Cloud Computing Final Report

1 Introduction

Large-scale deep neural networks (DNNs) have achieved great success in various fields, such as computer vision (CV), natural language processing (NLP), speech recognition and so on. With the yearning for deep learning democratization (Garvey, 2018), there are increasing demands to deploy DNNs on resource-constrained devices (i.e., FPGA, GPU, etc.). However, there are many challenges to accommodate DNN models on hardware, such as limited storage, limited computational speed and high demand of real-time inference. Based on these challenges, model compression has been investigated in recent years. The core idea of model compression is to generate a much sparse model thus we could get acceleration in computation and reduction on space. Several model compression techniques were proposed to find a sparse network on DNNs to meet the demands, such as weight pruning (Han et al., 2015a), knowledge distillation (Hinton et al., 2015), quantization (Zhang et al., 2020), low rank approximation (Yu et al., 2017), sparsity regularization (Louizos et al., 2017), etc.

Among all the compression techniques referred above, weight pruning (Zhang et al., 2018a) is one of the most simple but popular one and is widely explored in recent years. The technique is implemented by zero out certain weights and then optimize the rest, which is a destruction plus learning process. From the aspect of hardware friendly or not, it can be divided into two categories which are structured pruning (hardware friendly) (Anwar et al., 2017; Wang et al., 2019) and unstructured pruning (hardware unfriendly). For structured pruning, we will explore pruning methods such as row pruning, column pruning and whole block pruning while for the unstructured pruning, we will conduct experiment on irregular pruning. In this work, we will demonstrate the effectiveness of weight pruning in the fields of both CV and NLP.

2 Background

Advances in Machine Learning have enabled high accuracy in classifications, recommendations, natural language processing, etc. The success of modern deep neural networks is mainly dependent on the availability of advanced computing power and a large number of data.(Wang et al., 2021) Technically, a large enough neural network and dataset could result in a machine that can do anything a human brain is capable of. However, a high-performance neural network usually means complicated and has a huge amount of parameters to update. The enormous size of a network not only causes great challenges on hardware devices but also wastes unnecessary computation and time consumption. A simple way to avoid such wastes is to prune the parameters in the network that have slight impacts.

2.1 Deep Neural Network

A deep neural network (DNN) is a framework for deep learning, which is a neural network with at least one hidden layer. Similar to shallow neural networks, deep neural networks can also provide modeling for complex nonlinear systems, but the extra levels provide a higher level of abstraction for the model, thus improving the capabilities of the model(Chen et al., 2019).

The deep neural network is a discriminative model that can be trained using a backpropagation algorithm. The weight update can be solved by the stochastic gradient descent method using the following formula(Chen et al., 2019):

$$\Delta\omega_{ij}(t+1) = \Delta\omega_{ij}(t) + \eta\frac{\partial C}{\partial\omega_{ij}} \qquad (1)$$

Among them, η is the learning rate, and C represents the cost function. The choice of this function is related to the type of learning (such as supervised learning, unsupervised learning, and reinforcement learning) and the activation function.

DNN is currently the basis of many artificial intelligence applications. Due to the breakthrough applications of DNN in speech recognition and image recognition, the number of applications using DNN has exploded. These DNNs are deployed in various applications ranging from self-driving cars, cancer detection to complex games. In these many fields, DNN can surpass human accuracy. The outstanding performance of DNN comes from its ability to use statistical learning methods to extract high-level features from raw sensory data, and to obtain an effective representation of the input space from a large amount of data. This is different from previous methods that used manual feature extraction or expert design rules.

However, the price of DNN to obtain superior accuracy is high computational complexity cost. Although general-purpose computing engines (especially GPUs) have become the mainstay of DNN processing, it is becoming increasingly popular to provide specific acceleration methods for DNN computing.

According to different applications, the shape and size of deep neural networks are also different. Popular shapes and sizes are evolving rapidly to improve model accuracy and efficiency. The input of all deep neural networks is a set of information values that characterize the network to be analyzed and processed. These values can be the pixels of a picture, or the sample amplitude of a piece of audio, or a digital representation of a system or game state.

