Structured Compression by Weight Encryption for Unstructured Pruning and Quantization

Se Jung Kwon[1*] Dongsoo Lee[1*] Byeongwook Kim[1]
Parichay Kapoor[1] Baeseong Park[1] Gu-Yeon Wei[1,2]
[1]Samsung Research, Republic of Korea [2]Harvard University, MA
{sejung0.kwon, dongsoo3.lee, byeonguk.kim, pk.kapoor, bpbs.park, gy.wei}@samsung.com

Abstract

Model compression techniques, such as pruning and quantization, are becoming increasingly important to reduce the memory footprints and the amount of computations. Despite model size reduction, achieving performance enhancement on devices is, however, still challenging mainly due to the irregular representations of sparse matrix formats. This paper proposes a new weight representation scheme for Sparse Quantized Neural Networks, specifically achieved by fine-grained and unstructured pruning method. The representation is encrypted in a structured regular format, which can be efficiently decoded through XOR-gate network during inference in a parallel manner. We demonstrate various deep learning models that can be compressed and represented by our proposed format with fixed and high compression ratio. For example, for fully-connected layers of AlexNet on ImageNet dataset, we can represent the sparse weights by only 0.28 bits/weight for 1-bit quantization and 91% pruning rate with a fixed decoding rate and full memory bandwidth usage. Decoding through XOR-gate network can be performed without any model accuracy degradation with additional patch data associated with small overhead.

1. Introduction

Deep neural networks (DNNs) are evolving to solve increasingly complex and varied tasks with dramatically growing data size [7]. As a result, the growth rate of model sizes for recent DNNs leads to slower response times and higher power consumption during inference [9]. To mitigate such concerns, model compression techniques have been introduced to significantly reduce model size of DNNs while maintaining reasonable model accuracy.

It is well known that DNNs are designed to be over-parameterized in order to ease local minima exploration [5, 6]. Thus, various model compression techniques have been proposed for high-performance and/or low-power inference. For example, pruning techniques remove redundant weights (to zero) without compromising accuracy [18], in order to achieve memory and computation reduction on devices [11, 27, 37, 20]. As another model compression technique, non-zero weights can be quantized to fewer bits with comparable model accuracy of full-precision parameters, as discussed in [4, 28, 14, 32].

To achieve even higher compression ratios, pruning and quantization can be combined to form Sparse Quantized Neural Networks (SQNNs). Intuitively, quantization can leverage parameter pruning since pruning reduces the number of weights to be quantized and quantization loss decreases accordingly [21]. Deep compression [10], ternary weight networks (TWN) [23], trained ternary quantization (TTQ) [36], and viterbi-based compression [1, 19] represent recent efforts to synergistically combine pruning and quantization.

To benefit from sparsity, it is important to (1) represent pruned models in a format with small memory footprint and (2) implement fast computations with sparse matrices as input operands. Even if reduced SQNNs can be generated with a high pruning rate, it is challenging to gain performance enhancement without an inherently parallel sparse-matrix decoding process during inference. To illustrate such a challenge, Figure 1 presents DRAM bandwidth, the number of transactions, and execution time of a matrix multiplication using one random (2048×2048) sparse matrix (following CSR format) and a random (2048×64) dense matrix on NVIDIA Tesla V100 (supported by CUDA 9.1). Because of unbalanced workloads (note that pruning each weight is a somewhat independent and random operation) and additional data for index, sparse matrix computations using CSR format do not offer performance gain as much as sparsity. Moreover, if pruning rate is not high enough, sparse matrix operations can be even slower than dense matrix operations.

As such, structured and blocked-based pruning tech-

*Equal contribution

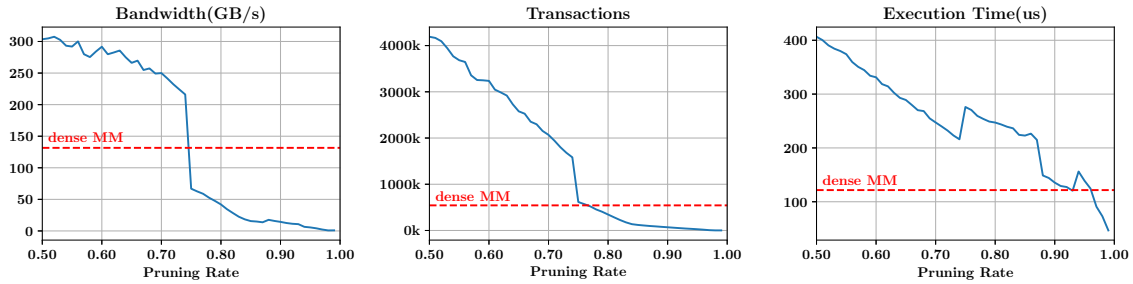


Figure 1: DRAM bandwidth, the number of transactions, and execution time of a matrix multiplication using one random (2048×2048) sparse matrix (following CSR format) and a random (2048×64) dense matrix using NVIDIA Tesla V100. CUDA 9.1 is used as a main computation library and analysis is supported by NVIDIA Profiler. Performance of a multiplication using two dense matrices (denoted as dense MM) without pruning is also provided as a baseline.

