Northeastern University
Department of Electrical and Computer Engineering

2021 PhD Qualification Examination
in
Computer Engineering

Student's Name: Timothy Rupprecht

Problem Number: Problem 6

Submission Date: May 9, 2021

Question 1

"What do you think about the pros and cons of different types of model compression techniques on various types of platforms?"

Non-structured Pruning

Non-structured pruning is one of the first methods proposed for model compression. Model compression is motivated by a desire to (i) increase inference speeds, (ii) increase power efficiency, (iii) decrease model storage size, and (iv) maintain the performance of the baseline model. Objectives (i) and (ii) can be attained indirectly by decreasing the model storage size. One of the earliest works in non-structured pruning [4] attempted to achieve all four of these objectives by heuristically removing (pruning) redundant model weights from the fully connected and convolutional layers of LeNet-5 and AlexNet. They trained a model, and looked for layer weights that were close to zero, and then set those weights to zero (therefore pruning them). They would retrain with the sparser network. After repeating this process several times these researchers saw a reduction in the number of connections by 9x to 13x without compromising accuracy. In fact, the top1 and top5 accuracies of their models increased. These researcher's refined work [5] have seen success in implementing their work on FPGA [6] [7] and ASIC [6] devices for speech recognition tasks.

Despite successes with non-structured pruning for model compression, this methodology suffers trade offs between decreasing model storage size, and maintaining the performance of the baseline model. Maintaining (and increasing) the accuracy of the baseline model is easier for non-structured pruning than alternative methods due to the flexibility of pruning arbitrary weights from the model. However, to prune arbitrary weights from a network requires storing indices of the weights pruned in memory. The model uses these indices during the inference step to know which weights to drop from computation. Recent research from 2018 [12] showed that the refined work done in [5] actually causes a slow down in performance in a pruned AlexNet by one third the speed of the baseline model. Additionally, when the pruning ratio is high, roughly 50% of the final model storage size is from the stored indices. This can compromise our objective to increase power efficiency and according to [14] creates difficulty in applying the irregular sparsity of non-structured pruning on devices that run on parrallel GPUs or multi-core CPU systems. In a hypothetical embedded system where there is not enough RAM to store both the indices and model, the excess calls to storage to access the indices would cause a decrease in power efficiency.

Structured Pruning

Structured Pruning has been advanced in recent years as an alternative to non-structured pruning. [10] [11] [14] The seminal works [10] [11] propose concepts like filter, channel, and filter shape pruning which seek to remove computational overhead without relying on indices for arbitrary weights. A recent 2020 paper [14] compares and contrasts structured and non-structured pruning in more detail. These methods prune

whole filters, channels, and elements of the filters; these are changes to entire channels, and filters rather than arbitrary layer weights for pruning. That means that the storage compression doesn't need to account for the stored indices which is an enormous performance gain when compared to non-structured pruning.

There are downsides to using structured pruning over non-structured pruning. In structured pruning researchers do not see accuracy improvements from the resulting regularization and enforced sparsity; in fact accuracy degradation occurs when the pruning ratio is high.[13] In addition to this, designing which connections to eliminate can be a matter of trial and error, or in the case of [11] choosing which channels to prune requires solving an additional optimization step. This is relatively more difficult compared to non-structured heuristic pruning where it can be said that the network is learning on its own which weights are redundant (which leads to its irregular nature.)

Weight Quantization

Weight quantization is the second technique used for model compression of neural network architectures. Very generally speaking, model weights are represented as floating point numbers, and stored as floating point data types in memory. Weight quantizing forces model weights to be binary values, ternary values, or powers of two which reduces the storage requirements needed for the individual model weights. In [12] the researchers study binary and ternary quantization for the model weights and completely eliminate the computational overhead from multiplication steps increasing inference speeds. For these reasons, weight quantization methods are very hardware friendly both in terms of storage and inference speeds. The authors of ADMM-NN in [12] performed weight quantizing after pruning as a separate process that uses the same procedure as weight pruning and is explained more in the next question.

