Structured Compression by Weight Encryption for Unstructured Pruning and Quantization

Se Jung Kwon
Dongsoo Lee
Byeongwook Kim
Parichay Kapoor
Baeseong Park
Gu-Yeon Wei
1Samsung Research, Republic of Korea
2Harvard University, MA

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 overparameterized 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-
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.
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 $W$ is quantized to be binary matrices $W^q$ ($1\leq i\leq n_q$) with $n_q$ as the number of bits for quantization. As the first step of our encryption algorithm, we reshape each binary matrix $W^q$ 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, $w^q$, including don’t care bits is encrypted to be a small vector $w^c$ (of $n_{in}$ size) without any don’t care bits. Through the XOR-gate network, each encrypted vector $w^c$ is decrypted to be original bits consisting of correct care bits and randomly filled don’t care bits with respect to $w^q$. 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 $\mathcal{M}^b\in\{0, 1\}^{n_{out}\times n_{in}}$ over Galois Field with two elements, $GF(2)$, using the connectivity information between the input vector $w^c\in\{0, 1\}^{n_{in}}$ (compressed and encrypted weights) and $w^q\in\{0, x, 1\}^{n_{out}}$. Note that $\mathcal{M}$

<table>
<thead>
<tr>
<th>Encryption</th>
<th>CSR Format</th>
<th>Viterbi-based Compr.</th>
<th>Proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Balance</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Decoding Rate</td>
<td>Uneven</td>
<td>Even</td>
<td>Even</td>
</tr>
<tr>
<td>Parallelism Limited by</td>
<td>Uneven Sparsity</td>
<td>Number of Decoders</td>
<td>Number of Decoders</td>
</tr>
<tr>
<td>Memory Access Pattern</td>
<td>Irregular (Gather-Scatter)</td>
<td>Regular</td>
<td>Regular</td>
</tr>
<tr>
<td>Compressed Memory Bandwidth</td>
<td>Depends on on-chip Buffer Structure</td>
<td>1 bit/decoder</td>
<td>Multi-bits/decoder</td>
</tr>
<tr>
<td>HW Resource for a Decoder</td>
<td>Large Buffer</td>
<td>XOR gates</td>
<td>XOR gates</td>
</tr>
</tbody>
</table>

![Table 1: Comparisons of CSR, Viterbi, and our proposed representation. For all of these representation schemes, compression ratio is upper-bounded by sparsity.](image-url)
Figure 4: An example illustrating the overall procedures of our proposed method using \((4 \times 4)\) full-precision weights. (a) We assume that a \((4 \times 4)\) weight matrix \(W\) is pruned and then quantized into 3 bits. (b) Quantized weights \((W^q_1, W^q_2, \text{and } W^q_3)\) are encrypted by using XOR-gate network which can be formulated as 4 XOR-based equations with 3 inputs \((x_1, x_2, \text{and } x_3)\). 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 \(W^q_1, W^q_2, \text{and } W^q_3\), decryption yields new quantized weights in which care bits are matched and don’t care bits are randomly filled.

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

\[
\mathcal{M}^{\oplus} w^q = w^q_{\{i_1, \ldots, i_k\}}, \quad \text{where} \quad \{i_1, \ldots, i_k\} \text{ is a set of indices indicating care bits of each vector } w^q \left(0 \leq k \leq n_{out}, 1 \leq i_k \leq n_{out}\right).
\]

\[
\mathcal{M}^{\oplus} := \mathcal{M}^{\oplus}_{\{i_1, \ldots, i_k\}}, \{i_1, \ldots, i_k\} \text{ exist in Figure 5, the } (8 \times 4) \text{ matrix } \mathcal{M}^{\oplus} \text{ is reduced to a } (4 \times 4) \text{ matrix, } \mathcal{M}^{\oplus}, \text{ by removing } 1^{\text{st}}, 2^{\text{nd}}, 6^{\text{th}}, \text{ and } 8^{\text{th}} \text{ rows.}
\]

Given the pruning rate \(S\), \(w^q\) 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 \(w^q\).

