INTERSPEECH 2020
October 25–29, 2020, Shanghai, China

Compressing LSTM Networks with Hierarchical Coarse-Grain Sparsity

Deepak Kadetotad[1,2], *Jian Meng*[1], *Visar Berisha*[1], *Chaitali Chakrabarti*[1], *and Jae-sun Seo*[1]

[1]School of ECEE, Arizona State University
[2]Starkey Hearing Technologies
jaesun.seo@asu.edu

Abstract

The long short-term memory (LSTM) network is one of the most widely used recurrent neural networks (RNNs) for automatic speech recognition (ASR), but is parametrized by millions of parameters. This makes it prohibitive for memory-constrained hardware accelerators as the storage demand causes higher dependence on off-chip memory, which bottlenecks latency and power. In this paper, we propose a new LSTM training technique based on hierarchical coarse-grain sparsity (HCGS), which enforces hierarchical structured sparsity by randomly dropping static block-wise connections between layers. HCGS maintains the same hierarchical structured sparsity throughout training and inference; this reduces weight storage for both training and inference hardware systems. We also jointly optimize in-training quantization with HCGS on 2-/3-layer LSTM networks for the TIMIT and TED-LIUM corpora. With 16× structured compression and 6-bit weight precision, we achieved a phoneme error rate (PER) of 16.9% for TIMIT and a word error rate (WER) of 18.9% for TED-LIUM, showing the best trade-off between error rate and LSTM memory compression compared to prior works.
Index Terms: long short-term memory, speech recognition, weight compression, structured sparsity

1. Introduction

The advent of internet of things (IoT) and edge computing has created a demand for energy-efficient deep neural networks (DNNs) for mobile devices. The particular challenge of performing on-device ASR is that state-of-the-art LSTMs for ASR contain tens of millions of weights [1, 2]. Considering that off-chip memory access consumes high energy, it is crucial to store most or all weights on-chip through sparsity/compression, weight quantization, and network size reduction.

DNN compression has been heavily studied in the literature [3–8]. Magnitude-based pruning has shown large compression [3,9], but the index storage can be a large burden, especially for the simple coordinate (COO) format that stores each non-zero weight's location. The compressed sparse row/column (CSR/CSC) format [10] reduces the index cost as only the distance between non-zero elements in a row/column is stored, but still requires non-negligible index memory and causes irregular memory access [6]. To that end, row-/column-/block-wise structured compression techniques have been proposed [5–7, 11, 12], which minimizes the index storage, enables regular memory access, and enhances hardware acceleration [13–15].

Several recent works have jointly optimized compression and low-precision quantization [13, 15, 16], where the relative cost of index storage for compressed DNNs with low-precision weights will be even higher. Using the three aforementioned compression methods, COO, CSC, and structured sparsity, Figure 1 shows the comparison of index memory overhead for a 512×512 weight matrix with 4-bit weight precision, for compression targets from 1× (dense network) to 16× (6.25% of weights are non-zero). In this work, we introduce hierarchical block-wise sparsity for weight matrices in LSTMs, which substantially reduces the index overhead to <1.3% (Figure 1).

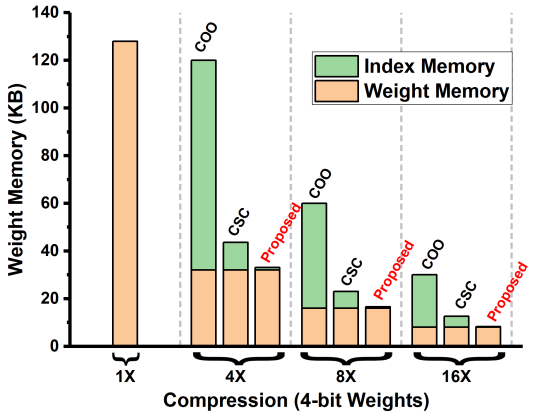

Figure 1: *Comparison of index and 512×512 weight memory for 4-bit weights with 4×, 8×, and 16× compression.*

For LSTM based RNNs, obtaining structured sparsity can be more challenging compared to multi-layer perceptrons (MLPs) or convolutional neural networks (CNNs) due to the temporal dependency of the recurrent units. In addition, aggressively compressing RNNs without taking into consideration the interconnected gates in LSTMs will lead to a mismatch in the dimension of weight matrices and adversely impact accuracy [11]. To minimize accuracy loss while maintaining a large compression rate, we propose hierarchical coarse-grain sparsity (HCGS) for LSTMs. We enforce a hierarchically sparse structure between LSTM layers before training, by randomly selecting large blocks and then randomly selecting small blocks recursively within the selected large blocks. Such hierarchical block-wise sparsity is maintained statically throughout training and inference, which can aid storage/computation reduction of hardware accelerators.