There are two main forms of input processing networks: feedforward and loop. In the feedforward network, all calculations are a series of operations based on the output of the previous layer. The final set of operations is the output of the network. In this type of deep neural network, the network has no memory, and the output is always independent of the input order of the previous network. In contrast, recurrent networks (LSTM is a popular variant) have internal memory, allowing long-term dependencies to affect the output. In these networks, the state values of some intermediate operations are stored in the network and are also used as inputs for other operations related to processing the latter input.

2.2 Transformer-based Language Model

Transformer is a sequence-to-sequence NLP model (Vaswani et al., 2017) which uses an attention mechanism to draw global dependencies between input and output.

It takes a sequence of word embeddings from one vocabulary set as input and generates the probability of tokens in the other vocabulary set. The model mainly consists of encoding and decoding components. The encoding component is a stack of encoders that are all identical in structure but their weights are trained independently.

Likewise, the decoding component is also a stack of decoders. Note, the number of encoders and decoders can be adjusted to arrive at different transformer models. For instance, BERT only contains encoders (Lan et al., 2019). BERT$_{BASE}$ has 12 layers in the encoder stack while BERT$_{LARGE}$ has 24 layers in the encoder stack. OpenAI GPT-3 (Brown et al., 2020) only has 12 layers of decoders.

Before the word embedding is processed by the encoder, we add the position information of the tokens to the sequence so that the model is aware of the position of the token in each sequence. We use sine and cosine functions to encode this position information (Vaswani et al., 2017):

$$\mathbf{PE}_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}), \quad (2)$$

$$\mathbf{PE}_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}), \quad (3)$$

where pos is the position of the token in the sequence; d_{model} is the dimension of each word's embedding, and i is the i-th dimension in the word embedding vector. These positional values are added to the embedding of each token.

Self-attention is the key for an encoder. During linear transformation, the input $\mathbf{X}$ is multiplied by three weight matrices, i.e., query ($\mathbf{W_Q}$), key ($\mathbf{W_K}$), and value ($\mathbf{W_V}$), to arrive at $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$, respectively. That is $\mathbf{Q} = \mathbf{X} \cdot \mathbf{W}_\mathbf{Q}^\mathrm{T}$, $\mathbf{K} = \mathbf{X} \cdot \mathbf{W}_\mathbf{K}^\mathrm{T}$ and $\mathbf{V} = \mathbf{X} \cdot \mathbf{W}_\mathbf{V}^\mathrm{T}$. Afterwards, we follow Equation 4 to perform attention computation:

$$\mathbf{Z} = Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax(\frac{\mathbf{Q} \cdot \mathbf{K}^\mathrm{T}}{\sqrt{d_k}}) \cdot \mathbf{V}. \quad (4)$$

Note, a mask is applied to exclude certain word-to-word interactions before softmax. One popular masking mechanism is to set the lower triangle part of the masking as 0 and the upper triangle part as negative infinity. When applied to $\mathbf{Q} \cdot \mathbf{K}^\mathrm{T}$, this mask actually prevents the position information of later words from affecting the earlier ones. Multi-head attention extends Equation 4. The attention model possesses multiple sets of $\mathbf{W_Q}$, $\mathbf{W_K}$, $\mathbf{W_V}$, which

allows the model to jointly attend to information from different representation subspaces at different positions and each set of weights is a head. Since there are multiple $\mathbf{Z}$'s of Equation 4, multi-head attention combines them by multiplying $\mathbf{Z}$ with a weight matrix $\mathbf{W_O}$. $\sqrt{d_k}$ is the dot product's scaling factor, where $d_k = \frac{d_{model}}{\mathrm{H}}$, and H is the number of heads.

After the self-attention module, there is a multi-layer perceptron (MLP) module consisting of two linear transformation layers with an activation layer between them. A layer normalization is applied after self-attention and MLP respectively, whose input will be added by the input of the module.

2.3 Network Pruning

As described in the background section, network pruning is an intuitive yet very effective strategy to reduce network size, thereby reduces the computation and time cost. Since there are countless researchers taking great effort on this, many pruning methods have been provided so far. For example, structure pruning, channel pruning, irregular pruning, and so on. Each algorithm has shown great performance. However, pruning usually causes a trade-off with an accuracy drop. Thus, our work becomes dealing with such trade-offs, more intuitively, pruning the network as much as possible and maintain the accuracy at the same time.

