Joint Optimization of Quantization and Structured Sparsity for Compressed Deep
Neural Networks

by

Gaurav Srivastava

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved July 2018 by the
Graduate Supervisory Committee:

Jae-Sun Seo, Chair
Chaitali Chakrabarti
Visar Berisha

ARIZONA STATE UNIVERSITY

August 2018

ABSTRACT

Deep neural networks (DNN) have shown tremendous success in various cognitive tasks, such as image classification, speech recognition, etc. However, their usage on resource-constrained edge devices has been limited due to high computation and large memory requirement. To overcome these challenges, recent works have extensively investigated model compression techniques such as element-wise sparsity, structured sparsity and quantization. While most of these works have applied these compression techniques in isolation, there have been very few studies on application of quantization and structured sparsity together on a DNN model.

This thesis co-optimizes structured sparsity and quantization constraints on DNN models during training. Specifically, it obtains optimal setting of 2-bit weight and 2-bit activation coupled with 4X structured compression by performing combined exploration of quantization and structured compression settings. The optimal DNN model achieves 50X weight memory reduction compared to floating-point uncompressed DNN. This memory saving is significant since applying only structured sparsity constraints achieves 2X memory savings and only quantization constraints achieves 16X memory savings. The algorithm has been validated on both high and low capacity DNNs and on wide-sparse and deep-sparse DNN models. Experiments demonstrated that deep-sparse DNN outperforms shallow-dense DNN with varying level of memory savings depending on DNN precision and sparsity levels. This work further proposed a Pareto-optimal approach to systematically extract optimal DNN models from a huge set of sparse and dense DNN models. The resulting 11 optimal designs were further evaluated by considering overall DNN memory which includes activation memory and weight memory. It was found that there is only a small change in the memory footprint of the optimal designs corresponding to the low sparsity DNNs. However, activation memory cannot be ignored for high sparsity DNNs.

ACKNOWLEDGMENTS

I would first like to thank my thesis advisor Dr. Jae-Sun Seo of the ECEE at Arizona State University. He has always been the guiding light during my thesis work by giving me the research direction whenever I was stuck with mediocre results. I thank him for encouraging me to explore better solutions for the research problems.

I am grateful to Dr. Chaitali Chakrabarti and Dr. Visar Berisha for showing such great interest in my research and giving me insightful ideas which have not only shaped this research but have also imbibed in me valuable research ethos.

I would also like to thank my lab colleagues for being available whenever I needed help, irrespective of their busy schedule, and especially Shihui Yin who was always my reliable guidance whenever any problem seemed unsolvable.

I would like to thank my friends who have intentionally and unintentionally contributed towards this research. Last but not least, I am thankful to my family without whose unconditional support none of this would had been possible.

TABLE OF CONTENTS

	Page
LIST OF TABLES	v
LIST OF FIGURES	vi
CHAPTER	
1 INTRODUCTION	1
2 BACKGROUND AND PRIOR WORKS	5
2.1 Compression Techniques	7
2.1.1 Element-wise Sparsity Techniques	8
2.1.2 Structured Sparsity Techniques	9
2.2 Quantization/ Low-Precision Techniques	14
2.3 Combination of Compression and Quantization Techniques	15
2.4 Pareto-Optimal Front	15
3 JOINT-OPTIMIZATION OF QUANTIZATION AND STRUCTURED SPARSITY FOR DNN	16
3.1 Motivation	16
3.2 Proposed Algorithm	16
3.2.1 Training of DNN with Coarse-Grained Sparsity (DNN-CGS)	17
3.2.2 Training of DNN-CGS with Quantization	18
3.3 Experimental Setup	20
3.3.1 Setup Nomenclature	20
3.3.2 Implementation Details	22
3.3.3 Test Details	25
4 EXPERIMENTAL RESULTS	30
4.1 Model Selection	30
4.2 Comparison of CGS Block Sizes	30

CHAPTER	Page
4.3 Comparison of Wide-Sparse versus Thin-Dense DNN	32
4.4 Comparison of Deep-Sparse versus Shallow-Dense DNN.....	34
4.5 Comparison of Multiple DNNs with Quantization and CGS	38
4.6 Pareto-Optimal Analysis	43
4.7 Memory Distribution of Optimal Designs	45
5 CONCLUSION AND DISCUSSION	48
REFERENCES	50

LIST OF TABLES

Table	Page
4.1 Optimal Designs Obtained from Pareto-Optimal Front	46

LIST OF FIGURES

Figure		Page
2.1	Artificial Neural Network	6
2.2	Dense DNN.	9
2.3	Sparse DNN.	9
2.4	Illustration of CGS Mask for Convolution Weight Tensor of Size $32 \times 32 \times 3 \times 3$. (left-top) 2-D Weight Filter of Size 3×3 Represented by an Element in the Mask Grid. (right-top) CGS Mask of Size 32×32 With Block Size 4×4 and CGS Compression Ratio 4X. (bottom) Selected Non-Sparse Elements on the CGS Mask.	13
2.5	Illustration of CGS Mask for Fully-Connected Weight Matrix of Size 32×32 . (left) CGS Mask of Size 32×32 with CGS Block Size 4×4 and CGS Compression Ratio 4X. (right) Selected Non-Sparse Weights on CGS Mask.	13
3.1	Illustration of CGS. (left) Weight Matrix of Size 1024×1024 has 87.5% Weight Blocks Dropped. (right) Selected 16×16 Weight Blocks are Compressed, Stored with Minimal Index Memory.	18
3.2	2-bit Quantization Example.	19
3.3	2-bit Straight-Through-Estimator Gradient.	19
3.4	1-bit Quantization Example.	20
3.5	1-bit Straight-Through-Estimator Gradient.	20
3.6	n_{128r4} DNN. The DNN has 12 Convolution Layers. There are 128 Fea- ture Maps in the First Convolution Layer With Layers of Same Width Repeated 4 Times.	23

Figure	Page
3.7 n_{128r3} DNN. The DNN has 9 Convolution Layers. There are 128 Feature Maps in the First Convolution Layer With Layers of Same Width Repeated 3 Times.	24
3.8 n_{128r2} DNN. The DNN has 6 Convolution Layers. There are 128 Feature Maps in the First Convolution Layer With Layers of Same Width Repeated 2 Times.	24
3.9 Sequence of Convolution, Max-pooling, Batchnorm, ReLU Activation in the DNN Architecture. Max-Pooling is Present Only After Selected Convolution Layers.	26
3.10 Dense DNN.	27
3.11 Wide-Sparse DNN.	27
3.12 Deep-Sparse DNN.	28
4.1 Model Selection: Test-Accuracy versus Weight Memory Comparison of different depth DNN models. CGS ratio: 16X-8X-4X-2X-1X (Left to Right).	31
4.2 Test Accuracy versus Weight Memory for Different Convolution CGS Block Sizes. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	31
4.3 Test Accuracy versus Weight Memory Comparison between Wide-Sparse and Thin-Dense Floating-Point DNNs (W-FP:A-FP) on Low-Capacity Sparse DNN. CGS ratio: 16X-8X-4X-2X-1X (Left to Right).	32
4.4 Test Accuracy versus Weight Memory Comparison between Wide-Sparse and Thin-Dense Floating-Point DNNs (W-FP:A-FP) on High-Capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	33

4.5	Test Accuracy versus Weight Memory Comparison between Wide-Sparse and Thin-Dense 8-bit Weight and 8-bit Activation DNNs (W8b:A8b) on High-Capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	34
4.6	Test Accuracy versus Weight Memory Comparison between Wide-Sparse and Thin-Dense 4-bit Weight and 4-bit Activation DNNs (W4b:A4b) on High-Capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	34
4.7	Test Accuracy versus Weight Memory Comparison between Wide-Sparse and Thin-Dense 2-bit Weight and 2-bit Activation DNNs (W2b:A2b) on High-Capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	35
4.8	Test Accuracy versus Weight Memory Comparison between Wide-Sparse vs Thin-Dense 1-bit Weight and 1-bit Activation DNNs (W1b:A1b) on High-Capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	35
4.9	Test Accuracy versus Weight Memory comparison between Deep-Sparse and Shallow-Dense Floating-Point DNNs (W-FP:A-FP) on Low-Capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	36
4.10	Test Accuracy versus Weight Memory Comparison between Deep-Sparse and Shallow-Dense 8-bit Weight and 8-bit Activation DNNs (W8b:A8b) on Low-capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	36