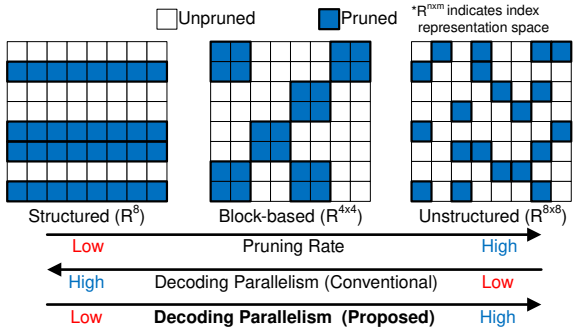


Figure 2: Several types of pruning granularity. In the conventional sparse formats, as a sparse matrix becomes more structured to gain parallelism in decoding, pruning rate becomes lower in general.

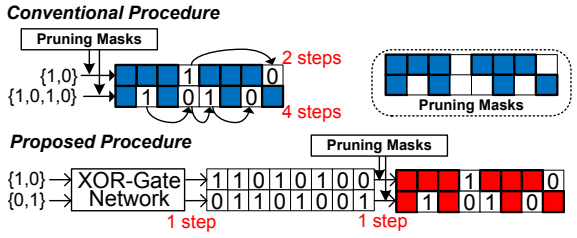


Figure 3: Comparison between conventional and proposed sparse matrix decoding procedures given a pruning mask. In the conventional approach, the number of decoding steps for each row can be different (i.e., degraded row-wise parallelism). On the contrary, the proposed approach decodes each row at one step by using XOR-gate network.

niques [24, 2, 34, 13, 33] for DNNs have been proposed to accelerate decoding of sparse matrices using reduced indexing space, as Figure 2 shows. However, coarse-grained pruning associated with reduced indexing space exhibits relatively lower pruning rates compared to unstructured pruning [25], which masks weights with fine-grained granularity. In conventional sparse matrix formats representing unstructured pruning (i.e., random weights can be pruned), decoding time can vastly differ if decoding processes are conducted in different blocks simultaneously, as shown in the conventional approach of Figure 3.

To enable inherently parallel computations using sparse matrices, this paper proposes a new sparse format. Our main objective is to remove all pruned weights such that the resulting compression ratio tracks the pruning rate, while maintaining a regular format. Interestingly, in VLSI testing, proposals for test-data compression have been developed from similar observations, i.e., there are lots of *don't care* bits (= pruned weights in the case of model compression) and the locations of such *don't care* bits seem to be random [30] (the locations of unstructurally pruned weights also seem to be random). We adopt XOR-gate network, previously used for test-data compression, to decode the compressed bits in a fixed rate during inference, as shown in Figure 3. XOR-gate network is small enough such that we can embed multiple XOR-gate networks to fully utilize memory bandwidth and decode many sparse blocks concurrently. Correspondingly, we propose an algorithm to find encrypted and compressed data to be fed into XOR-gate network as inputs.

2. Related Works and Comparison

In this section, we introduce two previous approaches to represent sparse matrices. Table 1 describes CSR format, Viterbi-based index format, and our proposed method.

Compressed Sparse Row (CSR): Deep compression [10] utilizes the Compressed Sparse Row (CSR) format to reduce memory footprint on devices. Unfortunately, CSR

	CSR Format	Viterbi-based Compr.	Proposed
Encryption	No	Yes	Yes
Load Balance	Uneven	Even	Even
Decoding Rate	Variable	Fixed	Fixed
Parallelism Limited by	Uneven Sparsity	Number of Decoders	Number of Decoders
Memory Access Pattern	Irregular (Gather-Scatter)	Regular	Regular
Compressed Memory Bandwidth	Depends on on-chip Buffer Structure	1 bit/decoder	Multi-bits/decoder
HW Resource for a Decoder	Large Buffer to improve load balance	XOR gates and Flip-Flops	XOR gates only

Table 1: Comparisons of CSR, Viterbi, and our proposed representation. For all of these representation schemes, compression ratio is upper-bounded by sparsity.

formats (including blocked CSR) present irregular data structures not readily supported by highly parallel computing systems such as CPUs and GPUs [19]. Due to uneven sparsity among rows, computation time of algorithms based on CSR is limited by the least sparse row [35], as illustrated in Figure 3. Although [9] suggested hardware supports via a large buffer to improve load balancing, performance is still determined by the lowest pruning rate of a particular row. In contrast, our scheme provides a perfectly structured format of weights after compression such that high parallelism is maintained.

Viterbi Approaches: Viterbi-based compression [19, 1] attempts to compress pruning-index data and quantized weights with a fixed compression ratio using *don't care* bits, similar to our approach. Quantized weights can be compressed by using the Viterbi algorithm to obtain a sequence of inputs to be fed into Viterbi encoders (one bit per cycle). Because only one bit is accepted for each Viterbi encoder, only an integer number (=number of Viterbi encoder outputs) is permitted as a compression ratio, while our proposed scheme allows any rational numbers.

Because only one bit is used as inputs for Viterbi encoders, Viterbi-based approaches require large hardware resources. For example, if a memory allows 1024 bits per cycle of bandwidth, then 1024 Viterbi encoders are required, where each Viterbi encoder entails multiple Flip-Flops to support sequence detection. On the other hand, our proposed scheme is resource-efficient to support large memory bandwidth because Flip-Flops are unnecessary.

3. Proposed Weight Representation for Structured Compression

Test-data compression usually generates random numbers as outputs using the input data as seed values. The outputs (test data containing *don't care* bits) can be compressed successfully if such outputs can be generated by the random number generator using at least one particular input seed data (which is the compressed test data). It is well known that memory reduction can be as high as the portion of *don't care* bits [3, 30] if randomness is good enough. Test data compression and SQNNs with fine-grained pruning share the following properties: 1) Parameter pruning induces *don't care* values as much as pruning rates and 2) If a weight is unpruned, then each quantization bit is assigned to 0 or 1 with equal probability [1]. In this section, we propose a new weight representation method exploiting such shared properties while fitting high memory bandwidth requirements and lossless compression to maintain accuracy.