Preliminary HCGS algorithm and hardware accelerator design was reported in [14], while this work presents algorithm enhancement with in-training quantization [17], further analysis on multi-tier HCGS, in-depth investigation on LSTM design with optimal memory, and comparison to learned sparsity methods. We train 2-/3-layer LSTMs for the TIMIT [18] and TED-LIUM [19] corpora. With 16× structured compression and 6-bit weight precision, we achieved 16.9% PER for TIMIT with 0.3 MB of total LSTM weight memory, and 18.9% WER for TED-LIUM with 1 MB weight memory. By evaluating various compression and quantization values on different sizes of LSTMs, we determine the Pareto-optimal designs, where HCGS-based LSTMs show the best trade-off between error rate and weight memory compared to prior works. Code for this work is available at `https://github.com/razor1179/`.

http://dx.doi.org/10.21437/Interspeech.2020-1270

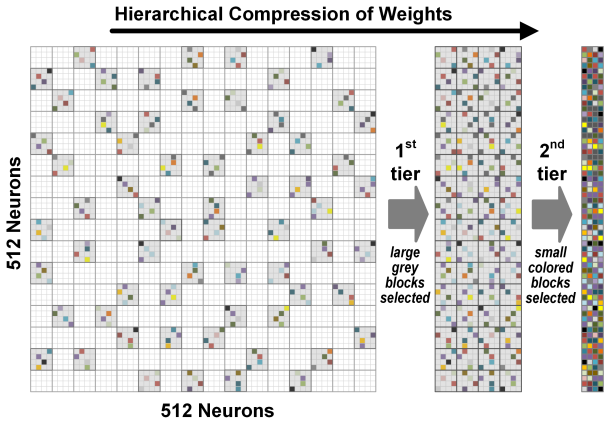

Figure 2: *Proposed HCGS-based structured compression.*

2. HCGS based LSTM training

2.1. Long short-term memory RNN

RNNs construct consecutive hidden states $\{h_1, h_2, ..., h_T\}$ by processing corresponding input sequence $\{x_1, x_2, ..., x_T\}$. Each hidden state is implicitly trained to remember task-relevant aspects of preceding inputs, and is incorporated with new inputs via a recurrent operator, T. This operation converts the previous hidden state and the present input into a new hidden state, e.g.,

$$h_t = T(h_{t-1}, x_t) = tanh(Wx_t + Uh_{t-1} + b), \quad (1)$$

where W and U are weights for the feed-forward and recurrent structures, respectively, and b is the bias parameter.

In addition to the hidden state h_t, LSTM introduces a memory cell c_t, intended for internal long-term storage. The parameters c_t and h_t are computed via input, output, and forget gate functions. The forget gate function f_t directly connects c_t to the memory cell c_{t-1} of the previous timestep via an element-wise multiplication. Large values of the forget gates cause the cell to remember most (if not all) of its previous values. Each gate function has a weight matrix and a bias vector; we use subscripts i, o and f to denote parameters for the input, output and forget gate functions, respectively, e.g., the parameters for the forget gate function are denoted by W_f, U_f, and b_f.

