minima in the discrete weight domain. The proposed ap-

proach intentionally changes the learning rate and the precision of the parameters in an alternating manner to reach a flat minimum despite abrupt increases in the training error. The experiments exhibit particularly good results in the quantization of parameter-size efficient convolutional neural networks (CNNs) and recurrent neural networks (RNNs). The contributions of this study are as follows:

- We derive that the quantization noise can play a role of escaping sharp minima in the training of QDNN.
- A high-low-high-low precision (HLHLp) training scheme is developed to encourage a QDNN arriving at a flat minimum that exhibits a high generalization capability.
- The proposed method is applied to the quantization of RNNs and CNNs. We achieve the results that significantly exceed those of previous designs for RNNs and competitive results for CNNs.

3.2 Related Works

3.2.1 Quantization of Deep Neural Networks

QDNN has been studied for a long time. However, earlier studies typically employed an 8-bit or higher precision partly because the networks were small and a direct quantization method was used. Stochastic gradient descent (SGD) for quantized DNNs to 2-bit ternary or 1-bit binary precision without significantly affecting the performance is proposed by [1] and [15]. It is difficult to update discrete weights directly because the gradients are much smaller than quantized weight values. Thus, the quantized weights are obtained by the error feedback quantization method. The method retains the high precision weights to accumulate gradients while the quantized weights are used in forward and backward propagation [15, 63, 1, 64, 65, 21].

Several quantization techniques are developed to optimize QDNNs, and these techniques mostly try to reduce quantization errors by considering the distribution of

weights. In particular, various elaborate techniques are developed for CNNs, which include weight cluster [68], stochastic rounding [69], data distribution [21], fittable quantization scale [70], or trainable quantization [71].

RNNs weights were also quantized with a binary format, which employed stochastic and deterministic ternarization, and pow2-ternarization methods [72]. Parameter-dependent adaptive threshold [73] or increasing the size of the neural network [74] is also investigated. Other studies formulated an optimization problem to determine the optimal quantization step size with greedy approximation [75] or alternating multi-bit quantization [64]. HitNet applies a different quantization algorithm to weight and activation [76]. Quantize only the weight using batch normalization between inputs and hidden state vectors shows high performance [77].

3.2.2 Flat Minima in Loss Surfaces

Most high performance deep neural networks contain a vast number of parameters, and thus the training error almost converges to zero in many cases. The stochastic gradient descent (SGD) algorithm updates the weights to minimize the training error. However, neural network training is non-convex optimization, and low-training error does not necessarily ensure good test performance capability. An early study proposed that the determination of flat minima in the loss surface is important to train high-performance networks [33]. Recent studies suggested that the increased amount of noise in gradients of a small-batch method aids in reaching a flat minimum in the loss surface [34]. Conversely, large-batch training wherein the gradient noise is low requires an increased learning rate to obtain a good performance [35].

The learning rate is the most important hyper-parameter in the SGD-based training. Typically, the learning rate is designed to monotonically decrease when the training proceeds. At the early stage of training, the weights should be updated coarsely, although they require fine-tuning at the final stage. However, [67] and [66] indicated that cyclically increasing and decreasing or warm-restarting the learning rate improves

test accuracy. It should be noted that the training error is also fluctuating albeit not necessarily decreasing monotonically when the learning rate is alternating. Understanding flat minima is very important in QDNN design because quantization is equivalent to injecting noise to weights, and flat minima imply resiliency in weight distortion.

3.3 Training QDNN for Improved Generalization Capability

In this section, we first briefly explain the conventional neural network quantization algorithm and derive that learning rate to quantization precision ratio controls the stochastic noise. We also present a new QDNN training technique that aids to encourage reaching flat minima in the quantization domain.

3.3.1 Analysis of Training with Quantized Weights

The number of bits representing the quantized values is denoted as b. b is usually from 1 to 8 and b-bit quantization can support up to 2^b levels. The quantization step size, Δ, is inversely proportional to the number of levels, 2^b. Thus, a low-precision weight needs a large Δ. When b is 2, a weight can be represented as 2-bit ternary, which is +Δ, 0, and -Δ. The b-bit symmetric uniform quantization including the 2-bit quantization can be generalized as follows:

$$\mathrm{Q}^b(\mathbf{w}) = \mathrm{sign}(\mathbf{w}) \cdot \Delta \cdot \min\left\{ \left\lfloor \left(\frac{|\mathbf{w}|}{\Delta} + 0.5 \right) \right\rfloor, \frac{(M-1)}{2} \right\} \tag{3.1}$$

where M is $2^b - 1$. We employ an L2-error minimization between floating and fixed-point weights to obtain the quantization step size Δ [1, 78, 65].

Quantization can be interpreted as injecting noise whose range is between $-\frac{\Delta}{2}$ and $+\frac{\Delta}{2}$. Thus, the retraining process is equivalent to injecting noise to weights, which has been known to improve the generalization capability [79]. As the number of bits, b, decreases, the amount of noise injection increases. Following the approach proposed

in [34], we analyze the relationship between flatness and precision of weights. Specifically, the weight update procedure in quantization retraining is expressed as follows:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla L(\mathrm{Q}(\mathbf{w}_t)), \tag{3.2}$$

where $\mathrm{Q}(\cdot)$ is the quantization function, L is the loss, and η is the learning rate. Loss surface surrounding the local minimum $\mathbf{w}^*$ is approximated via the Hessian of L at $\mathbf{w}^*$, and this is denoted as $\mathbf{H}$:

$$L(\mathbf{w}) \approx L(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \times \mathbf{H} \times (\mathbf{w} - \mathbf{w}^*) \tag{3.3}$$

$$\nabla L(\mathbf{w}) \approx \nabla L(\mathbf{w}^*) + \mathbf{H} \times (\mathbf{w} - \mathbf{w}^*) \tag{3.4}$$

We rewrite Equation (3.2) by using Equation (3.4) as follows:

$$\mathbf{w}_{t+1} \approx \mathbf{w}_t - \eta \mathbf{H} \times (Q(\mathbf{w}_t) - \mathbf{w}^*) \tag{3.5}$$

Let's consider that $\mathbf{n}_t$, the quantization noise, has a uniform distribution.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{H}((\mathbf{w}_t + \mathbf{n}_t) - \mathbf{w}^*) \tag{3.6}$$

We proceed the training for k step which is sufficiently training the model. Then the Equation (3.6) can be rewritten under the same assumption with [34] as follows:

$$\mathbf{w}_{t+k} = \mathbf{w}_t - \eta \mathbf{H}(\sum_{i=0}^{k-1}(\mathbf{w}_{t+i} + \mathbf{n}_{t+i}) - k\mathbf{w}^*) \tag{3.7}$$

$$= \mathbf{w}_t - \eta \mathbf{H}(\sum_{i=0}^{k-1}(\mathbf{w}_{t+i} - \mathbf{w}^*) + \sum_{i=0}^{k-1}\mathbf{n}_{t+i}) \tag{3.8}$$

For floating-point SGD training, we have the following equation

$$\mathbf{w}_{t+k} = \mathbf{w}_t - \eta \mathbf{H}(\sum_{i=0}^{k-1}(\mathbf{w}_{t+i} - \mathbf{w}^*)) \tag{3.9}$$

Only the difference between floating-point (Equation (3.9)) and fixed-point (Equation (3.8)) training is $\sum_{i=0}^{k-1}\mathbf{n}_{t+i}$. Note that quantization noise for each step, $\mathbf{n}_{t+i}$, is

IID. Hence under the central limit theorem, the resulting quantization noise becomes approximately a Gaussian. Thus, Equation (3.5) can be approximately expressed as

$$\mathbf{w}_t - \eta\mathbf{H} \times (\mathbf{w}_t + \mathcal{N}(0, (c^2/2^{2b})\boldsymbol{\mathcal{I}}) - \mathbf{w}^*) \tag{3.10}$$

$$= \mathbf{w}_t - \eta\mathbf{H} \times (\mathbf{w}_t - \mathbf{w}^*) - \eta\mathbf{H}\mathcal{N}(0, (c^2/2^{2b})\boldsymbol{\mathcal{I}}) \tag{3.11}$$

$$\approx \mathbf{w}_t - \eta\nabla L(\mathbf{w}_t) - \mathcal{N}(0, (\eta^2 c^2/2^{2b})\mathbf{H}^2), \tag{3.12}$$

where c is a constant related to models. Therefore, we consider the quantization retraining algorithm as the gradient descent with noisy gradients [80], and this corresponds to the Gaussian distribution with a covariance of $(\eta^2 c^2/2^{2b})\mathbf{H}^2$.

Eigendecomposition of Hessian matrix corresponds to $\mathbf{H} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T$, where $\boldsymbol{\Lambda}$ denotes the diagonal matrix of eigenvalues and $\mathbf{V}$ denotes an orthonormal matrix. We consider that Equation (3.12) implements the following stochastic differential equation (SDE) [34],

$$d\mathbf{w} = -\nabla L(\mathbf{w})dt + (c\sqrt{\eta}/2^b)\mathbf{V}\boldsymbol{\Lambda}^2 dW(t), \tag{3.13}$$

where $W(t)$ denotes Wiener process. We transform Equation (3.13) to general Ornstein-Uhlenbeck process (OUP). For this, we reparameterize $\mathbf{w}$ in terms of a new variable $\mathbf{z}$ which is defined as $\mathbf{z} = \mathbf{V}^T(\mathbf{w} - \mathbf{w}^*)$. The change of variable results in the following expression:

$$d\mathbf{z} = -2\boldsymbol{\Lambda}\mathbf{z}dt + (c\sqrt{\eta}/2^b)\boldsymbol{\Lambda}dW(t) \tag{3.14}$$

The stationary distribution of the OUP becomes the Gaussian $\mathcal{N}(0, (\eta c^2/2^{2b})\boldsymbol{\Lambda})$. The

expected loss is written as

$$\mathbb{E}(L(\mathbf{w}) - L(\mathbf{w}^*)) \tag{3.15}$$

$$= \frac{1}{2}\mathbb{E}((\mathbf{w} - \mathbf{w}^*)^{\mathrm{T}}\mathbf{H}(\mathbf{w} - \mathbf{w}^*)) \tag{3.16}$$

$$= \frac{1}{2}\mathbb{E}(\mathbf{z}^{\mathrm{T}}\mathbf{\Lambda}\mathbf{z}) \tag{3.17}$$

$$= \frac{1}{2}\sum \lambda_i \mathbb{E}(z_i^2) \tag{3.18}$$

$$= \frac{1}{2}\sum \lambda_i (\eta c^2/2^{2b})\lambda_i \tag{3.19}$$

$$= \frac{1}{2}(\eta c^2/2^{2b})Tr(\mathbf{\Lambda}^2) \tag{3.20}$$

$$= \frac{1}{2}(\eta c^2/2^{2b})Tr(\mathbf{H}^2) \tag{3.21}$$

The precision of weights b determines the trade-off between the expected loss and the squared sum of eigenvalues, with

$$\mathbb{E}(L(\mathbf{w}) - L(\mathbf{w}^*))/Tr(\mathbf{H}^2) \propto \eta c^2/2^{2b}. \tag{3.22}$$

The eigenvalues of Hessian matrix represent the flatness of the loss surface around the local minimum. Therefore, we conclude that the quantization precision also influences the minima in low-precision domain as well as the three factors (learning rate, batch size, and gradient covariance) found in [34].

Based on the above analysis, we propose a new quantization training scheme, high-low-high-low-precision (HLHLp) training, that manipulates the learning rate, η, and quantization precision, b, during training to reach flat minima of the QDNN.

3.3.2 High-low-high-low-precision Training

The proposed HLHLp optimization employs a multi-step training scheme, and this consists of floating-point training of a model from scratch, coarse-tuning on low-precision, fine-tuning on high-precision, and fine-tuning on low-precision. It should be noted that the low-precision means 2-bit weight representation, and the high-precision indicates 8-bit or floating-point weight representation. The coarse-tuning step employs

a high learning rate to escape from the current minimum point while the fine-tuning step proceeds with a low learning rate or decreasing learning rate. A detailed explanation of each step is given as follows.

High-precision Model Training (H-step)

The first step involves training a neural network in floating-point. Commonly known regularization techniques, such as dropout [81], batch normalization [82], and weight decay [83], can be employed. The learning rate is selected to obtain the optimal floating-point performance. Pretrained models can also be used. The initial learning rate in this step is denoted as $\eta^{\text{step 1}}$.

Coarse-tuning on Low-precision (L-step)

The second step performs retraining to 2-bit QDNN using the pretrained model from the first step. Activation quantization can also be employed. The learning rate for the conventional fine-tuning is $\alpha\eta^{\text{step 1}}$, where α is typically from 0.1 to 0.001 [19]. We employ relatively high learning rate in this step for the purpose of coarse-tuning, as opposed to fine-tuning. The coarse-tuning aids to escape from sharp minima by increasing the dynamics of η to 2^{2b} ratio in Equation (3.22). The new learning rate for this step is selected as approximately $\alpha\eta^{\text{step 1}} \times \frac{\Delta_{\text{2-bit}}}{\Delta_{\text{8-bit}}}$. The ratio is initially designed by considering the quantization step size ratios of 8-bit and 2-bit precision.

As we derived in Section 3.3.1, the eigenvalues of Hessian matrix represent the flatness of the loss surface. However, computation of the exact Hessian is super inefficient on large neural networks. To handle this problem, we approximate the Hessian by a diagonal matrix from the second moment of gradient **v** [84]. Since **v** is an estimator of diag($\mathbf{H}^2$), we can obtain a sum of eigenvalues s exploiting by trace(sqrt(**v**)). It should be noted that, to select the initial parameter of the third step, we measured both s and the validation error rate during the training. More specifically, during the training of the current step, we save three to five model parameters considering validation

results [1] and select the one which has the lowest value of s among them.

Fine-tuning on High-precision (H-step)

The third step performs retraining to 8-bit QDNN using the pretrained model from the second step. The initial learning rate for this step is lower than that in the second step. The fine-tuning decreases the dynamics of η to 2^{2b} ratio in Equation (3.22). This step involves descending to the maximum possible extent from the new local minimum. To select the initial model for the next step, we evaluate the validation results.

Fine-tuning on Low-precision (L-step)

The fourth step involves fine-tuning from the 8-bit weights obtained at the third step. The learning rate for this step is not extremely high and is decreasing. Thus, the final step is intended for fine-tuning and is similar to that in the conventional retraining-based method. We can repeat the second and third steps again. In this case, the total training is represented as HLHLHLp, and this denotes high-precision training, low-precision coarse-tuning, high-precision fine-tuning, low-precision coarse-tuning, high-precision fine-tuning, and final-tuning on low-precision. Additional HL steps may improve performance but increase training time. In our experiments, performance has converged in HLHLp in most cases.

The proposed HLHLp training scheme can employ various quantizers such as uniform quantizer [1] and asymmetric quantizer [17]. The experimental results that combine the proposed training algorithm with various quantizers are shown in Section 3.4.

3.4 Experimental Results

We evaluate the proposed HLHLp training scheme on the following three tasks: image classification (CIFAR-10/CIFAR-100 [48], ImageNet [85]), language modeling

[1] Accuracy for classification problem or perplexity for language modeling.

(PTB [86] and WikiText-2 [87]), and speech recognition (WSJ corpus [88]). The descriptions of each dataset are as follows:

- **CIFAR-10 & CIFAR-100:** CIFAR-10 and CIFAR-100 datasets [48] consist of 50K training and 10K test images on 10 and 100 classes, respectively. The image size is 32 x 32 with 3 channels (RGB). We use 45K and 5K images for training and validation, respectively. We employ a simple data augmentation, shifting and mirroring as suggested by [89].
- **ImageNet:** ILSVRC 2012 classification dataset [85] contains over 1.2M training and 50K validation images from 1,000 classes. We resize the images to 256x256 and conduct random or center crop to 224x224 during the training or evaluation, respectively.
- **PTB:** Penn Tree Bank (PTB) corpus [86] contains 929k training, 73k validation, and 82k test words with a vocabulary size of 10k. The dataset is widely used for the rapid evaluation of language models.
- **WikiText-2:** WikiText-2 corpus [87] consists of 2,088k training words, 217k validation words, and 245k test words. The vocabulary size is 33k.
- **WSJ corpus:** Wall Street Journal (WSJ) SI284 set [88] is composed of 81 hours of speech data. We evaluated our QDNN model with the WSJ eval92 set.

3.4.1 Image Classification with CNNs

Network and Hyper-parameter Configuration: We evaluate our method on CNNs for image classification. For the CIFAR-10 dataset, we train three different ResNets [5], namely ResNet-14, -20, and -32. Additionally, the same ResNet-20 and -32, and MobileNetV2 [90] are employed for the CIFAR-100 dataset. All models for both the CIFAR-10 and CIFAR-100 datasets are trained with the same hyper-parameters as follows. The batch size is 128, and the number of epochs trained is 175. An SGD optimizer with a momentum of 0.9 is used. The learning rate starts at 0.1 and decreases by 0.1 times at the 75th and 125th epochs. Additionally, L2-loss is added with the scale

Table 3.1: Test accuracy on CIFAR-10 and CIFAR-100 dataset. The numbers in the parenthesis are the accuracy difference between the floating and the 2-bit models. Both fine-tuning and HLHLp results are an average of five times running.