3.2. Extra Patches for Lossless Compression

In order to keep our proposed SQNNs representation lossless, we add extra bits to correct unavoidable errors,
Flipping Decompression generates a reduced row-echelon form to quickly to a
is a heuristic algorithm to yields more replace-
number of patches. Algorithm 3.3. Experiments Using Synthetic Data
searching algorithm to reduce the number of patches while
includes 1) An exhaustive search of patches requires exponential-
bit with the probability of 0.9 (=sparsity or pruning
care rate). If an element is a
bit makes the equations unsolvable, then a
don’t care
bit only when the enlarged
equation by including a
care
bit, then a 0 or 1 is assigned
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 w^q in 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_ref() 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-
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.

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
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].

<table>
<thead>
<tr>
<th>Model</th>
<th>DataSet</th>
<th>Size</th>
<th>Pre-trained</th>
<th>Pruning and Quantization</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeNet5 (FC1)</td>
<td>MNIST</td>
<td>800×500</td>
<td>99.1%</td>
<td>0.95 1-bit 99.1%</td>
</tr>
<tr>
<td>AlexNet (FC5, FC6)</td>
<td>ImageNet</td>
<td>9K×4K, 4K×4K</td>
<td>57.6% (T1)</td>
<td>0.91 1-bit 55.9% (T1)</td>
</tr>
<tr>
<td>ResNet32 (Conv Layers)</td>
<td>CIFAR10</td>
<td>460.76K</td>
<td>92.5%</td>
<td>0.70 2-bit 91.6%</td>
</tr>
<tr>
<td>LSTM</td>
<td>PTB</td>
<td>6.41M</td>
<td>89.6 PPW</td>
<td>0.60 2-bit 93.9 PPW</td>
</tr>
</tbody>
</table>

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.

The 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 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
process, we first assume that patch data $d_{\text{patch}}$ structure is decoupled from encrypted weights such that $d_{\text{patch}}$ is given as a stream data to be stored into buffers in a fixed rate. Then, for each XOR decryption cycle, $d_{\text{patch}}$ as much as $n_{\text{patch}}$ is read from buffers and used to fix XOR outputs. In Figure 11, $d_{\text{patch}}$ is stored into or loaded from FIFO buffers when the number of FIFO banks is $n_{\text{FIFO}}$. If $d_{\text{patch}}$ is needed, FIFO buffers deliver $d_{\text{patch}}$ to patch process logic while available $d_{\text{patch}}$ 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 $d_{\text{patch}}$ consumption rate is too low or too high.

Figure 12 presents relative execution time using CSR format or the proposed scheme with different $n_{\text{FIFO}}$ configurations. FIFO size can be small enough (say, 256 entries) to tolerate temporal high peak $d_{\text{patch}}$ usage. Hence, in Figure 12, stalls in the proposed scheme are due to high $d_{\text{patch}}$ throughput demands along with large $n_{\text{patch}}$. 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 $n_{\text{FIFO}}$ (at the cost of additional hardware design) enhances $d_{\text{patch}}$ throughput, and thus, reduces the number of stalls incurred by limited $d_{\text{patch}}$ bandwidth of FIFOs. In sum, reasonable $n_{\text{FIFO}}$ 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 $n_{\text{patch}}$

If $n_{\text{out}}$ is large enough, patching overhead is not supposed to disrupt the parallel decoding, ideally. However, even for large $n_{\text{out}}$, if the nonuniformity of pruning rates is observed over a wide range within a matrix, $n_{\text{patch}}$ may considerably increase. Large $n_{\text{patch}}$, 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 reduce $n_{\text{patch}}$.

**Blocked $n_{\text{patch}}$ 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 $w$ may have an exceptionally large number of care bits. In such a case, even if a quantized matrix $W^q$ consists of mostly don’t care bits and few patches are needed, all of the compressed vectors $w^c$ 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 $W^q$ into several blocks, and then max($p$) is obtained in each block independently. Different $n_{\text{patch}}$ is assigned to each block to reduce the overall $n_{\text{patch}}$ data size.

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