Weight quantization does not have drastic drawbacks. The 2018 work [12] claims that weight pruning has a higher potential for compression because there is more model redundancy in the redundancy of weights and layer connections than compared to redundancies in bit representation. Additionally it is claimed that the resulting sparse models from weight pruning can increase performance accuracy and this effect is not seen in weight quantization. However, so long as storage and memory remain valuable resources, weight quantization should be employed. The next technique surveyed for model compression relies on a process that combines both weight pruning and weight quantization.

A Combined Approach

The world of model compression was turned upside down in 2018 when researchers created the first combined weight pruning and weight quantization algorithm for model compression.[12] Because pruning and quantization perform compression using different methods, it was only natural to try and combine both methods into one framework. These researchers used non-structured pruning as their pruning technique, and saw a large increase in the amount of number of parameters pruned from their models compared to other non-structured pruning techniques such as [4]. This framework was called ADMM-NN because it solves both the pruning

and quantization problems using ADMM or alternating direction method of multipliers. ADMM works well with non-convex optimization problems with combinatorial constraints. Researchers in 2020 improved on this framework by combining weight quantization with structured pruning in a compression framework called ADMM-NN-S.[14] Combining quantizing with structured pruning proved to be an improvement over using the non-structured pruning of ADMM-NN and earlier works. Since the indices were no longer required, ADMM-NN-S results in models that require half to a quarter of the storage space compared to ADMM-NN depending on the datasets and models used in training.

The discoveries in [14] showed both ADMM-NN and ADMM-NN-S greatly increase the amount of compression from pruning, yet there is usually a negligible amount of accuracy loss - except in the case of AlexNet trained on ImageNet with ADMM-NN-S which actually sees an increase in top5 accuracy. The observed losses are comparable to alternative methods, but nonetheless it is a trade off between decreasing model storage size and maintaining the performance of the baseline model. It was also showed in [14] that the original framework of ADMM-NN from [12] saw a speed degradation of 20% and this was blamed on the non-structured nature of pruning and the overhead associated with using weight indices. As it turns out, the combined approaches do not cancel out the drawbacks associated with the pruning technique used. It was this observation that lead the authors of [14] to urge the research community to stop working with non-structured pruning.

Pattern-based Pruning

Pattern-based pruning proposed in [16] was presented in 2020, by some of the same researchers responsible for ADMM-NN and ADMM-NN-S. The introduction of pattern-based pruning seeks to balance the trade-offs between structured and non-structured pruning - combining the flexibility of non-structured pruning with the storage improvements of structured pruning. These patterns act as a mask for each kernel and allow only parts of the kernel to have an effect on the output. Connectivity pruning is also combined into this proposal which removes full kernels from the model similar in effect to filter pruning. Figure 2 in [16] shows some of these patterns and the effects of connectivity pruning. Pattern-based pruning combined with connectivity pruning performs extraordinarily well compared to alternative methods. On the CIFAR10 dataset the proposed pruning method was the only method to see an increase in performance in regards to accuracy for ResNet-18 and ResNet-50. In the cases where the methodology resulted in a loss of accuracy such as when training on ImageNet, the loss in accuracy is much smaller than competing or previous methods. Additionally, pattern-based pruning achieves a much higher pruning rate than competing methods. These experiments show how well pattern-based sparsity balances decreasing model storage size, and maintaining the performance of the baseline model. These advances see close to real-time inference speeds unmatched by competing algorithms.

When compared to previous works by the same authors, like the ADMM-NN-S framework [14] achieves higher compression rates compared to using pattern-based pruning. In the case of VGG-16 trained on

CIFAR10, ADMM-NN-S achieves a pruning rate of 50x while pattern-based pruning achieves a pruning rate of 8x to 19.7x (depending on the type of sparsity used). These differences may result in a model that requires less space; however it is hard to tell, because weight quantization was used in ADMM-NN-S unlike in the work with pattern-based pruning and may be responsible for the smaller storage size. Either way, ADMM-NN-S achieves a smaller overall model size. Additionally, the patterns themselves introduce a particular drawback: all the patterns are designed for 3x3 convolutional kernels! [16] Of course, a new pattern library could be developed for models that rely on 5x5 convolutional kernels, but it is better to focus on a strategy that will work for an arbitrary kernel size. That is where block-based pruning comes in.