4.11	Test Accuracy versus Weight Memory Comparison between Deep-Sparse and Shallow-Dense 4-bit Weight and 4-bit Activation DNNs (W4b:A4b) on Low-capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	37
4.12	Test Accuracy versus Weight Memory Comparison between Deep-Sparse and Shallow-Dense 2-bit Weight and 2-bit Activation DNNs (W2b:A2b) on Low-capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	37
4.13	Test Accuracy versus Weight Memory Comparison between Deep-Sparse and Shallow-Dense 1-bit Weight and 1-bit Activation DNNs (W1b:A1b) on Low-capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	38
4.14	Test Accuracy versus Weight Memory Comparison between Deep-Sparse and Shallow-Dense Floating-Point DNNs (W-FP:A-FP) on High-capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	38
4.15	Test Accuracy versus Weight Memory Comparison between Deep-Sparse and Shallow-Dense 8-bit Weight and 8-bit Activation DNNs (W8b:A8b) on High-Capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	39
4.16	Test Accuracy versus Weight Memory Comparison Between Deep-Sparse and Shallow-Dense 4-bit weight and 4-bit activation DNNs (W4b:A4b) on High-capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	39

4.17 Test Accuracy versus Weight Memory Comparison Between Deep-Sparse and Shallow-Dense 2-bit Weight and 2-bit Activation DNNs (W2b:a2b) on High-Capacity sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	40
4.18 Test Accuracy versus Weight Memory Comparison Between Deep-Sparse and Shallow-Dense 1-bit Weight and 1-bit Activation DNNs (W1b:A1b) on High-Capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	40
4.19 Test Accuracy versus Weight Memory Comparison for Different Precision and Structured Sparsity Settings for Low Capacity DNN models. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	41
4.20 Test Accuracy versus Weight Memory Comparison for Different Precision and Structured Sparsity Settings for Low Capacity DNN models, with extra W2b:A2b-2X DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	41
4.21 Test Accuracy versus Weight Memory Comparison for Different Precision and Sparsity Settings on High-Capacity DNN Model. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).	42
4.22 Test Accuracy versus Weight Memory Comparison for Different Precision and Sparsity Settings on High-Capacity DNN Model, with Extra W2b:A2b-2X DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right). . . .	42
4.23 Pareto Front of all the Dense and Sparse DNNs.	44
4.24 Optimal Designs (D1-D11) on Pareto Front.	44

Figure	Page
4.25 Test Accuracy versus Activation and Weight Memory for Optimal Designs on Pareto Front for Inference Batch Size of 1.....	45
4.26 Percentage Distribution of Convolution versus Fully-Connected Weight Memory for Optimal Designs on Pareto Front.	47
4.27 Percentage Distribution of Activation Memory versus Weight Memory for Optimal Designs on Pareto Front for Inference Batch Size of 1.....	47

Chapter 1

INTRODUCTION

Deep Neural Networks (DNN) have achieved tremendous success in various applications such as image classification (Simonyan and Zisserman (2014); He *et al.* (2016)), speech recognition (Chan *et al.* (2016)) etc. However, DNN models are computation and memory intensive. The state-of-the-art DNN models have millions of parameters and perform billions of operations per input. Thus they need large storage and high computational resources. On the other hand, embedded and edge devices are resource-constrained as they have limited memory and power. These challenges have limited the deployment of DNN on resource-constrained embedded and edge devices. Hence, compressed DNNs have become crucial to the design of hardware for Deep Learning applications.

There have been prior works which have tried to reduce the memory usage and computational requirement of DNNs while not losing on performance by exploring model compression techniques such as element-wise sparsity, structured sparsity and quantization. These techniques result in smaller trained DNN model. In some cases these techniques can help in generating simplified hardware design.

Element-wise sparsity techniques remove unimportant weights from DNN by making them zero. Optimal Brain Damage by LeCun *et al.* (1990) was an initial work in this area which used diagonal Hessian approximation to find low-saliency parameters and pruned them. Han *et al.* (2015b) pruned both neurons and weights in the network. Zhu and Gupta (2017) showed that large (wide) sparse networks perform better than small (thin) dense network. They demonstrated that the DNN incurs minimal performance degradation with 10X reduction in parameters. Even though

element-wise sparsity techniques reduce weight memory and computation, they come with an overhead cost of storing index memory of the non-sparse weights.

Structured Compression approach is based on removing a group of weights in the DNN model. This group of weights can be all the weights in a filter, channel or predefined blocks within the weight matrix/tensor. As structured sparsity techniques remove weights in a group, they do not need to store index of all the weights in a DNN model. Instead, they only need to store the index for the target non-sparse group of weights. Hence, the index memory required for structured sparsity techniques is usually small. Kadetotad *et al.* (2016) trained DNN by dropping blocks of weights before training the DNN and learned only the non-sparse blocks of weights in the DNN model. Wen *et al.* (2016) showed that generating scattered sparsity may not necessarily result in acceleration on hardware and applied group lasso Yuan and Lin (2006) to to make all the weights of a channel/filter/shapes/depth sparse.

Quantization techniques represent weights and activations with low-precision instead of floating point. This not only reduces the weight memory but also enables simplified hardware design. BinaryConnect by Courbariaux *et al.* (2015) reduced DNN weight precision to 1-bit. Binarized Neural Network by Hubara *et al.* (2016) binarized both the activations and weights, which made it possible to replace the MAC operations with bit-wise XNOR and accumulate operations. XNOR-Net by Rastegari *et al.* (2016) trained ImageNet dataset (Deng *et al.* (2009)) for 1-bit weight and 1-bit activation.

While these prior works have investigated quantization and compression techniques in isolation, there has been little work on systematically applying both these techniques together in a single framework. Deep compression by Han *et al.* (2015a) sequentially performed pruning, quantization and Huffman coding on the DNN to achieve compressed DNN. However, it suffered from large index memory as it used

element-wise sparsity. The work presented in this thesis applies quantization with structured compression, resulting in small index memory. Specifically, it aims at finding a precision and structured compression setting which saves DNN memory size without compromising on performance.

Below are the contribution of this thesis

- The proposed method co-optimizes structured sparsity and quantization constraints on DNN models during training.
- It comprehensively explored different precision and structured sparsity settings and compares performance of sparse low-precision DNN with that of dense uncompressed DNN.
- Experiments were performed on different capacity models, namely:
 - Wide-sparse and thin-dense DNN models,
 - Deep-sparse and shallow-dense DNN models

Experiments demonstrated that deep sparse DNN models outperform shallow dense DNN models. The magnitude of memory saving / performance gain diminishes with decreasing DNN precision and increasing DNN sparsity.

- Combined exploration of quantization and structured compression settings showed that 2-bit weight and 2-bit activation coupled with 4X structured compression achieves 50X weight memory reduction compared to floating-point uncompressed DNN. This memory saving is significant since applying only structured sparsity constraints achieves 2X memory savings and only quantization constraints achieves 16X memory savings.
- A Pareto-optimal approach was used to extract optimal models from a huge set of sparse and dense DNN models. These models achieved the highest test

accuracy for a given weight memory or lowest weight memory for a given test accuracy.

- Activation memory analysis was performed on 11 optimal designs extracted from Pareto-front. It was found that there is only a small change in the memory footprint of the optimal designs corresponding to the low sparsity networks (1X-4x compression). However, if the network has high sparsity (8X-32X compression) the effect of activation memory on overall DNN memory cannot be ignored.

The rest of the work is organized as follows: Section 2 provides the background and prior works. Section 3 gives details about the motivation of the problem, the proposed algorithm and the experimental setup used. Section 4 presents the experimental results. Finally, section 5 concludes my work with some guidelines and important results.

Chapter 2

BACKGROUND AND PRIOR WORKS

Artificial Neural Network (ANN) consists of stack of processing layers which learn representations of the data. Each layer consists of non linear processing units, called neurons, to generate a computational model that can extract the patterns in the data. Neurons in a layer are connected to neurons in the previous layer through trainable parameters, called weights and biases of the neural network model. Output of these neurons are called activations. The activation value of a layer is a function of neuron activation value of the previous layers and the weights which form the connection between the neurons in current layer and the previous layer. Fig. 2.1 shows an example neural network with two hidden layers. The circular nodes in the graph represent activations and the connection arrows between the activations represent the weights of the network. The weights of an ANN model are trained using a gradient descent method called backpropagation. The key of the learning process is that these layers are automatically learned and need not be designed.

Deep Neural Networks (DNN) are a type of Artificial Neural Network model what consisting of multiple layers of neurons. More the number of layers, deeper the DNN model. DNN models learn features at various levels of abstractions. For e.g. if a deep Convolutional Neural Network (CNN) is trained to classify images, the first layer would learn features like edges and blobs. The next layer would learn features about collections of edges, forming some shapes. The later layer would learn features about collections of shapes like wheel of car, ear on a face, etc., and the next layer would learn higher-level features such as faces etc. Here each layer generates a new feature obtained by composing and transforming features from previous layers, into a more

Figure 2.1: Artificial Neural Network

abstract feature. Multiple layers are much better at generalizing because they learn all the intermediate features between the raw data and the high-level classification. Among the latest high performing DNN architectures there has been a trend to make the network deeper, such as AlexNet (Krizhevsky *et al.* (2012)), VGGNet (Simonyan and Zisserman (2014)), GoogleNet (Szegedy *et al.* (2015)), ResNet (He *et al.* (2016)).

