On Building Efficient and Robust Neural Network Designs

Xiaoxuan Yang
Duke University
Durham, NC, USA
xy92@duke.edu

Huanrui Yang
Duke University
Durham, NC, USA
huanrui.yang@duke.edu

Jingchi Zhang
Duke University
Durham, NC, USA
jingchi.zhang@duke.edu

Hai Helen Li
Duke University
Durham, NC, USA
hai.li@duke.edu

Yiran Chen
Duke University
Durham, NC, USA
yiran.chen@duke.edu

2022 56th Asilomar Conference on Signals, Systems, and Computers | 978-1-6654-5906-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/IEEECONF56349.2022.10051891

***Abstract*—Neural network models have demonstrated outstanding performance in a variety of applications, from image classification to natural language processing. However, deploying the models to hardware raises efficiency and reliability issues. From the efficiency perspective, the storage, computation, and communication cost of neural network processing is considerably large because the neural network models have a large number of parameters and operations. From the standpoint of robustness, the perturbation in hardware is unavoidable and thus the performance of neural networks can be degraded. As a result, this paper investigates effective learning and optimization approaches as well as advanced hardware designs in order to build efficient and robust neural network designs.**

***Index Terms*—efficiency, robustness, neural network, hardware-software co-design,**

I. Introduction

In recent years, deep neural network (DNN) models have shown beyond-human performance in multiple tasks. However, there exists some outstanding issues in the efficient application of DNNs in the real world. For instance, modern DNN architectures often contain millions of parameters and require billions of operations to process a single input, which hinders the deployment of these models on mobile and edge devices [1]. In addition to that, the quantization process will introduce information loss to input and weight data, and may bring a gap to the software-based accuracy and the hardware-deployed accuracy [2], [3]. Therefore, it is necessary to develop efficient and robust neural network designs that reduce the number of parameters and operations in neural networks and protect the inference accuracy under quantization.

In this paper, we review the technical background of network pruning and quantization methods and focus on the challenging problems on building efficient and robust neural network design. Specifically, we investigate the structured pruning method for the long short-term memory (LSTM) and provide unique solutions for this sequential model. Then, we explore the bit-level sparsity quantization method that can efficiently perform the mixed-precision quantization and achieve better compression rates and inference accuracy compared with state-of-the-art (SOTA) solutions. Furthermore, we develop a second-order regularization scheme to provide quantization robustness and achieve excellent performance even under low-bit settings.

II. Background and Related Work

To tackle the challenge of high memory consumption and computation cost, model compression techniques for neural network models, such as pruning and quantization, have been extensively studied. Moreover, the sparse gradient pruning methods are also proposed to accelerate the training process [4]–[6]. Other than the algorithm development, the specialized hardware designs to support the efficient neural networks are also proposed and analyzed [7]–[9]. In this section, we will go over the representative neural network pruning and quantization methods.

A. Neural Network Pruning Techniques

To achieve highly sparse neural networks, there are effective approaches targeted at unstructured pruning and structured pruning. Unstructured pruning can boost the sparsity level of the neural network without the concern of structural information. For instance, deep compression method can prune weights by setting a threshold and achieve final results on convergence with iterative pruning and retraining [10]. Other than threshold-based pruning methods, the regularizer can induce the model sparsity by implementing additional penalization on the weights with large absolute values. Han et al. investigate setting an element-wise ℓ_0 constraint by iterative pruning a fixed ratio of smallest weight elements [11]. However, this method is heuristic and can hardly achieve an optimal compression rate. This ℓ_0 regularizer is only an indicator of the sparsity level but doesn't reflect the gradient magnitude information. Then, the following work focuses on combining the ℓ_0 regularizer with sensitivity analysis [12] and adaptive optimization methods [13]. It is worth noting that ℓ_1 regularizer reflects the weight amplitude and can easily optimize through the network training process. Therefore, multiple approaches take advantage of ℓ_1 regularizer to achieve a high sparsity level. For instance, Liu et al. [14] directly apply ℓ_1 regularization to all the weights to improve the sparsity level. Furthermore, Wen et al. [15] propose structural sparsity learning (SSL) via group lasso, which injects an ℓ_1 regularzation over the weight ℓ_2 norm. Additionally, this structural sparsity is proven to be more hardware-efficient than unstructural sparsity [16], [17]. Yang et al. [18] propose a Hoyer-Square regularizer to boost the sparsity level.