Dataset	CIFAR-10		
Model	ResNet-14	ResNet-20	ResNet-32
# of params	0.18M	0.27M	0.47M
Float	91.35	92.15	93.65
Fine-tuning	89.20 (-2.15)	90.86 (-1.29)	92.32 (-1.33)
HLHLp	**90.64 (-0.71)**	**91.58 (-0.57)**	**93.05 (-0.60)**
Dataset	CIFAR-100		
Model	ResNet-20	ResNet-32	MobileNetV2
# of params	0.28M	0.48M	2.45M
Float	68.01	69.97	75.98
Fine-tuning	64.47 (-3.54)	66.90 (-3.07)	74.97 (-1.01)
HLHLp	**66.44 (-1.57)**	**68.66 (-1.31)**	**75.51 (-0.47)**

of 5e-4. We employ simple symmetric uniform quantizer from [4]. The initial learning rate and the weight precision change as mentioned in Section 3.3.2 during HLHLp training. These changes in learning rate and precision are applied to all experiments in the rest of this paper.

Furthermore, we conduct the weight quantization of ResNet-18 on the ImageNet dataset using the proposed method. We employ a pretrained network as for the full precision model[2]. We set the batch size to 256 and conduct the retrain method for up to 20 epochs for each HLHLp step.

Results on CIFAR-10/CIFAR-100: The experimental results of the CIFAR-10 and CIFAR-100 datasets are presented in Table 3.1. Both the fine-tuned and the HLHLp-

[2]https://github.com/facebook/fb.resnet.torch

trained QDNNs are inherited from the same full-precision models. All layers in the models including the first and the last ones are quantized. In the case of the CIFAR-10 results, the performances of the 2-bit QDNNs improve when the HLHLp training is applied. Specifically, the HLHLp training results on ResNet-14 and ResNet-32 demonstrates 1.44% and 0.73% increase in the test accuracy when compared to the existing fine-tuning method [4]. The relative performance degradation of the 2-bit ResNet-32 for the full-precision model is 46% lower (0.6/1.3) when using the HLHLp training method. This small gap is due to the sufficiently large model size for the CIFAR-10 dataset. Large DNN models show a small difference between full-precision and low-precision networks.

The experiments with the more complex dataset (e.g. CIFAR-100) demonstrate more improvements. The test accuracy of the 2-bit ResNet-20 is 66.44% and 64.47% with the HLHLp training and the fine-tuning methods, respectively. The accuracy difference between our HLHLp and conventional training methods is reduced when the model size increases. However, HLHLp outperforms the fine-tuning method and reduces the gap between the floating-point and 2-bit networks significantly for all networks considered.

In the remaining experiments, we demonstrate the performance improvement when the HLHLp training scheme is applied, by using the same quantizers as those proposed in several previous studies.

Results on ImageNet: The experimental results of the ImageNet dataset are reported in Table 3.2. The compared low-precision models include TWN [16], TTQ [17], LQ-Nets [71], and ADMM [91]. These models employ 2-bit weights but we do not quantize the activations. Ours, TWN, and ADMM employ the symmetric ternary quantization. However, TTQ employs asymmetric ternary (AT) quantization, whereas LQ-Nets employs 4-level quantization. AT and 4-level quantization help in improving the performance but also make the inference more complex. The experimental results demonstrate that the proposed method is effective and that the top-1 accuracy with ternary

Table 3.2: HLHLp training results on ResNet-18 ImageNet. In this experiment, only the weights are quantized in 2-bit. The values in the parentheses are the difference between the full-precision and quantized accuracy (%) in literature. HLHLp result is an average of five times running.

W2/A32	Levels	Top-1 Acc	Top-5 Acc
TWN	3	61.8 (N/A)	84.2 (N/A)
TTQ	Asym3	66.6 (-3)	87.2 (-2)
LQ-Nets	4	68.0 (-2.3)	88.0 (-1.5)
ADMM	3	67.0 (-2.1)	87.5 (-1.5)
HLHLp (ours)	3	67.2 (**-1.6**)	87.8 (**-0.8**)

weights is better than ADMM. In the comparison of the accuracy difference between the full-precision model and the QDNN, our HLHLp training results demonstrate 1.6% degradation on Top-1 accuracy, which is much better than LQ-Nets (2.3%), TTQ (3%), and ADMM (2.1%). In LQ-Nets, the first and last layers of the model are not quantized, whereas, in ours, all the layers are quantized.

As part of the generalization test, we also evaluate our QDNN model with a contaminated dataset [92], which mixes various types of noise in the ImageNet validation set as shown in Figure 3.1.

The detailed results for each contaminated dataset are reported in Table 3.3. Our proposed HLHLp training scheme increases the average noise accuracy from 39.08% (HL) to 45.99% (HLHL) in 2-bit QDNNs. Thus, our HLHLp training helps to increase the generalization capability in QDNN.

3.4.2 Language Modeling on PTB and WikiText-2

Network and Hyper-parameter Configuration: For the quantitative comparison with previous works [75, 73, 74, 76, 64, 21], we constructed two word-level language mod-

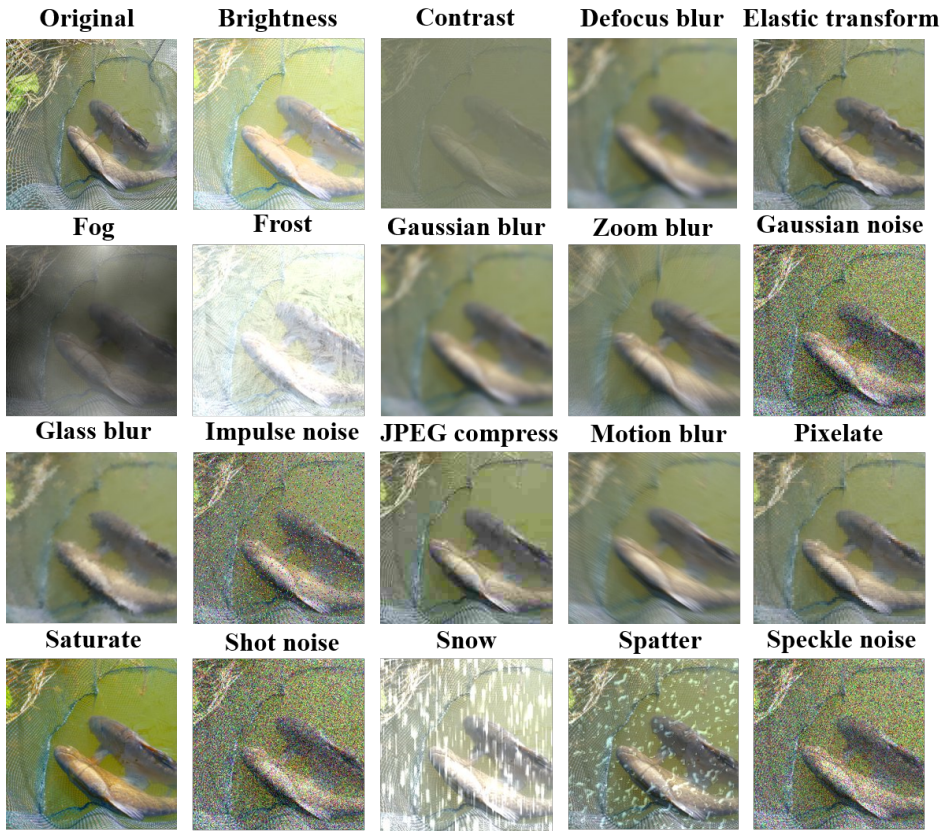


Figure 3.1: Examples of the added noises in the contaminated dataset.

Table 3.3: Detailed results on ImageNet contaminated test for Top-5 accuracy.

	Test Orig.	Avg. Noise	Fog	Shot	Glass	Zoom	Snow
H	**88.62**	**54.31**	59.06	35.67	40.32	50.75	45.79
HL	**84.26**	**39.08**	40.86	19.39	28.28	31.41	33.46
HLH	**88.41**	**47.56**	55.22	27.85	37.69	44.95	41.24
HLHL	**87.79**	**45.99**	51.96	26.08	36.32	41.94	39.71
	Spatter	Saturate	Elastic	Motion	Frost	Gauss B	Defocus
H	63.50	76.21	61.15	49.31	50.36	52.75	48.79
HL	46.78	59.37	48.85	31.71	38.60	32.22	27.30
HLH	56.34	70.38	58.08	43.87	46.45	41.70	37.16
HLHL	55.21	68.91	56.50	41.50	43.62	40.17	35.50
	Brighness	Pixelate	Contrast	Speckle	Impulse	JPEG	Gauss N
H	81.55	64.96	65.30	45.92	32.33	69.87	38.22
HL	69.33	61.14	50.78	28.26	16.15	58.07	20.60
HLH	76.12	59.44	58.04	38.38	22.33	59.27	29.10
HLHL	74.68	62.00	56.64	36.80	20.59	58.85	26.86

els (LMs) containing one long short-term memory (LSTM) [93] or one gated recurrent unit (GRU) [94]. Each LM has a 300-memory cell for PTB and 512-memory cell for WikiText-2. The initial learning rate for the floating-point network is 1.0. After 10 epochs, the learning rate decreases by a factor of 0.9 at each epoch. We clip the norm of the gradients by 1.0 for PTB and 3.5 for WikiText-2. Both the batch-size and unrolling steps are 20 for PTB, while, for WikiText-2, the values are 50 and 30, respectively. We apply dropout [81] only at non-recurrent connections as suggested in [95] with a keeping probability of 0.5 for PTB and 0.6 for WikiText-2. The performance of LM is measured via perplexity (PPL). An LM with low PPL is considered a good model.

Results on PTB: The comparison of the PPL of our HLHLp scheme and that of previous studies is presented in Table 3.4 for 2-bit ternary and 2-bit 4-level weight repre-

Table 3.4: PPL for 2-bit ternary and 2-bit 4-level weight quantized network of LSTM and GRU based language models on PTB test set. We denote 2-bit ternary and 4-level as 'T' and '4', respectively. The activations are also quantized in 2-bit 4-level. The number in the parenthesis represents that the gap of the PPL between the 2-bit and full-precision in literature. HLHLp result is an average of five times running.

W2 (T)/A2	LSTM	GRU	W2 (4)/A2	LSTM	GRU
[73]	152 (43)	150 (50)	[21]	126 (20)	142 (42)
[74]	152.2 (43.5)	N/A	[75]	100.3 (10.5)	105.1 (12.6)
[76]	110.3 (13.1)	113.5 (10.8)	[64]	95.8 (6.0)	101.2 (8.7)
HLHLp	**97.27 (7.82)**	**95.04 (1.80)**	HLHLp	**94.89 (5.44)**	**96.20 (2.96)**

sentations. We employ two previously developed quantizers for the 2-bit ternary [76] and 2-bit 4-level [75] weight respectively. The activations are also quantized in 2-bit. The HLHLp with ternary weights significantly outperforms the previous studies and also exhibits better results for the 2-bit 4-level representation. The GRU results also outperform both the 2-bit ternary and 2-bit 4-level representations. To the best of our knowledge, these results are the state-of-the-art when quantizing both weight and activation in 2-bit.

Results on WikiText-2: The PPL of WikiText-2 is reported in Table 3.5. We employ a simple uniform quantizer from [18]. The quantized network trained using HLHLp outperforms the other previous results. The LSTM model quantized with the proposed method shows lower (better) PPL when compared to the previous works in both ternary and 4-level weights. Especially for the 2-bit 4-level result, we achieve the test PPL of 105.5 that is 0.6 lower than the work of [64] although our full-precision PPL is 2.9 higher (worse) than the compared work. In the case of GRU, our HLHLp quantization scheme demonstrates a much better test PPL than [64]. We have improved the state-of-the-art PPL from 113.7 to 105.96. Surprisingly, the quantized network performs

Table 3.5: Quantization results on WikiText-2 test set for 2-bit quantized networks. 'FP' means full-precision and 'Difference' represents the gap between the PPL for 2-bit and full-precision in literature. HLHLp result is an average of five times running.

	Weight	Test PPL for LSTM			Test PPL for GRU		
	Levels	FP	2-bit	Difference	FP	2-bit	Difference
[76]	Ternary	114.37	126.72	12.35	124.50	132.49	7.99
HLHLp	Ternary	103.0	**107.66**	**4.66**	108.68	**105.96**	**-2.72**
[64]	4-level	**100.10**	106.10	6.00	**106.70**	113.70	7.00
HLHLp	4-level	103.0	**105.5**	**2.5**	-	-	-

better than the floating-point model, which suggest that the HLHLp scheme works as a regularizer.

3.4.3 Speech Recognition on WSJ Corpus

Network and Hyper-parameter Configuration: We construct three unidirectional LSTM layers with 512 memory cells. We employ the connectionist temporal classification (CTC) loss to train an RNN-based acoustic model (AM). We clip the norm of the gradients by 4, and train the AM with the Adam optimizer. A dropout with a keeping probability of 0.5 is applied for all the non-recurrent connections. The initial learning rate for the floating-point training is 3e-4, which decreases by a factor of 0.2 whenever the validation loss does not decrease thrice consecutively.

Results: The experiment results for WSJ are reported in Table 3.6. For comparison, we report another quantization result of the same RNN trained with the method suggested in [45]. The character error rate (CER) is measured using greedy decoding. To obtain the word error rate (WER), we follow the method used in [96] to decode the output of the CTC-AM using the weighted finite-state transducers (WFST) network. We used a retrained trigram LM with an extended vocabulary for decoding. The CER of the

Table 3.6: HLHLp training results on WSJ corpus. We quantize both weight and activations in 2-bit. 'CER' is character error rate (%) and 'WER' means word error rate (%). 'Clean' represents the results on Aurora-4 clean set, and 'Noisy' means the results on average of all noisy set.

W2/A2	WSJ		Aurora-4	
	Test CER	Test WER	Clean CER	Noisy CER
Float	8.18	11.16	7.37	51.95
[45]	9.76	11.32	8.58 (+1.21%)	51.65
HLHLp (ours)	**8.21**	**11.27**	**6.78 (-0.59%)**	**48.6**

full-precision model is 8.18%, and that of the 2-bit quantized model measured after the HLHLp training is 8.21%, which demonstrates almost no degradation. As part of the generalization test, we evaluate on the Aurora-4 noisy test corpus [97], which mixes the noises of a car, babble, restaurant, street, airport, and train to the WSJ eval clean test set. The evaluation results of this test are presented in Table 3.6. When full-precision was quantized to 2 bits using [45] method, CER increases by 1.21% on the clean set, however, our HLHLp training shows 0.59% higher accuracy than the full-precision result. A similar tendency is observed in the noise test. The performance of the 2-bit weight representation obtained by the HLHLp training is 3.35% better than the full-precision performance based on the average value of the noise test.

3.4.4 Discussion

Ablation Study: The experimental results demonstrate that the proposed HLHLp training method works exceptionally well for all experiments, including image classification, language modeling, and speech recognition. However, some questions still exist, such as "Are the improved results due to the longer training time?" and "Which

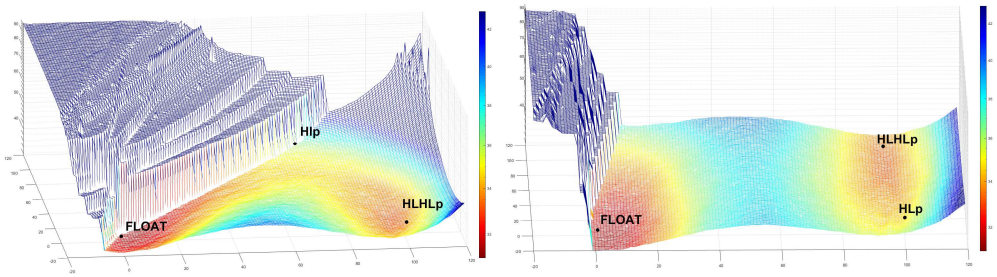
Table 3.7: Ablation study on GRU PTB LM. The results are reported in PPL. Results in the same column represent obtained PPL with the exactly same epochs.