3.1. Encryption and Decryption

We use an XOR-gate network as a random number generator due to its simple design and strong compression capability (such a generator is not desirable for test-data compression because it requires too many input bits). Suppose that a real-number weight matrix $\boldsymbol{W}$ is quantized to be binary matrices $\boldsymbol{W}_i^q$ ($1 \leq i \leq n_q$) with n_q as the number of bits for quantization. As the first step of our encryption algorithm, we reshape each binary matrix $\boldsymbol{W}_i^q$ to be a 1D vector, which is then evenly divided into smaller vector sequences of n_{out} size. Then, each of the evenly divided vectors, $\boldsymbol{w}^q$, including *don't care* bits is encrypted to be a small vector $\boldsymbol{w}^c$ (of n_{in} size) without any *don't care* bits. Through the XOR-gate network, each encrypted vector $\boldsymbol{w}^c$ is decrypted to be original bits consisting of correct *care* bits and randomly filled *don't care* bits with respect to $\boldsymbol{w}^q$. Figure 4 illustrates encryption and decryption procedure examples using a weight matrix. A 4D tensor (e.g. conv layers) can also be encrypted through the same procedures after flattening.

The structure of XOR-gate network is fixed during the entire process and, as depicted in Figure 5, can be described as a binary matrix $\mathcal{M}^{\oplus} \in \{0,1\}^{n_{out} \times n_{in}}$ over **Galois Field** with two elements, $GF(2)$, using the connectivity information between the input vector $\boldsymbol{w}^c \in \{0,1\}^{n_{in}}$ (compressed and encrypted weights) and $\boldsymbol{w}^q \in \{0, x, 1\}^{n_{out}}$. Note that $\mathcal{M}$

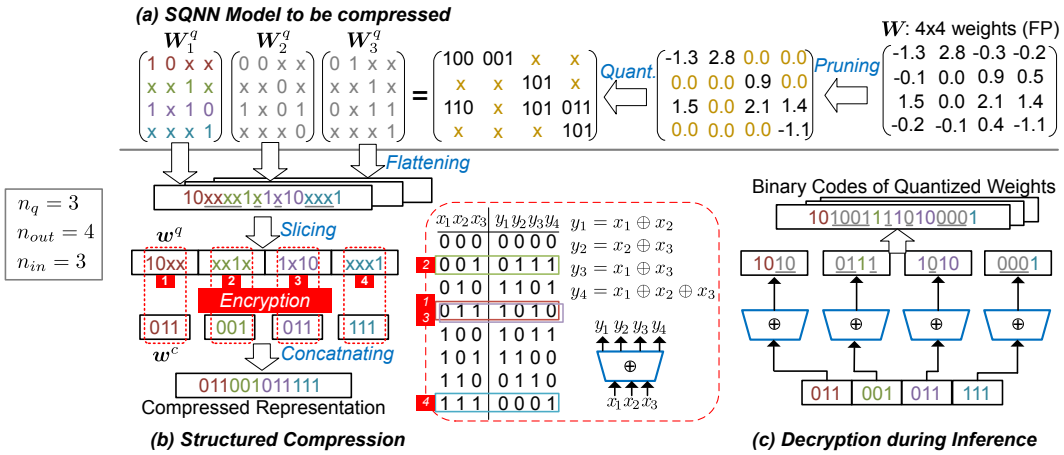


Figure 4: An example illustrating the overall procedures of our proposed method using (4×4) full-precision weights. (a) We assume that a (4×4) weight matrix ($\boldsymbol{W}$) is pruned and then quantized into 3 bits. (b) Quantized weights ($\boldsymbol{W}_1^q$, $\boldsymbol{W}_2^q$, and $\boldsymbol{W}_3^q$) are encrypted by using XOR-gate network which can be formulated as 4 XOR-based equations with 3 inputs (x_1, x_2, and x_3). We can assign a 3-bit encrypted vector to each sliced 4-bit vector through a look-up table constructed by all possible XOR-gate network input/output pairs. (c) During inference on devices, quantized weights are produced through decryption (that can be best implemented by ASIC or FPGA) from compactly encrypted weights. Note that compared with $\boldsymbol{W}_1^q$, $\boldsymbol{W}_2^q$, and $\boldsymbol{W}_3^q$, decryption yields new quntized weights in which *care* bits are matched and *don't care* bits are randomly filled.

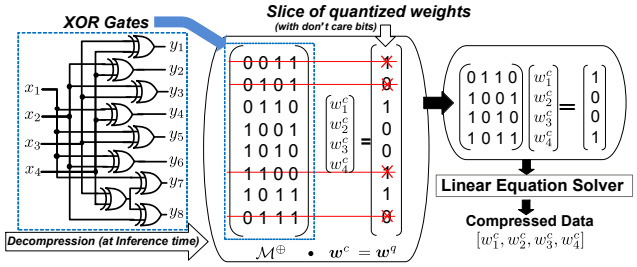


Figure 5: Given a fixed matrix $\mathcal{M}$ (n_{in}=4 and n_{out}=8) representing the XOR-gate network, encrypting $\boldsymbol{w}^q$ is equivalent to solving $\mathcal{M}^{\oplus}\boldsymbol{w}^c$=$\boldsymbol{w}^q$, which can be simplified after removing equations associated with *don't care* bits.

is pre-determined and designed in a way that each element is randomly assigned to 0 or 1 with the same probability.