2.2. Hierarchical coarse-grain sparsity (HCGS)

We propose hierarchical coarse-grain sparsity (HCGS) for LSTM training. Figure 2 illustrates the HCGS implementation for weight matrices in LSTMs, where the connections between feed-forward layers and recurrent layers are dropped in a hierarchical and recursive block-wise manner. The example shown in Figure 2 has a two-tier hierarchy, where the first tier connections are dropped randomly in large blocks (i.e. grey blocks). Within the preserved connections in the first tier (grey blocks), the second tier connections are then dropped randomly in smaller colored blocks to achieve further sparsity. Only the weights that are preserved through both tiers of hierarchy are trained and employed for inference. This random block selection is stored in a connection mask (C^W or C^U in Algorithm 1) at the start of training and fixed throughout training. The connection mask only contains $0's$ and $1's$, where $0's$ signify the deleted connections and $1's$ represent the preserved block-wise connections.

The indices needed for HCGS networks in Figure 2 also have two tiers. The first tier index stores the location of the grey block within the weight matrix, and the second tier index represents the smaller block's location within the larger grey block. The HCGS hierarchy can be expanded to have multiple tiers of block-wise sparse structure, recursively selecting even smaller blocks within smaller blocks.

Algorithm 1 Training LSTM with HCGS. $\circ$ indicates element-wise multiplication, C is the cost function for a minibatch, λ is the learning rate decay factor, and L is the number of layers.

Require: a minibatch of inputs and targets (x, a^*), previous weights W and U, HCGS mask C^W and C^U as well as previous learning rate η.
Ensure: updated weights W^{t+1} and U^{t+1} and updated learning rate η^{t+1}.
Forward Propagation:
for $k = 1$ to L **do**
  $W_{k_{i,f,o,c}} \leftarrow W_{k_{i,f,o,c}} \circ C_k^W$
  $U_{k_{i,f,o,c}} \leftarrow U_{k_{i,f,o,c}} \circ C_k^U$
  $h_{k,t} \leftarrow$ Compute$(W_{k_{i,f,o,c}}, U_{k_{i,f,o,c}}, x_{k,t})$ {via (1)-(5)}
  $x_{k+1,t} \leftarrow h_{k,t}$
end for
Backward Propagation:
$g_{W_{k_{i,f,o,c}}}$ and $g_{U_{k_{i,f,o,c}}}$ are the gradients calculated for each layer k from 1 to L and are represented below as g_{W_k} and g_{U_k} respectively for simplicity. Similarly $W_{k_{i,f,o,c}}$ and $U_{k_{i,f,o,c}}$ are represented as W_k and U_k.
Parameter Update:
for $k = 1$ to L **do**
  $g_{W_k} \leftarrow g_{W_k} \circ C_k^W$
  $W_k^{t+1} \leftarrow$ Update(W_k, η, g_{W_k})
  $g_{U_k} \leftarrow g_{U_k} \circ C_k^U$
  $U_k^{t+1} \leftarrow$ Update(U_k, η, g_{U_k})
  $\eta^{t+1} \leftarrow \lambda\eta$
end for

Algorithm 1 shows the computational changes required to incorporate HCGS in LSTM training. The binary connection mask is initialized for every layer of the feed-forward network (C^W) and the recurrent network (C^U), which forces the deleted weight connections to zero during the forward propagation. During back-propagation, the HCGS mask ensures that the deleted weights do not get updated and remain zero throughout training.

To further increase compression efficiency, weights associated with the four gates in each LSTM layer share the common connection mask that is randomly selected. Sharing the same random mask results in $4\times$ reduction of the index memory, and reduces the computations for decompression by $4\times$ as well. Compared to cases where different random masks were used for the four gates, sharing the same random mask did not affect PER or WER by more than 0.2% across all our LSTM experiments.

2.3. Guided coarse-grain sparsity (guided-CGS)

To benchmark the proposed pre-determined random sparsity against variants of learned sparsity methods, we introduce a guided block-wise sparsity method called guided coarse-grain sparsity (Guided-CGS). Unlike HCGS where the blocks are chosen randomly, Guided-CGS implements a magnitude-based selection criteria to select blocks that contain the largest absolute mean weights, and the unselected blocks will be zero. The magnitude-based selection is executed after one epoch of training with group Lasso [20]. This method ensures that the weight block selection is done through group Lasso based optimization, instead of being randomly chosen.