In our project, we have been working on a pruning method called pattern pruning. Pattern pruning prunes the whole row or column of the weight matrices. Therefore, such a pruning method benefits hardware even more. Nevertheless, pattern pruning also faces an accuracy drop as mentioned above. We also implement the ADMM algorithm to overcome the shortcoming.

ADMM is short for the alternating direction method of multipliers. The method is a regularized optimization during the DNN training, which can exactly enforce certain patterns in DNN weights (Yuan et al., 2021).

To be more explicit, we first generate masks with 0 and 1 in certain patterns. Then we apply masks to the weight matrix by multiplying them elementwise. Then the masks are also applied on the gradient matrix in the same way in each training step, thus the pruned weights do not update. The process is called hard-prune. One thing that's worth noting is that the algorithm also allows each weight matrix to select the best pattern from a pattern set according to its L2 norm.

3 Experiment

In this section, we make a description of our experiment process on both computer vision model pruning and Transformer-based language model pruning.

3.1 Computer Vision Model Pruning

Reading and writing weights in memory is an energy-consuming and time-consuming process. Approximately, reading $1.2MB$ data requires $1second$ and $1.47mj$ energy. Writing $1.3MB$ data requires $1second$ and $1.51mj$ energy. It is more frequent while operating in energy-aware inference models since the models are needed to be switched based on the energy level of the sensing devices. Therefore, technically, if the models of different sparsities have shared weights, the erasing and rewriting of the shared weights can be safely skipped, thus, the switching process could be more energy efficient.

Normally, model compressing process contains a retrain step to guarantee the mask is applied on the weight matrix. However, the retraining process naturally changes the weights remained while applying masks, thus shared weights by implementing conventional weight pruning strategy is not possible. Alternatively, we simplified the process through removing the retraining step. By sacrificing an acceptable accuracy drop, the shared weights among models of different sparsities become possible.

In our shared-weights training (for demonstration, we use three models), the weights of the pruned layer are:

- $HighSparsity: W_h = W_{Initial} \times M_h$
- $MediumSparsity: W_m = W_{Initial} \times M_{hm} + W_h$
- $LowSparsity: W_l = W_{Initial} \times M_{hm} + W_m$

In the equations shown above, W denotes the weight matrix, M represents the mask. The $footnote$ stands for the percentage of sparsity. h, m, l and hm stand for $high$, $medium$, low and $slightly\ higher\ than\ medium$ sparsity, respectively. It will be illustrated further in the Fig. 1, in which the shared-weights training workflow is illustrated.

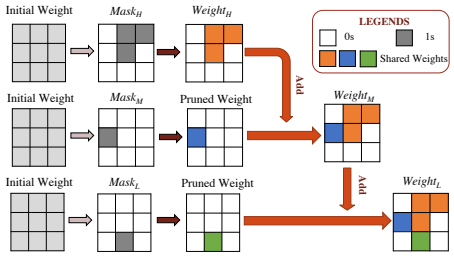

Figure 1: Shared-Weights Training Workflow

The white cells in the figure represent 0s. The grey cells represent 1s. Orange, green and blue cells are remained weights after pruning. The cells illustrated in the same color stand for shared weights.

The algorithm is simple yet effective, however, additional indices are required to store the positions of pruned weights. To compensate the memory spent on storing indices, pruning ratios have to be larger than a certain threshold. For the implementation of the LeNet5 model on CPU based devices, the sparsity needs to be no less than 4% to compensate the storage of indices, and the threshold is 20% when implementing on LEA based devices.

3.2 Transformer-based Language Model Pruning

In this work, various weight pruning methods, such as irregular pruning, column pruning, row pruning and whole block pruning will be explored on BERT$_{BASE}$ to demonstrate the effectiveness of weight pruning in shrinking model size, accelerating model inference time while retaining model performance.

We mainly focus on the backbone pruning of BERT$_{BASE}$, which means we prune the 6 weight matrices (i.e., query, key, value, attention.output.dense, intermediate.dense, output.dense) on 12 encoders. We follow the identical layer rule for pruning ratio, which implies we apply the same pruning ratio for each layer on each encoder. For each pruning method, we first finetune the pre-trained model on corresponding tasks. Then, we prune the model with the pruning method. Finally , we retrain the model until it's converged.