We intend to generate a random output vector using a seed vector $\boldsymbol{w}^c$ while targeting as many *care* bits of $\boldsymbol{w}^q$ as possible. In order to increase the number of successfully matched *care* bits, XOR-gate network should be able to generate various random outputs. In other words, when the sizes of a seed vector and an output vector are given as n_{in} and n_{out} respectively, all possible $2^{n_{in}}$ outputs need to be well distributed in $2^{n_{out}}$ solution space.

Before discussing how to choose n_{in} and n_{out}, let us first study how to find a seed vector $\boldsymbol{w}^c$, given $\boldsymbol{w}^q$. As shown in Figure 5, the overall operation can be expressed by linear equations $\mathcal{M}^{\oplus}\boldsymbol{w}^c$=$\boldsymbol{w}^q$ over $GF(2)$. Note that linear equations associated with *don't care* bits in $\boldsymbol{w}^q$ can be ignored, because decryption through XOR-gate network can produce any bits in the locations of *don't care* bits. By deleting unnecessary linear equations, the original equations can be reduced as below:

$$\hat{\mathcal{M}}^{\oplus}\boldsymbol{w}^c = \boldsymbol{w}^q_{\{i_1,\ldots,i_k\}}, \text{where} \quad (1)$$

- $\{i_1, \ldots, i_k\}$ is a set of indices indicating *care* bits of each vector $\boldsymbol{w}^q$ ($0 \le k \le n_{out}, 1 \le i_k \le n_{out}$).
- $\hat{\mathcal{M}}^{\oplus} := \mathcal{M}^{\oplus}[i_1, \ldots, i_k; 1, \ldots, n_{in}]$

For example, since only 4 *care* bits ($\boldsymbol{w}^q_{\{3,4,5,7\}}$) exist in Figure 5, the (8×4) matrix $\mathcal{M}^{\oplus}$ is reduced to a (4×4) matrix, $\hat{\mathcal{M}}^{\oplus}$, by removing 1st, 2nd, 6th, and 8th rows.

Given the pruning rate S, $\boldsymbol{w}^q$ contains $n_{out}\times(1-S)$ *care* bits on average. Assuming that $n_{out}\times(1-S)$ equations are independent and non-trivial, the required number of seed inputs (n_{in}) can be as small as $n_{out}\times(1-S)$, wherein the compression ratio n_{out}/n_{in} becomes $1/(1-S)$. As a result, higher pruning rates lead to higher compression ratios. However, note that the linear equations may not have a corresponding solution when there are too many 'local' *care* bits or there are conflicting equations for a given vector $\boldsymbol{w}^q$.

3.2. Extra Patches for Lossless Compression

In order to keep our proposed SQNNs representation lossless, we add extra bits to correct unavoidable errors,

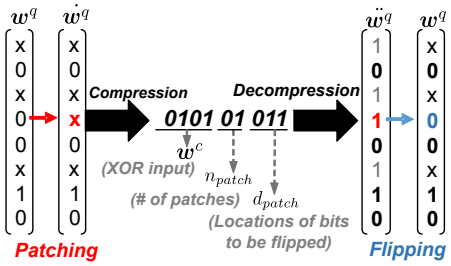


Figure 6: For every $\boldsymbol{w}^c$, additional n_{patch} and d_{patch} are attached to match all *care* bits.

i.e., patching. An unsolvable linear equation implies that the XOR random number generator cannot produce one or more matched *care* bits of $\boldsymbol{w}$. To resolve such an occasion, we can replace one or more *care* bits of $\boldsymbol{w}^q$ with *don't care* bits to remove conflicting linear equations, as depicted in Figure 6. We record the locations of replacements as d_{patch} which can be used to recover the original *care* bits of $\boldsymbol{w}$ by *flipping* the corresponding bits during decryption. For every $\boldsymbol{w}^q$, n_{patch} indicates the number of replacements for each $\boldsymbol{w}^c$. Since n_{patch} is always scanned prior to decryption using $\boldsymbol{w}^c$, the same number of bits is reserved to represent n_{patch} for all encrypted vectors in order to maintain a regular compressed format. On the other hand, the size of d_{patch} can be different for each $\boldsymbol{w}^q$ (overall parallelism is not disrupted by different d_{patch} sizes as flipping occurs infrequently).

At the expense of n_{patch} and d_{patch}, our compression technique can reproduce all *care* bits of $\boldsymbol{w}^q$ and, therefore, does not affect accuracy of DNN models and obviates retraining. In sum, the compressed format includes 1) $\boldsymbol{w}^c_{1\ldots l} \in \{0,1\}^{n_{in}}$ $(l = \lceil \frac{mn}{n_{out}} \rceil)$ encrypted from a quantized weight matrix $\boldsymbol{W}^q_i \in \{0,x,1\}^{m \times n}$ through $\mathcal{M}^{\oplus} \in \{0,1\}^{n_{out} \times n_{in}}$, 2) n_{patch}, and 3) d_{patch}. Hence, the resulting compression ratio r is

$$r = \frac{mn}{(\frac{n_{in}}{n_{out}} mn + l\lceil \lg \max(\boldsymbol{p}) \rceil + \sum_{j=1}^{l} p_j \lceil \lg n_{out} \rceil)}, \quad (2)$$

where p_j is the j^{th} n_{patch} and $\boldsymbol{p} = \{p_1, p_2, ..., p_l\}$. Improving r is enabled by increasing n_{out}/n_{in} and decreasing the amount of patches. We introduce a heuristic patch-searching algorithm to reduce the number of patches while also optimizing n_{in} and n_{out}.