B. Neural Network Quantization Approaches

Quantization techniques convert the floating-point parameters to fixed-point representations. Moreover, low-bit quantization can significantly reduce the memory consumption of the mode, and enable further acceleration on specialized hardware. However, directly quantizing all the weights to a lower precision may severely harm the performance of the neural network model. Polino et al. [19] observe that the weight of different layers may have various dynamic ranges, keeping the dynamic range of each layer is important for maintaining the performance of the model, especially after low-precision quantization. Straight-Through Estimator (STE) enables continuous gradient descent on the discrete values of fixed-point neural network model, which further improves the performance of model quantization [20]. However, model performance still degrades a lot when the quantization requirement for the whole model is set to low precision, such as three or two bits. Research work also demonstrates that the best tradeoff point between compression rate and accuracy can be achieved with mixed-precision quantization, where different layers of the neural network are quantized to different precision. The challenge lies in determining the quantization precision of each layer. Due to the large potential search space, previous methods propose to utilize neural architecture search (NAS) approach on determining the precision [21]. Another work proposes to rank each layer based on its importance and assign higher precision to a more important layer [22]. The exact precision of each layer, however, is still manually selected. It is hard to control the model size-performance tradeoff of the generated quantization scheme with these methods.

III. Structured Sparsity Exploration in Long Short-Term Memory

In this section, we present our study of Intrinsic Sparse Structures (ISS) Learning [23] and Efficient Sparsity Learning (ESS) [16] to regularize the structure and accelerate LSTM. We first introduce the computations within LSTM and cover the intrinsic challenges of LSTM sparsity regularization. Then, we propose novel methods and present evaluation results.

A common LSTM unit is composed of a cell ($\mathbf{c}$), an input gate ($\mathbf{i}$), an output gate ($\mathbf{o}$) and a forget gate ($\mathbf{f}$). The computation within LSTMs is as follows:

$$\begin{aligned}
\mathbf{i}_t &= \sigma\left(\mathbf{x}_t \cdot \mathbf{W}_{xi} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{hi} + \mathbf{b}_i\right) \\
\mathbf{f}_t &= \sigma\left(\mathbf{x}_t \cdot \mathbf{W}_{xf} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{hf} + \mathbf{b}_f\right) \\
\mathbf{o}_t &= \sigma\left(\mathbf{x}_t \cdot \mathbf{W}_{xo} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{ho} + \mathbf{b}_o\right) \\
\mathbf{u}_t &= tanh\left(\mathbf{x}_t \cdot \mathbf{W}_{xu} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{hu} + \mathbf{b}_u\right) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{u}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot tanh\left(\mathbf{c}_t\right)
\end{aligned} \tag{1}$$

where $\odot$ denotes Hadamard product (element-wise multiplication), $\sigma(\cdot)$ denotes sigmoid function, and $tanh(\cdot)$ denotes hyperbolic tangent function. $\mathbf{W}$s denotes weight matrices, which transform the concatenation (of hidden states $\mathbf{h}_{t-1}$ and inputs $\mathbf{x}_t$) to input updates $\mathbf{u}_t$ and gates ($\mathbf{i}_t$, $\mathbf{f}_t$ and $\mathbf{o}_t$). Due to the element-wise computation in LSTM, the involved