(A)	Float (H)	2-bit (L)	8-bit (H)	2-bit (L)
	95.32	99.12	92.25	96.97
(B)	Float (H)	Float (H)	Float (H)	2-bit (L)
	95.32	100.98	98.13	115.92
(C)	Float (H)	2-bit (L)	2-bit (L)	2-bit (L)
	95.32	99.46	97.48	97.84
(D)	Float (H)	Float (H)	Float (H)	2-bit (L)
	-	-	95.06	112.08

part helps in increasing the performance?". To answer these questions, we conduct ablation experiments that optimizes an LM with a GRU using four different approaches as follows:

- (A) is the proposed HLHLp training employing floating-point training for 50 epochs, 2-bit retraining for 30 epochs with high a learning rate, 8-bit retraining for 30 epochs with a low learning rate, and 2-bit retraining for 30 epochs with a low learning rate.
- (B) employs 110 (=50+30+30) epochs of floating-point training with a cyclic learning rate which is exactly the same as the learning rate of (A). Additionally, 2-bit retraining for 30 epochs is conducted with the same learning rate as that of the last step in (A).
- (C) adopts 50 epochs of floating-point training and 90 (=30+30+30) epochs of 2-bit retraining with exactly the same learning rate as that of (A). Therefore, this setting converts the high-precision in the third step of (A) into low-precision.
- (D) conducts floating-point training but monotonically decreases the learning rate during 110 epochs and performs the retraining of 30 epochs with 2-bit

weights representations. Thus, this method uses only fine-tuning.



(a) (b)

Figure 3.2: 3-D loss surface for test error on ResNet20 CIFAR-100. The three points in (a) indicate full-precision (FLOAT), 2-bit QDNN that trained with fine-tuning (Hlp), and 2-bit QDNN that trained with HLHLp (HLHLp). The three points in (b) represent full-precision (FLOAT), 2-bit QDNN that trained with HLHLp (HLHLp), and 2-bit QDNN that trained with HLp (HLp). Note that HLp means 2-bit QDNN after the second step of HLHLp training scheme.

The detailed results are presented in Table 3.7. The results clearly indicate that HLHLp ((A)) training performs much better than training with cyclical learning rate ((B)) or monotonic decreasing learning rate ((D)) without converting precision in the middle of steps in HLHLp training. The gap in PPL between (A) and (C) also indicates that fine-tuning in high-precision aids in improving the performance.

Visualization of Loss Surface: We employ 3-dimensional graphical visualization to demonstrate that our proposed HLHLp training scheme aids to reach flat minima in loss surface. We employ the method developed by [37]. This method can help compare the training or test loss of three neural network models on the same 3-D surface, while the previous visualization method in [36] only shows the loss surface of one model. Figure 1 depicts the loss surface of test error on ResNet20 using CIFAR-100 dataset. Figure 3.2 (a) compares the test loss of three models: full-precision ('FLOAT'), 2-bit quantized network retrained using a very small learning rate or fine-tuning ('Hlp'), and 2-bit quantized network trained with HLHLp ('HLHLp'). We can find a path con-

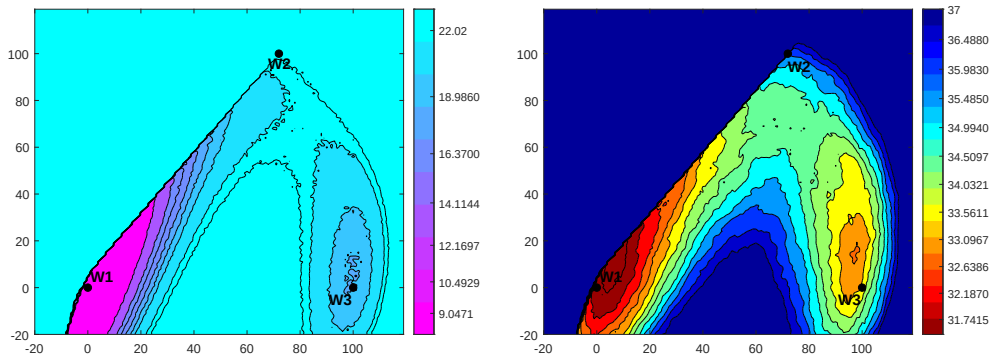
necting ‘FLOAT’ and ‘Hlp’. Note that ‘Hlp’ point is located very close to the steep loss wall, suggesting a poor generalization capability. On the other hand, connecting ‘FLOAT’ and ‘HLHLp’ seems more difficult because the loss surface between them is not flat. However, ‘HLHLp’ is located near the center of a wide basin or a flat minimum. Apparently, ‘HLHLp’ should be preferred for good generalization. In Figure 1 (b), we compare the full-precision (‘FLOAT’), 2-bit QDNN trained with HLHLp (‘HLHLp’), and 2-bit QDNN with HLp (‘HLp’). Note that ‘HLp’ means 2-bit QDNN after the second step of HLHLp training scheme. ‘HLp’ employs a very large learning rate for coarse tuning. Here, we can find that ‘HLp’ is at the same basin with the ‘HLHLp’, but is at the boundary. The remaining steps of HLHLp training help move ‘HLp’ to the near center of the basin. In Figure 3.2 (a) and (b), we show the 3-D test loss surface. The 2-D version of Figure 3.2 including training loss surface can be found in Figure 3.3.

We also employ ϵ-sharpness [35] and figures produced with another visualization method [36]. The flatness measurement using ϵ-sharpness is listed in Table 3.8, where a low sharpness value indicates a flat surface. We employ a quantizer from [4] and train QDNNs with and without HLHLp. The results clearly shows that our proposed method has flatten loss surface.

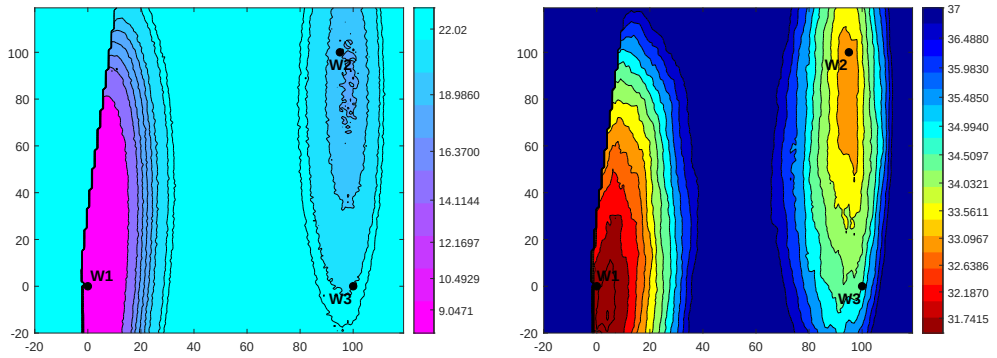
Table 3.8: ϵ-shaprness measurment. Lower value means flatter loss surface.

	FP	0.167
Without HLHLp	2-bit	0.059
With HLHLp	First L	0.03
	Final L	**0.008**

Figure 3.4 depicts the loss surface of the full-precision and quantized networks. To draw the loss surface we employ graphical approach suggested by [36]. The curvature of the loss surface is quite bent in Figure 3.4 (a). Figure 3.4 (b) and Figure 3.4 (c) exhibit a similar degree of bending irrespective of whether or not a high learning rate

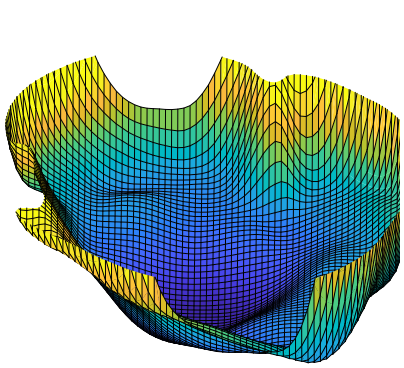


(a) Train (b) Test

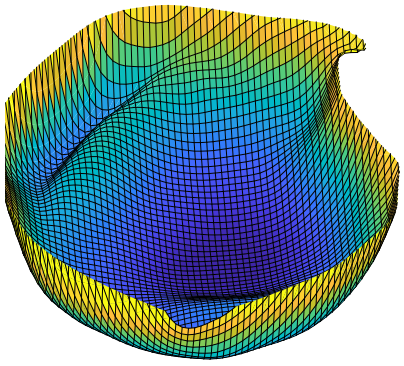


(c) Train (d) Test

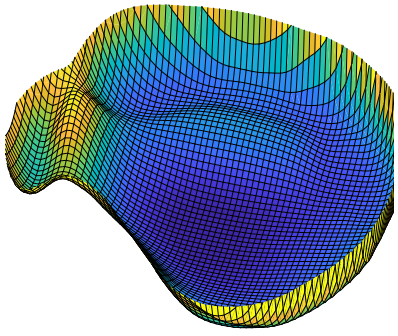
Figure 3.3: Illustrations of loss surface for train and test error on ResNet20 CIFAR-100. The three points in (a) and (b) denote that W1 (float), W2 (2-bit QDNN trained with [4]), and W3 (2-bit QDNN trained with HLHLp). The three points in (c) and (d) denote that W1 (float), W2 (2-bit QDNN trained with HLHLp), and W3 (2-bit QDNN trained with HLp).



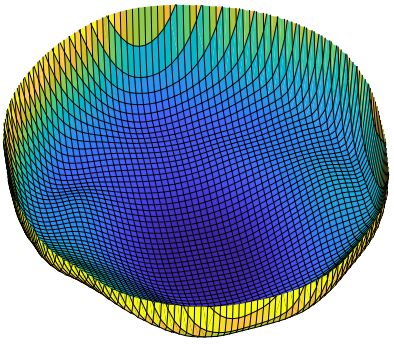
(a) Float



(b) 2-bit small lr



(c) 2-bit large lr



(d) Ours

Figure 3.4: Illustrations of loss surface for VGG-16 CIFAR-10. (a) 32-bit floating-point weight. (b) 2-bit ternary weight trained with a low learning rate (c) 2-bit ternary weight trained with a large learning rate. (d) 2-bit ternary weight obtained with HLHLp training.

is employed. The loss surface of the final results of the HLHLp training exhibits the most stable curvature among the four cases in Figure 3.4 (d).

3.5 Concluding Remarks

In this chapter, we developed a HLHLp training scheme to obtain high-performance quantized neural networks. At each training step, we employed different precisions and abruptly changing learning rates during training to escape from sharp minima and reach a flatter loss surface. Thus, the proposed approach significantly differs from conventional methods, wherein training with quantized networks is conducted for fine-tuning and the learning rates typically monotonically decrease. We applied the training scheme to the quantization of RNNs and CNNs and obtained very good performance closing the gap between full-precision and low-precision models. Specifically, the method exhibited extremely good results with respect to the quantization of RNNs.

Chapter 4

Knowledge Distillation for Optimization of Quantized Deep Neural Networks

4.1 Introduction

Quantization is a widely used compression technique, and even 1- or 2-bit models can show quite good performance. However, it is necessary to train the model very carefully not to lose the performance when only low-precision arithmetic is allowed. Many QDNN papers have suggested various types of quantizers or complex training algorithms [63, 1, 64, 65, 21].

Knowledge distillation (KD) that trains small networks using larger networks for improved performance [2, 98]. KD employs the soft-label generated by the teacher network to train the student network. Leveraging the knowledge contained in previously trained networks has attracted attention in many applications for model compression [99, 100, 101, 102] and learning algorithms [103, 104, 105, 106]. Recently, the use of KD for the training of QDNN has been studied [30, 29]. However, there are many design choices to explore when applying KD to QDNN training. The work in [30] studied the effects of simultaneous training or pre-training in the teacher model design. The result is rather expected; employing a pre-trained teacher model is ad-

vantageous when considering the performance of the student network. However, [107] mentioned that a too large teacher network does not help improve the performance of the student model.

In this chapter, we exploit KD with various types of teacher networks that include full-precision [30, 29] model, quantized one, and teacher-assistant based one [107]. The analysis results indicate that, rather than the type of the model, the distribution of the soft label is critical to the performance improvement of the student network. Since the distribution of the soft label can be controlled by the *temperature* and the *size of the teacher network*, we try to show how well-selected *temperature* can improve the QDNN performance dramatically even with a small teacher network. Further, we suggest a simple KD training scheme that adjusts the mixing ratio of hard and soft losses during training for obtaining stable performance improvements. We name it as the *gradual soft loss reducing* (GSLR) technique. GSLR employs both soft and hard losses equally at the beginning of the training, and gradually reduces the ratio of the soft loss as the training progresses.

The rest of the chapter is organized as follows. Section 4.2 describes how QDNNs can be trained with KD and explains why the hyperparameters of KD are important. Section 4.3 shows the experimental results and we conclude the paper in Section 4.4.

4.2 Quntized Deep Neural Network Training Using Knowledge Distillation

In this section, we first briefly describe the conventional neural network quantization method and also depict how QDNN training can be combined with KD. We also explain the hyperparameters of KD and their role in QDNN training.

4.2.1 Quantization of deep neural networks and knowledge distillation

The deep neural network parameter vector, $\mathbf{w}$, can be expressed in 2^b levels when quantized in b-bit. Since we usually use a symmetric quantizer, the quantized weight vector $\mathrm{Q}(\mathbf{w})$ can be represented using (4.1) or (4.2) for the case of $b = 1$ or $b > 1$ as follows:

$$\mathrm{Q}^1(\mathbf{w}) = \text{Binarize}(\mathbf{w}) = \Delta \cdot \text{sign}(\mathbf{w}) \tag{4.1}$$

$$\mathrm{Q}^b(\mathbf{w}) = \text{sign}(\mathbf{w}) \cdot \Delta \cdot \min\Big\{ \Big\lfloor \Big(\frac{|\mathbf{w}|}{\Delta} + 0.5 \Big) \Big\rfloor, \frac{(M-1)}{2} \Big\}, \tag{4.2}$$

where M is the number of quantization levels ($2^b - 1$) and Δ represents the quantization step size. Δ can be computed by L2-error minimization between floating and fixed-point weights or by the standard deviation of the weight vector [1, 78, 19].

Severe quantization such as 1- or 2-bit frequently incurs large performance degradation. Retraining technique is widely used to minimize the performance loss [20]. When retraining the student network, forward, backward, and gradient computations should be conducted using quantized weights but the computed gradients must be added to full-precision weights [63, 1, 64, 65, 21].

The probability computation in deep neural networks usually employs the softmax layer. Logit, $\mathbf{z}$, is fed into the softmax layer and generates the probability of each class, $\mathbf{p}$, using $p_i = \frac{exp(z_i/\tau)}{\sum_j exp(z_j/\tau)}$. τ is a hyperparameter of KD known as the *temperature*. A high value of τ softens the probability distribution. KD employs the probability generated by the teacher network as a soft label to train the student network, and the following loss function is minimized during training.

$$\mathcal{L}(\mathbf{w}_\mathrm{S}) = (1-\lambda)\mathcal{H}(y, p^\mathrm{S}) + \lambda\mathcal{H}(p^\mathrm{T}, p^\mathrm{S}), \tag{4.3}$$

where $\mathcal{H}(\cdot)$ denotes a loss function, y is the ground truth hard label, $\mathbf{w}_\mathrm{S}$ is the weight vector of the student network, p^T and p^S are the probabilities of the teacher and student networks, and λ is a *loss weighting factor* for adjusting the ratio of soft and hard losses.

A recent paper [107] investigates the relation among the teacher, teacher-assistant, and student models. The effect of KD gradually decreases when the size difference

Algorithm 1: QDNN training with KD

Initialization: $\mathbf{w}_{\mathrm{T}}$: Pretrained teacher model,

$\mathbf{w}_{\mathrm{S}}$: Pretrained student model,

λ: Loss wegithed factor, τ: Temperature

Output : $\mathbf{w}_{\mathrm{S}}^{\mathrm{q}}$: Quantized student model

while *not converged* **do**

$\mathbf{w}_{\mathrm{S}}^{\mathrm{q}} = \mathrm{Quant}(\mathbf{w}_{\mathrm{S}})$

Run forward teacher ($\mathbf{w}_{\mathrm{T}}$) and student model ($\mathbf{w}_{\mathrm{S}}^{\mathrm{q}}$)

Compute distillation loss $\mathcal{L}(\mathbf{w}_{\mathrm{S}}^{\mathrm{q}},\ \lambda)$

Run backward and compute gradients $\frac{\partial \mathcal{L}(\mathbf{w}_{\mathrm{S}}^{\mathrm{q}})}{\partial \mathbf{w}_{\mathrm{S}}^{\mathrm{q}}}$

$\mathbf{w}_{\mathrm{S}} = \mathbf{w}_{\mathrm{S}} - \eta \cdot \nabla \frac{\partial \mathcal{L}(\mathbf{w}_{\mathrm{S}}^{\mathrm{q}})}{\partial \mathbf{w}_{\mathrm{S}}^{\mathrm{q}}}$;

end

Return $\mathbf{w}_{S}^{q}$

between the teacher and student networks becomes too large. This performance degradation is due to the capacity limitation of the student model. Since QDNN limits the representation level of the weight parameters, the capacity of a quantized network is reduced when compared with the full-precision model. Therefore, QDNN training with KD is more sensitive to the *size of the teacher network*. We consider the optimization of three hyperparameters described above, *temperature*, *loss weighting factor*, and *size of the teacher network*. Algorithm 1 describes how to train QDNN with KD.