3.3. Experiments Using Synthetic Data

An exhaustive search of patches requires exponential-time complexity even though such a method minimizes the number of patches. Algorithm 1 is a heuristic algorithm to search encrypted bits including n_{patch} and d_{patch} for $\boldsymbol{w}^q$ in $\boldsymbol{W}$. The algorithm incrementally enlarges the reduced linear equation by including a *care* bit only when the enlarged equation is still solvable. Note that make_rref() in Algorithm 1 generates a reduced row-echelon form to quickly verify that the linear equations are solvable. If adding a certain *care* bit makes the equations unsolvable, then a *don't care* bit takes its place, and n_{patch} and d_{patch} are updated accordingly. Although Algorithm 1 yields more replacement of *care* bits than an exhaustive search (by up to 10% from our extensive experiments), our simple algorithm has time complexity of $O(n_{out})$, which is much faster.

To investigate the compression capability of our proposed scheme, we evaluate a large random weight matrix of 10,000 elements where each element becomes a *don't care* bit with the probability of 0.9 (=sparsity or pruning rate). If an element is a *care* bit, then a 0 or 1 is assigned with the same probability. Notice that randomness of loca-

Algorithm 1: Patch-Searching Algorithm

input : a set of indices of care bits $\{i_1, ..., i_k\}$,
a sliced weight vector $\boldsymbol{w}^q \in \{0, x, 1\}^{n_{out}}$,
a fixed matrix $\mathcal{M}^{\oplus} \in \{0,1\}^{n_{out} \times n_{in}}$

```
mat ← empty matrix which column size is n_in + 1
for i in {i_1, ..., i_k} do        // for only not-pruned bits
    Append a row (M^⊕_{i,1}, ...., M^⊕_{i,n_in}, w^q_i) to mat
    rref ← make_rref(mat)
    if rref.is_solved() is False then
        Remove the last row of mat from mat
    end if
end for
Solve linear equations using mat to find w^c
ẅ^q ← M^⊕ w^c
Compare ẅ^q with w^q to produce n_patch and d_patch
Return n_patch, d_patch, w^c
```

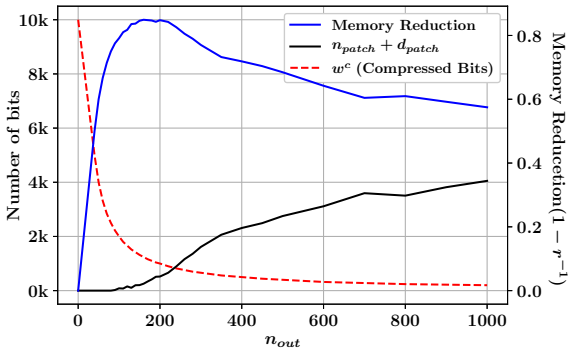
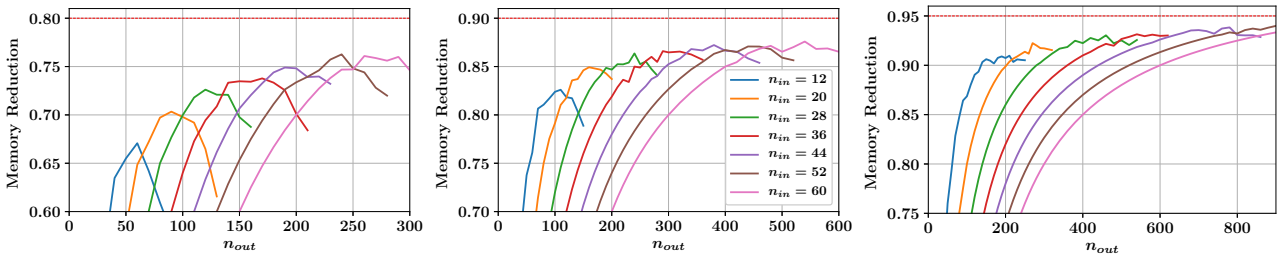


Figure 7: Memory reduction by applying Algorithm 1 to a random weight matrix with 10,000 elements with various n_{out} (pruning rate S=0.9, n_{in}=20)



(a) $S = 0.8$ (b) $S = 0.9$ (c) $S = 0.95$

Figure 8: Memory reduction using various n_{in} and pruning rates. n_{in} ranges from 12 to 60. Each line is stopped when the memory reduction begins to fall.

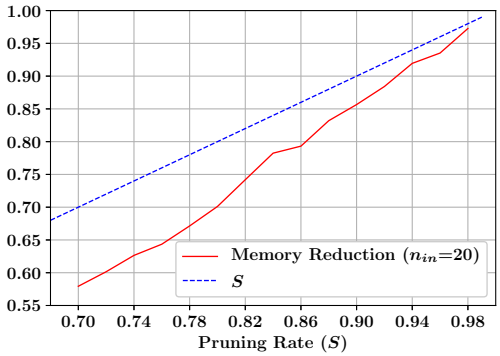


Figure 9: Graphs of memory reduction using n_{in}=20 (red line) and S (blue line). The gap between those two graphs is reduced with higher pruning rate.

tions in *don't care* bits is a feature of fine-grained pruning methods and assignment of 0 or 1 to weights with the same probability is obtainable using well-balanced quantization techniques [1, 19].