Block-based Pruning

Block-based pruning was proposed in 2020 in the development of YOLObile.[15] Essentially, the researchers break their YOLOv4 model into blocks, and assign the same pattern to each kernel within the block. It's harder to compare block-based pruning to pattern-based as the research done with pattern-based pruning was performed on models for classification tasks with 3x3 kernels, and the work for block-based pruning is applied to YOLOv4 for object detection which uses more varying sized kernels. It is clear that block-based fixes the problems presented by pattern-based pruning as block-based can be applied to any kernel size within a model. This can lead to increases in compression especially for models like MobileNetv1 where more than 95% of weights belong to convolutional layers with non 3x3 kernel sizes. These layers represent roughly 95% of the computations made during inference. This can be seen in figure 3 of [15]. Because of the problem differences the authors do not use ADMM, but instead choose to use a reweighted group lasso method.

Lightweight DNN Frameworks

I have neglected to speak to an ever present compression technique that has been present in some of this surveyed research. In early 2020, sparse convolution patterns and connectivity pruning were introduced alongside a lightweight neural network framework.[13] The lightweight neural network framework takes these sparse network models called PCONVs and creates an optimized computation plan that sees increased speeds during inference. PCONV takes advantage of the fine-grained patterns inside the coarse-grained structures to outperform competing lightweight DNN frameworks such as TensorFlow-lite, TVM, and Alibaba Mobile Neural Network. A lightweight framework also works well with the pattern based sparsity formalized in [16] to reorganize the order of kernels to achieve faster inference speeds by eliminating overhead associated with loading kernel data from storage and into RAM. Figure 6 in [13] demonstrates the steps of reorganization that clearly results in a smaller storage size. By reordering computations based on the kernel pattern used, overhead can be eliminated from loading and reloading different patterns from storage during inference. A different lightweight framework was used for the research that used the block-based pruning method to create YOLObile.[15] This framework works well by deploying the model on both the mobile GPU and CPU and can work with models that have larger than 3x3 kernels.

The drawback of using a custom designed lightweight neural network framework is that it is custom made. For smaller teams at smaller universities this can potentially represent a major hurdle: needing to create new frameworks, update existing frameworks, or spending time on either of these only for the framework to lose relevancy due to new discoveries. Northeastern and the students who attend are lucky to have a curriculum that balances computer architecture, and hardware courses with software development. At smaller schools or where research teams have a more specific focus, these teams may be unable to make these advances on their own.

Question 2

"How do you suggest to build an automatic, systematic framework from an arbitrary DNN to an arbitrary platform for the maximum implementation efficiency?"

System Outline

It is clear from the surveyed literature that the earliest state of the art designs combine weight pruning and weight quantization [12] [14] and all continued research has focused on improving the method of weight pruning or usage of a lightweight neural network framework. [13] [16] [15] The surveyed research peaks with the introduction of pattern-based pruning [16] which was successfully used to prune weights from 3x3 convolutional kernels in models used for classification tasks, and block-based pruning [15] which was successfully used to prune weights from arbitrary layer kernels in YOLOv4 used for object detection tasks.

In this way there is no single unified framework to take any model and deploy it on any system. This answer will bring the reader's attention to important mathematical concepts and will develop a set of questions researchers can ask themselves when deciding how to achieve maximum implementation efficiency for a platform using those mathematical concepts. The answer will end with an example of using the proposed questions.

ADMM Problem Formulation

A lot of the state of the art works in Model Compression are using an ADMM problem formulation. [12] [14] [16] This problem formulation begins by writing out the basic minimization problem with layer weight constraints. The basic minimization problem is seen below:

$$\begin{aligned} \underset{\{\boldsymbol{W}_i\},\{\boldsymbol{b}_i\}}{\text{minimize}} \quad & f(\{\boldsymbol{W}_i\}_{i=1}^N, \{\boldsymbol{b}_i\}_{i=1}^N), \\ \text{subject to} \quad & \boldsymbol{W}_i \in S_i, \quad i = 1, ..., N \end{aligned} \tag{1}$$

