Relaxed Pruning: Memory-Efficient LSTM Inference Engine by Limiting the Synaptic Connection Patterns

Jaeha Kung
DGIST, Republic of Korea
jhkung@dgist.ac.kr

Junki Park
POSTECH, Republic of Korea
junkipark@postech.ac.kr

Jae-Joon Kim
POSTECH, Republic of Korea
jaejoon@postech.ac.kr

ABSTRACT

This paper presents a memory-efficient LSTM inference engine using a new sparse matrix format with limited synaptic connection patterns. The proposed relaxed pruning uses soft thresholding which removes infrequent connection patterns. The initial experimental results show that memory requirement reduces by 1.52~2.05× over the widely used CSC format depending on the level of sparsity in weight matrices. This improvement comes with negligible accuracy degradation by allowing additional training epochs. Also, the relaxed pruning provides an opportunity to improve inference speed with proper hardware architecture support.

ACM Reference Format:
Jaeha Kung, Junki Park, and Jae-Joon Kim. 2018. Relaxed Pruning: Memory-Efficient LSTM Inference Engine by Limiting the Synaptic Connection Patterns. In *Proceedings of ACM SysML conference (SysML'18).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The capability of machine intelligence has been significantly improved over the past decade owing to the enhanced compute power of the recent processors [10, 12]. As deep learning has performed extremely well in vision-related tasks, inference engines for Convolutional Neural Networks (CNNs) were mainly focused and developed [1, 2, 6, 9]. According to the report from Google, however, the percentage of the server usage on CNN is negligible compared to that of LSTM/MLP [4]. This implies that the research effort needs to be expanded to improve performance/energy-efficiency of LSTM (or MLP) inference engines. Since the complexity of LSTM is higher than MLP, we will focus on improving energy-efficiency of the LSTM inference engine. The challenge here is that there is no opportunity of data reuse in fully-connected (FC) layers (*little chance of performance improvement in LSTM*). Note that LSTM consists of eight FC-layer computations [8].

There have been several approaches to improve efficiency of the LSTM inference engine [5, 11]. In [11], LUT-based multipliers are used to improve energy-efficiency of the LSTM inference engine. However, weight parameters are fixed to 4-bit limiting its actual use in LSTM applications requiring higher bit-precision. In [5], authors presented an algorithm to balance out the computation loads on processing elements (PEs) to improve performance by iteratively changing the connections to be pruned. Also, it handles the sparsity with the conventional CSC data format. In this paper, we present a new sparse matrix format with *relaxed pruning* that minimizes the data to be fetched from external memory regardless of the sparsity level. This helps the hardware accelerator keep performance even when the sparsity is lower.

SysML'18, February 2018, Stanford, California, USA
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 HUFFMAN-CODED NONZERO INDICATION

To overcome the memory bottleneck incurred by the large weight space in LSTM layers, weight quantization and pruning are normally used [5]. These techniques make the synaptic connections sparse resulting in a lighter neural network (Fig. 1(a)). The pruning simply thresholds the weight values to cut weak connections. In the simple example shown in Fig. 1(a), 0.4 is used as a threshold making the sparsity 62.5%.

The conventional CSC format can be used to efficiently represent the sparse weight matrix [5]. The CSC format, however, is inefficient when the network becomes less sparse as it needs to send nonzero elements as well as row indices. As the size of matrix becomes larger, the required bit-width to represent row index increases. When the density of a 1024×1024 matrix exceeds 45%, the size of data to be fetched by using CSC format becomes larger than simply sending all data (Fig. 1(b)).

Instead of sending row indices, we can simply send nonzero indication bit-stream and use simple on-chip counters to identify the locations. To make the bit stream memory-efficient, we encode the stream with Huffman coding (*Huffman-coded Nonzero Indication (HNI)*). This reduces the number of bytes to travel over the expensive external memory interface. The example in Fig. 1(a) encodes the symbols with length of 4 (*symLen*=4).