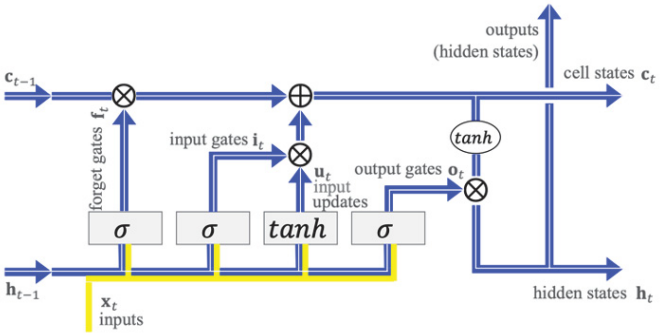

Fig. 1. Intrinsic Sparse Structures (ISS) in LSTMs that maintain the dimension consistency.

vectors should have the same dimension, as shown in the blue bands in Fig. 1. Therefore, the intrinsic challenge of LSTM is to simultaneously reduce the size of the basic structures of LSTM, including input updates, gates, hidden states, cell states, and outputs. For instance, given the circumstance that one selected element of a hidden state is pruned, we track the weight matrices of [$\mathbf{W}_{hi}$, $\mathbf{W}_{hf}$, $\mathbf{W}_{ho}$, $\mathbf{W}_{hu}$] and prune the selected rows. Then, we observe the generation of the hidden state and find that the selected dimensions of output gate and cell gate are also removable. This will result in the pruning of selected columns of weight matrices [$\mathbf{W}_{xi}$, $\mathbf{W}_{xf}$, $\mathbf{W}_{xo}$, $\mathbf{W}_{xu}$, $\mathbf{W}_{hi}$, $\mathbf{W}_{hf}$, $\mathbf{W}_{ho}$, $\mathbf{W}_{hu}$]. Moreover, if we consider the next layer where the hidden state is propagated to, the selected columns of weight matrices in the next layer can also be pruned. Therefore, these selected rows and columns are defined as ISS. The optimization goal is to increase the ISS without degrading the accuracy result.

Based on the analysis, we illustrate ISS that maintains the dimension consistency, as shown in Fig. 1. Suppose $\mathbf{w}_k^{(n)}$ is a vector of all weights in the k-th component of ISS in the n-th LSTM layer ($1 \le n \le N$ and $1 \le k \le K^{(n)}$), where N is the number of LSTM layers and $K^{(n)}$ is the number of ISS components (*i.e.*, hidden size) of the n-th LSTM layer. The regularization of ISS is constructed as follows:

$$R(\mathbf{w}) = \sum_{n=1}^{N} \sum_{k=1}^{K^{(n)}} \left|\left|\mathbf{w}_k^{(n)}\right|\right|_2, \tag{2}$$

where $\mathbf{w}$ denotes the vector of all weights and $||\cdot||_2$ is ℓ_2-norm (*i.e.*, Euclidean length). We develop our learning alogithm based on the Stochastic Gradient Descent (SGD) training, the update rule for each ISS weight group is

$$\mathbf{w}_k^{(n)} \leftarrow \mathbf{w}_k^{(n)} - \eta \cdot \left(\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}_k^{(n)}} + \lambda \cdot \frac{\mathbf{w}_k^{(n)}}{\left|\left|\mathbf{w}_k^{(n)}\right|\right|_2} \right), \tag{3}$$

where $E(\mathbf{w})$ is data loss, η is learning rate and $\lambda > 0$ is the coefficient of group Lasso regularization to trade off accuracy result and ISS sparsity.

The experimental results are evaluated with an RNN with two stacked LSTM layers and shown in Table I. Note that

TABLE I
LEARNING ISS SPARSITY FROM SCRATCH IN STACKED LSTMS.