2.4. Quantizing LSTM networks

A technique to achieve high accuracy with very low-precision quantization was proposed in [17], where weights of the DNN were quantized during training. We employed similar in-training quantization schemes that jointly optimize block-wise sparsity and low-precision quantization. During the forward propagation part of the LSTM training, each weight is quantized to n bits, while the backward propagation part employs full-precision weights. This way, the network is optimized to minimize the cost function with n-bit precision weights. The n-bit quantized weights are represented in (2) and steps to make quantized copies of the full-precision weights are shown in Algorithm 2.

$$W^{q_n} = Quantization(W, n) \tag{2}$$

Algorithm 2 Quantization. $\circ$ indicates element-wise multiplication and $/$ is element-wise division.

Require: weights W, quantize bits n.
$W \leftarrow clamp(W, -1, 1)$
$W^{sign} \leftarrow Sign(W)$
$W^{q_n} \leftarrow (\frac{ceil(abs(W) \circ 2^{n-1})}{2^{n-1}}) \circ W^{sign}$

The parameter update section in Algorithm 1 is adapted to include the process of updating the batch normalization parameters. Back-propagation through time (BPTT) [21] is used to compute the gradients by minimizing the cost function using the quantized weights W^{q_n}, but the full-precision weight copies (W) are updated to ensure the network is optimized to reduce the output error for quantized weights.

3. Experiments

3.1. Experimental setup

For the speech recognition tasks, the input consists of 440 feature space maximum likelihood linear regression (fMLLR) features [22] that are extracted using the s5 recipe of Kaldi [23]. The fMLLR features were computed using time window of 25ms with an overlap of 10ms. We use the PyTorch-Kaldi speech recognition toolkit [24] to train the LSTM networks. The final LSTM layer generates the acoustic posterior probabilities, which are normalized by their prior and then conveyed to a hidden Markov model (HMM) based decoder. An n-gram language model derived from the language probabilities is merged with the acoustic scores by the decoder. A beam search algorithm is then used to retrieve the sequence of words uttered in the speech signal. The final error rates for TIMIT and TED-LIUM corpora are computed with the NIST SCTK scoring toolkit [25].

For TIMIT, we considered the phoneme recognition task (aligned with the Kaldi s5 recipe) and trained 2-layer uni-directional LSTMs, with 256, 512, and 1,024 cells per layer. For TED-LIUM, we targeted the word recognition task (aligned with the Kaldi s5 recipe) and trained 3-layer uni-directional LSTMs, with 256, 512, and 1,024 cells per layer. We evaluated all possible combinations of power-of-2 block sizes, and the PER for TIMIT and WER for TED-LIUM were relatively constant, showing the robustness of HCGS across different block sizes.

3.2. Improvements due to HCGS

We observe improvements in error rates when we train LSTMs with HCGS. Figure 3 shows the PER improvement due to the hierarchical structure in two-tier HCGS scheme, compared to the single-tier CGS scheme reproduced from [5]. The results for LSTMs with 32-bit weight precision for different number of cells (256, 512, and 1,024) and compression rates (1× to 16×) are shown. In all experiments, LSTM networks trained with two-tier HCGS achieve noticeably lower PER than single-tier CGS for the same target compression. The three-tier HCGS shows marginal PER improvement over two-tier HCGS for LSTMs with 1,024 cells, but worse PER for LSTMs with 256 and 512 cells. Four-tier HCGS resulted in worse PER results compared to three-tier HCGS, hence was not included in Figure 3.

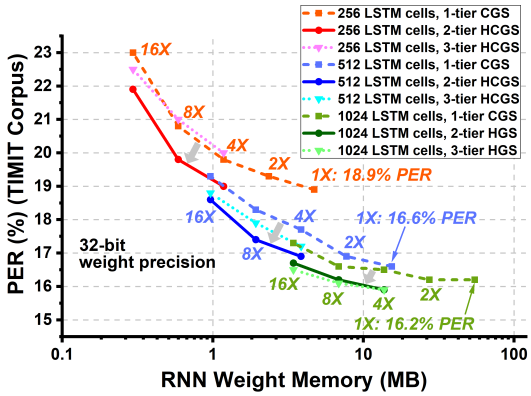

Figure 3: *PER (TIMIT) comparison between single-tier CGS and multi-tier HCGS schemes.*

We believe the hierarchical sparsity leads to the improved accuracy of the networks. Sparse weights with fine granularity tend to form a uniform sparsity distribution even within smaller regions of the weight matrix. This property will lead to extremely sporadic and isolated connections when the target compression rate is high. However, the grouping of sparse weights within the hierarchical structure of HCGS allows densely connected regions to be formed even when the target compression rate is high. As two-tier HCGS outperforms single-tier CGS in terms of accuracy and three-tier HCGS leads to marginal/worse performance than two-tier HCGS, we focused on LSTM training with two-tier HCGS for the reported results in Section 3.3 and 3.4.