4.2.2 Teacher model selection for KD

In this section, we try to find the best teacher model for QDNN training for KD. We consider three different approaches. The first one is training the full-precision teacher and student networks independently and applies KD when fine-tuning the quantized student model as suggested in [30, 29]. The second is training a medium-sized teacher assistant network with a very large teacher model, and then optimizing the student

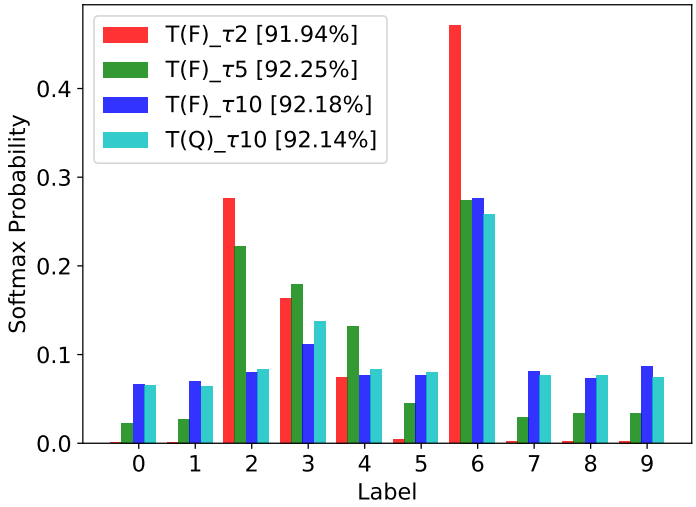


Figure 4.1: Example of the softmax distribution for label 6 from the teacher models in Table 4.1. The numbers in square brackets are the CIFAR-10 test accuracies of the student networks that trained by each teacher model.

network using the teacher assistant model as suggested by [107]. The last approach is using a quantized teacher model for the possibility of the student learning something on quantization.

Table 4.1 compares the results of these three approaches. Figure 4.1 also shows the softmax distributions when the teacher models and temperatures vary. The test accuracy of the quantized ResNet20 trained using hard loss was 91.71%. The results using various KD approaches indicate the following information. First, whether the teacher network is quantized or not, the performance of the student network is not much different. Secondly, employing the teacher assistant network [107] does not help increase the performance. The performance is similar to that of conventional KD. Thirdly, KD training with a full-precision teacher network [30, 29] is significantly better than conventional training when τ is 5. But, no performance increase is observed when τ is 2. Lastly, the teacher models that achieve the student network accuracy of 92.14%, 92.18%, and 92.25% shows a similar softmax distribution. However, T(F)-S with $\tau = 2$ shows a quite sharp softmax shape, and the resulting performance is similar

Table 4.1: Train and test accuracies of the quantized ResNet20 that was trained with various KD methods on CIFAR-10 dataset. 'T_L', 'T', 'S', '(F)', and '(Q)' denote large teacher, teacher, student, (full-precision), and (quantized), respectively. HD is a conventional training using hard loss. τ represents the *temperature*. Note that all the student networks are 2-bit QDNN and the results are the average of five times running.

Method	Status	Train Acc.	Test Acc.
T(F)-S	T (float)	99.95	94.02
	S (τ = 5)	98.02	92.25
	S (τ = 2)	98.76	91.94
T_L(F)-T(F)-S	T_L (float)	99.99	95.24
	T (float)	99.99	94.8
	S (τ = 10)	97.78	92.18
T_L(F)-T(Q)-S	T_L (float)	99.99	95.24
	T (4-bit)	99.99	94.46
	S (τ = 10)	97.79	92.14
T(Q)-S	T (8-bit)	99.99	94.34
	S (τ = 10)	97.53	92.02
HD	Conventional	98.91	91.71

to that of hard-target training.

These points indicate that the softmax distribution is the key to lead effective KD training. Although the different teacher models generate dissimilar softmax distributions, we can control the shapes by using the *temperature*. A detailed discussion about hyperparameters is provided in the following subsections.

4.2.3 Discussion on hyperparameters of KD

As we mentioned in Section 4.2.1 and 4.2.2, the hyperparameters *temperature* (τ), *loss weighting factor* (λ), and *size of teacher network* (N) can significantly affect the QDNN performance. Previous works usually fixed these hyperparameters when training QDNN with KD. For example, [30] always fixes τ to 1, and [29] holds it to 1 or 5 depending on the dataset. However, these three parameters are closely interrelated. For example, [107] points out that when the teacher model is very large compared to the student model, the softmax information produced by the teacher network become sharper, making it difficult to transfer the knowledge of the teacher network to the student model. However, even in this case, controlling the *temperature* may be able to make it possible. Therefore, when the value of one hyperparameter is changed, the others also need to be adjusted carefully. Thus, we empirically analyze the effect of KD's hyperparameters. In addition, we introduce the *gradual soft loss reducing* (GSLR) technique that aids to improve the performance of QDNN dramatically. The GSLR is a KD training method that gradually increases the reflection ratio of the hard loss.

4.3 Experimental Results

4.3.1 Experimental setup

Dataset: We employ CIFAR-10 and CIFAR-100 datasets for experiments. CIFAR-10 and CIFAR-100 consist of 10 and 100 classes, respectively [48]. Both datasets contain

Table 4.2: Train and test accuracies (%) of the teacher networks on the CIFAR-10 and the CIFAR-100 datasets. 'WRN20xN' denotes WideResNet with a wide factor of 'N'.

CIFAR-10	Train	Test	CIFAR-100	Train	Test
ResNet20	99.62	92.63	ResNet20	90.12	68.43
WRN20x1.2	99.83	92.93	WRN20x1.2	94.92	69.64
WRN20x1.5	99.93	93.48	WRN20x1.5	98.63	71.80
WRN20x1.7	99.95	94.02	WRN20x1.7	99.36	72.17
WRN20x2	99.95	94.36	WRN20x2	99.82	74.03
WRN20x5	100	95.24	WRN20x3	99.95	76.31
WRN20x10	100	95.23	WRN20x4	99.95	77.93
			WRN20x5	99.98	78.17
			WRN20x10	99.98	78.68

50K training images and 10K testing images. The size of each image is 32x32 with RGB channels.

Model configuration & training hyperparameter: To analyze the impact of hyperparameters of KD on QDNN training, we train WideResnet20xN (WRN20xN) [108] as the teacher networks, where N is set to 1, 1.2, 1.5, 1.7, 2, 3, 4, 5, and 10. When N is 1, the network structure is the same with ResNet20 [5]. All the train and the test accuracies of the teacher networks on CIFAR-10 and CIFAR-100 datasets are reported in Table 4.2. We employ ResNet20 as the student network for both the CIFAR-10 and CIFAR-100 datasets. If the network size is large enough considering that of the dataset, which means over-parameterized, most quantization method works well [20]. Therefore, to evaluate a quantization algorithm, we need to employ a small network that is located in the under-parameterized region [20, 109]. Although the full-precision ResNet20 model is over-parameterized, which means near 100% training accuracy, the 2-bit network becomes under-parameterized on the CIFAR-10 dataset. Likewise,

Table 4.3: Training results of full-precision and 2-bit quantized ResNet20 on CIFAR-10 and CIFAR-100 datasets in terms of accuracy (%). The models are trained with hard loss only.

		Train acc.	Test acc.
CIFAR-10	Full-precision	99.62	92.63
	2-bit quantized	98.92	91.71
CIFAR-100	Full-precision	90.12	68.43
	2-bit quantized	77.61	65.23

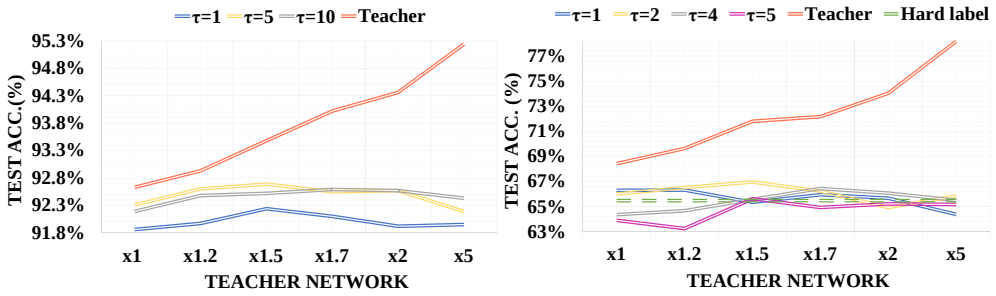
on the CIFAR-100 dataset, both the full-precision and the quantized models are under-parameterized. Thus, it is a good network configuration to evaluate the effect of KD on QDNN training. We report the train and the test accuracies for ResNet20 on CIFAR-10 and CIFAR-100 in Table 4.3.

4.3.2 Results on CIFAR-10 and CIFAR-100

We compare our models with the previous works in Table 4.4. The compared QDNN models trained with KD include QDistill [29], Apprentice [30], and Guided [110]. We achieve the results that significantly exceed those of previous studies. We compare our model (0.27M) with the 'student model 2' (SM 2) of QDistill that has 0.3 M parameters, and achieve an 18.32% of performance gap in the test accuracy. Also, it is about 1% better than the ResNet20 result reported by Apprentice and even achieved the same performance with their ResNet32 result. When quantized to 1-bit, the test accuracy of 91.3% is obtained, which is almost the same as Apprentice's ResNet20 2-bit model. In the case of CIFAR-100, QDistil and Guided student models use considerably large number, 17.2M and 22.0M, of parameters. Our student model only contains 0.28M parameters but achieve 17.7% and 2.4% higher accuracies than QDistill and Guided, respectively. These huge performance gaps show the importance of selecting hyperpa-

Table 4.4: Results of QDNN training with KD on ResNet-20 for CIFAR-10 and CIFAR-100 dataset. 'WRN','RN', 'SM', 'DS' represent WideResNet, ResNet, student model, and deeper student, respectively.

CIFAR10	Teacher (full-precision)		Student (2-bit)			
	# params (M) (model name)	Test (%)	# params (M) (model name)	Test (%)	τ	λ
QDistill	5.3 (small network)	89.7	0.3 (SM 2)	74.2	5	0.5
			5.8 (DS)	89.3	5	0.5
	145 (WRN28x20)	95.7	82.7 (WRN22x16)	94.23	5	0.5
Apprentice	0.66 (RN44)	93.8	0.27 (RN20)	91.6	1	0.5
	0.66 (RN44)	93.8	0.47 (RN32)	92.6	1	0.5
Ours	0.61 (WRN20x1.5)	93.5	0.27 (RN20)	92.52	10	0.5
	0.38 (WRN20x1.2)	92.9	0.27 (RN20) 1-bit	91.3	3	0.5
CIFAR100	Teacher (full-precision)		Student (2-bit)			
	# params (M) (model name)	Test (%)	# params (M) (model name)	Test (%)	τ	λ
QDistill	36.5 (WRN28x10)	77.2	17.2 (WRN22x8)	49.3	5	0.5
Guided	22.0 (AlexNet)	65.4	22.0 (AlexNet)	64.6	-	-
Ours	0.39 (WRN20x1.2)	69.64	0.28 (RN20)	66.6	2	0.5
	0.78 (WRN20x1.7)	72.17	0.28 (RN20)	67.0	3	GSLR



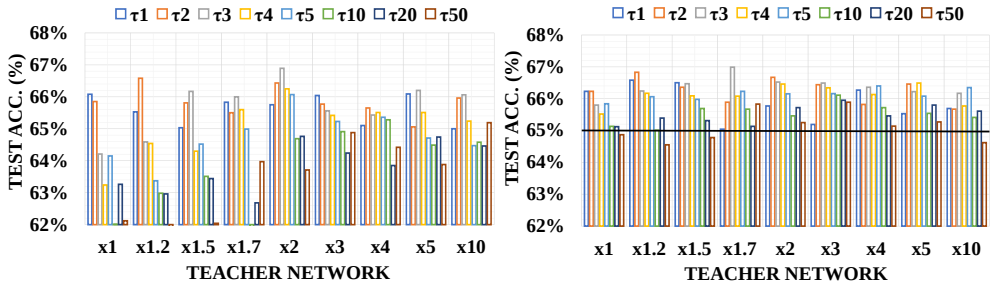
(a) CIFAR-10 (b) CIFAR-100

Figure 4.2: Results of 2-bit ResNet20 that trained with varying the *temperature* (τ) and the *size of the teacher network* on the CIFAR-10 and the CIFAR-100 datasets. The numbers in x-axis represent the wide factor (N) for WideResNet20xN.

rameters.

4.3.3 Model size and temperature

We report the test accuracies of 2-bit ResNet20 on CIFAR-10 in Figure 4.2 (a). To demonstrate the effect of the *temperature* for the QDNN training, we train 2-bit ResNet20 while varying the *size of the teacher network* from 'WRN20x1' to 'WRN20x5'. Each experiment was conducted for three τ values of 1, 5, and 10, which correspond to small, medium, and large one, respectively. Note that WRN20xN contains channel maps increased by N times. When the value of τ is small (blue line in the figure), the performance greatly depends on N, or the teacher model size. The performance change is much reduced as the value of τ increases to the medium (orange line) or the large value (blue line). This is related to the accuracy of the teacher model (red line). When the *size of the teacher model* increases, the shape of the soft label becomes similar to that of hard label. In this case, the KD training results are not much different from that trained with the hard label. Therefore, with $\tau = 1$, the performance decreases to 91.9% when the teacher network becomes larger than WRN20x2. This result is similar to the performance of a 2-bit ResNet20 trained with the hard loss (91.71%). The soft label needs to have a broad shape and it can be achieved either by increasing the



(a) KD (b) KD+GSLR

Figure 4.3: Results of 2-bit ResNet20 models that trained by the various *size of teacher networks* and the *temperature* on CIFAR-100. In (b), the black horizontal line represents the test accuracy when the student network is trained with hard label only.

temperature or limiting the *size of the teacher network*. A similar problem can occur for full-precision model KD training, but it is more important for QDNN since the model capacity is reduced due to quantization. Therefore, when training QDNN with KD, we need to consider the relationship between the *size of teacher model* and the *temperature*.

Figure 4.2 (b) shows the test accuracies of the 2-bit ResNet20 trained with KD on the CIFAR-100 dataset. Since the CIFAR-100 includes 100 classes, the soft label distribution is not sharp and the optimum value of τ is usually lower than that of the CIFAR-10. More specifically, when τ is larger than 5 (purple line), the test accuracies are lower than 65.49% (green dotted line), the accuracy of the 2-bit ResNet20 trained with the hard label. The soft label can easily become too flat even with a small τ, thus the teacher's knowledge does not transfer well to the student network. When τ is not large (e.g. less than 5), the tendency is similar to CIFAR-10 experiment. When τ is 1 (blue line), the best performance is observed with ResNet20. As τ increases to 2 (yellow line) and 4 (grey line), the size of the best performing teacher model also changes to WRN20x1.5 and WRN20x1.7, respectively. This demonstrates that a proper value of *temperature* can improve the performance, but it should not be too high since the knowledge from the teacher network can disappear.

4.3.4 Gradual Soft Loss Reducing

Throughout the paper, we have discussed the effects of the *temperature* and the *size of teacher network* on the QDNN training with KD. Since the two hyperparameters are interrelated, careful parameter selection is required and it makes the training challenging. We also have the risk of cherry picking if the outcome cannot be predicted well without using the test result. Thus, we need to have a parameter setting technique that is fail-proof.

We have developed a KD technique that is much less sensitive to specific parameter setting for KD. At the beginning of the training, where the gradient changes a lot, we use the soft and hard losses equally and then, gradually reduce the amount of the soft loss as the training proceeds. We name this simple method as the *gradual soft loss reducing* (GSLR) technique. To evaluate the effectiveness of the GSLR, we train 2-bit ResNet20 while varying the *size of the teacher* and the *temperature* as shown in Figure 4.3. The results clearly show that GSLR greatly aids to improve the performance or at least yields the comparable results with the hard loss (black horizontal line). When comparing the traditional KD, shown in Figure 4.3 (a), and GSLR KD, in Figure 4.3 (b), we can find that the latter yields much more predictable result, by which reducing the risk of cherry picking.

4.4 Concluding Remarks

In this chapter, we investigate the teacher model choice and the impact of the hyperparameters in quantized deep neural networks training with knowledge distillation. We found that the teacher needs not be a quantized neural network. Instead, hyperparameters that control the shape of softmax distribution is more important. The hyperparameters for KD, which are the *temperature*, *loss weighting factor*, and *size of the teacher network*, are closely interrelated. When the *size of the teacher network* grows, increasing the *temperature* aids to boost the performance to some extent. We introduce

a simple training technique, *gradual soft loss reducing* (GSLR) for fail-safe KD training. At the beginning of the training, GSLR equally employs the hard and soft losses, and then gradually reduces the soft loss as the training proceeds. With careful hyper-parameter selection and the GSLR technique, we achieve the far better performances than those of previous studies for designing 2-bit quantized deep neural networks on the CIFAR-10 and CIFAR-100 datasets.