Method	Dropout keep ratio	Perplexity (validate, test)	ISS # in (1st , 2nd) LSTM	Weight #	Total time	Speedup	Mult-add reduction
baseline	0.35	(82.57, 78.57)	(1500, 1500)	66.0M	157.0ms	1.00×	1.00×
ISS	0.60	(82.59, 78.65)	(373, 315)	21.8M	14.82ms	10.59×	7.48×
		(80.24, 76.03)	(381, 535)	25.2M	22.11ms	7.10×	5.01×
direct design	0.55	(90.31, 85.66)	(373, 315)	21.8M	14.82ms	10.59×	7.48×

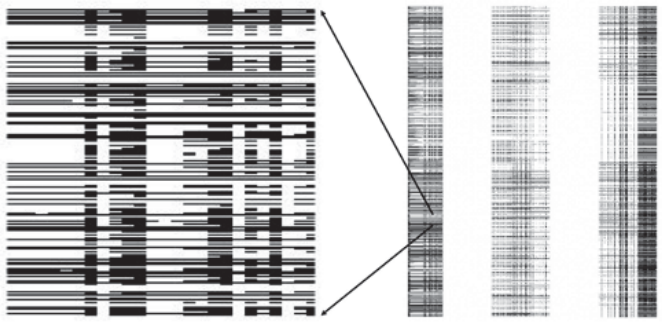

Fig. 2. The visualization of weight matrices in LSTM units learned by efficient structural sparsity (ESS).

the proposed ISS can avoid overfitting and thus the dropout keep ratio for ISS is larger than the baseline setting. With a weight size of 21.8M, the proposed method can achieve $10.59\times$ speedup and $7.48\times$ multi-add reduction. Other than that, the ISS can perform as structure regularization that achieves a smaller perplexity result with a faster and more compact model. Moreover, we compare our design with the direct design and show that ISS provides the best tradeoff between perplexity results and speedup.

Furthermore, ESS is developed to deal with real-world acoustic tasks and improve the computing efficiency [16]. The training process consists of a dedicated three-step training pipeline. Firstly, the network is trained from scratch with a structured-sparsity learning method until a sparse model at the desired sparsity level is generated. At this stage, the accuracy of the model may be relatively low. Therefore, the next two steps are proposed to mitigate the accuracy degradation. The second step fixes the zero parameters and protects the structured sparsity. In the third step, the SGD method is utilized to update the nonzero elements in the network.

In the evaluation, we use the open-source Kaldi toolkit and implement the network in Intel Gaussian Neural Accelerator (GNA). As shown in Fig. 2, ESS enables significant sparsity improvement, such as 72.5% pruning in weight groups.

IV. BIT LEVEL SPARSITY QUANTIZATION

Fixed point quantization is an important model compression technique for neural network models. To further reduce the bit level, finer-granularity quantization is preferred. The key problem of mix-precision quantization lies in determining the optimal mixed-precision quantization scheme. The whole design space for the quantization scheme is typically huge and discrete. Existing work either uses costly neural architecture search methods to explore the design space or utilizes a ranking of the importance of all layers and assign the precision manually, which may not be optimal [21], [22]. We aim to propose a differentiable regularizer to induce a mixed-precision quantization scheme [24].

For bit representation, we follow the dynamic scaling quantization procedure, where we utilize the element with the largest absolute value in each layer as the scaling factor and uniformly quantize all the elements to n bits and represent the quantized values in binary form. We consider scaling factor s and each bit in W_s as independent trainable variables.

Specifically, the STE for the bit representation training is defined as:

$$\begin{aligned} &\textbf{Forward: } W_q = \frac{1}{2^n-1} Round\left[\sum_{b=0}^{n-1} W_s^{(b)} 2^b\right]; \\ &\textbf{Backward: } \frac{\partial \mathcal{L}}{\partial W_s^{(b)}} = \frac{2^b}{2^n-1}\frac{\partial \mathcal{L}}{\partial W_q}. \end{aligned} \tag{4}$$

To enable gradient-based training of the bit representation, we allow each bit to take a continuous value between 0 and 2. The STE is then applied where it estimates the actual quantized value W_q of the weight to compute loss and gradient. As the gradient is back propagated to W_q, it will be passed directly to each bit of W_s.