When n_{in} is fixed, the optimal n_{out} maximizes the memory reduction offered by Algorithm 1. Figure 7 plots the corresponding memory reduction (=$1-r^{-1}$) from $\boldsymbol{W}^q$ on the right axis and the amount of $\boldsymbol{w}^c$ and n_{patch} + d_{patch} on the left axis across a range of n_{out} values when n_{in}=20. From Figure 7, it is clear that there exists a trade-off between the size of $\boldsymbol{w}^c$ and the sizes of n_{patch} and d_{patch}. Increasing n_{out} rapidly reduces $\boldsymbol{w}^c$ while n_{patch} and d_{patch} grow gradually. The highest memory reduction (≈ 0.83) is achieved when n_{out} is almost 200, which agrees with the observation that maximum compression is constrained by the relative number of *care* bits [30]. Consequently, the resulting compression ratio approaches $1/(1-S)$.

Given the relationship above, we can now optimize n_{in}. Figure 8 compares memory reduction for various n_{out} across different values of n_{in}. The resulting trend suggests that higher n_{in} yields more memory reduction. This is because increasing the number of bits used as seed values for the XOR-gate random number generator enables a larger solution space and, as a result, fewer d_{patch} are needed as n_{in} increases. The large solution space is especially useful when *don't care* bits are not evenly distributed throughout $\boldsymbol{W}^q$. Lastly, n_{in} is constrained by the maximum computation time available to run Algorithm 1.

Figure 9 presents the relationship between pruning rate S and memory reduction when n_{in}=20 and sweeping S. Since high pruning rates translate to fewer *care* bits and relatively fewer d_{patch}, Figure 9 confirms that memory reduction reaches S as S increases. In other words, maximizing pruning rate is key to compressing quantized weights with high compression ratio. In comparison, ternary quantization usually induces a lower pruning rate [36, 23]. Our proposed representation is best implemented by pruning first to maximize pruning rate and then quantizing the weights.

4. Experiments on various SQNNs

In this section, we show experimental results of the proposed representation for four popular datasets: MNIST, ImageNet [29], CIFAR10 [16], and Penn Tree Bank [26]. Though the compression ratio ideally reaches $1/(1-S)$, the actual results may not, because *don't care* bits can be less evenly distributed than the synthetic data we used for Section 3.3. Hence, we suggest several additional techniques in this section to handle uneven distributions.

Weights are pruned by the mask layer generated by binary-index matrix factorization [22] after pre-training, and then retrained. Quantization is performed by following the technique proposed in [20] and [15], where quantization-aware optimization is performed based on the quantization method from [32]. The number of bits per weight required by our method is compared with the case of n_q-bit quantization with an additional 1-bit indicating pruning index (e.g., ternary quantization consists of 1-bit quantization and 1-bit pruning indication per weight).

We first tested our representation using the LeNet-5

Target Model			Pre-trained	Pruning and Quantization		
Model	DataSet	Size	Acc.	S	n_q-bit	Acc.
LeNet5 (FC1)	MNIST	800×500	99.1%	0.95	1-bit	99.1%
AlexNet (FC5, FC6)	ImageNet	9K×4K, 4K×4K	57.6% (T1)	0.91	1-bit	55.9% (T1)
ResNet32 (Conv Layers)	CIFAR10	460.76K	92.5%	0.70	2-bit	91.6%
LSTM	PTB	6.41M	89.6 PPW	0.60	2-bit	93.9 PPW

Table 2: Descriptions of models to be compressed by our proposed method. The model accuracy after parameter pruning and quantization is obtained by a binary-index factorization [22] and alternating multi-bit quantization [32].

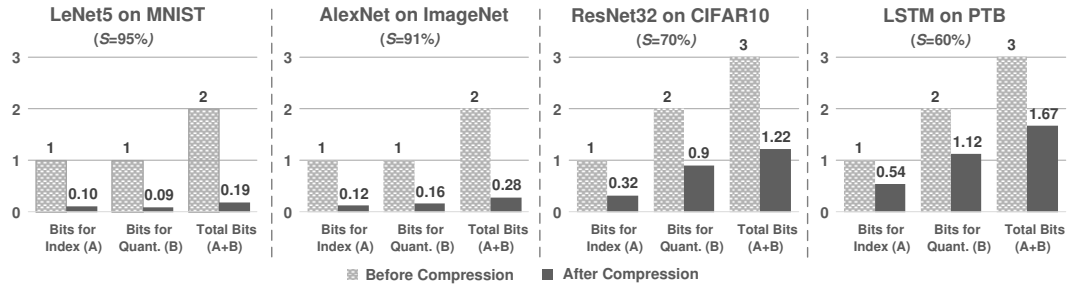


Figure 10: The number of bits to represent each weight for the models in Table 2 using our proposed SQNNs format. (A) means the number of bits for index (compressed by binary-index matrix factorization introduced in [22]). (B) indicates the number of bits for quantization by our proposed compression technique. Overall, we gain additional 2-11× memory footprint reduction according to sparsity. Note that memory overhead due to XOR-gate network is negligible because a relatively small XOR-gate network is pre-determined and fixed in advance.

model on MNIST. LeNet-5 consists of two convolutional layers and two fully-connected layers [11, 20]. Since the FC1 layer dominates the memory footprint (93%), we focus only on the FC1 layer whose parameters can be pruned by 95%. With our proposed method, the FC1 layer is effectively represented by only 0.19 bits per weight, which is 11× smaller than ternary quantization, as Figure 10 shows. We also tested our proposed compression techniques on large-scale models and datasets, namely, AlexNet [17] on the ImageNet dataset. We focused on compressing FC5 and FC6 fully-connected layers occupying ∼90% of the total model size for AlexNet. Both layers are pruned by a pruning rate of 91% [11] using binary-index matrix factorization [22] and compressed by 1-bit quantization. The high pruning rate lets us compress the quantized weights by ∼7×. Overall, FC5 and FC6 layers for AlexNet require only 0.28 bits per weight, which is substantially less than 2 bits per weight required by ternary quantization.