When considering a weight tensor $\boldsymbol{X}$ we must construct a set S_i that corresponds to the compression technique surveyed here. For non-structured pruning use $\boldsymbol{W}_i \in S_i := \{\boldsymbol{X} | \#\text{ nonzero elements in } \boldsymbol{X} \leq \alpha_i\}$. For filter pruning use $\boldsymbol{W}_i \in S_i := \{\boldsymbol{X} | \#\text{ nonzero filters in } \boldsymbol{X} \leq \beta_i\}$. For channel pruning use $\boldsymbol{W}_i \in S_i :=$

$\{\boldsymbol{X}|\#$ nonzero channels in $\boldsymbol{X} \leq \gamma_i\}$. For filter-shape pruning use $\boldsymbol{W}_i \in S_i := \{\boldsymbol{X}|\#$ nonzero vectors in $\boldsymbol{X}_{:,1,1,1} \leq \theta_i\}$. For weight quantization use $\boldsymbol{W}_i \in S_i := \{q_{i,1}, q_{i,2}, q_{i,3}, ..., q_{i,M}\}$. Here α_i, β_i, γ_i and θ_i are hyperparameters for each layer i and $\boldsymbol{q}_{i,:}$ are the M quantization levels for each layer i. Additionally, $f(\{\boldsymbol{W}_i\}_{i=1}^N, \{\boldsymbol{b}_i\}_{i=1}^N)$ is the differentiable loss function used during training and $\{\boldsymbol{W}_i\}$ and $\{\boldsymbol{b}_i\}$ are the weights and biases of the model at layer i.

Right now the problem as defined with its constraints is difficult to solve, especially in the context the problem presents itself in: deep learning. To get around this difficulty, we will apply a variant of the augmented Lagrangian called ADMM. This optimization procedure breaks equation (1) into two subproblems by following these steps:

1. Create an indicator function:

$$g_i(\boldsymbol{W}_i) = \begin{cases} 0 & \text{if} \quad \boldsymbol{W}_i \in S_i \\ \infty & \text{otherwise} \end{cases} \tag{2}$$

2. Create axuliary variable $\boldsymbol{Z}_i^k$ which equals $\boldsymbol{W}_i$

3. Transform the optimization problem seen in equation (1) into two new optimization problems seen in equations (3) and (4).

$$\underset{\{\boldsymbol{W}_i\},\{\boldsymbol{b}_i\}}{\text{minimize}} \quad f(\{\boldsymbol{W}_i\}_{i=1}^N, \{\boldsymbol{b}_i\}_{i=1}^N) + \sum_{i=1}^N \frac{\rho_i}{2} ||\boldsymbol{W}_i - \boldsymbol{Z}_i^k + \boldsymbol{U}_i^k||_F^2 \tag{3}$$

$$\underset{\{\boldsymbol{Z}_i\}}{\text{minimize}} \quad \sum_{i=1}^N g_i(\boldsymbol{Z}_i) + \sum_{i=1}^N \frac{\rho_i}{2} ||\boldsymbol{W}_i - \boldsymbol{Z}_i^k + \boldsymbol{U}_i^k||_F^2 \tag{4}$$

$$\text{Where} \quad \boldsymbol{U}_i^k := \boldsymbol{U}_i^{k-1} + \boldsymbol{W}_i^k - \boldsymbol{Z}_i^k \tag{5}$$

By reforming the question using ADMM into two new problems we can iteratively solve them by freezing the auxiliary variable in the first problem, and then freezing the actual variable as we work with the auxiliary variable in the second problem. [14] states that this problem can be solved optimally and analytically for each layer i for $i = 0, 1, ..., N$ by using euclidean projection to map $\boldsymbol{W}_i^{k+1} + \boldsymbol{U}_i^k$ to the proper S_i called for in the previous paragraphs. For non-structured pruning the euclidean projection results in keeping α_i elements of $(\boldsymbol{W}_i^{k+1} + \boldsymbol{U}_i^k)$ with the largest values and setting the remaining weights to zero. For filter pruning requires first calculating $O_a = ||(\boldsymbol{W}_i^{k+1} + \boldsymbol{U}_i^k)_{a,:,:,:}||_F^2$ for $a = 1, ..., A_i$. We can subsequently keep the β_i elements from $(\boldsymbol{W}_i^{k+1} + \boldsymbol{U}_i^k)_{a,:,:,:}$ corresponding the β_i largest values of O_a. We set the remaining elements of $(\boldsymbol{W}_i^{k+1} + \boldsymbol{U}_i^k)_{a,:,:,:}$ to zero. Channel pruning is similar, where we keep the γ_i elements from $(\boldsymbol{W}_i^{k+1} + \boldsymbol{U}_i^k)_{:,b,:,:}$ corresponding to the largest values from $O_b = ||(\boldsymbol{W}_i^{k+1} + \boldsymbol{U}_i^k)_{:,b,:,:}||_F^2$. It is said by the authors that filter-shape / column pruning is similar.