3.3. LSTM results for TIMIT

For the TIMIT corpus, we trained 2-layer LSTMs for a number of different LSTM cells (256, 512 and 1,024), compression rates (2×, 4×, 8× and 16×) and weight quantization schemes (32-bit, 6-bit and 3-bit), Figure 4. shows the compiled PER and weight memory curves of HCGS-based LSTMs. For a similar memory footprint, we observe that wider sparse networks perform better than narrower dense networks, similar to what was reported in [7, 26]. For example, a 1,024-cell network with 8× compression shows a lower PER than a 512-cell network with 2× compression. The Pareto front curve in Figure 4 offers the lowest PER for the smallest memory in the search space.

3.4. LSTM results for TED-LIUM

For the TED-LIUM corpus, we trained 3-layer LSTMs for a number of different LSTM cells (256, 512 and 1,024), compression rates (2×, 4×, 8× and 16×) and weight precision schemes (32-bit, 6-bit and 3-bit). Figure 5 shows the compiled WER and RNN weight memory curves. Similar to Figure 4, we find that a 1,024-cell network with 8× compression results in a lower WER than a 512-cell network with 2× compression. The Pareto front curve is extracted and shown in Figure 5.

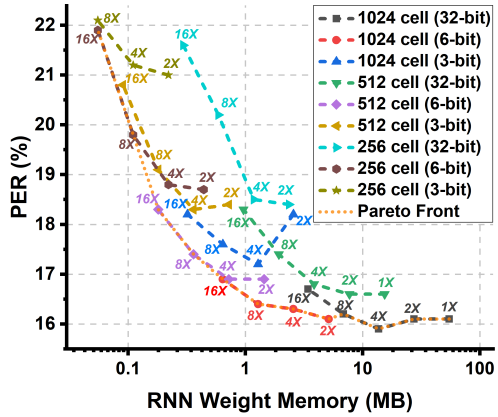

Figure 4: *PER vs. RNN weight memory results for various 2-layer LSTMs for TIMIT.*

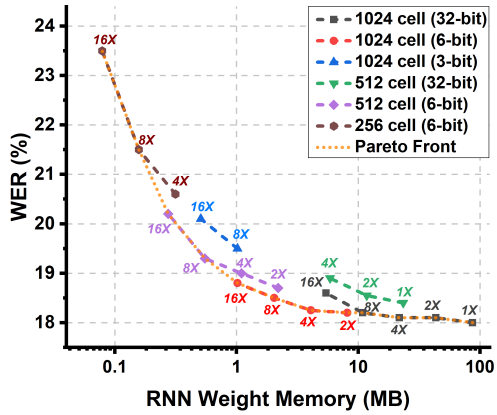

Figure 5: *WER vs. RNN weight memory results for various 3-layer LSTMs for TED-LIUM.*

The authors of [27] reported that wider CNNs can lower the precision of activations/weights than shallower counterparts, for the same or even better accuracy. However, in this work, we do not see such trends with LSTMs for TIMIT or TED-LIUM. Especially when combined with structured compression, we find that LSTMs are more sensitive to low-precision quantization, so that LSTMs with medium (e.g. 6-bit) precision show the best trade-off between PER/WER and weight memory compression.