This approach to make neural networks deeper has led to remarkable success of DNN in various applications such as image classification (Simonyan and Zisserman (2014); He *et al.* (2016)), speech recognition (Chan *et al.* (2016)), language translation (Jean *et al.* (2014)), gene expression (Leung *et al.* (2014)). DNN technology is being used from self-driving cars to face recognition to search engines. This revolution in deep learning has been possible due to higher compute power by Graphical Processing Units (GPU) (Raina *et al.* (2009)) and availability of large datasets (Deng *et al.* (2009)).

By making the network deeper the test accuracy improves. However, it also makes the DNN computation intensive and memory intensive. These challenges have limited the implementation of DNN on resource-constrained embedded and edge devices. For example, AlexNet has 60 million parameters, VGGNet has 138 million with 19 layers

and GoogleNet has 4 million parameters with 152 layers. These large models provide high performance but are usually run on remote high resource machines. This is because running these models on-device is a challenge both in terms of memory needed and limited battery power. For e.g. Tesla performs all the high end computations on the servers and periodically copies the updated model parameters to the customers' cars. Another example is that DNN inference cannot be run on mobile phones for state-of-the-art DNNs because mobile phones have limited battery life. To sum up, performing on-device inference on embedded devices is a challenge.

To overcome the challenges to run DNN on embedded devices, there have been works which have tried to reduce the memory usage and computation requirement by exploring compression and quantization techniques. These techniques try to generate compact trained DNN models and in some cases help in generating simplified hardware design.

2.1 Compression Techniques

Compression techniques in DNN try to utilize lesser number of parameters by removing redundant parameters from DNN. DNNs are usually over-parameterized, this means that the DNN model has huge number of parameters which learn the representations from the training data. Due to over-population of the parameters in a DNN model, some parameters will have insignificant contribution towards test accuracy. The compression techniques tend to figure out these redundant parameters and remove them from the model. The question arises as to why there is over-parameterized model in the first place. DNN have over-parameterized models as they try to solve optimization problem for a non-convex loss function. To solve a non-convex loss function a relatively deeper DNN model is needed which will have large number of parameters. Moreover, having a deeper DNN model with large number of

parameters along with random initialization of these parameters prevents DNN from getting stuck in local minima of the loss function during training. The redundancy in DNN weights can be countered by using compression techniques. Below are two compression techniques.

- **Element-wise Sparsity:** This technique removes/prunes the insignificant parameters (parameters which have less effect on DNN output) from the DNN model by making the value of the parameters zero. The pruning of each DNN parameter is performed independently.
- **Structured Sparsity:** Structured sparsity technique prunes parameters in groups. Parameters of the DNN are divided in certain group based on the desired sparsity structures. Sparsity constraint is applied on the weights within the group. Structured sparsity is also referred to as structured compression.

2.1.1 *Element-wise Sparsity Techniques*

Element-wise pruning techniques try to find unimportant weights within DNN and make their values zero. These techniques have employed different constraints to figure out the insignificant parameters. Fig. 2.2 and Fig. 2.3 show example of dense and sparse DNN, respectively. During inference, the sparse weights are not stored in memory; however, the index of the non-sparse weights need to be stored. As the sparse weights do not contribute to Multiply And Accumulate (MAC) operations in DNN, they result in reducing the computations. LeCun *et al.* (1990) proposed Optimal Brain Damage which used diagonal Hessian approximation to find low-saliency parameters and pruned them. Some of the recent works have used magnitude based pruning, pruning the weights which are below a threshold level. These works have used L1 and L2 regularization to prune the weights. Han *et al.* (2015b) performed magnitude based

Figure 2.2: Dense DNN.

Figure 2.3: Sparse DNN.

pruning on weights without incurring any accuracy degradation. Zhu and Gupta (2017) showed that large (wide) sparse networks perform better than small (thin) dense network. The authors demonstrated that the DNN incurs minimal performance degradation with 10X reduction in parameters.

2.1.2 Structured Sparsity Techniques

Structured Compression approach applies sparsity constraints on a group of weights in a DNN model. This group of weights can consists of all the weights in a filter,

channel or predefined blocks within the weight matrix/tensor. This is in contrast to element-wise sparsity technique, which applies sparsity constraints on each parameter of the DNN model independently. For e.g. if the target sparsity structure is a filter, structured sparsity technique will find the unimportant filters in the DNN model and will remove/zero-out all the parameters within the unimportant filter. Structured sparsity imposes more stringent sparsity constraints on the network than element-wise sparsity. All these techniques need to store the index of the non-sparse target structures for inference. As the number of target non-sparse structures will be considerably less than the number of parameters in a DNN model, structured sparsity techniques need smaller index memory compared to element-wise sparsity techniques. Moreover, hardware can utilize the predefined sparsity structures to speedup computations. Kadetotad *et al.* (2016) proposed a Coarse-Grain sparsity technique to randomly drop blocks of weights before training. The value of the weights within the sparse blocks remain zero during trained. Yin *et al.* (2017) used Coarse-Grain sparsity for spike-based classification for speech recognition application. Wen *et al.* (2016) proposed Structured sparsity learning using group lasso regularizer to make all the weights of a channel/filter/shapes/depth sparse. Authors also experimentally showed that generating scattered sparsity may not necessarily result in acceleration on hardware. Li *et al.* (2016) pruned whole filters and the connecting feature maps which were insignificant within the DNN model. The work used absolute sum of weights of a filter to find saliency of the filter. Scalpel by Yu *et al.* (2017) proposed customized DNN pruning technique such that the sparse DNN structure would match the data-parallel hardware organization.

Coarse-Grain-Sparsity

Coarse-Grain sparsity (CGS) by Kadetotad *et al.* (2016) is a structured sparsity technique which prunes randomly-selected blocks of weights within DNN weight matrix/tensor prior to training and these pruned blocks remain sparse for training as well as for inference. CGS technique prunes only DNN weights and not the biases. The sparsity of the DNN model depends on CGS parameters - CGS block size (CGS size) and CGS compression ratio (CGS ratio). Prior to training, CGS Mask is generated for each layer of the DNN model, depending on CGS parameters. The definition of CGS block is different for convolution and fully-connected layers.

A sample CGS mask for convolution layers is illustrated in Fig. 2.4. The sample mask shown in Fig. 2.4(right-top) is for a convolution weight tensor of shape $32 \times 32 \times 3 \times 3$, with 32 input channels, 32 output channels and each 2D filter of shape 3×3 . A minor-element in the mask grid represents the mask value associated with a 2D filter in the weight tensor. Grey elements in the grid represent sparse weight filter and white elements in the grid represent non-sparse weight filter. Elements in a row represent a 2D filters coming from the same 3D filter. Elements in a column represent 2D filters at a particular location in each of the 3D filters. As there are 32 feature maps in the example 3D filter, there are 32 columns. Also, the 32 3D filters correspond to 32 rows in the mask grid. The CGS block size for convolution layer is defined as the number of 2D filters that will be grouped together in a block. In this example with block size of 4×4 , 16 2D filters or $16 \times 3 \times 3 = 144$ weights belong to one block. As the example CGS compression ratio is 4X, 16 blocks will remain non-sparse out of total 64 blocks. An equal-fanout constraint is imposed while training convolution layers with CGS. The constraint allows to only have equal number of non-sparse blocks for each channel. Hence, for each column there are two

blocks selected. This property can be utilized to design efficient parallel hardware. Also, weight tensor will have a uniform shape, which makes storage and decoding of weight tensor hardware-friendly. Similarly, if required equal-fanin constraint can be imposed. Fig. 2.4(bottom) represents the non-sparse 2D weight filters after applying CGS compression.

Sample CGS mask for fully-connected layers is illustrated in Fig. 2.5. The sample mask shown in Fig. 2.5(left) is for fully-connected weight matrix of shape 32x32, with 32 input neurons and 32 output neurons. A minor-element in the mask grid represents the mask value associated with a weight in the weight matrix. Similar to convolution mask grid, grey elements in the grid represent sparse weight and white grid elements represent non-sparse weight. Elements in a row of the CGS mask correspond to all the outgoing weights for a neuron in input layer of weight matrix. Elements in a column of CGS mask correspond to all the incoming weights to a neuron in the output layer of the weight matrix. As there are 32 input neurons in the example, there are 32 rows and 32 columns correspond to 32 output neurons. The example has block size of 4x4, which means 16 weights belong to one block. As compression ratio is 4X, 16 blocks will remain non-sparse out of total 64 blocks. An equal-fanout constraint is imposed while training convolution layers with CGS. The constraint allows for equal number of resources accessing a neuron on input side of weight matrix. Also, weight matrix will have a uniform shape, which makes storage and decoding of weight matrix hardware-friendly. This property can be changed to equal-fanin, depending on the available hardware resources. Fig. 2.5(right) represents the non-sparse weights in the weight matrix.