The problem is not so simple for pattern-based pruning. The procedure for using pattern-based pruning requires a different initial problem formulation due to the differences in constraints associated with selecting a pattern from a library. Additionally, creating a pattern library is a process by itself. Simplifying the notation of [16], and changing variables to match the previous equations, we arrive at the new equations used for pattern-based pruning. The initial minimization problem is below:

$$\begin{aligned} \underset{\{\boldsymbol{W}\},\{\boldsymbol{z}\}}{\text{minimize}} \quad & f(\{\boldsymbol{W}_i \circ (\sum_{j=1}^{K} z_j \boldsymbol{M}_j)\}_{i=1}^{N}), \\ \text{subject to} \quad & z_j \in \{0,1\}, \forall J, \quad \sum_{j=1}^{K} z_j = 1; \end{aligned} \tag{6}$$

Here, z_j is the boolean selection variable for a pattern in a pattern library with K patterns. Also for pattern-based pruning, instead of applying ADMM by tying the weights to an auxiliary variable, we are going to tie the boolean selection variable to a continuous probabilistic auxiliary variable u. We also will use a new indicator function in light of the constraint changes; seen below:

$$g(\boldsymbol{u}) = \begin{cases} 0 & \text{if} \quad u \in [0,1], \forall j, \sum_{j=1}^{K} u_j = 1 \\ \infty & \text{otherwise} \end{cases} \tag{7}$$

Before applying ADMM we can now rewrite equation (6) in light of our new constraints and indicator function and arrive at equation (8) seen below:

$$\begin{aligned} \underset{\{\boldsymbol{W}\},\{\boldsymbol{u}\}}{\text{minimize}} \quad & g(\boldsymbol{u}) + f(\{W_i \circ (\sum_{j=1}^{K} z_j \boldsymbol{M}_j)\}_{i=1}^{N}), \\ \text{subject to} \quad & \boldsymbol{z} = \boldsymbol{u}; \end{aligned} \tag{8}$$

The augmented lagrangian of equation (8) can be given by:

$$\begin{aligned} \mathcal{L}(\boldsymbol{W}, \boldsymbol{z}, \boldsymbol{u}, \boldsymbol{\mu}) = & g(\boldsymbol{u}) + f(\{\boldsymbol{W}_i \circ (\sum_{j=1}^{K} z_j \boldsymbol{M}_j)\}_{i=1}^{N}) \\ & + \boldsymbol{\mu}^{(T)}(\boldsymbol{z} - \boldsymbol{u}) + \frac{\rho}{2} ||\boldsymbol{z} - \boldsymbol{u}||_F^2 \end{aligned} \tag{9}$$

It is here we can now break equation (9) into two subproblems that we can iteratively solve.

$$\boldsymbol{W}^{(t)}, \boldsymbol{z}^{(t)} = \underset{\boldsymbol{W},\boldsymbol{z}}{\text{minimize}} \quad \mathcal{L}(\boldsymbol{W}, \boldsymbol{z}, \boldsymbol{u}^{(t-1)}, \boldsymbol{\mu}^{(t-1)}) \tag{10}$$

$$\boldsymbol{u}^{(t)} = \underset{\boldsymbol{u}}{\text{minimize}} \quad \mathcal{L}(\boldsymbol{W}^{(t)}, \boldsymbol{z}^{(t)}, \boldsymbol{u}, \boldsymbol{\mu}^{(t-1)}) \tag{11}$$

$$\text{Where} \quad \boldsymbol{\mu}^{(t)} = \boldsymbol{\mu}^{(t-1)} + \rho(\boldsymbol{z}^{(t)} - \boldsymbol{u}^{(t)}) \tag{12}$$

When ADMM has converged we will have simultaneously selected patterns for each kernel, and trained the non-zero weights. Please see [16] for details on the analytical solution to the problem.