For irregular pruning, we eliminate the least absolute value of weights (Han et al., 2015b) after finetuning. For column pruning, we calculated the l2 norm of each column of a specific weight and fill the least l2 norm value corresponding column values with zeros. Row pruning and whole block pruning follow the same way with different pattern.

3.2.1 Dataset

We conduct experiments on GLUE benchmark (Wang et al., 2018), a comprehensive collection of natural language understanding tasks covering three NLP categories, i.e., inference tasks (MNLI (Williams et al., 2018), Quora Question Pairs (QQP) (Zhang et al., 2018b), QNLI (Wang et al., 2018) (a set of over 100,000+ question-answer pairs from SQuAD (Rajpurkar et al., 2016)), single-sentence (SST-2 (Socher et al., 2013)), paraphrase similarity matching (STS-B (Cer et al., 2017), Microsoft Research Paraphrase Corpus (MRPC (Dolan and Brockett, 2005)).

3.2.2 Baseline Models

The baseline model is our own fine-tuned unpruned BERT$_{\text{BASE}}$ (Devlin et al., 2019). We report our results (from the offical bert-base-uncased) as Full **BERT**$_{\text{BASE}}$. We use Huggingface Transformer toolkit (Wolf et al., 2019) to conduct our experiment. There are 12 layers (L =12; hidden size H = 768; self-attention heads A = 12), with 110 million parameters.

3.2.3 Evaluation Metrics

We apply the same metrics of the tasks as the GLUE paper (Wang et al., 2018), i.e., accuracy scores are reported for RTE and QNLI; F1 scores are reported for MRPC; Spearman correlations are reported for STS-B.

3.2.4 Platforms

We use Python 3.6.10 with PyTorch 1.4.0 and CUDA 11.1 on Quadro RTX6000 GPU and Intel(R) Xeon(R) Gold 6244 @ 3.60GHz CPU.

3.2.5 Experimental Results

- CV pruning: Table 2 shows the accuracy of different sparsities after shared-weights training. We can tell from the table that shared-weights training can help retain weights among different sparsities to have them shared with a comparable accuracy. The patterns applied on the weight matrix are selected manually according to the rule proposed in the PCONV paper(Ma and et.al., 2020) and the highest accuracy is given. As for $LeNet5$, the high sparsity is 60.24% and the corresponding accuracy is 98.94%. The medium Sparsity is 20.093% and its accuracy is 98.96%. The low sparsity is 8.47% and its accuracy is

Models	MNLI	QQP	QNLI	SST-2	STS-B	MRPC	RTE
BERT$_{\text{BASE}}$ (Devlin et al., 2018)	84.6	91.2	90.5	93.5	85.8	88.9	66.4
BERT$_{\text{BASE}}$ (ours)	83.9	91.4	91.1	92.7	85.8	89.8	66.4
BERT$_{\text{BASE}}$ irregular prune (ours)	81.1	86.3	88.9	89.3	83.1	89.3	66.1
BERT$_{\text{BASE}}$ row prune (ours)	78.1	86.8	88.1	85.5	78.7	77.7	53.2
BERT$_{\text{BASE}}$ column prune (ours)	78.5	87	88.3	85.9	78.7	77.8	53.8
BERT$_{\text{BASE}}$ whole block prune (ours)	78.8	87.3	86.2	87.3	85.3	79.2	53.1
Compression rate	1.428×	1.428×	1.428×	1.428×	1.428×	1.428×	1.428×

Table 1: Comparison of test accuracy using BERT$_{\text{BASE}}$ among the nine GLUE benchmark tasks.