Figure 2.4: Illustration of CGS Mask for Convolution Weight Tensor of Size $32 \times 32 \times 3 \times 3$. (left-top) 2-D Weight Filter of Size 3×3 Represented by an Element in the Mask Grid. (right-top) CGS Mask of Size 32×32 With Block Size 4×4 and CGS Compression Ratio $4X$. (bottom) Selected Non-Sparse Elements on the CGS Mask.

Figure 2.5: Illustration of CGS Mask for Fully-Connected Weight Matrix of Size 32×32 . (left) CGS Mask of Size 32×32 with CGS Block Size 4×4 and CGS Compression Ratio $4X$. (right) Selected Non-Sparse Weights on CGS Mask.

2.2 Quantization/ Low-Precision Techniques

DNN Quantization techniques represent weights and activations of DNN with small number of memory bits (1-8 bits) instead of using 32/64 bit floating-point. Quantization not only reduces the weight memory but also results in simplified hardware design, converting expensive multiplications to bitwise logical and bitwise-shift operations. BinaryConnect by Courbariaux *et al.* (2015) used binarized value of real-valued weights for forward and backward phases of backpropagation. As weights are constrained to +1 and -1 during propagation, many of the MAC operations can be replaced with simple addition. Binarized Neural Network (BNN) by Hubara *et al.* (2016) binarized both the activations and weights, which made it possible to replace the MAC operations with bit-wise XNOR and accumulate operations. Authors of BNN have argued that the noise introduced by quantization of weights and activations acts as a regularizer which helps to generalize the DNN model. XNOR-Net by Rastegari *et al.* (2016) also trained DNN with binary weights and activations and demonstrated that their quantization algorithm performs considerably better than BNN, when trained for ImageNet classification task. Zhu *et al.* (2016) showed that DNN model trained with ternary quantized weights could outperform full-precision models. LightNN by Ding *et al.* (2017) quantized the weights to the power of 2 (1-bit to 8-bits) and used binary activations, which enabled the hardware architecture to replace the multiplication operations with shift and add operations. Moons *et al.* (2017) showed that low precision (2-bit 4-bit) can give performance comparable to than of floating-point DNN. Mishra *et al.* (2017) reduced the precision of the DNN model while making the filters in the network wider. The thesis showed that the performance of wide reduced-precision networks surpasses the performance of baseline full-precision network.

2.3 Combination of Compression and Quantization Techniques

Some of the prior works have optimized compression and quantization together on the same DNN model; however, they are significantly different from the proposed algorithm. CGS by Kadetotad *et al.* (2016) used structured sparsity during training and inference, and used low precision weights for classification. However, this thesis research applies CGS constraints and low-precision constraints together during training. Kadetotad *et al.* (2016) applied CGS only on fully-connected layers, while this thesis has extended the CGS technique for convolution layers. Deep compression by Han *et al.* (2015a) sequentially performed pruning, quantization and Huffman coding on the DNN to achieve compressed DNN. However, it suffered from large index memory as it used element-wise sparsity. 50% of the final DNN storage was index memory. This thesis applies quantization with structured compression, resulting in small index memory.

A part of the research from this thesis is presented in (Yin *et al.* (2018)), where CGS and quantization constraints were optimized together for convolution and fully-connected layers, which would be a preface for this thesis work.

2.4 Pareto-Optimal Front

Many of the prior works have used Pareto-optimal approach to solve multi-objective problems in various fields, ranging from machine learning to biological computation (Jin (2008)). Jin (2007) applied Pareto-optimality for machine learning problem. This thesis has used Pareto-optimal front to extract optimal designs which maximize test accuracy and minimize weight memory for dense and sparse DNNs.

Chapter 3

JOINT-OPTIMIZATION OF QUANTIZATION AND STRUCTURED SPARSITY FOR DNN

3.1 Motivation

Some of the prior works have applied structured sparsity and quantization in isolation or in a combination. However, few works have systematically applied and optimized both structured sparsity and quantization techniques in a single framework. Compared to element-wise sparsity, structured sparsity does not have an overhead of large weight index memory. While, structured sparsity in Wen *et al.* (2016) generates a regular sparsity pattern, it still uses 32/64-bit floating-point weights and activations. This regular sparsity pattern can be utilized for speedup on hardware. Quantization techniques reduce the memory by using low precision weights and activations. However, they still have redundant parameters in the model which can be pruned by using structured sparsity techniques. This thesis work has simultaneously applied both structured sparsity and low-precision constraints on DNN model during training.

3.2 Proposed Algorithm

The proposed training algorithm is an extension on BNN training algorithm with additional CGS structured sparsity constraints. The structured sparsity technique used in this thesis work is inspired from Coarse-Grain Sparsity (CGS) introduced in Kadetotad *et al.* (2016). Prior to training, blocks of weights are randomly dropped off depending on the CGS block size and CGS compression ratio settings. The weights

within the dropped blocks remain zero during training and inference. CGS mask as show in Fig. 2.4 and Fig. 2.5 are used to drop the sparse blocks of weights. The mask is also used to prevent the dropped blocks from learning during training. Training algorithm for non-sparse blocks of weights is similar to that of BNN training using back propagation.

3.2.1 *Training of DNN with Coarse-Grained Sparsity (DNN-CGS)*

For fully-connected layers, the weight matrix is divided into square blocks. For $x \times x$ CGS block size, each block contains x^2 weights. For convolution layers, $x \times x$ CGS block size, contains x^2 2D filters of size $l \times w$, where l is the length and w is the width of the 2D filter. Once the weights are segregated into blocks, large number of blocks are randomly dropped off with probability equal to the CGS ratio. These blocks remain zero during training and inference and hence do not contribute to the physical memory. Fig. 3.1 shows an example weight matrix of a fully-connected layer of size 1024×1024 , where each square represents a block of weights of size 16×16 . Grey squares represent blocks where eligible connections are present and white squares represent blocks with absence of connections. Fig. 3.1 (right) illustrates the blocks with active connections, compressed along row, after applying 8X CGS ratio.

Figure 3.1: Illustration of CGS. (left) Weight Matrix of Size 1024×1024 has 87.5% Weight Blocks Dropped. (right) Selected 16×16 Weight Blocks are Compressed, Stored with Minimal Index Memory.

3.2.2 Training of DNN-CGS with Quantization

The sparse weight matrix/tensor, generated after applying CGS, is trained using back propagation. There are three phases of backpropagation algorithm: forward phase, backward phase and weight update phase. During forward phase, quantized weights are generated from high-precision weights using 1-/2-/4-/8-bit quantization function. Once the activations are computed, they are quantized to the specified activation precision. Fig. 3.4 and Fig. 3.2 show activation output for 1-bit and 2-bit quantized activations, respectively. The quantized version of weights and activations is used for the forward pass. Loss is computed using target labels and the output from the forward path. During backward phase, gradients of cost function with respect to activations and weights are computed starting from output to input layer. Straight-through-estimator introduced by Hinton (2012) and later studied in (Bengio *et al.* (2013)) is used to estimate gradient with respect to quantized activations. Fig. 3.5 and Fig. 3.3 show the estimated gradient for 1-bit and 2-bit quantization cases using straight-through-estimator. During weight update phase, the high-precision weights

Figure 3.2: 2-bit Quantization Example.

Figure 3.3: 2-bit Straight-Through-Estimator Gradient.

are updated only for non-sparse blocks of weights, using Eq. 3.1.

$$(W_{ij})_{k+1} = (W_{ij})_k + \{(\Delta W_{ij})_k + m * (\Delta W_{ij})_{k-1}\} * lr * C_{ij} \quad (3.1)$$

where $(W_{ij})_k$ is weight in the weight matrix at k^{th} iteration, m is momentum, lr is learning rate, and C_{ij} is CGS connection coefficient between two consecutive DNN layers. $C_{ij} = 0$ for weights corresponding to the non-selected blocks and $C_{ij} = 1$ for weights corresponding to selected non-zero blocks.

Figure 3.4: 1-bit Quantization Example.

Figure 3.5: 1-bit Straight-Through-Estimator Gradient.

3.3 Experimental Setup

3.3.1 Setup Nomenclature

This thesis uses below mentioned nomenclature for referring to DNN architectures.

- **C** denotes convolution layer and is followed by number of feature maps in the convolution layer. For e.g. C128 denotes convolution layer with 128 channels.
- **P** denotes max pooling layer and is followed by pooling filter side length. For e.g. P2 denotes Max pooling layer with pooling filter of side length 2.

- **F** denoted fully-connected layer followed by the number of neurons in the fully-connected layer. For e.g. F1024 is a fully-connected layer with 1024 neurons.

Above notations will be used to name a DNN architecture. For e.g. DNN C128-P2-C256-P2-C512-P2-F1024-F10 has 3 convolution layer with 128, 256 and 512 feature maps, 2 fully-connected layers with 1024, and 10 neurons and 3 max-pooling layer with max-pool filters of side length 2. Apart from the long notation for DNNs, a short notation will be used in the document.