Chapter 5

SQWA: Stochastic Quantized Weight Averaging for Improving the Generalization Capability of Low-Precision Deep Neural Networks

5.1 Introduction

The purpose of DNN training is to achieve good generalization capability. Thus, it may not be optimal to use quantized DNN (QDNN) designs that approximate floating-point weights using elaborate coding techniques. In recent years, loss surface visualization has helped to improve the generalization capability of DNNs [3, 37, 111]. Fast geometric ensemble (FGE) [37] and stochastic weight averaging (SWA) [3] have been proposed based on the observation that local minima attained by stochastic gradient descent (SGD) training are closely connected [37]. FGE and SWA capture multiple models during training and ensemble or average the models to obtain a well-generalized network. Model averaging moves the averaged model to the center of the loss surface especially when training with SGD causes sticking at the local minimum.

In this study, we employed the model averaging technique to design a QDNN with improved generalization capability. We used cyclical learning rate scheduling for retraining of directly quantized networks, and captured multiple low-precision models

near the end of training. However, it is not straightforward to apply the previously developed SWA or FGE to QDNN design because the weight precision of the averaged model increases. For example, if we take the average of seven 2-bit models with ternary weights (-Δ, 0, and +Δ), then a 4-bit model is obtained(-7Δ, -6Δ,$\cdots$, 0, $\cdots$, +6Δ, and +7Δ). Thus, we must quantize it again to obtain a 2-bit model. Loss-surface aware DNN training is facilitated significantly by recently developed visualization techniques. However, the loss-surface of a QDNN is different from that of a floating-point model because the representation capability of a low-precision network is limited. In this study, we developed a new visualization technique for QDNNs by applying the quantization training algorithm. The new visualization method can successfully explain the mechanism of the proposed SQWA.

Our main contributions are as follows:

- We presented a new QDNN training technique, SQWA, to improve the generalization capability of QDNNs.
- With the proposed SQWA training scheme, we achieved state-of-the-art results on CIFAR-100 and ImageNet datasets.
- We proposed a loss visualization method for low-precision quantized DNNs.

5.2 Related works

5.2.1 Quantization of deep neural networks for efficient implementations

Typically, the precision of parameters and data is reduced for efficient implementations in real-time signal processing system designs. While audio and video signal processing demands precision exceeding eight bits, many DNNs function well with lower precisions, such as one or two bits. Particularly, the performance of low-precision DNNs can be improved considerably by conducting retraining after quantization. Thus, quantization is a highly promising approach for the efficient implementation of DNNs. The quantization training algorithm, first proposed by [1] and [15], has been combined

with various types of quantization methods such as symmetric uniform [16], asymmetric uniform [21], non-uniform [22], and differentiable [23, 25, 26, 27] quantizers. In recent years, a few elaborate techniques have been developed, such as employing knowledge distillation and carefully controlling the learning rate and bit-precision for improved generalization [30, 29, 39]. Weight normalization is adopted to avoid a long tail distribution of the model weights [31].

In this section, we focus on obtaining a good training scheme to optimize QDNNs. This approach is focused on developing well-generalized low-precision DNNs instead of developing elaborate quantization schemes. It is noteworthy that we only used the uniform quantization scheme for simplifying the hardware [112, 113] for inference. Non-uniform quantization can yield improved QDNN performance when the precision is the same; however, it demands additional operations, which can be time-consuming when hardware with conventional arithmetic blocks is involved.

5.2.2 Stochastic weight averaging and loss-surface visualization

Stochastic gradient descent (SGD) is the most widely used method for DNN training. However, the loss surface for SGD contains many sharp minima [33]; thus, SGD-based training exhibits overfitting frequently. Many regularization techniques can be applied for alleviating this problem, such as L2-loss, dropout, and cyclical learning rate scheduling [114, 81, 67]. The ensemble of models is known to increase the generalization capability. However, this method typically demands increased cost for training and inference. Fast geometric ensemble (FGE) is a recent technique for the ensemble of models [37]. Dropout [81] and dropconnect [115] can be interpreted as building an ensemble of models by weight averaging.

SWA is a recently developed regularization technique; that is based on the weight averaging of models captured during training with cyclical learning rate scheduling [3]. SWA demonstrates excellent performances in many CNN models on various datasets.

SWA can be explained using the loss visualization technique. The visualization

method for representing three models in a single loss surface has been suggested in [37] and [3]. In those studies, training algorithms SWA and FGE were presented by discovering that the local minima trained with SGD were interconnected. Furthermore, because the loss surface for training and test were different, the minima found by SGD during training were not necessarily the best for the test data. Instead, the average of the models indicated a significantly improved generalization capability. Figure 5.2 (a) depicts three models captured during cyclical learning rate scheduling and shows that the average is located near the center in the loss surface [3].

SWA has been applied to low-precision training. Stochastic weight averaging in low-precision (SWALP) employs SWA for a cost-efficient training, where a low-precision (e.g., 8-bit) model is trained with cyclical learning rate scheduling and models are captured during training at the lowest learning rate in the cycle. The captured models are then averaged to obtain the final full-precision model. Thus, SWALP is intended to design high-precision models and is vastly different from our work, which optimizes severely quantized models (e.g., 2-bit) for inference.

5.3 Quantization of DNN and loss surface visualization

In this section, we explain how DNNs can be optimally quantized using a retraining method and then present a loss surface visualization method for QDNNs. To this end, we first revisit a previous method [3, 37] to visualize three weight vectors in a single loss surface and explain the limitation when applied to QDNN loss surface visualization.

5.3.1 Quantization of deep neural networks

The weight vector, $\mathbf{w}$, of a deep neural network can be quantized in b-bit using a symmetric uniform quantizer as follows:

$$\mathrm{Q}^b(\mathbf{w}) = \mathrm{sign}(\mathbf{w}) \cdot \Delta \cdot \min\Big\{ \Big\lfloor \Big(\frac{|\mathbf{w}|}{\Delta} + 0.5 \Big) \Big\rfloor, \frac{(M-1)}{2} \Big\}, \tag{5.1}$$

where sign$(\cdot)$ is the sign function, Δ is the quantization step size, M is the number of quantization levels that can be computed with $2^b - 1$. A trained full-precision model can be directly quantized with Equation (5.1), but the performance will be significantly degraded when severe quantization such as 1- or 2-bit is employed. To relieve this problem, retraining on quantization domain is adopted in previous studies [1, 15, 19, 65] as follows:

$$l_i = \sum_{j \in A_i} w_{ij}^{(q)} y_j^{(q)} \tag{5.2}$$

$$y_i^{(q)} = \phi_i(l_i) \tag{5.3}$$

$$\delta_j = \phi_j^{'}(l_j) \sum_{i \in P_j} \delta_i w_{ij}^{(q)} \tag{5.4}$$

$$\frac{\partial E}{\partial w_{ij}} = -\delta_i y_j^{(q)} \tag{5.5}$$

$$w_{ij,new} = w_{ij} - \eta \left\langle \frac{\partial E}{\partial w_{ij}} \right\rangle \tag{5.6}$$

$$w_{ij,new}^{(q)} = Q_{ij}(w_{ij,new}) \tag{5.7}$$

where l_i is the logit of the unit i, δ_i is the error signal of the unit i, w_{ij} is the weight from the unit j to the unit i, y_j is the output activation of the unit j. η is the learning rate, A_i is the set of units anterior to the unit i, P_j is the set of units posterior to the unit j, $Q(\cdot)$ is the weight quantizer, $\phi(\cdot)$ is the activation function. The superscript (q) indicates the value is quantized, and $\langle \cdot \rangle$ is the average operation over the mini-batch. As described in Equation (5.2) to (5.7), the forward, backward, and gradient calculation is conducted with the quantized weights, but weight update adopts the full-precision parameters. This is because that the quantization step size, Δ, is usually much larger than the computed gradients, $\frac{\partial E}{\partial \mathbf{w}}$. The weights are not changed if the gradient is directly updated to the quantized weights.

5.3.2 Loss surface visualization for QDNNs

The visualization method in [3, 37] shows the location of three weight vectors $\mathbf{w}_1$, $\mathbf{w}_2$, and $\mathbf{w}_3$. For locating these three models on the same loss surface, the projection vectors $\mathbf{u}$ and $\mathbf{v}$ are formed as follows:

$$\mathbf{u} = (\mathbf{w}_2 - \mathbf{w}_1) \tag{5.8}$$

$$\mathbf{v} = (\mathbf{w}_3 - \mathbf{w}_1) - \langle \mathbf{w}_3 - \mathbf{w}_1, \mathbf{w}_2 - \mathbf{w}_1 \rangle / \left\| \mathbf{w}_2 - \mathbf{w}_1 \right\|^2 \tag{5.9}$$

$$\hat{\mathbf{u}} = \frac{\mathbf{u}}{\|\mathbf{u}\|} \tag{5.10}$$

$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|} \tag{5.11}$$

The normalized vectors $\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}$ form an orthonormal basis in the plane containing $\mathbf{w}_1$, $\mathbf{w}_2$, and $\mathbf{w}_3$. These three vectors can be visualized on a Cartesian grid in the basis $\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}$ using a set of points P.

$$P = \mathbf{w}_1 + x \cdot \hat{\mathbf{u}} + y \cdot \hat{\mathbf{v}}, \tag{5.12}$$

where x and y are the coordinates.

We trained the 2-bit ternary quantized ResNet-20 [5] on the CIFAR-100 dataset [48] using the retraining algorithm [1]. After the performance has fully converged during the retraining process, we captured three quantized models $\mathbf{w}_1^{(q)}$, $\mathbf{w}_2^{(q)}$, and $\mathbf{w}_3^{(q)}$ with time intervals[1] on the training epoch. Figure 5.1 (a) visualizes the three quantized networks using Equations (5.8) to (5.12).

It is noted that $\mathbf{w}_1^{(q)}$, $\mathbf{w}_2^{(q)}$, and $\mathbf{w}_3^{(q)}$ are located at '$\mathbf{w1}$', '$\mathbf{w2}$', and '$\mathbf{w3}$', respectively. The limitation of this visualization method when applied to QDNNs is obvious. Even though $\mathbf{w}_1^{(q)}$, $\mathbf{w}_2^{(q)}$, and $\mathbf{w}_3^{(q)}$ are quantized weights, the other points between them are represented in full-precision. Thus, the exact shape of the loss surface cannot be determined when the weights are quantized. Hence, we plot the loss surface after quantizing the high-precision location vector, P. It is noteworthy that we can employ the full-precision weight vectors from Equation (5.6) to compute Equation (5.10)

[1] $\mathbf{w}_1^{(q)}$, $\mathbf{w}_2^{(q)}$, and $\mathbf{w}_3^{(q)}$ that were captured at epochs 214, 232, and 250, respectively

and (5.11). We denote these two normalized vectors as $\hat{\mathbf{u}}^f$ and $\hat{\mathbf{v}}^f$ to avoid confusion; subsequently, the three quantized vectors can be visualized on a Cartesian grid using a set of quantized points, P^q, for QDNNs as follows:

$$\mathbf{w}^f = \mathbf{w}_1^f + x \cdot \hat{\mathbf{u}}^f + y \cdot \hat{\mathbf{v}}^f \tag{5.13}$$

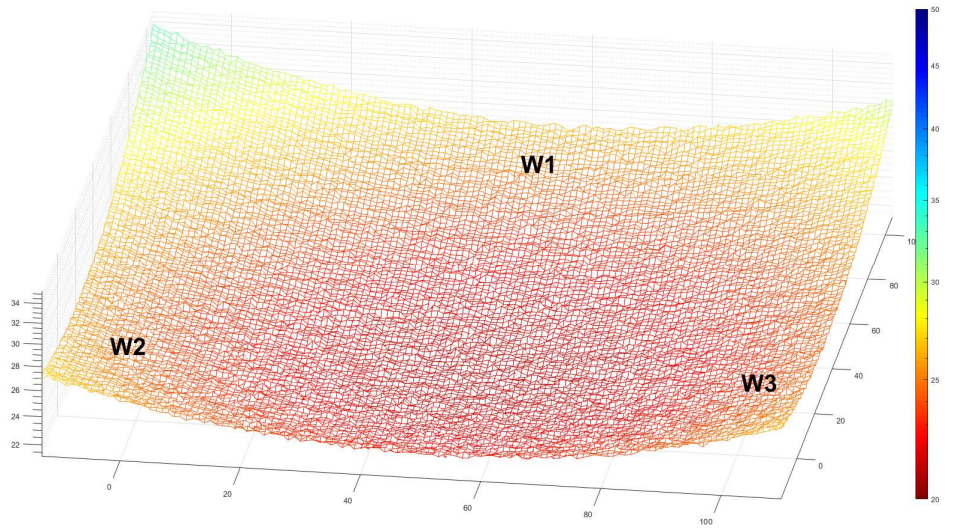
$$P^q = \text{sign}(\mathbf{w}^f) \cdot \Delta \cdot \min\Big\{ \Big\lfloor \Big(\frac{|\mathbf{w}^f|}{\Delta} + 0.5 \Big) \Big\rfloor, \frac{(M-1)}{2} \Big\} \tag{5.14}$$

where $\mathbf{w}_1^f$ is the full-precision weight vector that can be employed during retraining. We report the relationship of the three vectors $\mathbf{w}_1^{(q)}$, $\mathbf{w}_2^{(q)}$, and $\mathbf{w}_3^{(q)}$ obtained using the modified visualization method in Figure 5.1 (b). The relationship of the three quantized weight vectors, which cannot be observed in Figure 5.1 (a), is well represented. As they were captured in the epoch order ('$\mathbf{w1}$' $\rightarrow$ '$\mathbf{w2}$' $\rightarrow$ '$\mathbf{w3}$') during the retraining, a path along $\mathbf{w}_1^{(q)}$, $\mathbf{w}_2^{(q)}$ and $\mathbf{w}_3^{(q)}$ appeared. Because all of the points, $P^{(q)}$, were expressed in 2-bit quantized weights, the surface fluctuated strongly owing to quantization noise.

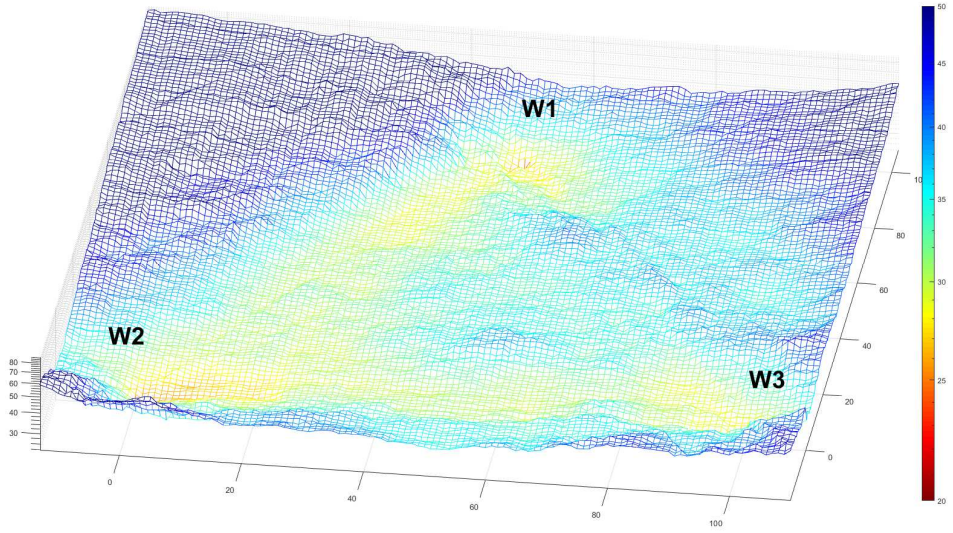
5.4 SQWA algorithm

Training a DNN can be regarded as guiding a model to near the center of the loss-surface of the training data. The weight quantization of a DNN incurs a large perturbation to the model, and even a well-trained DNN exhibits poor performance after a severe quantization. Thus, retraining is typically employed to return a model to the center of the training loss surface. The conventional fine-tuning approach that employs a low learning rate seeks to obtain a permissible nearby minimum in the quantized domain. In our opinion, this can be improved by employing more aggressive training methods.

The proposed SQWA retrains the quantized model using cyclical learning rate scheduling instead of low learning rates for fine-tuning. We captured multiple models during retraining and obtained the average of the captured models. It is noteworthy that the averaging process increases the bit-precision of the model. For example, if

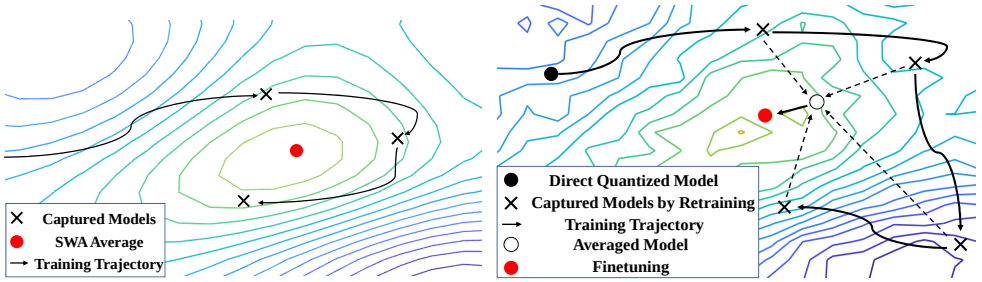


(a) Conventional [3]



(b) Ours

Figure 5.1: Visualization of three QDNNs in a single loss surface with the conventional method [3] (a) and ours (b). Three models are captured during fixed-point retraining. The points of $\mathbf{w1}$, $\mathbf{w2}$, and $\mathbf{w3}$ represent the captured models at 214th, 232th, and 250th epochs, respectively.