An Alternative Approach

YOLObile presented in [15] uses block pruning, and does not rely on ADMM for its implementation on a mobile device. In light of these problem differences and differences in constraints, the researchers pruned their models with a different approach as previous works. These researchers choose to adopt the reweighted group lasso method seen below:

$$\begin{aligned} &\underset{\boldsymbol{W}_i, \boldsymbol{b}_i}{\text{minimize}} \quad f(\boldsymbol{W}; \boldsymbol{b}_i) + \lambda \sum_{i=0}^{N} R(\boldsymbol{\alpha}_i^{(t)}, \boldsymbol{W}_i) \\ &\text{Where} \quad R(\boldsymbol{\alpha}_i^{(t)}, \boldsymbol{W}_i) = \sum_{j=1}^{K} \sum_{h=1}^{g_m^i} \sum_{w=1}^{g_n^i} ||\alpha_{ijn}^{(t)} \circ [\boldsymbol{W}_{ij}]_{h,w}||_F^2 \\ &\text{Where } \alpha_{ijn}^{(t)} \text{ is updated by} \quad \alpha_{ijn}^{(t)} = \frac{1}{||[\boldsymbol{W}_{ij}]_{h,w}||_F^2 + \epsilon} \end{aligned} \tag{13}$$

A Systematic Framework

Researchers can follow these systematic set of questions to determine the proper compression techniques to employ when implementing an arbitrary DNN model onto an arbitrary target system:

1. Is minimizing the final model storage size the most important consideration?
    - If yes, use ADMM-NN-S with structured pruning finishing with binary weight quantization, then skip to question 4.
      (See: equations (1) - (5) and follow [14])
    - If no, continue
2. Is the model mostly comprised of 3x3 convolutional layers?
    - If yes, use pattern-based pruning and continue.
      (See: equations (6) - (12) and follow [16])
    - If no, use block-based pruning and continue.
      (See: equation (13) and follow [15])
3. Does the target hardware support binary multiplication?
    - If yes, use ADMM-NN-S for binary weight quantization
      (See: equations (1) - (5) and follow [14])
    - if no, continue.
4. Does the model need to be deployed on a mobile platform?
    - If yes, use the lightweight framework used in [15]
    - If no, use the lightweight framework proposed in [13] or [16].

- If no framework is compatible with your target system ignore

As one example, consider compressing a VGG-16 model for performance on an FPGA. The proposed questions direct us to follow the ADMM-NN-S framework both for structured weight pruning, and binary weight quantization using equations (1) - (5). The researchers in [12] followed a similar path and successfully implemented a much larger VGG-16 model that used non-structured pruning onto a FPGA.

Question 3

"How do you think model compression can facilitate training acceleration of DNNs besides inference acceleration?"

Using ADMM

As stated in the answers to the prior questions, many state of the art methods rely on a ADMM problem formulation to both quantize their weights, and prune their model weights. In general, ADMM methods are good for convergence, as the original weight pruning method to use ADMM [12] cites both [1] [2] in saying ADMM has fast convergence properties. As [1] states "it is often the case that ADMM converges to modest accuracy—sufficient for many applications—within a few tens of iterations."

The 2020 work [14] introduces at least two concepts to theoretically increase training speeds while using ADMM. They propose (i) increasing the ρ hyperparamter in equations (3) and (4) as ADMM iterates and (ii) a concept called progressive pruning. Increasing the ρ hyperparamter is self-explanatory. Progressive weight pruning is also a simple step with enormous payoffs. After one round of ADMM iterations there should be a large amount of very small weights that have not been pruned. If one forces those weights to become zero (thus pruning them) before continuing to the next iteration we see accelerated convergence during training. In practice the authors of [14] suggest two rounds of progressive weight pruning. [14] states that using a pretrained model and training it in this manner to prune and quantize its weights takes 8 - 14 ADMM iterations which corresponds to 100 - 150 epochs of training in the pytorch framework. This is comparable to training a model from scratch using the pytorch framework and with the resulting performance enhancements the time commitment is clearly worth it.