Nomenclature for a test setup uses the short notation of DNN along with parameters to describe the test settings. Below notation is used to refer to a particular test:

$$\mathbf{n}_{(\text{nf})\text{r}(\text{nrl})} \mathbf{W}_{\text{wp}} \mathbf{A}_{\text{ap}} \mathbf{C}_{(\text{ch}) \times (\text{cw})} \mathbf{f}_{(\text{fh}) \times (\text{fw})} \mathbf{d}, \text{ where}$$

- nf: number of feature maps in first convolution layer
- nrl: number of times convolution layers with same width is repeated
- wp: number of weight precision bits. If weights are floating-point, this tag is omitted
- ap: number of activation precision bits. If activations are floating-point, this tag is omitted
- ch: height of convolution CGS block, if CGS sparsity is applied on convolution layers. If CGS sparsity is not applied on convolution layers, this tag is omitted
- cw: width of convolution CGS block, if CGS sparsity is applied on convolution layers. If CGS sparsity is not applied on convolution layers, this tag is omitted

- fh: height of fully-connected CGS block, if CGS sparsity is applied on fully-connected layers. If CGS sparsity is not applied on fully-connected layers, this tag is omitted
- fw: width of fully-connected CGS block, if CGS sparsity is applied on fully-connected layers. If CGS sparsity is not applied on fully-connected layers, this tag is omitted
- d: this tag is added if dropout is applied to the DNN model. Default dropout was 20% on convolution layer and 50% on fully-connected layer. Any different dropout values will be reported in the DNN names

Below are few example DNNs and DNN settings:

1. $n_{128r2}d$ represents (C128-C128-P2-C256-C256-P2-C512-C512-P2-F1024-F1024-F10) network with floating-point weights and activations and no CGS sparsity.
2. $n_{128r2}W_{1b}A_{1b}c_{4\times4}f_{16\times16}d$ is the above-mentioned DNN with 1-bit weights and 1-bit activations. CGS sparsity is applied on the network. CGS block size on convolution layer is 4×4 and CGS block size on fully-connected layer is 16×16 .

Quantized networks will be referred with notation $W<\text{number of weight bits}>:A<\text{number of activation bits}>$. For e.g. 1-bit weight and 1-bit activation DNN will be referred as $W1b:A1b$. Floating-point DNN will be referred as $W-FP:A-FP$.

3.3.2 Implementation Details

Tests were performed using a CNN architecture inspired by VGG-19 architecture (Simonyan and Zisserman (2014)), which was used by Binarized Neural Network. Architecture of VGG-19, which was used for ImageNet Large Scale Visual Recognition

Figure 3.6: n_{128r4} DNN. The DNN has 12 Convolution Layers. There are 128 Feature Maps in the First Convolution Layer With Layers of Same Width Repeated 4 Times.

Competition (ILSVRC), has 16 convolution layers, 5 max-pooling layers and 3 fully-connected layers (C64-C64-P2-C128-C128-P2-C256-C256-C256-C256-P2-C512-C512-C512-C512-P2-F4096-F4096-F4096-F4096-Softmax). Instead, four variation of VGGNet were used for the experiments.

- n_{128r4} : It consists of 12 convolution, 3 max-pooling and 5 fully-connected layers (C128-C128-C128-C128-P2-C256-C256-C256-C256-P2-C512-C512-C512-C512-P2-F1024-F1024-F1024-F1024-F10). DNN n_{128r4} is illustrated in Fig. 3.6, showing the first convolution layer with 128 channels and has four repeated convolution layers with same width. This network is used to validate the algorithm for training CIFAR-10 dataset on very deep network.
- n_{128r3} : It consists of 9 convolution, 3 max-pooling and 4 fully connect layers (C128-C128-C128-P2-C256-C256-C256-P2-C512-C512-C512-P2-F1024-F1024-F1024-F10). DNN n_{128r3} is illustrated in Fig. 3.7, showing the first convolution layer with 128 channels and has three repeated convolution layers with same width. The DNN model provides a high capacity (deep DNN) model for the training of CIFAR10 dataset. Variations of this architecture will be used by varying number of feature maps in convolution layers. The first convolution layer can have 16, 24, 32, 48, 64 and 128 feature map, where the number of neurons in subsequent layers are scaled accordingly. For e.g. archi-

Figure 3.7: n_{128r3} DNN. The DNN has 9 Convolution Layers. There are 128 Feature Maps in the First Convolution Layer With Layers of Same Width Repeated 3 Times.

Figure 3.8: n_{128r2} DNN. The DNN has 6 Convolution Layers. There are 128 Feature Maps in the First Convolution Layer With Layers of Same Width Repeated 2 Times.

texture of n_{64r3} is (C64-C64-C64-P2-C128-C128-C128-P2-C256-C256-C256-P2-F512-F512-F512-F10).

- n_{128r2} : It consists of 6 convolution, 3 max-pooling and 3 fully connected layers (C128-C128-P2-C256-C256-P2-C512-C512-P2-F1024-F1024-F10). DNN n_{n128r2} is illustrated in Fig. 3.8, showing the first convolution layer with 128 channels and has two repeated convolution layers with same width. This architecture is selected to validate the experiments against low capacity (small DNN) model for training CIFAR-10 dataset. The architecture will be varied by changing the number of feature maps in the DNN. The first convolution layer can have 16, 24, 32, 48, 64 and 128 feature map, with the number of neurons in subsequent layers scaled accordingly.

- n_{128r1} : This DNN consists of 3 convolution, 3 max-pooling and 3 fully connected layers (C128-P2-C256-P2-C512-P2-F1024-F10). This DNN will be used as a dense model. Similar to 128r3 network, variations in this architecture will be used by varying the number of feature maps in the DNN. First convolution layer can have 16, 24, 32, 48, 64 and 128 feature map, with the number of neurons in subsequent layers scaled accordingly.

Above variation of VGGNet architecture were used as the DNN model, as they can be scaled conveniently. When tests will be performed to compare deep-sparse vs. shallow dense and wide-sparse vs. thin dense DNN, scaling the models proportionally and making them sparse would be easier.

The DNN architectures used for the experiments use convolution filters of size 3×3 . Batch-normalization layers and Rectified Linear Unit (ReLU) activation function are used after each convolution operation. Max-pooling is present only after selected convolution operations. Fig. 3.9 shows the sequence of convolution, max-pooling, batchnorm and activation function in the architecture. Mini-batch stochastic gradient descent is used to train the model with batch size of 50. Square hinge loss function and Adam optimizer are used to learn the weights. Dropout ratio of 20% is employed on convolution layers and 50% on fully-connected layers. The proposed algorithm is used to train CIFAR-10 datasets for image classification tasks. The image dataset has 10 classes. Deep learning framework Theano (Theano Development Team (2016)) and toolbox Lasagne (Dieleman *et al.* (2015)) is used for training and testing of the models.

3.3.3 Test Details

The primary objective of this research is to minimize weight memory while increasing test accuracy. **The weight memory mentioned in results include the**

Figure 3.9: Sequence of Convolution, Max-pooling, Batchnorm, ReLU Activation in the DNN Architecture. Max-Pooling is Present Only After Selected Convolution Layers.

memory needed for storing the weight index of CGS non-sparse blocks.

Tests were performed for different weight and activation precision, such as floating-point DNN (W-FP:A-FP), 8-bit DNN (W8b:A8b), 4-bit DNN (W4b:A4b), 2-bit DNN (W2b:A2b), 1-bit DNN (W1b:A1b). Tests were performed for different CGS block sizes. Tests were performed for Convolution CGS block sizes of 8×8 , 8×4 , 4×4 , 2×2 , 4×1 , 1×1 . CGS block size for fully-connected layers was fixed to 16×16 . Tests were performed for CGS compression ratio 32X, 16X, 8X, 4X, 2X, 1X, where 32X is the maximum compressed DNN and 1X is uncompressed DNN.

Tests were performed to compare performance of:

- **Wide-Sparse vs. Thin-Dense:** A Dense DNN will not have any sparsity constraints applied on the the learning of the DNN model. Fig. 3.10 shows a representation of dense DNN model. First a wider DNN model is generated from this dense DNN by increasing the number of feature maps in a convolution layer and increasing the number of neurons in fully-connected layers. To generate a wide-sparse DNN, weight connections are dropped/ zeroed-out from this wide

Figure 3.10: Dense DNN.

Figure 3.11: Wide-Sparse DNN.

DNN, to make the total number of weight connections same as in original dense DNN. Fig. 3.11 shows a representation of wise-sparse DNN. In this scenario, the original dense DNN is called thin-dense DNN when compared to wide-sparse DNN. Weight memory savings were compared between wide-sparse DNN and corresponding thin-dense DNN at iso-test accuracy condition.

Figure 3.12: Deep-Sparse DNN.