(a) SWA (b) SQWA (ours)

Figure 5.2: Intuitions of the SWA and the SQWA.

we take the average of seven ternary models, then a 4-bit model is obtained. Thus, we must re-quantize the averaged model, followed by fine-tuning using low learning rates.

SQWA can be explained as shown in Figure 5.2. The difference is that optimization using the quantized loss surface is required. As shown in Section 5.3, the quantized loss surface is rough when compared with its high-precision counterpart. Thus, optimization is more difficult with low learning rates. Cyclical learning rate scheduling, which uses high and low learning rates alternately, and weight averaging are more effective than fine-tuning for traversing the rugged loss surface.

We demonstrate the entire workflow of the SQWA in Figure 5.2 (b). The details are provided as follows.

Pretrain a full-precision model: We used high-performance floating-point models for the design of the QDNN, instead of directly designing a QDNN from scratch. This approach is more convenient considering the GPU-dominant training facilities available. According to our experiments, the performance of a quantized model is closely related to that of the original floating-point network. Thus, good training methods such as knowledge distillation (KD) [2] or SWA [3] are necessitated.

Quantize the full-precision model and retraining with cyclical LR scheduling: We first quantized the full-precision model from step 1 and then conducted retraining on the quantization domain with cyclical learning rate scheduling. We adopted discrete

cyclical learning rate scheduling for a better generalization [34]. Detailed guidelines for scheduling are as follows. First, we define all values of the learning rates for the full-precision model as η_f. Then, the maximum and minimum values of the cyclic learning rate scheduling are determined as $\eta_{\text{cycleMax}} = \frac{\max(\eta_f)}{10}$ and $\eta_{\text{cycleMin}} = \frac{\min(\eta_f)}{10}$, respectively. These values of the learning rate are highly related to the quantization error. The quantized weights, $\mathbf{w}^{(q)}$, can be interpreted as adding a quantization noise, $\mathbf{n}$, to the full-precision weight $\mathbf{w}^{(f)}$. The quantization noise $\mathbf{n}$ increases as the number of quantization bit, b, decreases. It is noteworthy that performing a direct quantization with low-precision, such as one or two bits, typically degrades the performance significantly. Thus, the smaller the number of bits, the larger is the required learning rate for recovering the performance. Because our SQWA training method is designed for severe quantizations (i.e., a 2-bit ternary model), $\frac{\max(\eta_f)}{10}$ would be a good choice.

One period of the discrete cyclical learning rate, c, is a hyperparameter that affects the training performance. The appropriate value of c is four to six epochs in our experiments. Thus, one or two learning rate steps can be considered between η_{cycleMax} and η_{cycleMin} to form discrete cyclical learning rate scheduling. We captured the models during training at the lowest learning rate (i.e., η_{cycleMin}).

Averaging the captured models: The third step is averaging the captured low precision models. Model averaging improves the generalization capability by moving the averaged model to the middle of the loss surface [3]. The number of captured models for averaging affects SQWA training. When employing a 2-bit ternary symmetric uniform quantizer, for example, each captured weight is represented as $-\Delta$, 0, and Δ. If we select seven captured weights for averaging, the averaged model has the representation level of -7Δ, -6Δ,$\cdots$, 0,$\cdots$, 6Δ, and 7Δ, which is a 4-bit QDNN. Averaging too few models will degrade the final performance, whereas averaging too many networks will render the training less efficient.

Re-quantization and fine-tuning of the averaged model: The final goal of SQWA is to obtain a low-precision model, such as a 2-bit model; thus, we must quantize the

Table 5.1: Train and test accuracies (%) of the full-precision ResNet-20 for the SQWA training on CIFAR-100 dataset. 'Conventional' means training without special techniques, 'KD' represents knowledge distillation [2], and 'SWA' is stochastic weight averaging technique [3].

	Train Acc.	Test Acc.
Conventional	90.12	68.43
KD [2]	87.55	71.06
SWA [3]	90.44	70.45
KD + SWA	87.02	**71.26**

averaged model into a low-precision one and fine-tune it with relatively low learning rates. We employed a monotonically decreasing learning rate scheduling for this step. Thus, we adopted the initial learning rate of $0.1\eta_{\text{cycleMax}}$ and trained three or four epochs. It is noteworthy that we decreased the learning rate at every epoch.

More detailed information and experimental results of our proposed method are reported in Section 5.5.

5.5 Experimental results

We evaluate the proposed SQWA method using the CIFAR-100 [48] and ImageNet [85] datasets.

5.5.1 CIFAR-100

Network and hyperparameter configuration: We trained ResNet-20 [5] and MobileNetV2 [90] for the CIFAR-100 dataset. The training hyperparameters are as follows. For full-precision training, the batch size was 128 and the number of epochs trained was 175. An SGD optimizer with a momentum of 0.9 was used. The learning rate began at 0.1 and decreased by 0.1 times at the 75th and 125th epochs. Addition-

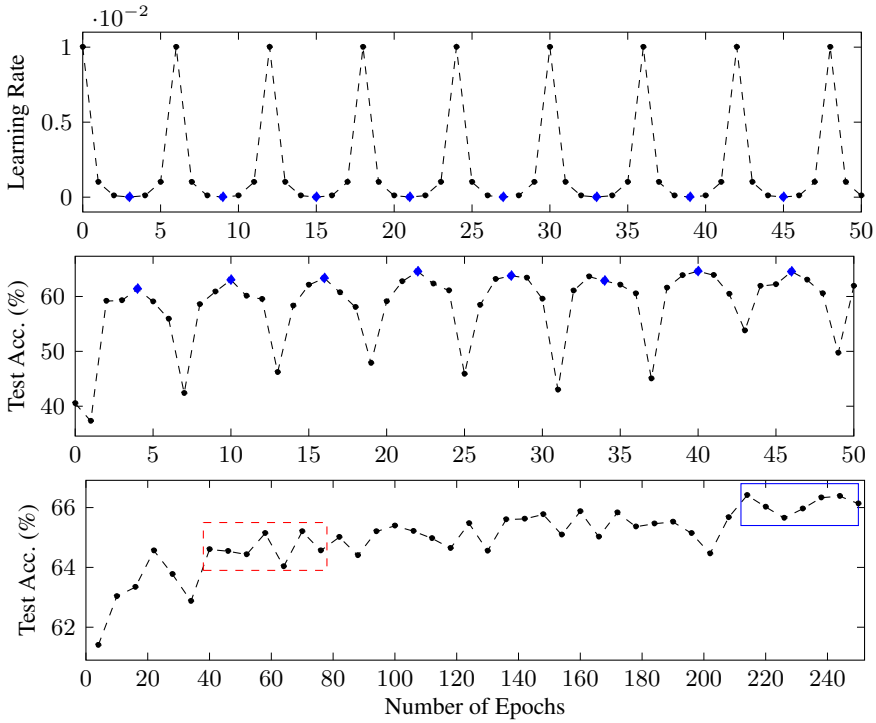


Figure 5.3: (**Top**): Cyclical learning rate scheduling for CIFAR-100 dataset, (**Middle**): the test accuracy curve with ResNet20, (**Bottom**): the sampled test accuracy curve from the every minimum learning rates with ResNet20.

ally, L2-loss was added with a scale of 5e-4. For QDNN retraining, the batch size and optimizer were the same as those of the full-precision one. The cyclical learning rate scheduling for retraining is described in Figure 5.3 (Top).

We captured the quantized models at the minimum points of the cyclical learning rate scheduling and obtained their average. To fine-tune the averaged model, the initial learning rate was set as 0.001 and decreased by 0.1 times at every epoch. We only performed three to four epochs for the fine-tuning. We did not employ L2-loss for the QDNN training as it conflicted with the clipping of the quantization.

Results: As described in Section 5.4, SQWA requires a pretrained full-precision model. We compare the full-precision ResNet20 that was developed with KD [2] and SWA [3]

in Table 5.1. The best full-precision model was trained by applying both KD and SWA. Its test accuracy was 71.26%. We selected this network as the original full-precision model.

In the next step of the SQWA training process, we performed QDNN training with cyclical learning rate scheduling, as depicted in Figure 5.3 (Top). We captured the models when the learning rates were the lowest in the cycles (i.e., blue diamonds in Figure 5.3 (Top and Middle)). To select the models for averaging, we considered two groups of the networks, as depicted in Figure 5.3 (Bottom). The first group (dashed red box) was selected at the beginning of the training and the other group (solid blue box) was captured after a sufficient number of training epochs has elapsed. We employed seven models for both groups, and their performances are compared in Table 5.2.

More specifically, the models in the first group were captured between the 40th and 76th[2] epochs. Their test accuracies were approximately 64.7% and the averaged model demonstrated an accuracy of 67.94%. It is noteworthy that we took the average of seven 2-bit ternary QDNNs ($-\Delta$, 0, and Δ), of which the resultant model was a 4-bit ($-7\hat{\Delta}$, $-6\hat{\Delta}, \cdots$, $0, \cdots$, $6\hat{\Delta}$, and $7\hat{\Delta}$) QDNN. We conducted re-quantization and fine-tuned the averaged model to obtain a final 2-bit QDNN, which yielded a test accuracy of 66.75%. The result of the second group was significantly better than that of the first group. The averaged 4-bit model yielded a test accuracy of 68.81%. After the fine-tuning, the final performance of the 2-bit QDNN was 67.75%. From these results, we can deduce the following:

- SQWA can be fully utilized when the models are captured after a sufficient convergence.
- Based on the observation of the direct quantization results in Table 5.2, the second group forms a wider minima in the loss surface than the first group. Because direct quantization can be interpreted as a noise injection operation, less performance degradation suggests that the model is laying in a wider minimum or at

[2]The training performance was too low to capture for models earlier than 40th epochs.

Table 5.2: Train and test accuracies (%) of the quantized model during retraining with cyclical learning rate scheduling on CIFAR-100 dataset. The left column represents the result obtained at the beginning phase of retraining, while the right shows that at the last phase, 214th to 250th epochs. 'Avg.' means the averaged model using 7 models during cyclical learning rate training with specific epochs, 'Direct' represents the direct quantization results of the averaged model, and 'Fine-tune' is the result after fine-tuning of direct quantized network.

Epoch (precision)	Train Acc.	Test Acc.	Epoch (precision)	Train Acc.	Test Acc.
76 (2-bit)	72.80	64.56	250 (2-bit)	76.12	66.13
70 (2-bit)	73.28	65.20	244 (2-bit)	75.68	66.38
64 (2-bit)	72.32	64.03	238 (2-bit)	75.33	66.33
58 (2-bit)	73.16	65.14	232 (2-bit)	75.32	65.96
52 (2-bit)	72.03	64.43	226 (2-bit)	75.14	65.65
46 (2-bit)	72.00	64.54	220 (2-bit)	75.08	66.02
40 (2-bit)	72.27	64.60	214 (2-bit)	75.78	66.41
Avg. (4-bit)	76.53	67.94	Avg. (4-bit)	78.95	68.81
Direct (2-bit)	62.89	56.93	Direct (2-bit)	70.31	62.52
Fine-tune (2-bit)	74.25	66.75	Fine-tune (2-bit)	**76.83**	**67.75**

Table 5.3: Comparison with literature in terms of the test accuracy (%) for quantized ResNet20 and MobileNetV2 on CIFAR-100.

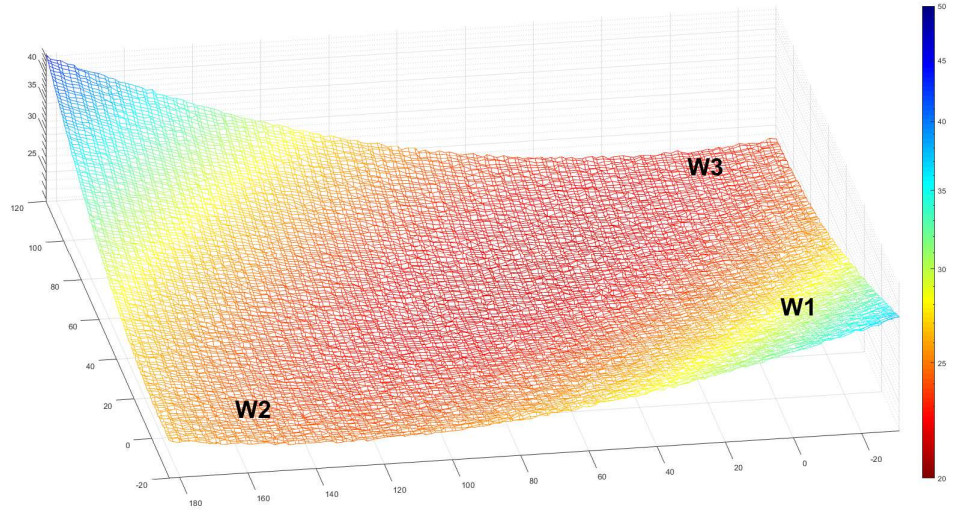
ResNet20	Quant Level	Test Acc.
DoReFa-Net [65]	4-level	66.95
Residual [75]	4-level	65.97
LQ-Net [71]	4-level	66.53
WNQ [31]	4-level	67.42
HLHLp [39]	Ternary	66.44
KDQ [116]	Ternary	67.00
SQWA (ours)	Ternary	**67.75**
KDQ [116]	Binary	60.14
SQWA (ours)	Binary	**62.32**
MobileNetV2	Quantization Level	Test Acc. (%)
L2Quant [4]	Ternary	74.97
HLHLp [39]	Ternary	75.51
SQWA (ours)	Ternary	**76.73**

the center of the loss surface.

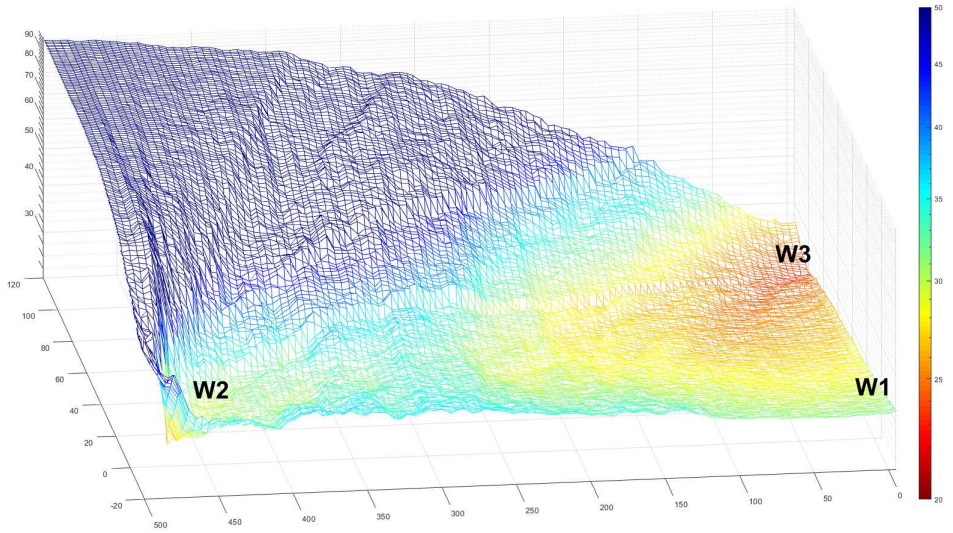
We compare our SQWA results with those of previous studies in Table 5.3. Our proposed SQWA outperforms the methods of previous studies. In particular, SQWA indicated 0.8%, 1.78%, 1.22%, and 0.33% higher test accuracies than DoReFa-Net [65], Residual [75], LQ-Net [71], and WNQ [31], respectively. This result is encouraging as the previous studies employed 2-bit 4-level quantizers, whereas we adopted 2-bit ternary and 1-bit binary quantizer. Furthermore, we compare our result to those involving 2-bit ternary and 1-bit binary quantizer. Our method with 2-bit ternary quantizer outperformed HLHLp [39] and KDQ [116] in terms of test accuracy by 1.31% and 0.75%, respectively. For the binary weights, SQWA achieves 2.18% higher accuracy than KDQ. It should be noted that KDQ improves the performance of the QDNN us-

ing the KD. Additionally, we exploit the KD technique to obtain a high-performance full-precision model. Because SQWA outperforms KDQ, it suggests that SQWA training methods can combine with KD. Furthermore, we evaluated the proposed SQWA method using MobileNetV2, which has a larger number of parameters than ResNet20. We trained a full-precision MobileNetV2 with KD and SWA and achieved a test accuracy of 77.64%. We exploited SQWA with the same cyclical learning rate scheduling used in the ResNet20 experiment. After a sufficient number of epochs, we captured seven models to establish a 4-bit averaged model and fine-tuned it. Our final 2-bit MobileNetV2 yielded the test accuracy that was 1.76% and 1.22% higher than those of L2Quant [4] and HLHLp [39], respectively.