The bit-level sparse quantization (BSQ) regularizer is defined as:

$$B_{GL}(W^g) = \sum_{b=0}^{n-1} \left\| \left[W_p^{(b)}; W_n^{(b)} \right] \right\|_2, \tag{5}$$

where $W_p^{(b)}$ and $W_n^{(b)}$ are bit representations converted from W^g, and $[\cdot;\cdot]$ denotes the concatenation of matrices. B_{GL} could guarantee a certain bit b of all elements in both $W_p^{(b)}$ and $W_n^{(b)}$ to be zero simultaneously.

We show the results of BSQ in Table II and prove that our method can figure out the desired mixed-precision quantization scheme and provide a model with higher performance under the same quantization scheme.

TABLE II
ACCURACY-#BITS TRADEOFF UNDER DIFFERENT REGULARIZATION STRENGTHS.

Strength α	3e-3	5e-3	7e-3	1e-2	2e-2
#Bits per Para / Comp ($\times$)	3.02 / 10.60	2.25 / 14.24	1.66 / 19.24	1.37 / 23.44	0.87 / 36.63
BSQ acc before / after FT (%)	91.30 / 92.60	90.98 / 92.32	90.42 / 91.48	90.35 / 91.16	85.77 / 89.49
Train from scratch acc (%)	91.72	91.45	91.12	89.57	89.14

V. ROBUST ALGORITHM TOWARDS QUANTIZATION ROBUSTNESS

For neural network models to be deployed into the real world, we expect the model to be generalizable to unseen data and to be flexible to potential compression required by the runtime environment. We observe that both model generalizability and performance under compression can be unified as performance under weight perturbation. Specifically, sharpness-aware minimization work notes that the generalization gap is bounded by ℓ_2 weight perturbation [25]. Moreover, common model compression techniques can also be understood as adding perturbation to the pre-trained weights. For instance, uniform fixed-precision quantization can be modeled as ℓ_∞ weight perturbation. In general, we can unify generalization and quantization performance as improving the model's robustness to ℓ_p norm bounded weight perturbation.

To improve such robustness, we first observe how the model is performing under weight perturbation. Here we introduce the idea of a perturbation lower bound, which is the minimal perturbation strength needed to induce a loss increase larger than our tolerance. In this setting, a larger perturbation lower bound indicates that the model is more robust against weight perturbation, which is our optimization goal. With additional analysis utilizing Taylor expansion, we derive the bounds for ℓ_2 and ℓ_∞ perturbation and find that a smaller Hessian eigenvalue can lead to better generalization performance and quantization robustness.

We propose Hessian-Enhanced Robust Optimization (HERO) [26] to regularize the hessian eigenvalue and improve the generalization ability and quantization robustness. We compute the Hessian norm with a finite difference approximation, which is formulated as the gradient difference regarding the original and perturbed weight. We simplify the gradient computation of the Hessian regularization by only keeping the gradient terms related to the perturbed weight. For the overall gradient optimization step, we utilize the perturbed weight gradient from SAM, which is the first term in the gradient formulation, SAM term behaves as a first-order regularization to the optimization process for better stability in the training.

For a neural network model, L_r needs to be computed on the weight tensors from all the layers, each having distinct dimensions and gradient value ranges. To accommodate the diversity among layers, we propose to compute L_r layer by layer and scale the ℓ_2 norm of the perturbation $\boldsymbol{z}$ to match the weight value range in each layer. Specifically, for layer i, we have

$$L_r^i(\boldsymbol{W}^i) = ||\nabla L(\boldsymbol{W}^i + h\boldsymbol{z}^i) - \nabla L(\boldsymbol{W}^i)||^2, \quad \boldsymbol{z}^i = \frac{\boldsymbol{W}^{i^2}}{||\boldsymbol{W}^i||_2} \frac{\nabla L(\boldsymbol{W}^i)}{||\nabla L(\boldsymbol{W}^i)||_2}. \tag{6}$$