The difference between CSC and HNI format is in the position identifier. Note that nonzero elements must be sent in both formats (Fig. 1 assumes 12-bit per element). Since the row/column length is four, 2-bit is required to represent row index in the CSC format. As the size of the weight matrix gets bigger, the required bit-width increases as well. For instance, 10-bit is needed for each entry in row index vector of a 1024×1024 matrix. The compression ratios are compared in Fig. 1(b) for the 1024×1024 weight matrix. For highly sparse matrix (10% density), the improvement of HNI format over the CSC format is 1.41×. The improvement in compression ratio increases to 1.90× when the density becomes 63%.

3 RELAXED PRUNING

The proposed HNI format provides additional benefit in memory efficiency. This is done by the modified pruning algorithm, named *relaxed pruning*. So far, pruning algorithms were applied by using hard thresholding as shown in Fig. 2(a). Instead, it is possible to apply pruning by allowing a don't-care region (Fig. 2(a)). In this region, the synaptic connection can be either kept or disconnected (analogous to don't-care term in digital logic). The relaxed pruning keeps the sparsity level same while finding the synaptic connection

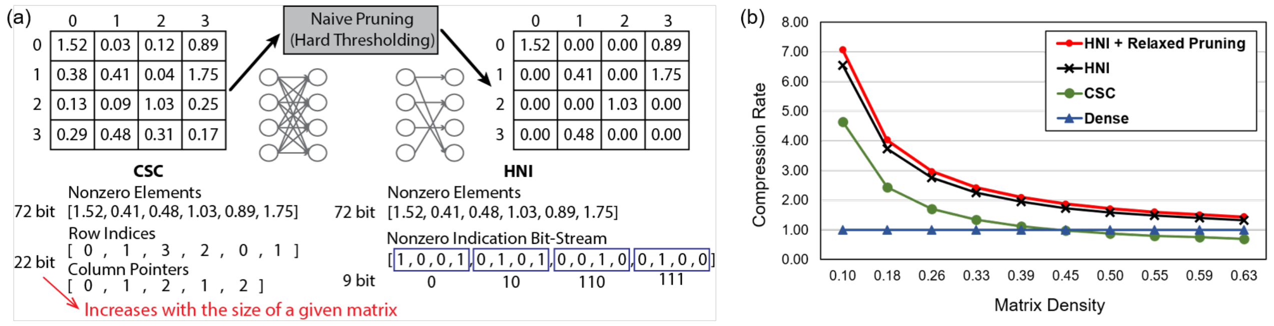

Figure 1: The proposed sparse matrix format: (a) Data reduction by using Huffman-coded Nonzero Indication (HNI) format after naive pruning and (b) compression ratio using sparse matrix formats depending on the sparsity of a 1024×1024 weight matrix.

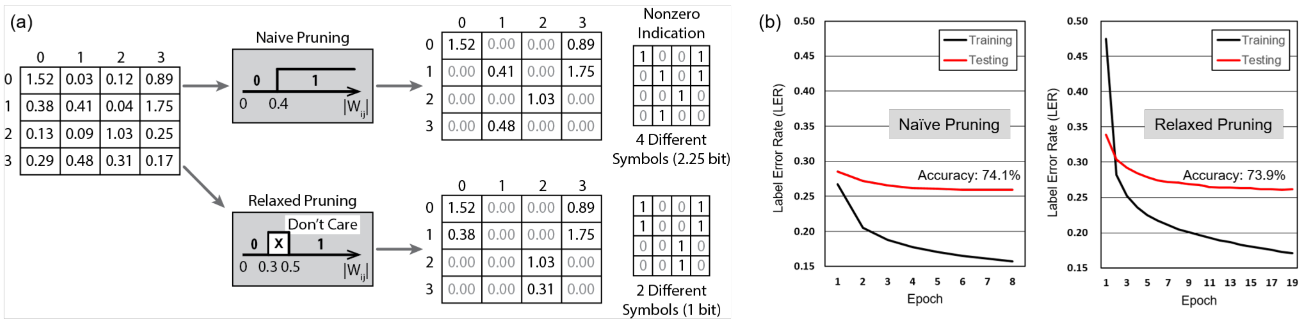

Figure 2: Relaxed pruning: (a) Huffman symbol reduction by limiting the synaptic connection patterns using relaxed pruning and (b) negligible accuracy degradation on speech recognition after allowing additional retraining epochs.

pattern that is more frequent in occurrence. This reduces the average length of Huffman codes. The detailed algorithm of relaxed pruning is not provided here due to the limited space.