3.5. Comparison with learned sparsity and prior works

For comprehensive comparison, we implemented learned sparsity methods of Guided-CGS (Section 2.3), group Lasso [20], L1 normalization [7] and magnitude-based pruning (MP). To obtain block-wise sparsity for group Lasso scheme, block sizes similar to that of single-tier CGS are chosen. The sparsity for group Lasso and L1 schemes are obtained through a final pruning operation conducted after training. For every scheme, we applied the same sparsity for all weight matrices (32-bit precision) in 2-layer 512-cell LSTMs, and Figure 6 shows the comparison results. Single-tier Guided-CGS shows better PER than HCGS for compression ratios up to 4×, but PER worsens substantially for larger compression ratios. This sharp increase in PER is observed for group Lasso, L1 and MP schemes as well, which can be attributed to the congestion of selected groups in small regions of weight matrices caused by the regularization function. The pre-determined random sparsity in HCGS ensures that congestion is avoided when selecting blocks within weight matrices, resulting in a much more graceful PER degradation for large

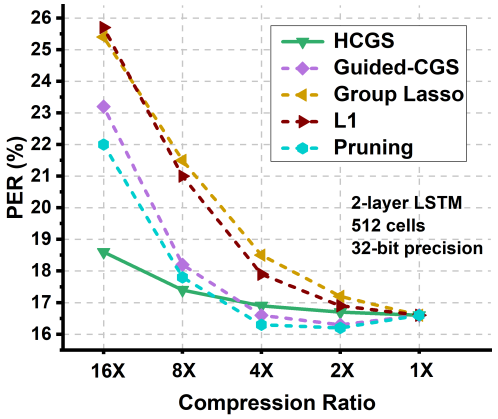

Figure 6: *PER (TIMIT) comparison between HCGS and learned sparsity methods.*

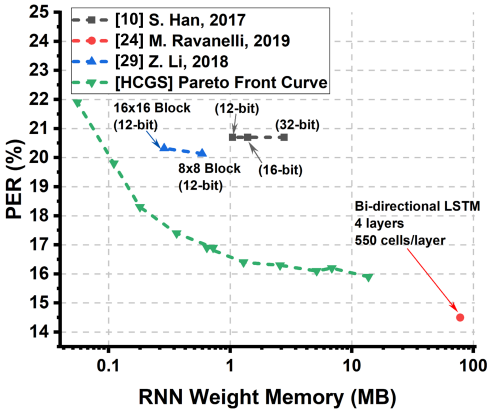

Figure 7: *PER and memory comparison with prior LSTM works.*

(>4×) compression ratios. The effectiveness of random pruning was also demonstrated in [28], where the pruned DNN recovered the accuracy loss by fine-tuning the remaining weights.

Figure 7 compares the total RNN weight memory and PER for prior LSTM works with structured compression [10, 29] and a baseline uncompressed LSTM [24], with the Pareto front curve obtained with the proposed HCGS-based LSTMs. It can be seen that the datapoints in the Pareto front of HCGS provide lower PER while requiring less storage for the LSTM weights.

4. Conclusion

In this paper, we present HCGS as a new training algorithm for LSTMs, targeting hierarchical structured sparsity and low-precision quantization. HCGS allows large compression (16×) of LSTM weights with graceful error rate degradation, while minimizing the index memory (∼1%). Experiments conducted on the TIMIT and TED-LIUM corpora demonstrated the effectiveness of HCGS across various LSTM RNNs. We also derived the Pareto front by jointly optimizing HCGS-based structured compression, low-precision quantization and the number of LSTM cells in RNNs. HCGS results in the best trade-off between accuracy and LSTM memory when compared to prior compression works and other learned sparsity methods for LSTMs.

5. Acknowledgements

This work was in part supported by NSF grant 1652866, Samsung, ONR, and C-BRIC, one of six centers in JUMP, a SRC program sponsored by DARPA.

6. References

[1] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke, "The Microsoft 2017 Conversational Speech Recognition System," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2018, pp. 5934–5938.

[2] K. J. Han, A. Chandrashekaran, J. Kim, and I. Lane, "The CAPIO 2017 conversational speech recognition system," *arXiv preprint arXiv:1801.00059*, 2018.

[3] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[4] M. Tu, V. Berisha, M. Woolf, J. Seo, and Y. Cao, "Ranking the parameters of deep neural networks using the fisher information," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 2647–2651.

[5] D. Kadetotad, S. Arunachalam, C. Chakrabarti, and J. Seo, "Efficient memory compression in deep neural networks using coarse-grain sparsification for speech applications," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016.