The overall Hessian regularization is then computed as $L_r(\boldsymbol{W}) = \sum_{i=1}^N L_r^i(\boldsymbol{W}^i)$, summing over all the N layers in the model. We derive the gradient of our Hessian-enhanced robust optimization as

$$\nabla_{\boldsymbol{W}^i} = \nabla_{(\boldsymbol{W}^i + h\boldsymbol{z}^i)} L(\boldsymbol{W}^i + h\boldsymbol{z}^i) + \alpha \boldsymbol{W} + \gamma \sum_{i=1}^N \nabla_{(\boldsymbol{W}^i + h\boldsymbol{z}^i)} G(\boldsymbol{W}^i + h\boldsymbol{z}^i), \tag{7}$$

where $\alpha > 0$ denotes the weight decay and $\gamma > 0$ denotes the regularization strength of the Hessian regularization. Besides, we summarize the HERO algorithm as shown in Algorithm 1.

Algorithm 1 Hessian-Enhanced Robust Optimization (HERO)

Randomly initialize model weights $\boldsymbol{W}_0^i$ for all layer i;
Set total step T, perturbation strength h, learning rate η;
Set weight decay α and Hessian regularization strength γ;
for $t = 0, \ldots, T$ **do**
  Sample batch $\mathcal{B}$ from training set;
  Compute batch loss's gradient $g^i = \nabla L_{\mathcal{B}}(\boldsymbol{W}_t^i)$;
  Compute weight perturbation $\boldsymbol{z}^i$ with g^i per Equation (6);
  Weight perturbation $\boldsymbol{W}^{i*} = \boldsymbol{W}^i + h\boldsymbol{z}^i$;
  Compute perturbed gradient $\nabla L_{\mathcal{B}}(\boldsymbol{W}^{i*})$
  Hessian regularization $G(\boldsymbol{W}^{i*}) = ||\nabla L_{\mathcal{B}}(\boldsymbol{W}^{i*}) - g^i||^2$
  Compute HERO gradient $\nabla_{\boldsymbol{W}^i}$ per Equation (7);
  Weight update $\boldsymbol{W}_{t+1}^i = \boldsymbol{W}_t^i - \eta \nabla_{\boldsymbol{W}^i}$
return $\boldsymbol{W}_T^i$ for all layer i

To verify our theoretical analysis, we perform neural network training with the proposed HERO optimizer. In Fig. 3, we compare the SGD and first-order gradient regularization, and it shows that HERO reaches a smaller Hessian norm towards the end of training, thus showing a smaller generalization gap. As shown in Table III, we provide the quantization results and full-precision results on three representative methods. It shows that adding the Hessian regularization leads to better testing accuracy and higher robustness against post-training

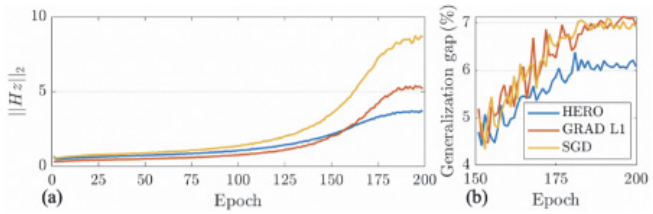

Fig. 3. Hessian norm and generalization gap evolution through the training with HERO, GRAD L1, and SGD.

TABLE III
ABLATION STUDY ON HERO, FIRST-ORDER ONLY, AND SGD GRADIENT UPDATE RULE. RESULTS REPORTED WITH MOBILENETV2 NETWORK ON THE CIFAR-10 DATASET.