- **Deep-Sparse vs. Shallow-Dense:** Fig. 3.10 shows a representation of dense DNN model. First a deeper DNN model is generated from this dense DNN by increasing the number of convolution and fully-connected layers in the DNN. To generate a deep-sparse DNN, weight connections are dropped/ zeroed-out from this deep DNN, to make the total number of weight connections same as in original dense DNN. Fig. 3.12 shows a representation of deep-sparse DNN. In this scenario, the original dense DNN is called shallow-dense DNN when compared to deep-sparse DNN. Weight memory savings were compared between deep-sparse DNN and corresponding shallow-dense DNN at iso-test accuracy condition.

Further Pareto-optimal analysis was performed on all the trained DNN models, both dense and sparse DNNs. The Pareto-optimal front was used to extract the optimal designs. Further, tests were limited to the designs extracted from Pareto-front. Optimal designs were validated to still be optimal when taking activation memory into consideration along with weight memory. Activation memory versus weight mem-

ory usage was studied for the optimal designs. Convolution weight memory versus fully-connected weight memory distribution was studied for the optimal designs.

Chapter 4

EXPERIMENTAL RESULTS

4.1 Model Selection

Tests were performed to select the models on which CGS sparsity and quantization experiments would be performed. Experiments were performed on three DNN models of varying depths: n_{128r2} , n_{128r3} and n_{128r4} . Model n_{128r1} was used only as a dense DNN model, to compare with large sparse models. Fig. 4.1 shows the test accuracy versus weight memory comparison of the three models n_{128r2} , n_{128r3} and n_{128r4} . It is evident that when the number of layers in DNN is increased from 12 to 15, the performance of the DNN improves. Here, DNN n_{128r2} is a low capacity or small DNN model for CIFAR-10 image classification task. This model was used as a representative low capacity DNNs and tests performed on n_{128r2} were used to understand how proposed algorithm would perform on a low capacity DNN. Increasing DNN depth of model from 12 to 15 reduces the test accuracy. This shows that DNN n_{128r3} has enough capacity to represent features in the training data. DNN n_{128r4} seems to be overfit and was not used for any future tests. In absence of n_{128r4} achieving good performance, DNN n_{128r3} was used as a representative high capacity/ large DNNs and tests performed in n_{128r3} were used to understand how proposed algorithm would perform on a high capacity DNN.

4.2 Comparison of CGS Block Sizes

Test accuracy versus weight memory relationship study was performed for different CGS block sizes in the convolution layers. As seen in Fig 4.2, **although CGS block**

Figure 4.1: Model Selection: Test-Accuracy versus Weight Memory Comparison of different depth DNN models. CGS ratio: 16X-8X-4X-2X-1X (Left to Right).

Figure 4.2: Test Accuracy versus Weight Memory for Different Convolution CGS Block Sizes. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

size 2x2 shows the best performance for most of the compression ratios, the block sizes do not have a significant effect on the performance. The effect of CGS block size on test accuracy is evident at high sparsity (CGS ratio 16X). At high sparsity (at CGS ratio 16X), smaller CGS block size performs better than larger CGS block size. For all the CGS block size experiments, CGS block size is fixed to 16x16 for the fully-connected layers.

Figure 4.3: Test Accuracy versus Weight Memory Comparison between Wide-Sparse and Thin-Dense Floating-Point DNNs (W-FP:A-FP) on Low-Capacity Sparse DNN. CGS ratio: 16X-8X-4X-2X-1X (Left to Right).

4.3 Comparison of Wide-Sparse versus Thin-Dense DNN

As explained in Section 3.3.3, Wide-sparse DNN models are generated by increasing the number of feature maps of the DNN filters and at the same time increasing the CGS compression ratio. Test accuracy versus weight memory comparison was performed between wide-sparse and thin-dense DNN models for various sparsity and precision settings.

Wide-sparse versus thin-dense comparison was performed on low-capacity sparse DNN. As seen in Fig. 4.3, wide-sparse low-capacity floating-point DNN is similar to thin-dense floating-point DNN in performance. **Hence, it can be concluded that wide-sparse DNN is not competitive for low-capacity DNN models.** As low-capacity wide-sparse model is not competitive for floating point DNN, wide-sparse is expected to be worse for low-precision DNNs. No further comparison was performed on wide-sparse low-capacity models for low-precision settings.

Further, wide-sparse versus thin-dense comparison was performed on high-capacity sparse DNN. At first, comparison was made between floating-point wide-sparse DNN with floating-point thin-dense DNN. As seen in Fig. 4.4, **wide-sparse model for**

Figure 4.4: Test Accuracy versus Weight Memory Comparison between Wide-Sparse and Thin-Dense Floating-Point DNNs (W-FP:A-FP) on High-Capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

floating-point demonstrates significant memory savings of approximately 2X memory gain at 4X CGS ratio at iso-test accuracy settings compared to thin-dense model. However, these memory savings diminish at high sparsity (at CGS ratio 16X and greater).

As weight memory savings are observed for floating-point wide-sparse DNN, next step was to validate if similar memory savings are observed on low-precision wide-sparse DNN. Fig. 4.5, shows that wide-sparse 8-bit (W8b:A8b) high-capacity DNN achieves no memory savings for iso-test accuracy settings. At high sparsity levels (CGS ratio 16X) wide-sparse under-performs compared to thin-dense 8-bit (W8b:A8b) DNN. The tests were extended to other low-precision (1-bit, 2-bit, 4-bit) DNNs. The fact was corroborated for 4-bit DNN in Fig. 4.6, 2-bit DNN in Fig. 4.7 and 1-bit DNN in Fig. 4.8, that wide-sparse DNN is not competitive against thin-dense DNN.

It was observed that memory gains are achievable by using wide-sparse DNN compared to thin-dense DNN only by using high-capacity high-precision DNN model.

Figure 4.5: Test Accuracy versus Weight Memory Comparison between Wide-Sparse and Thin-Dense 8-bit Weight and 8-bit Activation DNNs (W8b:A8b) on High-Capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

Figure 4.6: Test Accuracy versus Weight Memory Comparison between Wide-Sparse and Thin-Dense 4-bit Weight and 4-bit Activation DNNs (W4b:A4b) on High-Capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

4.4 Comparison of Deep-Sparse versus Shallow-Dense DNN

Deep-sparse DNN models are generated by increasing the depth of dense DNN and applying CGS sparsity on it, while keeping the memory footprint similar to that of original dense DNN model. Section 3.3.3 explains deep-sparse DNN models. Weight memory savings of deep-sparse DNN models were compared to that of shallow-dense

Figure 4.7: Test Accuracy versus Weight Memory Comparison between Wide-Sparse and Thin-Dense 2-bit Weight and 2-bit Activation DNNs (W2b:A2b) on High-Capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

Figure 4.8: Test Accuracy versus Weight Memory Comparison between Wide-Sparse vs Thin-Dense 1-bit Weight and 1-bit Activation DNNs (W1b:A1b) on High-Capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

DNN models at iso-test accuracy settings. Similarly test accuracy of deep-sparse were compared to that of shallow-dense DNN at iso-weight memory settings.

At first, deep-sparse versus shallow-dense DNN comparison was performed on low-capacity sparse DNNs. As seen in Fig. 4.9, test accuracy of floating-point deep-sparse DNN (W-FP:A-FP) is approximately 6% higher than that of shallow-dense DNN even at 16X CGS ratio. Similar observations of deep-sparse outperforming shallow-dense DNN models were made for low-capacity low-precision DNN models.

Figure 4.9: Test Accuracy versus Weight Memory comparison between Deep-Sparse and Shallow-Dense Floating-Point DNNs (W-FP:A-FP) on Low-Capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

Figure 4.10: Test Accuracy versus Weight Memory Comparison between Deep-Sparse and Shallow-Dense 8-bit Weight and 8-bit Activation DNNs (W8b:A8b) on Low-capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

The phenomenon is demonstrated for 8-bit DNN in Fig. 4.10, 4-bit DNN in Fig. 4.11, 2-bit DNN in Fig. 4.12 and 1-bit DNN in Fig. 4.13.

Next experiments were performed to verify the advantages of deep-sparse DNNs on high-capacity models. As seen in Fig. 4.14, high-capacity deep-sparse floating point DNN (W-FP:A-FP) demonstrates performance gain compared to shallow-dense floating-point DNN. Even at 10X sparsity level test accuracy of deep sparse model is higher by 0.5% than shallow-dense DNN.