Dataset/Task		MNIST (Image Classification)			HAR (Human Activity Recognition)			OKG (Speech Recognition)			ImageNet (Image Classification)		
Models		LeNet-5 (LeCun et al., 1998)			HAR-Net(Anguita and et.al., 2013)			OKG-Net(Warden, 2018)			SqNxt-23 (Gholami and et. al., 2018)		
		M1	M2	M3	M1	M2	M3	M1	M2	M3	M1	M2	M3
SW-Train	Sparsity	60.24%	40.50%	20.70%	24.13%	5.31%	2.69%	83.76%	56.32%	30.78%	55.39%	54.30%	54.07%
	Accuracy	98.94%	98.96%	99.02%	90.19%	89.78%	87.21%	82.01%	83.43%	83.5%	73.80%	75.02%	77.36%
	Latency (s)	1.1	1.3	1.5	0.66	0.83	0.85	0.82	1.3	1.62	3.4	3.6	3.7

Table 2: Results of Shared Weight Training

99.02%. Part of the remaining weights are shared, as illustrated in Figure 1. Similarly in the $SqNxt-23$ model, the accuracy slightly increases as the sparsity reduces. Apart from image classification tasks, our shared-weights training shows satisfying accuracy on recognition tasks as well. For example, as for the $HAR-Net$ model, the high, medium and low sparsities are 24.13%, 5.31% and 2.69%, respectively and their corresponding accuracy are 90.19%, 89.78% and 87.21%. Comparably, the accuracy and sparsity of the OKG-Net show the same trend as $HAR-Net$. Thus, with sacrificing little accuracy drop, the shared-weights training algorithm could keep weights unchanged, and further benefit the inference process on energy harvesting devices.

4 Conclusion

In order to tackle the problem of accommodating DNN models on hardware, pruning is a key point. This can usually causes a trade-off with an accuracy drop. In this project, we focus on dealing with such trade-offs, more intuitively, pruning the network and maintain the accuracy at the same time.The experiments are based on the computer vision model pruning and Transformer-based language model pruning. As for the results, we can make a conclusion that pruning strategy has benefits on multiple-tasks in both hardware and software level, by accelerating inference process and saving computation costs.

References

Davide Anguita and et.al. 2013. A public domain dataset for human activity recognition using smartphones. In *Esann*, volume 3, page 3.

Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2017. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):1–18.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, Virtual. Curran Associates, Inc.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada. The Association for Computer Linguistics.

Xue Chen, Lanyong Zhang, Tong Liu, and MM Kamruzzaman. 2019. Research on deep learning in the field of mechanical equipment fault diagnosis image quality. *Journal of Visual Communication and Image Representation*, 62:402–409.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jacob Devlin et al. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*.

Bill Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Third International Workshop on Paraphrasing*. Asia Federation of Natural Language Processing.

Colin Garvey. 2018. A framework for evaluating barriers to the democratization of artificial intelligence. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Amir Gholami and et. al. 2018. Squeezenext: Hardware-aware neural network design. In *CVPR*, pages 1638–1647.

Song Han, Huizi Mao, and William J Dally. 2015a. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Song Han et al. 2015b. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Christos Louizos, Max Welling, and Diederik P Kingma. 2017. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*.

Xiaolong Ma and et.al. 2020. Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices. *ArXiv*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, Red Hook, NY, USA. Curran Associates Inc.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Yijue Wang, Chenghong Wang, Zigeng Wang, Shanglin Zhou, Hang Liu, Jinbo Bi, Caiwen Ding, and Sanguthevar Rajasekaran. 2021. Against membership inference attack: Pruning is all you need.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. 2019. Structured pruning of large language models. *arXiv preprint arXiv:1910.04732*.

Pete Warden. 2018. Speech commands: A dataset for limited-vocabulary speech recognition. *ArXiv*, abs/1804.03209.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Thomas Wolf et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv, abs/1910.03771*.

Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. 2017. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7370–7379.

Geng Yuan, Payman Behnam, Zhengang Li, Ali Shafiee, Sheng Lin, Xiaolong Ma, Hang Liu, Xuehai Qian, Mahdi Nazm Bojnordi, Yanzhi Wang, and Caiwen Ding. 2021. Forms: Fine-grained polarized reram-based in-situ computation for mixed-signal dnn accelerator.

Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. 2018a. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 184–199.

Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. Ternarybert: Distillation-aware ultra-low bit bert. *arXiv preprint arXiv:2009.12812*.

Xiaodong Zhang, Xu Sun, and Houfeng Wang. 2018b. Duplicate question identification by integrating framenet with neural networks. In *Proceedings of the Conference on Artificial Intelligence*, volume 32, New Orleans, Louisiana, USA. AAAI Press.