[6] S. Wang, Z. Li, C. Ding, B. Yuan, Q. Qiu, Y. Wang, and Y. Liang, "C-LSTM: Enabling efficient LSTM using structured compression techniques on FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2018, pp. 11–20.

[7] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2074–2082.

[8] F. Zhu, J. Pool, M. Andersch, J. Appleyard, and F. Xie, "Sparse persistent RNNs: Squeezing large recurrent networks on-chip," *arXiv preprint arXiv:1804.10223*, 2018.

[9] S. Narang, E. Elsen, G. Diamos, and S. Sengupta, "Exploring sparsity in recurrent neural networks," *arXiv preprint arXiv:1704.05119*, 2017.

[10] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang *et al.*, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017, pp. 75–84.

[11] W. Wen, Y. He, S. Rajbhandari, M. Zhang, W. Wang, F. Liu, B. Hu, Y. Chen, and H. Li, "Learning intrinsic sparse structures within long short-term memory," *arXiv preprint arXiv:1709.05027*, 2017.

[12] S. Gray, A. Radford, and D. P. Kingma, "Gpu kernels for block-sparse weights," *arXiv preprint arXiv:1711.09224*, 2017.

[13] S. Yin, G. Srivastava, S. K. Venkataramanaiah, C. Chakrabarti, V. Berisha, and J. Seo, "Minimizing area and energy of deep learning hardware design using collective low precision and structured compression," in *Proceedings of the 51st Asilomar Conference on Signals, Systems, and Computers*, 2017, pp. 1907–1911.

[14] D. Kadetotad, V. Berisha, C. Chakrabarti, and J. Seo, "A 8.93-TOPS/W LSTM recurrent neural network accelerator featuring hierarchical coarse-grain sparsity with all parameters stored on-chip," in *IEEE European Solid State Circuits Conference (ESSCIRC)*, 2019, pp. 119–122.

[15] S. Ye, T. Zhang, K. Zhang, J. Li, J. Xie, Y. Liang, S. Liu, X. Lin, and Y. Wang, "A unified framework of DNN weight pruning and weight clustering/quantization using ADMM," *arXiv preprint arXiv:1811.01907*, 2018.

[16] G. Srivastava, D. Kadetotad, S. Yin, V. Berisha, C. Chakrabarti, and J. Seo, "Joint optimization of quantization and structured sparsity for compressed deep neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2019, pp. 1393–1397.

[17] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations." *Journal of Machine Learning Research*, vol. 18, no. 187, pp. 1–30, 2017.

[18] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren, "The DARPA TIMIT acoustic-phonetic continuous speech corpus," *National Institute of Standards and Technology*, 1990.

[19] A. Rousseau, P. Deleglise, and Y. Esteve, "TED-LIUM: an automatic speech recognition dedicated corpus," in *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC)*, May 2012.

[20] S. Narang, E. Undersander, and G. Diamos, "Block-sparse recurrent neural networks," *arXiv preprint arXiv:1711.02782*, 2017.

[21] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[22] M. J. F. Gales, "Maximum likelihood linear transformations for HMM-based speech recognition," *Computer Speech & Language*, vol. 12, no. 2, pp. 75–98, 1998.

[23] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The Kaldi speech recognition toolkit," in *IEEE Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, 2011.

[24] M. Ravanelli, T. Parcollet, and Y. Bengio, "The PyTorch-Kaldi speech recognition toolkit," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 6465–6469.

[25] SCTK, "The NIST scoring toolkit," 2008. [Online]. Available: https://github.com/usnistgov/SCTK

[26] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.

[27] A. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr, "WRPN: Wide reduced-precision networks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[28] D. Mittal, S. Bhardwaj, M. M. Khapra, and B. Ravindran, "Recovering from random pruning: On the plasticity of deep convolutional neural networks," in *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2018, pp. 848–857.

[29] Z. Li, S. Wang, C. Ding, Q. Qiu, Y. Wang, and Y. Liang, "Efficient recurrent neural networks using structured matrices in FPGAs," in *Proceedings of the International Conference on Learning Representations (ICLR), Workshop Track*, 2018.