Hardware Speed-ups

It has already been alluded to in question 1's answer but there exists a lot of opportunity to improve model compression by making design decisions at the hardware level. Two options arise (i) adopting/developing a lightweight neural network framework for implementation [13][15] and (ii) by using weight binary or ternary quantization to eliminate floating point multiplication which can be computationally inefficient compared to binary multiplication.[3][8][9][12] It is the opinion of this author that so long as the target hardware supports

binary multiplication binary weight quantization should be used by employing ADMM following equations (1) - (5) using the correct S_i for weight quantization: $W_i \in S_i := \{q_{i,1}, q_{i,2}, q_{i,3}, ..., q_{i,M}\}$.

Bibliography

[1] S. Boyd et al. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Foundations and Trends in Machine Learning* 3 (2010). URL: https://stanford.edu/class/ee367/reading/admm_distr_stats.pdf.

[2] Hua Ouyang et al. "Stochastic Alternating Direction Method of Multipliers". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 1. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 80–88. URL: http://proceedings.mlr.press/v28/ouyang13.html.

[3] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "BinaryConnect: Training Deep Neural Networks with binary weights during propagations". In: *CoRR* abs/1511.00363 (2015). arXiv: 1511.00363. URL: http://arxiv.org/abs/1511.00363.

[4] Song Han et al. "Learning both Weights and Connections for Efficient Neural Networks". In: *CoRR* abs/1506.02626 (2015). arXiv: 1506.02626. URL: http://arxiv.org/abs/1506.02626.

[5] Song Han, Huizi Mao, and William J. Dally. *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. 2016. arXiv: 1510.00149 [cs.CV].

[6] Song Han et al. "EIE: Efficient Inference Engine on Compressed Deep Neural Network". In: *CoRR* abs/1602.01528 (2016). arXiv: 1602.01528. URL: http://arxiv.org/abs/1602.01528.

[7] Song Han et al. "ESE: Efficient Speech Recognition Engine with Compressed LSTM on FPGA". In: *CoRR* abs/1612.00694 (2016). arXiv: 1612.00694. URL: http://arxiv.org/abs/1612.00694.

[8] Itay Hubara et al. "Binarized Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper/2016/file/d8330f857a17c53d217014ee776bfd50-Paper.pdf.

[9] Mohammad Rastegari et al. "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks". In: *CoRR* abs/1603.05279 (2016). arXiv: 1603.05279. URL: http://arxiv.org/abs/1603.05279.

[10] Wei Wen et al. "Learning Structured Sparsity in Deep Neural Networks". In: *CoRR* abs/1608.03665 (2016). arXiv: 1608.03665. URL: http://arxiv.org/abs/1608.03665.

[11] Yihui He, Xiangyu Zhang, and Jian Sun. "Channel Pruning for Accelerating Very Deep Neural Networks". In: *CoRR* abs/1707.06168 (2017). arXiv: 1707.06168. URL: http://arxiv.org/abs/1707.06168.

[12] Ao Ren et al. "ADMM-NN: An Algorithm-Hardware Co-Design Framework of DNNs Using Alternating Direction Method of Multipliers". In: *CoRR* abs/1812.11677 (2018). arXiv: 1812.11677. URL: http://arxiv.org/abs/1812.11677.

[13] Xiaolong Ma et al. "PCONV: The Missing but Desirable Sparsity in DNN Weight Pruning for Real-time Execution on Mobile Devices". In: *CoRR* abs/1909.05073 (2019). arXiv: 1909.05073. URL: http://arxiv.org/abs/1909.05073.

[14] Yetang Wang et al. *Non-structured DNN Weight Pruning Considered Harmful*. July 2019.

[15] Yuxuan Cai et al. "YOLObile: Real-Time Object Detection on Mobile Devices via Compression-Compilation Co-Design". In: *CoRR* abs/2009.05697 (2020). arXiv: 2009.05697. URL: https://arxiv.org/abs/2009.05697.

[16] Xiaolong Ma et al. "An Image Enhancing Pattern-based Sparsity for Real-time Inference on Mobile Devices". In: *CoRR* abs/2001.07710 (2020). arXiv: 2001.07710. URL: https://arxiv.org/abs/2001.07710.