Quantization (bit)	4	6	8	Full
HERO	**93.45%**	**94.90%**	**95.03%**	**95.03%**
First-order only	91.61%	93.92%	94.00%	94.06%
SGD	85.88%	91.81%	92.33%	92.45%

quantization, showing the necessity of the Hessian enhancement. Moreover, the 4-bit quantization result shows that the HERO method can preserve the quantization robustness even under a relatively low precision setting.

VI. CONCLUSION

This paper covers structured sparsity exploration in long short-term memory and includes the ISS and ESS methods. Moreover, we discuss bit-level quantization work BSQ for more advanced compression rates and inference accuracy compared with SOTA designs. Furthermore, we build generalized robust algorithms HERO, which can protect the neural network against weight perturbation. For future exploration, we believe that execution acceleration and design flexibility enhancement will significantly benefit the efficient and robust neural network designs.

ACKNOWLEDGMENT

This work is supported in part by NSF 2112562, NSF 1955246, and ARO W911NF-19-2-0107.

REFERENCES

[1] F. Chen, W. Wen, L. Song, J. Zhang, H. H. Li, and Y. Chen, "How to obtain and run light and efficient deep learning networks," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–5.

[2] X. Yang, S. Belakaria, B. K. Joardar, H. Yang, J. R. Doppa, P. P. Pande, K. Chakrabarty, and H. H. Li, "Multi-objective optimization of reram crossbars for robust dnn inferencing under stochastic noise," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[3] X. Yang, C. Wu, M. Li, and Y. Chen, "Tolerating noise effects in processing-in-memory systems for neural networks: A hardware–software codesign perspective," *Advanced Intelligent Systems*, p. 2200029, 2022.

[4] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," *Advances in neural information processing systems*, vol. 30, 2017.

[5] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *arXiv preprint arXiv:1712.01887*, 2017.

[6] X. Yang, H. Yang, J. R. Doppa, P. P. Pande, K. Chakrabarty, and H. Li, "Essence: Exploiting structured stochastic gradient pruning for endurance-aware reram-based in-memory training systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[7] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[8] S. Li, E. Hanson, X. Qian, H. H. Li, and Y. Chen, "Escalate: Boosting the efficiency of sparse cnn accelerator with kernel decomposition," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 992–1004.

[9] E. Hanson, S. Li, H. Li, and Y. Chen, "Cascading structured pruning: Enabling high data reuse for sparse dnn accelerators," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: ACM, 2022, p. 522–535.

[10] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[11] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[12] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through l_0 regularization," *arXiv preprint arXiv:1712.01312*, 2017.

[13] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic dnn weight pruning framework using alternating direction method of multipliers," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[14] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 806–814.

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

[16] J. Zhang, W. Wen, M. Deisher, H.-P. Cheng, H. Li, and Y. Chen, "Learning efficient sparse structures in speech recognition," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 2717–2721.

[17] J. Zhang, J. Huang, M. Deisher, H. Li, and Y. Chen, "Structural sparsification for far-field speaker recognition with gna," *arXiv preprint arXiv:1910.11488*, 2019.

[18] H. Yang, W. Wen, and H. Li, "Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures," *arXiv preprint arXiv:1908.09979*, 2019.

[19] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," *arXiv preprint arXiv:1802.05668*, 2018.

[20] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 365–382.

[21] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8612–8620.

[22] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Hawq: Hessian aware quantization of neural networks with mixed-precision," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 293–302.

[23] 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.

[24] H. Yang, L. Duan, Y. Chen, and H. Li, "Bsq: Exploring bit-level sparsity for mixed-precision neural network quantization," *arXiv preprint arXiv:2102.10462*, 2021.

[25] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, "Sharpness-aware minimization for efficiently improving generalization," *arXiv preprint arXiv:2010.01412*, 2020.

[26] H. Yang, X. Yang, N. Z. Gong, and Y. Chen, "Hero: Hessian-enhanced robust optimization for unifying and improving generalization and quantization performance," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 25–30.