We further verify our compression techniques using ResNet32 [12] on the CIFAR10 dataset with a baseline accuracy of 92.5%. The model is pruned and quantized to 2 bits, reaching 91.6% accuracy after retraining. Further compression with our proposed SQNN format yields 1.22 bits per weight, while 3 bits would be required without our proposed compression techniques.

Additionally, an RNN model with one LSTM layer of size 300 [32] on the PTB dataset, with performance measured by using Perplexity Per Word, is compressed by our representation. Following our proposed representation scheme along with pruning and 2-bit quantization, such PTB LSTM model requires only 1.67 bits per weight.

For various types of layers, our proposed technique, supported by weight sparsity, provides additional compression over ternary quantization. Compression ratios can be further improved by using more advanced pruning and quantization methods (e.g., [31, 8]) since the principles of our compression methods do not rely on the specific pruning and quantization methods used.

5. Discussion

5.1. Variation on Execution Time

While decryption process through XOR-gate network provides a fixed output rate (thus, high parallelism), if all mismatched bits are supposed to be corrected by patches, then entire decoding (including decryption and patch correction) may result in variation in execution time due to irregular patch size. In order to mitigate variation in patch

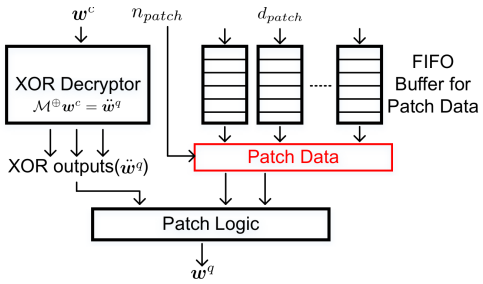


Figure 11: Patch data process structure corresponding to each XOR-gate network with a multi-bank FIFO to store patch data.

process, we first assume that patch data d_{patch} structure is decoupled from encrypted weights such that d_{patch} is given as a stream data to be stored into buffers in a fixed rate. Then, for each XOR decryption cycle, d_{patch} as much as n_{patch} is read from buffers and used to fix XOR outputs. In Figure 11, d_{patch} is stored into or loaded from FIFO buffers when the number of FIFO banks is n_{FIFO}. If d_{patch} is needed, FIFO buffers deliver d_{patch} to patch process logic while available d_{patch} throughput for load/store is determined by the number of FIFO banks. Decoding process can be stalled when the FIFO is either full or empty if temporal d_{patch} consumption rate is too low or too high.

Figure 12 presents relative execution time using CSR format or the proposed scheme with different n_{FIFO} configurations. FIFO size can be small enough (say, 256 entries) to tolerate temporal high peak d_{patch} usage. Hence, in Figure 12, stalls in the proposed scheme are due to high d_{patch} throughput demands along with large n_{patch}. Note that CSR format yields high variations in execution time because each row exhibits various numbers of unpruned weights. On the other hand, in the case of the proposed scheme, increasing n_{FIFO} (at the cost of additional hardware design) enhances d_{patch} throughput, and thus, reduces the number of stalls incurred by limited d_{patch} bandwidth of FIFOs. In sum, reasonable n_{FIFO} size significantly reduces execution time variations, which has been a major bottleneck in implementing fine-grained pruning schemes, as demonstrated in Figure 1.

5.2. Practical Techniques to Reduce n_{patch}

If n_{out} is large enough, patching overhead is not supposed to disrupt the parallel decoding, ideally. However, even for large n_{out}, if the nonuniformity of pruning rates is observed over a wide range within a matrix, n_{patch} may considerably increase. Large n_{patch}, then, leads to not only degraded compression ratio compared with synthetic data experiments, but also deteriorated parallelism in the decoding process. The following techniques can be considered to

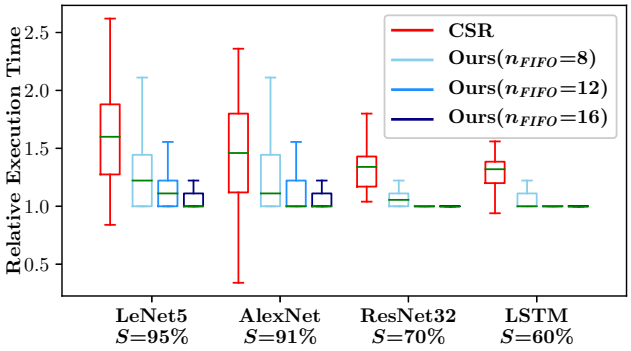


Figure 12: Relative execution time with CSR and the proposed scheme with various n_{FIFO} size. 1.0 in y-axis means no pruning rate variation in each row in the case of CSR format, or no stall due to limited d_{patch} load bandwidth in the case of the proposed scheme.

reduce n_{patch}.