Figure 4.11: Test Accuracy versus Weight Memory Comparison between Deep-Sparse and Shallow-Dense 4-bit Weight and 4-bit Activation DNNs (W4b:A4b) on Low-capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

Figure 4.12: Test Accuracy versus Weight Memory Comparison between Deep-Sparse and Shallow-Dense 2-bit Weight and 2-bit Activation DNNs (W2b:A2b) on Low-capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

Similar experiments were performed by lowering the weight and activation precision. As seen in Fig. 4.15, for 8-bit DNN (W8b:A8b) there is significant performance gain on deep sparse models compared to shallow-dense models at low sparsity levels (CGS ratio 8X and lower). However, at high sparsity levels (CGS ratio 16X) the gain becomes insignificant. Decreasing the precision further, to 4-bits (W4b:A4b), as seen in Fig. 4.16, at low sparsity levels (CGS ratio 6X) accuracy of deep-sparse DNN is approximately 0.5% higher than the shallow-dense DNN model. 2-bit DNN

Figure 4.13: Test Accuracy versus Weight Memory Comparison between Deep-Sparse and Shallow-Dense 1-bit Weight and 1-bit Activation DNNs (W1b:A1b) on Low-capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

Figure 4.14: Test Accuracy versus Weight Memory Comparison between Deep-Sparse and Shallow-Dense Floating-Point DNNs (W-FP:A-FP) on High-capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

in Fig. 4.17 and 1-bit DNN in Fig. 4.18 show minor weight memory savings; however, this memory saving keeps reducing with decreasing weight/activation precision.

4.5 Comparison of Multiple DNNs with Quantization and CGS

To extract the optimal quantization and structured sparsity constraints DNN models were plotted and analyzed together for iso-test accuracy setting. Fig. 4.19 presents test accuracy versus weight memory relationship between low-capacity DNN models

Figure 4.15: Test Accuracy versus Weight Memory Comparison between Deep-Sparse and Shallow-Dense 8-bit Weight and 8-bit Activation DNNs (W8b:A8b) on High-Capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

Figure 4.16: Test Accuracy versus Weight Memory Comparison Between Deep-Sparse and Shallow-Dense 4-bit weight and 4-bit activation DNNs (W4b:A4b) on High-capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

with different CGS ratio and precision settings. Floating-point uncompressed model is the baseline DNN model on which low-precision and CGS constraints were applied. First only CGS sparsity constraints was applied on the baseline DNN. Approximately 2X weight memory compression is achieved using only CGS compression, with minimal test accuracy degradation ($<0.33\%$). Using only low-precision but uncompressed DNN resulted in approximately 16X weight memory savings with performance gain of 0.80%. 2-bit (W2b:A2b) DNN is the best low-precision DNN. Finally using both

Figure 4.17: Test Accuracy versus Weight Memory Comparison Between Deep-Sparse and Shallow-Dense 2-bit Weight and 2-bit Activation DNNs (W2b:a2b) on High-Capacity sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

Figure 4.18: Test Accuracy versus Weight Memory Comparison Between Deep-Sparse and Shallow-Dense 1-bit Weight and 1-bit Activation DNNs (W1b:A1b) on High-Capacity Sparse DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

quantization and CGS together, 32X weight savings is achieved with similar test accuracy as that of baseline floating-point uncompressed DNN. The optimal quantization CGS settings for baseline iso-accuracy condition is achieved by 2-bit quantized and 4X compressed DNN.

As 2-bit DNN achieves favorable compression for a range of CGS ratio settings, extra data point was collected on the network with 32X compression, as shown in Fig. 4.20.

Figure 4.19: Test Accuracy versus Weight Memory Comparison for Different Precision and Structured Sparsity Settings for Low Capacity DNN models. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

Figure 4.20: Test Accuracy versus Weight Memory Comparison for Different Precision and Structured Sparsity Settings for Low Capacity DNN models, with extra W2b:A2b-2X DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

Similar analysis was performed on high-capacity DNN models to extract optimal low-precision and structured sparsity constraints. Fig. 4.21 presents a relationship between DNN models with different CGS ratio and precision settings on high-capacity DNN models. Similar to the analysis on low-capacity DNN, applying only CGS constraints on floating-point uncompressed DNN achieves 2X compression with minor test accuracy degradation ($<0.16\%$). Using quantization only DNNs, 16X compression was achieved with performance gains (0.80% improvement in test accuracy) for

Figure 4.21: Test Accuracy versus Weight Memory Comparison for Different Precision and Sparsity Settings on High-Capacity DNN Model. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

Figure 4.22: Test Accuracy versus Weight Memory Comparison for Different Precision and Sparsity Settings on High-Capacity DNN Model, with Extra W2b:A2b-2X DNN. CGS Ratio: 16X-8X-4X-2X-1X (Left to Right).

2-bit DNN. Applying quantization and low-precision constraints together, memory saving of approximately 50X was achieved by 2-bit quantized and 4X compressed DNN, compared to floating-point uncompressed DNN.

As 2-bit DNN achieves favorable compression for a range of CGS ratio on high-capacity DNN, extra data point was collected on the network with 32X compression, as shown in Fig. 4.22.

4.6 Pareto-Optimal Analysis

The large number of DNN models trained over the experiments provided a trade-off between higher performance (high test accuracy) and low resource utilization (low weight memory). DNN test accuracy versus weight memory optimization problem is multi-objective. The desired DNN model should have low weight memory and should also have high test accuracy. In order to arrive at such a model, we employ Pareto-optimal analysis. Such a technique is a systematic approach to extract the optimum quantization and structured sparsity settings amongst all the trained DNN models.

Pareto-optimal analysis involved generating a Pareto-front and using the Pareto-front to extract the optimal design settings. As a first step, all the DNN designs were dumped on a test accuracy versus weight memory plot, as shown in Fig. 4.23. All the trained DNNs including the dense and sparse models are indicated by a blue '+' symbol. The Pareto-front shown by the red line, maximizes test accuracy and minimizes weight memory. All the designs close to Pareto-front are then chosen as optimal design settings. If the only objective was to achieve high test accuracy, then the Pareto-front saturates at approximately 93% test accuracy. Also Pareto-front drops vertically for weight memory at approximately 1 Mbits of weight memory, as none of the experimented DNN had weight memory less than this value.

11 optimal designs on/near-to Pareto-front were chosen and numbered D1-D11. As these designs are either highest accuracy for a given weight memory or lowest weight memory for a given test accuracy. Fig. 4.24 shows the designs which lie on the Pareto-front.

Total DNN memory is the sum of activation memory and weight memory. While only weight memory was considered for extracting the optimal designs from Pareto-front, next contribution of activation memory was analyzed for the optimal designs.

Figure 4.23: Pareto Front of all the Dense and Sparse DNNs.

Figure 4.24: Optimal Designs (D1-D11) on Pareto Front.

The analysis with respect to activation memory was performed using batch size of 1. Batch size of 1 during inference is a practical choice on embedded devices.

Fig. 4.25 shows test accuracy as a function of sum of activation memory and weight memory for inference batch size of 1. After adding activation memory to weight memory, there is only a small change in the memory footprint of the optimal

Figure 4.25: Test Accuracy versus Activation and Weight Memory for Optimal Designs on Pareto Front for Inference Batch Size of 1.

designs corresponding to the low sparsity networks (CGS ratio 1X-4x). However, if the network has high sparsity (CGS ratio 8X-32X) the effect of activation memory on overall DNN memory cannot be ignored.

Table. 4.1 provides the configuration details of the optimal designs extracted using Pareto-front.

4.7 Memory Distribution of Optimal Designs

DNN weight memory consists of both convolution weights and fully-connected weights. Usually convolution weights contribute toward heavy computations in DNN and fully-connected weights dominate the DNN memory footprint. Memory distribution analysis of convolution weights and fully-connected weights with respect to overall weight memory can give a better understanding of the hardware design constraints for the optimal DNN models. Fig. 4.26 gives the distribution of convolution versus fully-connected weights for the optimal designs. The ratio of convolution weights versus fully-connected weights is slightly lower for high-capacity DNN models. When the

Table 4.1: Optimal Designs Obtained from Pareto-Optimal Front

Optimal Designs	Network	Activation Precision	Weight Precision	CGS ratio
D1	n _{128r3}	4b	4b	1X
D2	n _{128r3}	4b	4b	2X
D3	n _{128r3}	2b	2b	1X
D4	n _{128r3}	2b	2b	2X
D5	n _{128r3}	2b	2b	4X
D6	n _{128r3}	2b	2b	8X
D7	n _{128r3}	2b	2b	16X
D8	n _{128r3}	2b	2b	32X
D9	n _{128r2}	2b	2b	16X
D10	n _{128r2}	2b	2b	32X
D11	n _{32r3}	2b	2b	1X

optimal design is low-capacity DNN, the contribution from fully-connected weights dominate the weight memory distribution.

Fig. 4.27 shows the distribution of activation memory and weight memory for the optimal designs for inference batch size of 1. The ratio of activation memory to that overall memory is insignificant for low sparsity DNNs (CGS ratio 1X-4X). However, at high sparsity (CGS ratio 8X-16X) the activation memory becomes a significant component of the overall DNN memory as weights get pruned while all the activations remain non-sparse.

Figure 4.26: Percentage Distribution of Convolution versus Fully-Connected Weight Memory for Optimal Designs on Pareto Front.

Figure 4.27: Percentage Distribution of Activation Memory versus Weight Memory for Optimal Designs on Pareto Front for Inference Batch Size of 1.