Discussion: We visualize the SQWA training results using the previous method [3] and our method in Figure 5.4 (a) and (b), respectively. The results show similar trends as reported in Figure 5.1 (a) and (b). The original visualization method [3] cannot demonstrate the relationship between the QDNNs. However, our modified method clearly depicts the relationship of the three quantized models. More specifically, we visualized three models from "the final SQWA model" ($\mathbf{w3}$), "the 2-bit quantized version of the averaged model" ($\mathbf{w1}$), and "one of the captured models during the cyclical learning rate" ($\mathbf{w2}$). Thus, $\mathbf{w3}$ can be obtained by fine-tuning $\mathbf{w1}$, and $\mathbf{w2}$ is one of the models to obtain $\mathbf{w1}$. It is noteworthy that all three models were 2-bit ternary QDNNs. Figure 5.4 (a) does not provide a clear correlation of $\mathbf{w1}$, $\mathbf{w2}$, and $\mathbf{w3}$. It shows that the relationship between $\mathbf{w1}$ and $\mathbf{w2}$ is almost similar to that between $\mathbf{w1}$ and $\mathbf{w3}$. Our proposed visualization method, as shown in Figure 5.4 (b), clearly distinguishes the difference between them. $\mathbf{w1}$ is fine-tuned with a low learning rate to obtain $\mathbf{w3}$, and they should exist in the same basin of the loss surface. Furthermore, it is clear that the distance between $\mathbf{w2}$ and $\mathbf{w1}$ is much larger than that between $\mathbf{w3}$ and $\mathbf{w1}$. We expect the proposed visualization method for the QDNNs is very useful for understanding the relationship between quantized networks in future studies.



(a) Train ([3])



(b) Train (ours)

Figure 5.4: Visualization in terms of train accuracies of three quantized models on a single loss surface. (a) is depicted by [3] and (b) is by ours. The points of '$\mathbf{w2}$', '$\mathbf{w1}$', and '$\mathbf{w3}$' represent 'Epoch 214', 'Direct', and 'Fine-tune' in Table 5.2, respectively.

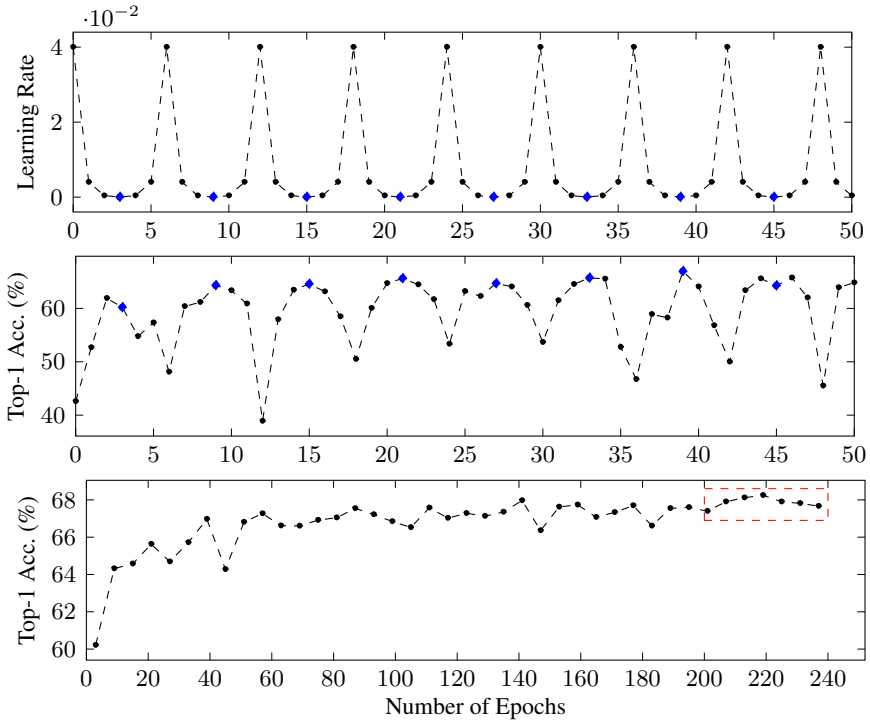


Figure 5.5: (**Top**): Cyclical learning rate scheduling for ImageNet dataset, (**Middle**): a validation top-1 accuracy curve with ResNet18, (**Bottom**): the sampled top-1 accuracy curve from the every minimum learning rates in the cycle.

5.5.2 ImageNet

Network and hyperparameter configuration: We trained ResNet-18 [5] for the ILSVRC 2012 classification dataset [117]. The training hyperparameters are as follows. We trained the full-precision model with a batch size of 1024 on 90 epochs, with the initial learning rate of 0.4 and decreased it by 0.1 times at the 30th, 60th, and 80th epochs. It is noteworthy that the initial learning rate of 0.4 was determined using the *linear scaling rule*, as suggested in [118]. We used the SGD optimizer with a momentum of 0.9. Additionally, L2-loss was added with a scale of 1e-4. For the QDNN retraining, the batch size and optimizer were the same as those of the full-precision training. Because we employed SQWA training, cyclical learning rate scheduling was

Table 5.4: Detailed ImageNet Top-1 and Top-5 accuracies (%) of the quantized model during retraining with cyclical learning rate scheduling for ResNet18. 'Avg.' means the averaged model using seven models that obtained from 202th to 238th epochs, 'Direct' represents the direct quantization results of the averaged model, and 'Fine-tune' is the result after fine-tuning of direct quantized network.

Epoch (precision)	Top-1 Acc.	Top-5 Acc.
238 (2-bit)	67.66	87.83
232 (2-bit)	67.81	88.04
226 (2-bit)	67.90	88.00
220 (2-bit)	68.25	88.10
214 (2-bit)	68.12	88.09
208 (2-bit)	67.90	87.89
202 (2-bit)	67.40	87.75
Avg. (4-bit)	69.66	89.12
Direct (2-bit)	60.78	83.01
Fine-tune (2-bit)	**69.34**	**88.77**

employed, as shown in Figure 5.5 (Top). The maximum and minimum values of the learning rates were determined by considering the learning rate of the full-precision training, as suggested in Section 5.4.

We captured the models at the minimum points of the cyclical learning rate scheduling and obtained their average. To fine-tune the averaged model, the initial learning rate was set to 0.004 and decreased by 0.1 times at every epoch. We executed five epochs for the fine-tuning and did not employ L2-loss for the QDNN training.

Results: Apprentice [30] and QKD [29] employed KD to improve the performance of QDNNs. Thus, we employed KD loss for the full-precision training and achieved a top-1 accuracy of 71.68%. With this full-precision model, we performed SQWA with cyclical learning rate scheduling and captured the quantized models at the lowest

leaning rate in the cycles, as described in Figure 5.5 (Middle). The accuracy curve of the captured models is depicted in Figure 5.5 (Bottom). We adopted the last seven models for averaging and fine-tuning it to obtain the final 2-bit QDNN. The results are reported in Table 5.4. The performance of the averaged model is 69.66%, and we obtained 60.78% as the direct quantization results. It is noteworthy that the averaged model has a 4-bit precision. After the fine-tuning, the accuracy improved to 69.34%, which is significantly better than those of the captured models.

We conducted additional experiments to investigate the effect of number of models on averaging. As discussed in Section 5.4, the number of captured models is related to the precision of the averaged model. More specifically, the averaged model using three 2-bit ternary models becomes a 3-bit QDNN. Thus, we employed 3, 7, 15, and 31 models such that the precision of the averaged model was 3, 4, 5, and 6 bits, respectively, and fine-tuned each model. The results are reported in Table 5.5. Adopting three models afforded a top-1 accuracy of 69.18%, which is 0.22% worse than the seven models. When 15 models were employed, the top-1 accuracy was similar to that of the 7 models but the top-5 accuracy was 0.2% higher. Using 31 models did not improve the performance.

We compare our results with those of previous studies in Table 5.6. Our result outperformed those of previous studies including the 2-bit 4-level (LQ-NET [71] and WNQ [31]) and ternary (TWN [16], TTQ [17], INQ [119], ADMM [91], QNet [25], QIL [24], and Apprentice [30]). More specifically, we achieved a top-1 accuracy of 69.4%. Only the QNet result is comparable with our result, although it is 0.3% lower. This result is significant because QNet employs non-linear quantizer while we adopt uniform quantization.

Table 5.5: Effect of the number of captured models for averaging. The results are reported in terms of top-1 accuracy after fine-tuning to achive final 2-bit QDNN model on the ImageNet dataset.

# of models	3	7	15	31
(bit-precision)	(3-bit)	(4-bit)	(5-bit)	(6-bit)
Top-1 Acc.	69.2	69.4	69.4	69.4
Top-5 Acc.	88.7	88.7	88.9	88.8

5.6 Concluding remarks

We proposed an SQWA algorithm for the optimum quantization of deep neural networks. The model averaging technique was employed to improve the generalization capability of QDNNs by moving them to the wide minimum of the loss surface. Because SQWA captures multiple models for averaging using only a single training with cyclical learning rate scheduling, it is easy to implement and can be applied to many different models. Although we only used a uniform quantization scheme, our results far exceeded the performances of existing non-uniform quantized models in the CIFAR-100 and ImageNet datasets. Additionally, we presented a visualization technique that showed the location of three QDNNs on a single loss surface. Because the proposed method is a training scheme to improve the generalization of QDNNs, it can be combined with other elaborate and non-uniform quantization schemes.

Table 5.6: Comparison with literature in terms of the validation accuracy (%) for 2-bit ResNet18 on ImageNet.

Methods	Quant Level	Top-1	Top-5
TWN [16]	Ternary	61.8	84.2
TTQ [17]	Ternary	66.6	87.2
INQ [119]	Ternary	66.0	87.1
ADMM [91]	Ternary	67.0	87.5
LQ-Net [71]	4-level	68.0	88.0
QNet [25]	Ternary	69.1	**88.9**
WNQ [31]	4-level	67.7	87.9
QIL [24]	Ternary	68.1	88.3
Apprentice [30]	Ternary	68.5	88.4
SQWA (ours)	Ternary	**69.4**	**88.9**

Chapter 6

Conclusion

In this dissertation, we studied the design of quantized deep neural networks (QDNNs) to improve their generalization capability. In particular, by analyzing the performance resiliency of QDNNs, high-low-high-low-precision (HLHLp) training, QDNNs training with knowledge distillation (KD), and stochastic quantized weight averaging (SQWA) techniques were developed.

We analyzed the performance of QDNNs by not only changing the arithmetic precision, but also varying network complexity. In addition, we employed layer-wise sensitivity analysis for quantization, and our results clearly showed that the input or output layers of DNNs are most sensitive to quantization. When the complexity of DNNs is reduced by lowering either the number of units, feature maps, or hidden layers, the performance gap between the full-precision and quantized model increases. Thus, a large network that contains redundant representation capability for given training data is not considerably negatively affected by lowered precision; however, a considerably compact network is. Furthermore, we presented two simple quantization techniques, namely the adaptive step size retraining and gradual quantization schemes, both of which led to increased performance for compact networks.

We also proposed the HLHLp training scheme that can improve the generalization capability of QDNNs. This training scheme employs high-low-high-low bit precision

with cyclical learning rate scheduling. Our results indicate that QDNNs trained using the HLHLp scheme have considerably better performance compared with those that are trained using conventional method.

Quantization training with KD was also performend in our study. In particular, we explored the effect of teacher network configuration on the quantization of DNNs, and found that the softmax distribution generated by the teacher network plays a key role in KD training. In addition, we showed that the softmax distribution of the teacher model could be controlled by the hyperparameters of KD. Moreover, we presented the gradual soft loss reducing (GSLR) technique to avoid cherry picking during the QKD training.

The SQWA training technique was also introduced in our study. The SQWA technique involves captures multiple models that are then averaged during a single training phase; it is easy to implement and can be applied to different models. As a result of SQWA training, the trained QDNNs showed significantly improved performances. In addition, we proposed a visualization method for three QDNNs on a single loss surface in the quantization domain.

In this doctoral study, we show that improving the generalization capability of QDNNs can lead to a significant reduction in performance degradation. This approach is quite different from that of previous works that have tried to employ sophisticated quantization methods to improve the performance of QDNNs. In summary, our study indicates that in the cases wherein only limited resources are available for DNN model design, our proposed quantization training schemes, including HLHLp, KDQ, and SQWA methods can be exploited for improving the generalization capability. .

Bibliography

[1] Kyuyeon Hwang and Wonyong Sung, "Fixed-point feedforward deep neural network design using weights +1, 0, and -1," in *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*. IEEE, 2014, pp. 1–6.

[2] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[3] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson, "Averaging weights leads to wider optima and better generalization," *arXiv preprint arXiv:1803.05407*, 2018.

[4] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1131–1135.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 770–778.

[6] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al., "Deep speech 2: End-to-end speech recognition in english

and mandarin," in *International conference on machine learning*, 2016, pp. 173–182.

[7] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria, "Recent trends in deep learning based natural language processing," *ieee Computational intelligenCe magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[8] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[9] Dongyoon Han, Jiwhan Kim, and Junmo Kim, "Deep pyramidal residual networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5927–5935.

[10] Wonyong Sung and Ki-II Kum, "Simulation-based word-length optimization method for fixed-point digital signal processing systems," *Signal Processing, IEEE Transactions on*, vol. 43, no. 12, pp. 3087–3090, 1995.

[11] B Zahir M Hussain et al., "Short word-length LMS filtering," in *Signal Processing and Its Applications, 2007. ISSPA 2007. 9th International Symposium on*. IEEE, 2007, pp. 1–4.

[12] Perry Moerland and Emile Fiesler, "Neural network adaptations to hardware implementations," Tech. Rep., IDIAP, 1997.

[13] Jordan L Holt and Thomas E Baker, "Back propagation simulations using limited precision calculations," in *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*. IEEE, 1991, vol. 2, pp. 121–126.

[14] Emile Fiesler, Amar Choudry, and H John Caulfield, "Weight discretization paradigm for optical neural networks," in *The Hague'90, 12-16 April*. International Society for Optics and Photonics, 1990, pp. 164–173.

[15] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 3123–3131.

[16] Li Fengfu, Zhang Bo, and Liu Bin, "Ternary weight networks," in *NIPS Workshop on EMDNN*, 2016, vol. 118, p. 119.

[17] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally, "Trained ternary quantization," *International Conference on Learning Representations (ICLR)*, 2017.

[18] Sungho Shin, Kyuyeon Hwang, and Wonyong Sung, "Fixed-point performance analysis of recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 976–980.

[19] Sungho Shin, Yoonho Boo, and Wonyong Sung, "Fixed-point optimization of deep neural networks with adaptive step size retraining," in *2017 IEEE International conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2017, pp. 1203–1207.

[20] Wonyong Sung, Sungho Shin, and Kyuyeon Hwang, "Resiliency of deep neural networks under quantization," *arXiv preprint arXiv:1511.06488*, 2015.

[21] Shu-Chang Zhou, Yu-Zhi Wang, He Wen, Qin-Yao He, and Yu-Heng Zou, "Balanced quantization: An effective and efficient approach to quantized neural networks," *Journal of Computer Science and Technology*, vol. 32, no. 4, pp. 667–682, 2017.

[22] Daisuke Miyashita, Edward H Lee, and Boris Murmann, "Convolutional neural networks using logarithmic data representation," *arXiv preprint arXiv:1603.01025*, 2016.

[23] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan, "PACT: Parameterized clipping activation for quantized neural networks," *arXiv preprint arXiv:1805.06085*, 2018.

[24] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi, "Learning to quantize deep networks by optimizing quantization intervals with task loss," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4350–4359.

[25] Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua, "Quantization networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7308–7316.

[26] Lu Hou, Quanming Yao, and James T Kwok, "Loss-aware binarization of deep networks," *arXiv preprint arXiv:1611.01600*, 2016.

[27] Lu Hou and James T Kwok, "Loss-aware weight quantization of deep networks," *arXiv preprint arXiv:1802.08635*, 2018.

[28] Tim Salimans and Durk P Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 901–909.

[29] Antonio Polino, Razvan Pascanu, and Dan Alistarh, "Model compression via distillation and quantization," in *International Conference on Learning Representations*, 2018.

[30] Asit Mishra and Debbie Marr, "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy," in *International Conference on Learning Representations*, 2018.