Blocked n_{patch} Assignment: The compression ratio r in the Eq. (8) of Section 3.2 is affected by the maximum of $\{p_1, p_2, ..., p_l\}$. Note that a particular vector $\boldsymbol{w}$ may have an exceptionally large number of *care* bits. In such a case, even if a quantized matrix $\boldsymbol{W}^q$ consists of mostly *don't care* bits and few patches are needed, all of the compressed vectors $\boldsymbol{w}^c$ must employ a large number of bits to track the number of patches. To mitigate such a problem and enhance the compression ratio r, we divide a binary matrix $\boldsymbol{W}^q$ into several blocks, and then $\max(\boldsymbol{p})$ is obtained in each block independently. Different n_{patch} is assigned to each block to reduce the overall n_{patch} data size.

Minimizing n_{patch} for Small n_{in}: One simple patch-minimizing algorithm is to list all possible $2^{n_{in}}$ inputs (for $\boldsymbol{w}^c$) and corresponding outputs through $\mathcal{M}^{\oplus}$ on memory and find a particular $\boldsymbol{w}^c$ that minimizes the number of patches. At the cost of high space complexity and memory consumption, such an exhaustive optimization guarantees minimized n_{patch}. n_{in} below 30 is a practical value.

6. Conclusion

This paper proposes a compressed representation for Sparse Quantized Neural Networks based on an idea used for test-data compression. Through XOR-gate network and solving linear equations, we can remove most *don't care* bits and a quantized model is further compressed by sparsity. Since our representation provides a regular compressed-weight format with fixed and high compression ratios, SQNNs enable not only memory footprint reduction but also inference performance improvement due to inherently parallelizable computations.

References

[1] Daehyun Ahn, Dongsoo Lee, Taesu Kim, and Jae-Joon Kim. Double Viterbi: Weight encoding for high compression ratio and fast on-chip reconstruction for deep neural network. In *International Conference on Learning Representations (ICLR)*, 2019. 1, 3, 6

[2] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):32, 2017. 2

[3] Ismet Bayraktaroglu and Alex Orailoglu. Test volume and application time reduction through scan chain concealment. In *Proceedings of the 38th Annual Design Automation Conference*, 2001. 3

[4] 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*, pages 3123–3131, 2015. 1

[5] Misha Denil, Babak Shakibi, Laurent Dinh, Nando De Freitas, et al. Predicting parameters in deep learning. In *Advances in neural information processing systems*, pages 2148–2156, 2013. 1

[6] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018. 1

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 1

[8] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient DNNs. In *Advances in Neural Information Processing Systems*, 2016. 7

[9] 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 *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 243–254, 2016. 1, 3

[10] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016. 1, 2

[11] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015. 1, 7

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 7

[13] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017. 2

[14] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: training neural networks with low precision weights and activations. *arXiv:1609.07061*, 2016. 1

[15] Parichay Kapoor, Dongsoo Lee, Byeongwook Kim, and Saehyung Lee. Computation-efficient quantization method for deep neural networks, 2019. 6

[16] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 6

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 7

[18] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pages 598–605, 1990. 1

[19] Dongsoo Lee, Daehyun Ahn, Taesu Kim, Pierce I. Chuang, and Jae-Joon Kim. Viterbi-based pruning for sparse matrix with fixed and high index compression ratio. In *International Conference on Learning Representations (ICLR)*, 2018. 1, 3, 6

[20] Dongsoo Lee, Parichay Kapoor, and Byeongwook Kim. Deeptwist: Learning model compression via occasional weight distortion. *arXiv:1810.12823*, 2018. 1, 6, 7

[21] Dongsoo Lee and Byeongwook Kim. Retraining-based iterative weight quantization for deep neural networks. *arXiv:1805.11233*, 2018. 1

[22] Dongsoo Lee, Se Jung Kwon, Parichay Kapoor, Byeongwook Kim, and Gu-Yeon Wei. Network pruning for low-rank binary indexing. *arXiv:1905.05686*, 2019. 6, 7

[23] Fengfu Li and Bin Liu. Ternary weight networks. *arXiv:1605.04711*, 2016. 1, 6

[24] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, 2017. 2

[25] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J Dally. Exploring the regularity of sparse structure in convolutional neural networks. *arXiv preprint arXiv:1705.08922*, 2017. 2

[26] 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*, HLT '94, pages 114–119, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics. 6

[27] Dmitry Molchanov, Arsenii Ashukha, and Dmitry P. Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning (ICML)*, pages 2498–2507, 2017. 1

[28] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016. 1

[29] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 6

[30] Nur A. Touba. Survey of test vector compression techniques. *IEEE Design & Test of Computers*, 23:294–303, 2006. 2, 3, 6

[31] 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*, 2018. 7

[32] Chen Xu, Jianqiang Yao, Zhouchen Lin, Wenwu Ou, Yuanbin Cao, Zhirong Wang, and Hongbin Zha. Alternating multi-bit quantization for recurrent neural networks. In *International Conference on Learning Representations (ICLR)*, 2018. 1, 6, 7

[33] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *arXiv preprint arXiv:1802.00124*, 2018. 2

[34] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. Scalpel: Customizing DNN pruning to the underlying hardware parallelism. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 548–560, 2017. 2

[35] Xuda Zhou, Zidong Du, Qi Guo, Shaoli Liu, Chengsi Liu, Chao Wang, Xuehai Zhou, Ling Li, Tianshi Chen, and Yunji Chen. Cambricon-s: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 15–28. IEEE, 2018. 3

[36] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization. In *International Conference on Learning Representations (ICLR)*, 2017. 1, 6

[37] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *CoRR*, abs/1710.01878, 2017. 1