CONCLUSION AND DISCUSSION

DNNs have drastically improved the performance of various machine learning applications. However, their usage on resource-constrained edge devices has been limited due to high computation and large memory requirement. Hence, compressed DNNs have become crucial for the deployment of DNNs on embedded and edge-devices. Many prior works have tried to overcome these challenges by exploring model compression techniques such as element-wise sparsity, structured sparsity and quantization. While these prior works have investigated quantization and compression techniques in isolation, there has been little work on systematically applying both these techniques together in a single framework. This thesis work co-optimizes structured sparsity and quantization constraints on DNN models during training. Various precision and structured sparsity settings were comprehensively explored and performance of sparse low-precision DNN was compared with that of dense uncompressed DNN.

There are few conclusions/guidelines based on the experiments:

- Experiments using different CGS block size demonstrated that CGS block size do not have significant effect on DNN performance on low sparsity DNNs. However, high sparsity DNNs with smaller block size performs better than larger block size DNNs.
- Wide-sparse DNNs could achieve weight memory gains only for high-capacity DNN using high-precision. Hence, it is advised to use wide-sparse models only with high-capacity high-precision DNNs.

- Deep-sparse DNN outperform shallow dense DNN with comparable weight memory savings both on low-capacity and high-capacity sparse DNNs. However, the weight memory savings diminish as the precision of DNN is reduced. Summing up, it is preferred to use deep-sparse DNNs than shallow-dense DNNs.
- Optimal setting of 2-bit weight and 2-bit activation coupled with 4X structured compression achieved 50X weight memory reduction compared to floating-point uncompressed DNNs. This memory saving is significant since applying only structured sparsity constraints achieved 2X weight memory savings and only quantization constraints achieved 16X weight memory savings.
- A Pareto-optimal approach was employed to systematically extract optimal DNN models from a huge set of sparse and dense DNN models. 11 optimal designs were extracted from the Pareto-front. These designs either had the highest accuracy for a given weight memory or lowest weight memory for a given test accuracy.
- The optimal designs were further evaluated by considering the total DNN memory which includes activation memory and weight memory. It was found that there is only a small change in the memory footprint of the optimal designs corresponding to the low sparsity networks (CGS ratio 1X-4x). However, if the network has high sparsity (CGS ratio 8X-32X) the effect of activation memory on overall DNN memory cannot be ignored.

Future work would apply the algorithm on another DNN architecture to ensure the algorithm is architecture-robust. The algorithm would be validated on a larger dataset such as ImageNet.

REFERENCES

- Bengio, Y., N. Léonard and A. Courville, “Estimating or propagating gradients through stochastic neurons for conditional computation”, arXiv preprint arXiv:1308.3432 (2013).
- Chan, W., N. Jaitly, Q. Le and O. Vinyals, “Listen, attend and spell: A neural network for large vocabulary conversational speech recognition”, in “Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on”, pp. 4960–4964 (IEEE, 2016).
- Courbariaux, M., Y. Bengio and J.-P. David, “Binaryconnect: Training deep neural networks with binary weights during propagations”, in “Advances in neural information processing systems”, pp. 3123–3131 (2015).
- Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database”, in “CVPR09”, (2009).
- Dieleman, S., J. Schlter, C. Raffel, E. Olson, S. K. Snderby, D. Nouri *et al.*, “Lasagne: First release.”, URL <http://dx.doi.org/10.5281/zenodo.27878> (2015).
- Ding, R., Z. Liu, R. Shi, D. Marculescu and R. Blanton, “Lightnn: Filling the gap between conventional deep neural networks and binarized networks”, in “Proceedings of the on Great Lakes Symposium on VLSI 2017”, pp. 35–40 (ACM, 2017).
- Han, S., H. Mao and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding”, arXiv preprint arXiv:1510.00149 (2015a).
- Han, S., J. Pool, J. Tran and W. Dally, “Learning both weights and connections for efficient neural network”, in “Advances in neural information processing systems”, pp. 1135–1143 (2015b).
- He, K., X. Zhang, S. Ren and J. Sun, “Deep residual learning for image recognition”, in “Proceedings of the IEEE conference on computer vision and pattern recognition”, pp. 770–778 (2016).
- Hinton, G., “Neural networks for machine learning”, (2012).
- Hubara, I., M. Courbariaux, D. Soudry, R. El-Yaniv and Y. Bengio, “Binarized neural networks”, in “Advances in neural information processing systems”, pp. 4107–4115 (2016).
- Jean, S., K. Cho, R. Memisevic and Y. Bengio, “On using very large target vocabulary for neural machine translation”, arXiv preprint arXiv:1412.2007 (2014).
- Jin, Y., “Pareto-based multi-objective machine learning”, in “Hybrid Intelligent Systems, 2007. HIS 2007. 7th International Conference on”, pp. 2–2 (IEEE, 2007).

- Jin, Y., “Pareto-optimality is everywhere: From engineering design, machine learning, to biological systems”, in “Genetic and Evolving Systems, 2008. GEFS 2008. 3rd International Workshop on”, pp. 1–1 (IEEE, 2008).
- Kadetotad, D., S. Arunachalam, C. Chakrabarti and J.-s. Seo, “Efficient memory compression in deep neural networks using coarse-grain sparsification for speech applications”, in “Proceedings of the 35th International Conference on Computer-Aided Design”, p. 78 (ACM, 2016).
- Krizhevsky, A., I. Sutskever and G. E. Hinton, “Imagenet classification with deep convolutional neural networks”, in “Advances in neural information processing systems”, pp. 1097–1105 (2012).
- LeCun, Y., J. S. Denker and S. A. Solla, “Optimal brain damage”, in “Advances in neural information processing systems”, pp. 598–605 (1990).
- Leung, M. K., H. Y. Xiong, L. J. Lee and B. J. Frey, “Deep learning of the tissue-regulated splicing code”, *Bioinformatics* **30**, 12, i121–i129 (2014).
- Li, H., A. Kadav, I. Durdanovic, H. Samet and H. P. Graf, “Pruning filters for efficient convnets”, arXiv preprint arXiv:1608.08710 (2016).
- Mishra, A., E. Nurvitadhi, J. J. Cook and D. Marr, “Wrpn: Wide reduced-precision networks”, arXiv preprint arXiv:1709.01134 (2017).
- Moons, B., K. Goetschalckx, N. Van Berckelaer and M. Verhelst, “Minimum energy quantized neural networks”, in “Signals, Systems, and Computers, 2017 51st Asilomar Conference on”, pp. 1921–1925 (IEEE, 2017).
- Raina, R., A. Madhavan and A. Y. Ng, “Large-scale deep unsupervised learning using graphics processors”, in “Proceedings of the 26th annual international conference on machine learning”, pp. 873–880 (ACM, 2009).
- Rastegari, M., V. Ordonez, J. Redmon and A. Farhadi, “Xnor-net: Imagenet classification using binary convolutional neural networks”, in “European Conference on Computer Vision”, pp. 525–542 (Springer, 2016).
- Simonyan, K. and A. Zisserman, “Very deep convolutional networks for large-scale image recognition”, arXiv preprint arXiv:1409.1556 (2014).
- Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, “Going deeper with convolutions”, in “Proceedings of the IEEE conference on computer vision and pattern recognition”, pp. 1–9 (2015).
- Theano Development Team, “Theano: A Python framework for fast computation of mathematical expressions”, arXiv e-prints **abs/1605.02688**, URL <http://arxiv.org/abs/1605.02688> (2016).
- Wen, W., C. Wu, Y. Wang, Y. Chen and H. Li, “Learning structured sparsity in deep neural networks”, in “Advances in Neural Information Processing Systems”, pp. 2074–2082 (2016).

- Yin, S., D. Kadetotad, B. Yan, C. Song, Y. Chen, C. Chakrabarti and J.-s. Seo, “Low-power neuromorphic speech recognition engine with coarse-grain sparsity”, in “Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific”, pp. 111–114 (IEEE, 2017).
- Yin, S., G. Srivastava, S. K. Venkataramanaiah, C. Chakrabarti, V. Berisha and J.-s. Seo, “Minimizing area and energy of deep learning hardware design using collective low precision and structured compression”, arXiv preprint arXiv:1804.07370 (2018).
- Yu, J., A. Lukefahr, D. Palframan, G. Dasika, R. Das and S. Mahlke, “Scalpel: Customizing dnn pruning to the underlying hardware parallelism”, in “ACM SIGARCH Computer Architecture News”, vol. 45, pp. 548–560 (ACM, 2017).
- Yuan, M. and Y. Lin, “Model selection and estimation in regression with grouped variables”, *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **68**, 1, 49–67 (2006).
- Zhu, C., S. Han, H. Mao and W. J. Dally, “Trained ternary quantization”, arXiv preprint arXiv:1612.01064 (2016).
- Zhu, M. and S. Gupta, “To prune, or not to prune: exploring the efficacy of pruning for model compression”, arXiv preprint arXiv:1710.01878 (2017).