[31] Wen-Pu Cai and Wu-Jun Li, "Weight normalization based quantization for deep neural network compression," *arXiv preprint arXiv:1907.00593*, 2019.

[32] Guandao Yang, Tianyi Zhang, Polina Kirichenko, Junwen Bai, Andrew Gordon Wilson, and Christopher De Sa, "Swalp: Stochastic weight averaging in low-precision training," *arXiv preprint arXiv:1904.11943*, 2019.

[33] Sepp Hochreiter and Jürgen Schmidhuber, "Flat minima," *Neural Computation*, vol. 9, no. 1, pp. 1–42, 1997.

[34] Stanisław Jastrzkebski, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey, "Three factors influencing minima in SGD," *arXiv preprint arXiv:1711.04623*, 2017.

[35] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *International Conference on Learning Representations (ICLR)*, 2017.

[36] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein, "Visualizing the loss landscape of neural nets," in *Advances in Neural Information Processing Systems (NIPS)*, 2018, pp. 6391–6401.

[37] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson, "Loss surfaces, mode connectivity, and fast ensembling of dnns," in *Advances in Neural Information Processing Systems*, 2018, pp. 8789–8798.

[38] Sungho Shin, Kyuyeon Hwang, and Wonyong Sung, "Fixed point performance analysis of recurrent neural networks," *arXiv preprint arXiv:1512.01322*, 2015.

[39] Sungho Shin, Jinhwan Park, Yoonho Boo, and Wonyong Sung, "HLHLp: Quantized neural networks training for reaching flat minima in loss surface," in *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[40] Darryl D Lin, Sachin S Talathi, and V Sreekanth Annapureddy, "Fixed point quantization of deep convolutional networks," in *ICML 2016: 33rd International Conf. Machine Learning*, 2016.

[41] Song Han, Huizi Mao, and William J Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[42] Jonghong Kim, Kyuyeon Hwang, and Wonyong Sung, "X1000 real-time phoneme recognition vlsi using feed-forward deep neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 7510–7514.

[43] Jinhwan Park and Wonyong Sung, "FPGA based implementation of deep neural networks using on-chip memory only," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 1011–1015.

[44] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally, "EIE: efficient inference engine on compressed deep neural network," in *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*, 2016, pp. 243–254.

[45] Minjae Lee, Kyuyeon Hwang, Jinhwan Park, Sungwook Choi, Sungho Shin, and Wonyong Sung, "Fpga-based low-power speech recognition with recurrent neural networks," in *2016 IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2016, pp. 230–235.

[46] Nicholas J Fraser, Yaman Umuroglu, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers, "Scaling binarized neural networks on reconfigurable logic," in *To appear in the PARMA-DITAM workshop at HiPEAC*, 2017, vol. 2017.

[47] A Krizhevskey, "CUDA-convnet," 2014.

[48] Alex Krizhevsky, Geoffrey Hinton, et al., "Learning multiple layers of features from tiny images," Tech. Rep., Citeseer, 2009.

[49] Dong Yu, Alex Acero Deng, George Dahl, Frank Seide, and Gang Li, "More data + deeper model = better accuracy," in *keynote at International Workshop on Statistical Machine Learning for Speech Processing*, 2012.

[50] Dong Yu, Frank Seide, Gang Li, and Li Deng, "Exploiting sparseness in deep neural networks for large vocabulary speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, 2012, pp. 4409–4412.

[51] Jian Xue, Jinyu Li, and Yifan Gong, "Restructuring of deep neural network acoustic models with singular value decomposition.," in *INTERSPEECH*, 2013, pp. 2365–2369.

[52] Roberto Rigamonti, Amos Sironi, Vincent Lepetit, and Pascal Fua, "Learning separable filters," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE, 2013, pp. 2754–2761.

[53] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 41–48.

[54] John S Garofolo, Lori F Lamel, William M Fisher, Jonathon G Fiscus, and David S Pallett, "Darpa timit acoustic-phonetic continous speech corpus cd-

rom. nist speech disc 1-1.1," *NASA STI/Recon Technical Report N*, vol. 93, pp. 27403, 1993.

[55] Alan Graves, Abdel-rahman Mohamed, and Geoffrey Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 6645–6649.

[56] Ilya Sutskever, James Martens, George E Dahl, and Geoffrey E Hinton, "On the importance of initialization and momentum in deep learning.," *ICML (3)*, vol. 28, pp. 1139–1147, 2013.

[57] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015, pp. 448–456.

[58] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng, "Reading digits in natural images with unsupervised feature learning," *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[59] Pierre Sermanet, Soumith Chintala, and Yann LeCun, "Convolutional neural networks applied to house numbers digit classification," in *Pattern Recognition (ICPR), 2012 21st International Conference on*. IEEE, 2012, pp. 3288–3291.

[60] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[61] Alex Krizhevsky, "cuda-convnet: High-performance c++/cuda implementation of convolutional neural networks," 2012.

[62] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber, "Learning precise timing with lstm recurrent networks," *Journal of machine learning research*, vol. 3, no. Aug, pp. 115–143, 2002.

[63] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations.," *The Journal of Machine Learning Research*, vol. 18, no. 187, pp. 1–30, 2017.

[64] Chen Xu, Jianqiang Yao, Zhouchen Lin, Wenwu Ou, Yuanbin Cao, Zhirong Wang, and Hongbin Zha, "Alternating multi-bit quantization for recurrent neural networks," *International Conference on Learning Representations (ICLR)*, 2018.

[65] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[66] Ilya Loshchilov and Frank Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *International Conference on Learning Representations (ICLR)*, 2017.

[67] Leslie N Smith, "Cyclical learning rates for training neural networks," in *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*. IEEE, 2017, pp. 464–472.

[68] Eunhyeok Park, Junwhan Ahn, and Sungjoo Yoo, "Weighted-entropy-based quantization for deep neural networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 7197–7205.

[69] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning (ICML)*, 2015, pp. 1737–1746.

[70] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos, "Deep learning with low precision by half-wave gaussian quantization," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 5406–5414.

[71] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 365–382.

[72] Joachim Ott, Zhouhan Lin, Ying Zhang, Shih-Chii Liu, and Yoshua Bengio, "Recurrent neural networks with limited numerical precision," *arXiv preprint arXiv:1608.06902*, 2016.

[73] Qinyao He, He Wen, Shuchang Zhou, Yuxin Wu, Cong Yao, Xinyu Zhou, and Yuheng Zou, "Effective quantization methods for recurrent neural networks," *arXiv preprint arXiv:1611.10176*, 2016.

[74] Supriya Kapur, Asit Mishra, and Debbie Marr, "Low precision RNNs: Quantizing RNNs without losing accuracy," *arXiv preprint arXiv:1710.07706*, 2017.

[75] Yiwen Guo, Anbang Yao, Hao Zhao, and Yurong Chen, "Network sketching: Exploiting binary structure in deep CNNs," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, vol. 2.

[76] Peiqi Wang, Xinfeng Xie, Lei Deng, Guoqi Li, Dongsheng Wang, and Yuan Xie, "Hitnet: Hybrid ternary recurrent neural network," in *Advances in Neural Information Processing Systems (NIPS)*, 2018, pp. 602–612.

[77] Arash Ardakani, Zhengyun Ji, Sean C Smithson, Brett H Meyer, and Warren J Gross, "Learning recurrent binary/ternary weights," *International Conference on Learning Representations (ICLR)*, 2019.

[78] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 2016, pp. 525–542.

[79] Wei Wen, Yandan Wang, Feng Yan, Cong Xu, Chunpeng Wu, Yiran Chen, and Hai Li, "Smoothout: Smoothing out sharp minima to improve generalization in deep learning," *arXiv preprint arXiv:1805.07898*, 2018.

[80] Robert Kleinberg, Yuanzhi Li, and Yang Yuan, "An alternative view: When does sgd escape local minima?," *arXiv preprint arXiv:1802.06175*, 2018.

[81] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan 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.

[82] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[83] Anders Krogh and John A Hertz, "A simple weight decay can improve generalization," in *Advances in Neural Information Processing Systems (NIPS)*, 1992, pp. 950–957.

[84] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[85] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al., "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[86] Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger, "The penn treebank: annotating predicate argument structure," in *Proceedings of the workshop on Human Language Technology*. Association for Computational Linguistics, 1994, pp. 114–119.

[87] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[88] Douglas B Paul and Janet M Baker, "The design for the Wall Street Journal-based CSR corpus," in *Proceedings of the workshop on Speech and Natural Language*. Association for Computational Linguistics, 1992, pp. 357–362.

[89] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu, "Deeply-supervised nets," in *Artificial Intelligence and Statistics*, 2015, pp. 562–570.

[90] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[91] Cong Leng, Zesheng Dou, Hao Li, Shenghuo Zhu, and Rong Jin, "Extremely low bit neural network: Squeeze the last bit out with admm," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[92] Dan Hendrycks and Thomas G Dietterich, "Benchmarking neural network robustness to common corruptions and surface variations," *arXiv preprint arXiv:1807.01697*, 2018.

[93] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[94] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[95] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals, "Recurrent neural network regularization," *International Conference on Learning Representations (ICLR)*, 2015.

[96] Yajie Miao, Mohammad Gowayyed, and Florian Metze, "EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding," in *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*. IEEE, 2015, pp. 167–174.

[97] Naveen Parihar, Joseph Picone, David Pearce, and Hans-Günter Hirsch, "Performance analysis of the aurora large vocabulary baseline system," in *2004 12th European Signal Processing Conference*. IEEE, 2004, pp. 553–556.

[98] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 535–541.

[99] Zhiyuan Tang, Dong Wang, and Zhiyong Zhang, "Recurrent neural network training with dark knowledge transfer," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 5900–5904.

[100] Xuemeng Song, Fuli Feng, Xianjing Han, Xin Yang, Wei Liu, and Liqiang Nie, "Neural compatibility modeling with attentive knowledge distillation," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, 2018, pp. 5–14.

[101] Taichi Asami, Ryo Masumura, Yoshikazu Yamaguchi, Hirokazu Masataki, and Yushi Aono, "Domain adaptation of dnn acoustic models using knowledge

distillation," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 5185–5189.

[102] Junpeng Wang, Liang Gou, Wei Zhang, Hao Yang, and Han-Wei Shen, "Deepvid: Deep visual interpretation and diagnosis for image classifiers via knowledge distillation," *IEEE transactions on visualization and computer graphics*, vol. 25, no. 6, pp. 2168–2180, 2019.

[103] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio, "Fitnets: Hints for thin deep nets," *arXiv preprint arXiv:1412.6550*, 2014.

[104] Mandar Kulkarni, Kalpesh Patil, and Shirish Karande, "Knowledge distillation using unlabeled mismatched images," *arXiv preprint arXiv:1703.07131*, 2017.

[105] Wonpyo Park, Dongju Kim, Yan Lu, and Minsu Cho, "Relational knowledge distillation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3967–3976.

[106] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4133–4141.

[107] Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, and Hassan Ghasemzadeh, "Improved knowledge distillation via teacher assistant: Bridging the gap between student and teacher," *arXiv preprint arXiv:1902.03393*, 2019.

[108] Sergey Zagoruyko and Nikos Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[109] Yoonho Boo, Sungho Shin, and Wonyong Sung, "Memorization capacity of deep neural networks under parameter quantization," in *ICASSP 2019-2019*

IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2019, pp. 1383–1387.

[110] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian Reid, "Towards effective low-bitwidth convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7920–7928.

[111] Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred A Hamprecht, "Essentially no barriers in neural network energy landscape," *arXiv preprint arXiv:1803.00885*, 2018.

[112] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 65–74.

[113] Kota Ando, Kodai Ueyoshi, Kentaro Orimo, Haruyoshi Yonekawa, Shimpei Sato, Hiroki Nakahara, Shinya Takamaeda-Yamazaki, Masayuki Ikebe, Tetsuya Asai, Tadahiro Kuroda, et al., "Brein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 tops at 0.6 w," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 4, pp. 983–994, 2017.

[114] Twan van Laarhoven, "L2 regularization versus batch and weight normalization," *arXiv preprint arXiv:1706.05350*, 2017.

[115] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus, "Regularization of neural networks using dropconnect," in *International conference on machine learning*, 2013, pp. 1058–1066.

[116] Sungho Shin, Yoonho Boo, and Wonyong Sung, "Knowledge distillation for optimization of quantized deep neural networks," *arXiv preprint arXiv:1909.01688*, 2019.

[117] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.

[118] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[119] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.

초 록

최근 깊은 신경망(deep neural network, DNN)은 영상, 음성 인식 및 합성 등 다양한 분야에서 좋은 성능을 보이고 있다. 하지만 대부분의 인공신경망은 많은 가중치(parameter) 수와 계산량을 요구하여 임베디드 시스템에서의 동작을 방해한다. 인공신경망은 낮은 정밀도에서도 잘 동작하는 인간의 신경세포를 모방하였기 때문에 낮은 정밀도에서도 잘 동작할 가능성을 가지고 있다. 인공신경망의 양자화(quantization)는 이러한 특징을 이용한다. 일반적으로 깊은 신경망 고정소수점 양자화는 8-bit 이상의 단어길이에서 부동소수점과 유사한 성능을 얻을 수있지만, 그보다 낮은 1-, 2-bit에서는 성능이 떨어진다. 이러한 문제를 해결하기 위해 기존 연구들은 불균형 양자화기나 적응적 양자화 등의 더 정밀한 인공신경망 양자화 방법을 사용하였다.

본 논문은 기존의 연구와 매우 다른 방법을 제시한다. 본 연구는 고정 소수점 네트워크의 일반화능력을 향상시키는데 초점을 맞추었으며, 이를 위해 재훈련(retraining) 알고리즘에 기반하여 양자화된 인공신경망의 성능을 분석한다. 성능 분석은 레이어별 민감도 측정(layer-wise sensitivity analysis)에 기반한다. 또한 양자화 모델의 넓이와 깊이에 따른 성능도 분석한다. 분석된 결과를 바탕으로 양자화 스텝 적응 훈련법(quantization step size adaptation)과 점진적 양자화 훈련 방법(gradual quantization)을 제안한다. 양자화된 신경망 훈련시 양자화 노이즈를 적당히 조정하여 손실 평면(loss surface)상에 평평한 미니마(minima)에 도달 할 수 있는 양자화 훈련 방법 또한 제안한다. HLHLp (high-low-high-low-precision)로 명명된 훈련 방법은 양자화 정밀도를 훈련중에 높게-낮게-높게-낮게 바꾸면서 훈련한다. 훈련률

(learning rate)도 양자화 스텝 사이즈를 고려하여 유동적으로 바뀐다. 제안하는 훈련 방법은 일반적인 방법으로 훈련된 양자화 모델에 비해 상당히 좋은 성능을 보였다.

또한 선훈련된 선생 모델로 학생 모델을 훈련하는 지식 증류(knowledge distillation, KD) 기술을 이용하여 양자화의 성능을 높이는 방법을 제안한다. 특히 선생 모델을 선택하는 방법과 지식 증류의 하이퍼파라미터가 성능에 미치는 영향을 분석한다. 부동소수점 선생모델과 양자화 된 선생 모델을 사용하여 훈련 시킨 결과 선생 모델이 만들어내는 소프트맥스(softmax) 분포가 지식증류학습 결과에 크게 영향을 주는 것을 발견하였다. 소프트맥스 분포는 지식증류의 하이퍼파라미터들을 통해 조절될수 있으므로 지식증류 하이퍼파라미터들간의 연관관계 분석을 통해 높은 성능을 얻을 수 있었다. 또한 점진적으로 소프트 손실 함수 반영 비율을 훈련중에 줄여가는 점진적 소프트 손실 감소(gradual soft loss reducing)방법을 제안하였다.

뿐만 아니라 여러 양자화모델을 평균내어 높은 일반화 능력을 갖는 양자화 모델을 얻는 훈련 방법인 확률 양자화 가중치 평균(stochastic quantized weight averaging, SQWA) 훈련법을 제안한다. 제안하는 방법은 (1) 부동소수점 훈련, (2) 부동소수점 모델의 직접 양자화(direct quantization), (3) 재훈련(retraining)과정에서 진동 훈련율(cyclical learning rate)을 사용하여 훈련율이 진동내에서 가장 낮을 때 모델들을 저장, (4) 저장된 모델들을 평균, (5) 평균 된 모델을 낮은 훈련율로 재조정 하는 다중 단계 훈련법이다. 추가로 양자화 가중치 도메인에서 여러 양자화 모델들을 하나의 손실평면내에 동시에 나타낼 수 있는 심상(visualization) 방법을 제안한다. 제안하는 심상 방법을 통해 SQWA로 훈련된 양자화 모델은 손실평면의 가운데 부분에 있다는 것을 보였다.

주요어: 양자화된 깊은 인공 신경망, 일반화 능력, 고정소수점 최적화, HLHLp 훈련법, SQWA 훈련법, 지식 증류법

학번: 2013-23122