One may argue that the accuracy of LSTM algorithm may degrade as the relaxed pruning limits the synaptic connection patterns. The experimental result in Fig. 2(b) shows that the accuracy degradation is negligible (only 0.2%) in speech recognition application by allowing 11 additional retraining epochs. The dataset used for this simulation is TIMIT dataset [3] and the LSTM network has a structure of (Input:39 - LSTM:512 - LSTM:512 - Output:62).

The impact of relaxed pruning may vary by symbol length. Table 1 summarizes the improvement in memory requirement with different symbol lengths. Here, *symLen=8* means that nonzero indication bit-stream is grouped by 8-bit to make one symbol. With *symLen=8*, the relaxed pruning additionally reduces the total memory footprint by 8% compared to the HNI format. This improvement is influential as the energy (pJ/bit) of external memory access is 23× higher than that of PE operation [6]. The relaxed pruning not only reduces memory footprint, but also improves decoding speed as the average code-length reduces by 27.9% with *symLen*=8. As the Huffman decoding is a sequential process, the reduction in code-length is extremely important in improving hardware throughput. Also, the reduction in the number of symbols (21.3%) lowers the on-chip LUT size for storing the Huffman tree [7]. These additional savings with relaxed pruning make the HNI format suitable for hardware implementation. The parallel Huffman decoders and detailed data controllers should be designed to make a high-performance LSTM inference engine. The parallel computations (*optimized dataflow*) of independent LSTM cell gates will maximize the inference speed as well.

Table 1: Improvement of memory-efficiency by using relaxed pruning over HNI format

symLen	Nonzero Element	Nonzero Indication	Total Data Size	Huffman Code Length	Symbol Reduction
4	0.00%	6.31%	1.72%	4.67%	0.00%
6	0.00%	20.50%	5.56%	18.08%	1.88%
8	0.00%	29.58%	8.00%	27.91%	21.33%

4 CONCLUSION

In this brief paper, we proposed a new sparse matrix format to reduce required memory footprint of LSTM inference engines. The relaxed pruning is developed to minimize memory footprint even further and to possibly improve LSTM hardware for faster inference. The performance improvement can be expected with the proper design of parallel Huffman decoders and the exploration of system parameters to maximize computation speed. Thus, the extensive study on architecture support needs to be followed to design a highly-efficient LSTM inference engine.

REFERENCES

[1] Y. H. Chen, J. Emer, and V. Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 367–379.

[2] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. 92–104.

[3] John S. Garofolo et al. 1993. TIMIT Acoustic-Phonetic Continuous Speech Corpus. (1993). https://catalog.ldc.upenn.edu/ldc93s1/

[4] Norman P. Jouppi et al. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA'17)*. ACM, New York, NY, USA, 1–12.

[5] Song Han et al. 2017. ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17)*. ACM, New York, NY, USA, 75–84.

[6] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'17)*. ACM, New York, NY, USA, 751–764.

[7] Reza Hashemian. 1994. Design and hardware implementation of a memory efficient Huffman decoding. *IEEE Transactions on Consumer Electronics* 40, 3 (Aug 1994), 345–352.

[8] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (Nov. 1997), 1735–1780.

[9] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst. 2017. Envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable Convolutional Neural Network processor in 28nm FDSOI. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. 246–247.

[10] NVIDIA. 2017. Volta GPU Architecture. (2017). https://www.nvidia.com/en-us/data-center/volta-gpu-architecture/

[11] D. Shin, J. Lee, J. Lee, and H. J. Yoo. 2017. DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. 240–241.

[12] Yang You, Aydin Buluc, and James Demmel. 2017. Scaling Deep Learning on GPU and Knights Landing Clusters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'17)*. ACM, New York, NY, USA, 9:1–9:12.